Université de Montréal

Feature Extraction on Faces: From Landmark Localization to Depth Estimation

par

Sina Honari

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

Décembre 2018

Resumé

Le sujet de cette thèse porte sur les algorithmes d'apprentissage qui extraient les caractéristiques importantes des visages. Les caractéristiques d'intérêt principal sont des points clés; La localisation en deux dimensions (2D) ou en trois dimensions (3D) de traits importants du visage telles que le centre des yeux, le bout du nez et les coins de la bouche. Les points clés sont utilisés pour résoudre des tâches complexes qui ne peuvent pas être résolues directement ou qui requièrent du guidage pour l'obtention de performances améliorées, telles que la reconnaissance de poses ou de gestes, le suivi ou la vérification du visage. L'application des modèles présentés dans cette thèse concerne les images du visage; cependant, les algorithmes proposés sont plus généraux et peuvent être appliqués aux points clés de d'autres objets, tels que les mains, le corps ou des objets fabriqués par l'homme. Cette thèse est écrite par article et explore différentes techniques pour résoudre plusieurs aspects de la localisation de points clés.

Dans le premier article, nous démêlons l'identité et l'expression d'un visage donné pour apprendre une distribution à priori sur l'ensemble des points clés. Cette distribution à priori est ensuite combinée avec un classifieur discriminant qui apprend une distribution de probabilité indépendante par point clé. Le modèle combiné est capable d'expliquer les différences dans les expressions pour une même représentation d'identité.

Dans le deuxième article, nous proposons une architecture qui vise à conserver les caractéristiques d'images pour effectuer des tâches qui nécessitent une haute précision au niveau des pixels, telles que la localisation de points clés ou la segmentation d'images. L'architecture proposée extrait progressivement les caractéristiques les plus grossières dans les étapes d'encodage pour obtenir des informations plus globales sur l'image. Ensuite, il étend les caractéristiques grossières pour revenir à la résolution de l'image originale en recombinant les caractéristiques du chemin d'encodage. Le modèle, appelé Réseaux de Recombinaison, a obtenu l'état de l'art sur plusieurs jeux de données, tout en accélérant le temps d'apprentissage.

Dans le troisième article, nous visons à améliorer la localisation des points clés lorsque peu d'images comportent des étiquettes sur des points clés. En particulier, nous exploitons une forme plus faible d'étiquettes qui sont plus faciles à acquérir ou plus abondantes tel que l'émotion ou la pose de la tête. Pour ce faire, nous proposons une architecture permettant la rétropropagation du gradient des étiquettes les plus faibles à travers des points clés, ainsi entraînant le réseau de localisation des points clés. Nous proposons également une composante de coût non supervisée qui permet des prédictions de points clés équivariantes en fonction des transformations appliquées à l'image, sans avoir les vraies étiquettes des points clés. Ces techniques ont considérablement amélioré les performances tout en réduisant le pourcentage d'images étiquetées par points clés.

Finalement, dans le dernier article, nous proposons un algorithme d'apprentissage permettant d'estimer la profondeur des points clés sans aucune supervision de la profondeur. Nous y parvenons en faisant correspondre les points clés de deux visages en les transformant l'un vers l'autre. Cette transformation nécessite une estimation de la profondeur sur un visage, ainsi que une transformation affine qui transforme le premier visage au deuxième. Nous démontrons que notre formulation ne nécessite que la profondeur et que les paramètres affines peuvent être estimés avec un solution analytique impliquant les points clés augmentés par profondeur. Même en l'absence de supervision directe de la profondeur, la technique proposée extrait des valeurs de profondeur raisonnables qui diffèrent des vraies valeurs de profondeur par un facteur d'échelle et de décalage. Nous démontrons des applications d'estimation de profondeur pour la tâche de rotation de visage, ainsi que celle d'échange de visage.

Mots-clés: réseaux neuronaux, apprentissage profond, réseaux neuronaux de convolution, apprentissage supervisé, apprentissage non-supervisé, apprentissage semi-supervisé, architectures grossières à fines, localisation de points clés, estimation de la profondeur, rotation de visage, échange de visage.

Abstract

This thesis focuses on learning algorithms that extract important features from faces. The features of main interest are landmarks; the two dimensional (2D) or three dimensional (3D) locations of important facial features such as eye centers, nose tip, and mouth corners. Landmarks are used to solve complex tasks that cannot be solved directly or require guidance for enhanced performance, such as pose or gesture recognition, tracking, or face verification. The application of the models presented in this thesis is on facial images; however, the algorithms proposed are more general and can be applied to the landmarks of other forms of objects, such as hands, full body or man-made objects. This thesis is written by article and explores different techniques to solve various aspects of landmark localization.

In the first article, we disentangle identity and expression of a given face to learn a prior distribution over the joint set of landmarks. This prior is then merged with a discriminative classifier that learns an independent probability distribution per landmark. The merged model is capable of explaining differences in expressions for the same identity representation.

In the second article, we propose an architecture that aims at uncovering image features to do tasks that require high pixel-level accuracy, such as landmark localization or image segmentation. The proposed architecture gradually extracts coarser features in its encoding steps to get more global information over the image and then it expands the coarse features back to the image resolution by recombining the features of the encoding path. The model, termed Recombinator Networks, obtained state-of-the-art on several datasets, while also speeding up training.

In the third article, we aim at improving landmark localization when only a few images with labelled landmarks are available. In particular, we leverage a weaker form of data labels that are easier to acquire or more abundantly available such as emotion or head pose. To do so, we propose an architecture to backpropagate gradients of the weaker labels through landmarks, effectively training the landmark localization network. We also propose an unsupervised loss component

which makes equivariant landmark predictions with respect to transformations applied to the image without having ground truth landmark labels. These techniques improved performance considerably when we have a low percentage of labelled images with landmarks.

Finally, in the last article, we propose a learning algorithm to estimate the depth of the landmarks without any depth supervision. We do so by matching landmarks of two faces through transforming one to another. This transformation requires estimation of depth on one face and an affine transformation that maps the first face to the second one. Our formulation, which only requires depth estimation and affine parameters, can be estimated as a closed form solution of the 2D landmarks and the estimated depth. Even without direct depth supervision, the proposed technique extracts reasonable depth values that differ from the ground truth depth values by a scale and a shift. We demonstrate applications of the estimated depth in face rotation and face replacement tasks.

Keywords: neural networks, deep learning, convolutional networks, supervised learning, unsupervised learning, semi-supervised learning, coarse-to-fine architectures, landmark localization, depth estimation, face rotation, face replacement.

Contents

List of Tables

List of Figures

Acknowledgments

The journey of my PhD was one of the biggest challenges of my life, filled with ups and downs, with experiences both pleasant and difficult. Accomplishing this journey was not possible without the help of many people. I would like to thank my advisors Pascal Vincent and Christopher Pal, who accepted to supervise me and were supportive during the difficult moments of my PhD. I especially appreciate all of the constructive discussions we had and the confidence they gave me to pursue my research. I am also thankful for all the freedom they gave me in pursuit of my interests, which made my PhD an enjoyable and constructive experience.

I would also like to thank Jason Yosinski for his help on my FQRNT grant application and also his guidance and insights for the work we did together. His feedback was very constructive in helping me understand the research cycle and how to deal with the application challenges. I am also thankful to my other co-authors, Pavlo Molchanov, Stephen Tyree, Jan Kautz, Christopher Beckham, Joel Moniz, Simon Rajotte, and Md Kamrul Hasan, without whom this thesis could not have been written. Especially, I would like to thank Christopher Beckham for being both a loyal collaborator and a good friend. I am also thankful to him for proofreading this thesis.

I would also like to thank to Pascal Lamblin, Frédéric Bastien, and Simon Lefrancois for maintaining the computing and software infrastructure at Mila. I am grateful for their patience on many occasions regarding software and resource related issues throughout the course of my PhD. I would like to thank my Mila colleagues and friends, who made my PhD an enjoyable and unforgettable experience, such as Mohammad Pezeshki, Reyhane Askari, Joseph Paul Cohen, Margaux Luck, Tristan Sylvain, David Vazquez, Faruk Ahmed, Caglar Gulcehre, Soroush Mehri, Alexandre de Brébisson, Samira Shabanian, and Chiheb Trabelsi just to name a few. I am also thankful to Gabriele Prato and Samuel Lavoie for their help in translating the abstract of this thesis to French.

I am grateful to my Montréal based friends Maziar, Ali, Susan, Reza, and Shima for their entire support and the courage they gave me during my difficult moments in Montréal.

Finally, I would like to thank my family for their unconditional support throughout the entire course of my studies, without which I could not have made it so far.

Chapter 1

Background

Faces convey a lot of information about others. People are identified by their faces and we understand how others feel through facial expressions. Extracting features on faces can help better address face related tasks. The features of interest that are extracted in this thesis are landmarks or keypoints; the two dimensional (2D) or three dimensional (3D) features that explain important facial locations such as eye centers, nose tip, mouth corners or even face contour. Finding landmarks on images, which corresponds to finding their 2D or 3D locations is often termed landmark localization or keypoint detection.

Landmark localization is used to solve other tasks of interest that cannot be solved directly or when the results on them are of inferior quality without usage of landmarks. In the case of faces, many tasks including identity recognition, head pose estimation, face rotation, face replacement, and eye gaze estimation can benefit from landmarks. In the case of hands, gesture recognition and tracking in videos are applications that are currently solved mainly by using landmarks. In the case of entire bodies, human pose estimation, activity recognition in videos, and virtual clothes replacement of a person (e.g. for the fashion and clothing industry) are applications of full body landmark localization. While landmark localization can be also applied to other objects, we pursue only applications of landmark localization on faces in this thesis.

Machine learning algorithms and more recently deep learning algorithms are used to solve landmark localization problems, with the latter currently achieving state-of-the-art results. When I started my PhD, deep learning algorithms were not yet the de facto standard techniques for landmark localization. My first work, presented in Chapter 4, in this thesis uses more general machine learning algorithms. However, my later works, presented in Chapters 6, 8, and 10, leverage deep learning methods to propose new learning techniques. Due to usage of learning algorithms in this thesis, I

provide an introduction to machine and deep learning algorithms in this chapter, focusing more on the algorithms that are used in later chapters.

This thesis is written by article, and whenever the narrator is referred to as "we" it refers to the collective opinion of the authors on that article. Any other writing outside the articles is my own and reflects my personal opinion. Throughout this thesis the terms ‘landmark’ and ‘keypoint’ are used interchangeably.

1.1. Machine Learning

Learning is the process of obtaining new knowledge, skills, capabilities or improving them over time (Gross, 2015). Artificial Intelligence or machine intelligence is the process of bringing intelligence to machines to make them capable of tasks that are usually associated with intelligence beings, mainly humans, such as reasoning, decision making or rational behaviours in new situations (Copeland, 2018). By bringing learning into machines, one can minimize the degree of human intervention and solve problems where rule-based methods fail.

Machine Learning (ML) is the study of algorithms that are capable of improving their performance over time on specific sets of tasks (which require some level of intelligence to be performed) by learning from examples (their training data) such that they can generalize to new and unseen data. These algorithms mostly learn by optimizing a given objective function in the course of training. ML methods are used to learn meaningful latent patterns in data, to predict or forecast a phenomenon on new and unseen data, to learn a distribution of the data (e.g. to generate new examples from the data distribution or to detect outliers in the data), to act autonomously in a complex environment (such as navigation, autonomous driving, or gaming), and also to interact autonomously and rationally with other entities in an environment such as in dialogue systems.

Depending on whether the data is labeled or not, the ML techniques are broadly divided into supervised, unsupervised, and semi-supervised learning methods. In supervised learning, a set of labeled examples is provided in the form of $(\mathbf{x}, \mathbf{y})$ pairs, where the goal is to learn a function $f : \mathbf{X} \rightarrow \mathbf{Y}$, such that it can predict accurately the output $\mathbf{y}' \in \mathbf{Y}$ on an unseen example $\mathbf{x}' \in \mathbf{X}$. On the other hand, in unsupervised learning, the data is not labeled and the goal is to find some pattern in the data (e.g. cluster), to reduce the dimensionality of the data (e.g. for visualization or compression tasks), to obtain the main directions of variations in data, or to model its distribution in some way.

In semi-supervised learning methods the task to learn is the same as in supervised learning, but only part of the data is labeled and the rest of the data is unlabeled. The unlabeled data can be used to find some intermediate representations or to first train a model in an unsupervised style in order to put its parameters in a space where the subsequent supervised optimization is simplified, as a way to regularize the network in the case of a limited amount of labels.

There are also reinforcement learning (RL) techniques, where the "labels" (usually in form of rewards) come sparsely in the course of training. In RL the learner is dealing interactively with an environment and the data arrives sequentially in time, as a function of the learner's exploration policy in the environment. The learner has to find a way to assign the reward to the actions leading to that reward. This is known as the credit assignment problem in RL methods. The learner's goal is to enhance its performance as more data arrives in order to maximize its overall reward. In RL methods the policy pursued by the agent in the environment directly affects the data it collects. Thereupon, the learning process also involves finding a trade-off in between exploitation, where the learner uses its knowledge greedily to get higher rewards, and exploration, where the learner risks taking new actions or examining new states with the hope of finding more optimal solutions that will lead to higher rewards in the future.

1.1.1. Components of a Learning Algorithm

The most important elements of a learning algorithm are data, the hypothesis function or the model, the objective function, and the optimization algorithm. Below I describe each one of these elements.

Data:

Any learning algorithm leverages a set of data (or dataset) that is gathered from the underlying, commonly unknown, distribution of data p_{data}. The data is often assumed to be gathered with i.i.d assumption, which stands for independently and identically distributed. It implies that each example (or data point) in the dataset is gathered independently from other examples in the dataset and also sampled from the same underlying data distribution p_{data}. The identically distributed assumption is important in having the training algorithm generalize to unseen data at test time. Otherwise, if the underlying distribution of train and test data are different, the model can generally not be expected to perform well (or generalize) to test data.

In unsupervised learning, the dataset is in the form of $\mathbf{x}_n,\ n \in \{1, \ldots, N\}$, with N data points, each with dimensionality M. In case of images, for example, M is $3 \times row \times column$ with $\mathbf{x}_n$ containing the intensity of each pixel. The dataset can be presented as a matrix $\mathbf{X}$ of size $N \times M$,

with each row representing a different data point with M features. Formally, the i.i.d assumption on such data can be expressed as:

$$p_{data}(\mathbf{X}) = \prod_{n=1}^{N} p_{data}(\mathbf{x}_n) \tag{1.1.1}$$

In supervised learning, for each data point, there is also a label or a set of labels $\mathbf{y}$, e.g. indicating the class of the data. The dataset is in form of $(\mathbf{x}_n, \mathbf{y}_n),\ n \in \{1, \ldots, N\}$ where $(\mathbf{x}_n, \mathbf{y}_n)$ is the n^{th} paired labeled data. The set of labels can be shown as a vector $\mathbf{Y}$ of size N, with each row providing the label for a different data point. Similarly, the i.i.d assumption on such data can be shown as:

$$p_{data}(\mathbf{X}, \mathbf{Y}) = \prod_{n=1}^{N} p_{data}(\mathbf{x}_n, \mathbf{y}_n) \tag{1.1.2}$$

A common practice in training is to split a dataset into three subsets, a training, a validation, and a test set, with no intersection. The training set is the only subset on which the learning algorithm is trained. The validation set, represents a subset of data that comes from the same underlying distribution p_{data} to indicate how the learning algorithm will perform on an unseen subset at test time. The validation set is not used directly to train a model, however, it is used to select the parameters of the training algorithm that impact the model's performance but cannot be optimized by the learning algorithm itself. These sets of variables are known as hyper-parameters. For example, the learning rate and the number of iterations used to train an algorithm are hyper-parameters. The validation set, is therefore used to pick the best hyper-parameters, by informing us on the performance of the model on an unseen subset of data from the same distribution. Finally once the model is trained and the hyper-parameters are selected, the model's performance is reported on the test set. The test set is not touched during training or hyper-parameter selection such that the reported performance on the test set represents to best approximation the performance of the model on an unseen subset of data.

Model:

A model or hypothesis is a function or a set of functions that are potential solutions for the task of interest. It is the central component of a learning algorithms. In supervised learning, one can

think of a hypothesis as learning a function $f_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} : \mathbf{X} \to \mathbf{Y}$ that for each data point $\mathbf{x}_n$ estimates the corresponding label $\mathbf{y}_n$. A large variety of hypothesis functions are introduced including linear regression, logistic regression, kernel methods, support vector machines, multi-layer perceptron, convolutional neural networks, and recurrent neural networks, where all estimate the labels $\mathbf{y}$, but differ in their model representation and the characteristics of their training algorithm.

In unsupervised learning there are also a large body of models, associated with various unsupervised learning tasks such as:

- *Clustering*: the training data are grouped into subsets based on their similarity.
- *Anomaly Detection*: examples outside of the data distribution are detected, such as for detecting spams in e-mails.
- *Dimensionality Reduction*: data is projected from a higher dimensional space to a lower orthogonal dimensional space for data compression, or finding the main components in data that can explain the observed variations. Examples of such methods are Principal Component Analysis (PCA) and auto-encoders.
- *Learning Explicit Distribution of Data*: a probability distribution is fit to the training data. One can maximize the probability of data under some probability distribution function, e.g. a Gaussian mixture model:

$$\underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\arg\max}\ p_{model}(\mathbf{X};\boldsymbol{\theta}) \tag{1.1.3}$$

  Once we have the distribution of data, we can easily generate new samples from it, however it is not trivial to have a probability distribution that explains all the variations in data.

Objective Function:

The objective function indicates which criteria to minimize or maximize during the course of training. If the objective is minimized, it is referred to as the loss or cost function $\mathcal{L}$. If it is maximized, it is referred to as utility or fitness function.

In supervised learning, the goal is to learn a function $f : \mathbf{X} \to \mathbf{Y}$ which predicts the output $\mathbf{y} \in \mathbf{Y}$ for a given input $\mathbf{x} \in \mathbf{X}$. The loss function $\mathcal{L}$ measures the average difference between the

function's output $f_{\boldsymbol{\theta}}(\mathbf{x}) = \tilde{\mathbf{y}}$ and the ground truth (gt) label $\mathbf{y}$. The parameters $\boldsymbol{\theta}$ in f are optimized in order to minimize the loss function defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{y} \in \mathcal{D}_{\text{train}}} \mathcal{L}(\mathbf{y}, \tilde{\mathbf{y}}; \boldsymbol{\theta}). \tag{1.1.4}$$

Depending on the nature of data in $\mathbf{Y}$ and the model being used different loss functions can be defined. If $\mathbf{y}$ is discrete the following loss functions are commonly defined:

- If $y \in \{0, 1\}$ and $f_{\boldsymbol{\theta}}(\mathbf{x})$ measures the probability of each data point $\mathbf{x}$, a *binary cross-entropy* loss can be defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}, y \in \mathcal{D}_{\text{train}}} y \log f_{\boldsymbol{\theta}}(\mathbf{x}) + (1 - y) \log(1 - f_{\boldsymbol{\theta}}(\mathbf{x})). \tag{1.1.5}$$

- If $y \in \{-1, 1\}$ and $f_{\boldsymbol{\theta}}(\mathbf{x})$ is a real valued output, a *hinge* loss can be defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}, y \in \mathcal{D}_{\text{train}}} \max(0, 1 - y \cdot f_{\boldsymbol{\theta}}(\mathbf{x})). \tag{1.1.6}$$

  here if y and $f_{\boldsymbol{\theta}}(\mathbf{x})$ have the same sign and $|f_{\boldsymbol{\theta}}(\mathbf{x})| > 1$, the loss is zero. Otherwise, the greater the distance between y and $f_{\theta}(\mathbf{x})$, the higher the loss.

- If y can take a categorical label over K classes, a *multi-class cross-entropy* loss can be defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}, y \in \mathcal{D}_{\text{train}}} \sum_{k=1}^{K} 1(y = k) \left[-\log(f_{\boldsymbol{\theta}}(\mathbf{x})_k) \right], \tag{1.1.7}$$

  with $1(y = k)$ being an indicator function that equals to one if y has class label k, and otherwise it is zero. $f_{\boldsymbol{\theta}}(\mathbf{x})_k$ indicates the probability output by the model for class k.

If y is a continuous value, the following loss functions are commonly used:

- L^P *distance function*: for a label $\mathbf{y}$ of dimensionality J, L^P distance is defined as

$$L^P(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x})) = \big(\sum_{j \in J} |\mathbf{y}_j - f_{\boldsymbol{\theta}}(\mathbf{x})_j|^p \big)^{1/p}. \tag{1.1.8}$$

If $p = 1$, the loss over the training data measures $L1$ distance between $\mathbf{y}$ and $\tilde{\mathbf{y}}$ and is defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x},\mathbf{y} \in \mathcal{D}_{\text{train}}} \sum_{j \in J} |\mathbf{y}_j - f_{\boldsymbol{\theta}}(\mathbf{x})_j|. \tag{1.1.9}$$

If $p = 2$, the loss over the training data measures Euclidean distance between $\mathbf{y}$ and $\tilde{\mathbf{y}}$ and is defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x},\mathbf{y} \in \mathcal{D}_{\text{train}}} \big(\sum_{j \in J} [\mathbf{y}_j - f_{\boldsymbol{\theta}}(\mathbf{x})_j]^2 \big)^{\frac{1}{2}}. \tag{1.1.10}$$

Usually the square-root in the above equation is not taken, in which case the loss is termed as mean squared error (MSE).

- *Cosine similarity*: It measures the cosine between two vectors $\mathbf{y}$ and $\tilde{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x})$. This metric measures how much the orientation of the two vectors are similar regardless of their magnitude. If the angle between the two vectors is zero, it gets a similarity value of 1, indicating the highest similarity, and if the angle is π, it gets a similarity value of 0, indicating no orthogonality or decorrelation. If the angle is 2π, it gets a similarity value of -1, indicating opposite similarity. The similarity changes monotonically between zero to 2π angles. The objective based on minimizing the cosine distance is defined as:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) =& \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{y} \in \mathcal{D}_{\text{train}}} \frac{\mathbf{y} \cdot f_{\boldsymbol{\theta}}(\mathbf{x})}{\|\mathbf{y}\| \|f_{\boldsymbol{\theta}}(\mathbf{x})\|} \\ =& \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{y} \in \mathcal{D}_{\text{train}}} \frac{\sum_{j \in J} \mathbf{y}_j f_{\boldsymbol{\theta}}(\mathbf{x})_j}{[\sum_{j \in J} {\mathbf{y}_j}^2]^{\frac{1}{2}} [\sum_{j \in J} {f_{\boldsymbol{\theta}}(\mathbf{x})_j}^2]^{\frac{1}{2}}}. \end{aligned} \tag{1.1.11}$$

Optimization:

Once the data is available and the model and the objective function are defined, the model is fit to the training data using an optimization algorithm that maximizes the objective function or minimizes the loss. This means finding the parameters of the model that minimizes or maximizes

the function of interest. In the remaining part of this section, I assume the loss function is minimized. The arguments can be extended to the objective functions that should be maximized.

To train the parameters of the model, first the loss is measured over training data, which indicates the degree of error of the model. Then, the gradient of the loss with respect to (w.r.t) the parameters is estimated. If the optimization solution for the parameters can be written as a closed form solution of the training data, the parameters can be directly estimated. However, for most of the models the optimal solution cannot be expressed in closed form, or there exist also unknown variables, such as the latent variables in the EM algorithm (explained in Section 1.1.2 in Factor Analysis sub-section). In such cases the parameters are updated in multiples steps until the error cannot be further minimized.

If $\mathcal{L}$ is the loss and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i^t}$ indicates the partial derivative of the loss w.r.t. to parameter $\boldsymbol{\theta}_i$ after t updates, then $\boldsymbol{\theta}_i$ at time step $t+1$ is updated by

$$\boldsymbol{\theta}_i^{t+1} = \boldsymbol{\theta}_i^t - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i^t}, \tag{1.1.12}$$

with η indicating the learning rate. In *batch gradient descent* the loss over the entire training data is measured first and then parameters are updated. By using the batch gradient descent the parameters are updated in the direction that minimizes the loss over the entire dataset. However, the computation can be slow, since it requires a full sweep over the entire training data. *Stochastic or online gradient descent*, on the other hand, updates the parameters after seeing only one training example. While it updates the parameters faster, it is more noisy since the gradient update direction for each data point is not the same as the direction over the entire training data. However, it is an unbiased estimator of the gradient direction and also the noise helps bypassing plateaus and local minimas. In practice, people commonly use a mini-batch, e.g. of size 32.

Gradient descent methods finally converge when they reach a local minima, however, they are not guaranteed to reach the global minima if the loss function is not convex w.r.t its parameters. A common remedy is to use gradient descent by initializing parameters randomly with different values and let each model converge and finally choose the best performing model.

Generalization:

The goal of many machine learning algorithms is to generalize to unseen data from the same distribution as the training set p_{data}, that have not been seen during training. However, if the training algorithm is able to fit too perfectly the model to the training data, it can memorize the data, or it can fit to the noise of the training data rather than learning the general patterns in the data. In such cases, the trained model will not generalize well to unseen data. This phenomenon is known as over-fitting.

The validation set is used to measure over-fitting. Formally, the model h_1 is said to be over-fitting compared to h_2 if on the training set $\mathcal{L}_{h_1}(\mathcal{D}_{\text{train}}) < \mathcal{L}_{h_2}(\mathcal{D}_{\text{train}})$ but on the validation set $\mathcal{L}_{h_1}(\mathcal{D}_{\text{valid}}) > \mathcal{L}_{h_2}(\mathcal{D}_{\text{valid}})$. In such cases a regularization method is applied to avoid overfitting. The following methods are commonly applied as regularization techniques:

- **Reducing the training time (early stopping)**: models that are optimized using iterative optimization procedures fit more to the training data as the training goes on. If they have too much capacity they can overfit to the training data. One regularization trick is to do early stopping, which is stopping the training procedure when the error on the validation set keeps increasing significantly. The model that gives the lowest error on the validation set is selected and retained as the trained model.
- **Increasing the training data (data augmentation)**: The more training examples we have relative to the number of learnable parameters of the model, the less likely the model will overfit. If more training data cannot be gathered, data augmentation techniques can be used, where some transformations are applied to the input given that it does not invalidate the input data (such as adding too much noise) or change the class label. An example of valid data augmentation is applying an affine transformation such as scaling, rotation, and translation to the training images in image classification tasks. Such transformations will increase the variation of the data observed by the model which helps reducing overfitting.
- **Restricting Model's parameters:** The more a model's parameters are free to change, the more it can adapt to smaller details of the data and the more easily it can overfit to the data. The following regularizations are applied to the model's parameters to restrict their freedom to change. From a Bayesian point of view, these regularizations act as a prior that shapes the posterior distribution of the hypothesis learned by the learning algorithm.

– **L1 regularization**: It applies a L1 loss to the model's parameters. The total loss for training the model is then written as

$$\mathcal{L}_{total} = \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) + \gamma \sum_{p=1}^{P} |\boldsymbol{\theta}_p|, \tag{1.1.13}$$

where P is the number of parameters of the model and γ is the regularization coefficient. L1 loss encourages the parameters that do not impact the model's prediction to be as close to zero as possible.

– **L2 regularization**: It applies a L2 loss to the model's parameters. The total loss for training the model is then written as

$$\mathcal{L}_{total} = \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}) + \gamma \sum_{p=1}^{P} \boldsymbol{\theta}_p{}^2. \tag{1.1.14}$$

L2 penalizes mostly the big parameters, therefore constraining them to be small, but not necessarily zero.

- **Dropout (Srivastava et al., 2014)**: This regularization, which is mostly used in feed-forward neural networks, applies multiplicative binary noise to intermediate representations of the model in each training iteration. This is equivalent to sparsifying the parameters of the network (applied randomly in each training iteration), which encourages the model to rely on different parameters for learning the same feature from data, eventually limiting the model's free parameters. Since the model is trained on a different subset of parameters in each training iteration, the model implicitly contains an ensemble of sub-networks (Baldi and Sadowski, 2014), which in turn reduces variance and improves the accuracy.
- **Pre-training**: Pre-training a model on a different task or dataset can help put the parameters of the model in a space such that in addition to easing optimization of the model on the second task, may also reduce overfitting on the second task in a low data regime. Pre-training can boost performance when the model learns on the distribution of data $P(\mathbf{X}_1)$ of the first task that shares the same statistics with the data $P(\mathbf{X}_2)$ of the second task, or when optimizing on $P(\mathbf{Y}_1|\mathbf{X}_1)$ helps better learning $P(\mathbf{Y}_2|\mathbf{X}_2)$.

- **Multitask learning**: Multitasking is learning multiple tasks using the same model that share parameters across tasks. It reduces overfitting by leveraging extra tasks that put more load on the parameters of the model, eventually stopping it from memorizing features overly specialized for a specific task and allowing it to generalize to solve several tasks. Multitasking can improve performance of the model on a specific task i only if learning $P(\mathbf{Y}_j|\mathbf{X}), j \neq i$ can help improving $P(\mathbf{Y}_i|\mathbf{X})$, or if the model can learn useful statistic on $P(\mathbf{X})$ when few paired labeled data $P(\mathbf{X}, \mathbf{Y}_i)$ exist. Otherwise, if irrelevant tasks are used, multitasking can deteriorate the performance compared to training only on an individual task. Applying multiple loss functions has a similar impact as multitasking.

In this section, I discussed important elements of a learning algorithm, namely the data, the model, the objective function, and the optimization procedure. In the following sections of this chapter I talk about unsupervised and supervised learning techniques and focus on the models that are used in this thesis.

1.1.2. Unsupervised Learning

Unsupervised learning methods are used on data that doesn't have any labels. The data is represented as $\mathbf{x}_n,\ n \in \{1, \ldots, N\}$, with N data points and each data point having a dimensionality n. Unsupervised learning methods are used to find some statistical structure in the data, or to reduce the dimensionality of the data by considering the main directions of variation, e.g. finding a lower dimensional latent space that generates the data, or to train a network in an unsupervised way either to initialize the parameters of the network or to regularize it in semi-supervised methods. In the following part of this section I review the unsupervised learning methods that are used in this thesis.

Principal Component Analysis:

The goal of principal component analysis (PCA) is to project data from its original space of dimensionality $\mathbb{R}^m$ to a lower dimensional space of dimensionality $\mathbb{R}^d$ (d is usually much smaller than $m,\ d \ll m$), where we find the d most important directions of variation in the data and project it to d orthogonal axes, yielding uncorrelated features. PCA can be seen as fitting a d-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a princinpal component.

Let the data come from a training set with N instances, where each instance $\mathbf{x}$ has dimensionality $\mathbb{R}^m$. If we compute the mean and the convariance matrix of the mean subtracted data using

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{1.1.15}$$

$$\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu})^T(\mathbf{x}_n - \boldsymbol{\mu}), \tag{1.1.16}$$

then by computing the largest eigenvalues $\lambda_j, j \in \{1, \ldots, d\}$ and the corresponding eigenvectors $\mathbf{v}_j, j \in \{1, \ldots, d\}$ of $\boldsymbol{\Sigma}$, we can write each element $\mathbf{x}_n$ as a function of the mean plus a residual (mean subtracted $\mathbf{x}_n$) that is projected from the latent space $\mathbf{z}_n$ using the eigenvectors:

$$\mathbf{x}_n = \boldsymbol{\mu} + \mathbf{W}\mathbf{z}_n \tag{1.1.17}$$

$$\tag{1.1.18}$$

where $\mathbf{W}$ is a $d \times m$ matrix containing the d eigenvectors that explain the most variations in the data. Each column of $\mathbf{W}$ is an eigenvector. Equivalently $\mathbf{z}_n$ can be written as

$$\mathbf{z}_n = \mathbf{W}^T(\mathbf{x}_n - \boldsymbol{\mu}). \tag{1.1.19}$$

$\mathbf{z}_n$ is the projection of $\mathbf{x}_n$ onto the d principal components found by the PCA model.

The complete process of computing PCA is the following:

(1) Obtain the mean of the data $\boldsymbol{\mu} = \frac{1}{N}\sum_{\mathbf{x}=1}^{N}\mathbf{x}$.

(2) Subtract the mean from each instance $\mathbf{x}$, this yields $\mathbf{s} = \mathbf{x} - \boldsymbol{\mu}$.

(3) Get the empirical covariance matrix, which is equal to

$$\boldsymbol{\Sigma} = \frac{1}{N-1}\mathbf{B}^T\mathbf{B} \tag{1.1.20}$$

where $\mathbf{B}$ is a matrix of dimensionality $N \times m$ with each row containing a mean subtracted instance $\mathbf{s}$.

(4) Find the eigenvalue and eigenvector of the covariance matrix i.e. its eigendecomposition, which amount to solving for

$$\mathbf{V}^{-1}\mathbf{\Sigma}\mathbf{V} = \mathbf{E} \tag{1.1.21}$$

where $\mathbf{V}$ is the orthogonal matrix of eigenvectors and $\mathbf{E}$ is a diagonal matrix whose elements are eigenvalues of $\mathbf{\Sigma}$ corresponding in the same order to the columns (eigenvectors) of $\mathbf{V}$.

(5) Rearrange the items in $\mathbf{V}$ and $\mathbf{E}$ in the decreasing order of the eigenvalues.

(6) Get the first d eigenvectors of $\mathbf{V}$ and label them as $\mathbf{W}$

(7) Normalize the mean subtracted data $\mathbf{s}$ by dividing it to the standard deviation which yields $\mathbf{u} = \mathbf{s}/\mathbf{d}$ where $\mathbf{d} = \{\sigma_1,\ \sigma_2,\ \ldots,\ \sigma_m\}$ and $\sigma_j = \sqrt{C_{jj}}$ for $j \in m$.

(8) Project the data to the new space $\mathbf{z} = \mathbf{u} \times \mathbf{W}$.

Probabilistic PCA:

PCA projects data from the original space to a lower dimensional space representing the d most important directions of variation in data. However, it does not give a proper probabilistic representation of data in the original space. Using maximum likelihood estimation, one can fit a Gaussian distribution to the data in $\mathcal{D}_{\text{train}}$ by training a full covariance matrix with the dimensionality of the input space. However, if the dimensionality of the input space is high this requires keeping $O(n^2)$ parameters. This approach is also not feasible when the data is scarce which leads to overfitting. Probabilistic principal component analysis (PPCA) assumes that the data is generated at a lower dimensional latent space $\mathbf{z}$ and then is projected to the observed space $\mathbf{x} \in \mathbb{R}^n$ using $\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$, where $\mathbf{W}$ is the loading matrix projecting data from the latent space to the observable space, $\boldsymbol{\mu}$ is the mean of the observed data and $\boldsymbol{\epsilon}$ is the noise generated from $\mathcal{N}(0, \sigma^2\mathbf{I})$. The data in the latent space is distributed with a zero mean and a diagonal covariance matrix $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$. Given the latent variable $\mathbf{z}$, the conditional is Gaussian with distribution $\mathbf{x}|\mathbf{z} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$. Since both the prior and conditional distributions are Gaussian, the joint distribution is also Gaussian. By integrating out the latent variable, the observable $\mathbf{x}$ is represented by a normal distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})$. Using the maximum likelihood estimation, the

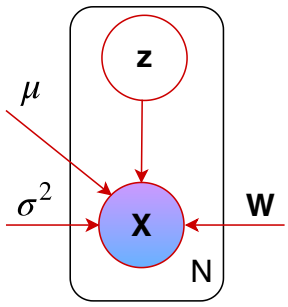


Fig. 1.1. The graphical model of the probabilistic PCA with N observations of $\mathbf{x}$ as the visible and $\mathbf{z}$ as the latent variables. The parameters of the model are $\boldsymbol{\theta} = \{\sigma^2,\ \boldsymbol{\mu},\ \mathbf{W}\}$.

parameters $\boldsymbol{\mu}$, $\mathbf{W}$ and σ^2 are estimated by

$$\boldsymbol{\mu}_{ML} = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} \mathbf{x} \tag{1.1.22}$$

$$\mathbf{W}_{ML} = \mathbf{U}_d (\boldsymbol{\Lambda}_d - \sigma^2 \mathbf{I})^{1/2} \boldsymbol{\Omega} \tag{1.1.23}$$

$$\sigma^2_{ML} = \frac{1}{m-d} \sum_{i=d+1}^{m} \lambda_i, \tag{1.1.24}$$

where $\mathbf{U}_d$ is a matrix of dimensionality $m \times d$ with its columns containing the d principal eigenvectors and $\boldsymbol{\Lambda}_d$ is a $d \times d$ diagonal matrix with eigenvalues $\lambda_i, i \in \{1, \ldots, d\}$ along its diagonal which correspond respectively to the columns of $\mathbf{U}_d$. The term $\boldsymbol{\Omega}$ is an arbitrary orthogonal matrix. Note that σ^2_{ML} is the maximum likelihood estimate of σ^2 and takes the average of the remaining eigenvalues that correspond to the least significant directions of variation in data. Using the PPCA formulation, one can sample data from the distribution. Also, the number of parameters required by PPCA is $O(m \times d)$, which is smaller compared to learning a full covariance matrix in the original representation of $\mathbf{x}$, which requires $O(m^2)$ parameters. PPCA is used in the building block of probabilistic PCA constrained local models (PPCA-CLMs) in Section 4.4.2 to construct an energy function for landmark localization.

Factor Analysis:

Similar to PPCA, factor analysis (FA) is a Gaussian model constructed linearly by

$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon}$. The latent variable comes from the same distribution as in PPCA. However, the conditional distribution $p(\mathbf{x}|\mathbf{z})$ is constructed by $\mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \boldsymbol{\Psi})$ with a diagonal covariance matrix $\boldsymbol{\Psi}$ instead of an isotropic one, namely $\sigma^2\mathbf{I}$, used in PPCA. The columns in $\mathbf{W}$ are called factor loadings and the elements in the diagonal of $\boldsymbol{\Psi}$ are referred to as uniquenesses. Since the elements outside the diagonal of the covariance matrix are all zero, only the variance along each dimension is captured in $\boldsymbol{\Psi}$ and the correlation in-between different dimensions of the input space is captured by the loading matrix $\mathbf{W}$. The marginal distribution of the observed variable is given by $\mathcal{N}(\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi})$. The parameters $\boldsymbol{\mu}$, $\mathbf{W}$, and $\boldsymbol{\Psi}$ can be computed using maximum likelihood. As in PPCA, the maximum likelihood of the mean is given by the average of the data

$$\boldsymbol{\mu}_{ML} = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} \mathbf{x}. \tag{1.1.25}$$

The parameters $\mathbf{W}$ and $\boldsymbol{\Psi}$ can be computed by using the expectation maximization (EM) algorithm. EM algorithm is an iterative method, where the model depends on some latent variables $(\mathbf{z})$ and the optimization of both latent variables and the parameters cannot be solved directly. Therefore, an iterative approach is used, where in the E-step the parameters $\mathbf{W}$ and $\boldsymbol{\Psi}$ of the model are kept fixed and the latent variables $\mathbf{z}$ and $\mathbf{z}\mathbf{z}^T$ are estimated:

$$\mathbb{E}\big[\mathbf{z}\big] = \mathbf{F}\mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu}) \tag{1.1.26}$$

$$\mathbb{E}\big[\mathbf{z}\mathbf{z}^T\big] = \mathbf{F} + \mathbb{E}\big[\mathbf{z}\big]\mathbb{E}\big[\mathbf{z}\big]^T \tag{1.1.27}$$

with $\mathbf{F}$ being defined as

$$\mathbf{F} = (\mathbf{I} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1} \tag{1.1.28}$$

In the M-step the latent variables $\mathbf{z}$ and $\mathbf{z}\mathbf{z}^T$ are kept fixed and the parameters $\mathbf{W}$ and $\boldsymbol{\Psi}$ are updated:

$$\mathbf{W} = \Big[\sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} (\mathbf{x} - \boldsymbol{\mu})\mathbb{E}\big[\mathbf{z}\big]^T\Big]\Big[\sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} \mathbb{E}\big[\mathbf{z}\mathbf{z}^T\big]\Big]^{-1} \tag{1.1.29}$$

$$\boldsymbol{\Psi} = diag\Big[\mathbf{S} - \mathbf{W}\frac{1}{m}\sum_{\mathbf{x} \in \mathcal{D}_{\text{train}}} \mathbb{E}\big[\mathbf{z}\big](\mathbf{x} - \boldsymbol{\mu})^T\Big] \tag{1.1.30}$$

where $\mathbf{S} = \mathbf{U}_d \mathbf{\Lambda}_d \mathbf{U}_d$. The terms $\mathbf{U}_d$ and $\mathbf{\Lambda}_d$ were described in Probabilistic PCA Section. The E and M steps are repeated until the latent variables and the parameters converge, meaning they do not change anymore up to some tolerance, at which point the parameters of the FA model are considered as trained. Similar to PPCA, FA also gives a probabilistic representation over the data distribution such that we can generate new samples from the data. However, FA is more expressive than PPCA since the covariance matrix in the conditional distribution $p(\mathbf{x}|\mathbf{z})$ has a different variance per dimension. This allows FA to explain more variations in the data distribution $p(\mathbf{x})$. FA is used in the building block of the identity-expression factorized constrained local models (IE-CLMs) in Section 4.4.3, which builds an energy function for landmark localization. The identity and expression model uses a FA formulation in the energy function.

1.1.3. Supervised Learning

In supervised learning, data is given in the form $\mathcal{D}_{\text{train}} = \{\mathbf{x}_n, \mathbf{y}_n\},\ n \in \{1, \ldots, N\}$ where $\{\mathbf{x}_n, \mathbf{y}_n\}$ is the n^{th} paired labeled data. The goal is to learn a function $f : \mathbf{X} \rightarrow \mathbf{Y}$ which predicts a output or a set of outputs $\mathbf{y} \in \mathbf{Y}$ for a given input $\mathbf{x} \in \mathbf{X}$.

In regression problems each label $y_n \in \mathbf{y}_n$ is a real number $y_n \in \mathbb{R}$, while in classification problems y_n is discrete; $y_n \in \{1,\ \ldots, K\}$, with K being the total number of classes. There are a large body of supervised learning techniques such as regression models, kernel methods, support vector machines, naive Bayes and decision trees. Since these classical models are out of the scope of this thesis, I do not review them. We will instead focus on supervised learning approaches that use (deep) artificial neural networks, a.k.a deep learning, which is the subject of the next section.

1.2. Deep Learning

Deep learning (DL) methods are a family of machine learning algorithms that learn a hierarchy of representations usually learned through different layers of features, where the layers closer to the input data learn lower level features (e.g. detecting edges on images) and layers further away from the input data learn more abstract concepts (e.g. general shape representation or whether an object is present or not).

Prior to the deep learning era, a lot of effort was put into extracting pre-processed or hand-crafted features from data to get high performance on specific tasks, as extracting these features was key to improving the performance of those models. The arrival of graphical processing units (GPUs) – which allowed processing data in parallel using thousands of cores – and also the abundance of data on the internet, allowed deep learning models to flourish. This breakthrough was most prominent in the ImageNet 2012 Challenge, where AlexNet (Krizhevsky et al., 2012) reduced error by more than 10.8% compared to the second model on top-5 classification error, which brought a lot of attention to deep learning approaches. Krizhevsky trained his neural network on 1.2 million labelled images using a highly optimized GPU implementation of 2D convolutional neural networks, showing the neural networks capability by leveraging both GPUs and a big dataset.

Another important reason for the success of DL algorithms is their ability to automatically learn a hierarchy of representations. Depth allows these models to reuse features and learn more abstract representations in higher levels of the network (Bengio et al., 2013a). This allows DL algorithms to produce their outputs based on multiple non-linear transformation of the input, usually by extracting lower level features first and reusing them to compute more global features higher in the representations until producing the final result. DL algorithms are also more efficient in learning a distributed representation (Bengio et al., 2012) meaning a smaller number of parameters can explain the variations in many data, as opposed to local representation algorithms where O(N) parameters are needed to explain the variations in O(N) data, which is the case e.g. for Gaussian mixture models, nearest neighbor methods, and Gaussian SVM.

Since the birth of DL algorithms, a lot of effort has been invested into designing DL architectures, losses, or training techniques to improve the performance of learning algorithms.

1.2.1. Artificial Neural Networks

Artificial neural networks are the building blocks of deep learning algorithms which are broadly inspired by biological neural networks. Biological neurons signal other neurons through synapses. If the excitation received by a neuron through synapses over a short period of time is large enough, the neuron generates a pulse called action potential which is then passed through synapses to other neurons. While artificial neural networks are inspired by biological neurons, they have diverged separately through proposal of different components of the artificial neural networks that yielded better performance in practice such as sharing parameters in the convolutional neural networks that does not happen in biological neural systems.

Deep Learning methods are deeper variants of artificial neural networks when multiple layers are used between inputs and outputs, potentially learning a hierarchy of features and more abstract concepts in the deeper layers. The simplest form of an artificial neural network is a multi-layer perceptron (MLP), where the network is composed of multiple fully connected (FC) layers. Each FC layer is parameterized by a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$ and computes its output $\mathbf{o}$ as:

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{1.2.1}$$

$$\mathbf{o} = f(\mathbf{h}) \tag{1.2.2}$$

where $\mathbf{x}$ is the input of the layer, $\mathbf{W}$ is a weight matrix of shape $m \times n$, where n and m are input and output layer dimensions and $\mathbf{b}$ is the bias vector. This yields a pre-activation vector $\mathbf{h}$. A non-linearity function f is applied to $\mathbf{h}$ to yield the layer's output. The commonly used non-linearities are sigmoid, tanh, ReLU, leaky-ReLU (Maas et al., 2013), and exponential linear units (ELU) (Clevert et al., 2015):

$$\text{sigmoid}(\mathbf{h}_i) = \frac{1}{1 + \exp(-\mathbf{h}_i)} \tag{1.2.3}$$

$$\tanh(\mathbf{h}_i) = \frac{\exp(2\mathbf{h}_i) - 1}{\exp(2\mathbf{h}_i) + 1} \tag{1.2.4}$$

$$\text{ReLU}(\mathbf{h}_i) = \begin{cases} 0, & \text{if } \mathbf{h}_i < 0 \\ \mathbf{h}_i, & \text{if } \mathbf{h}_I >= 0 \end{cases} \tag{1.2.5}$$

$$\text{leaky-ReLU}(\mathbf{h}_i) = \begin{cases} 0.01, & \text{if } \mathbf{h}_i < 0 \\ \mathbf{h}_i, & \text{if } \mathbf{h}_i >= 0 \end{cases} \tag{1.2.6}$$

$$\text{ELU}(\mathbf{h}_i) = \begin{cases} \alpha(\exp(\mathbf{h}_i) - 1), & \text{if } \mathbf{h}_i < 0 \\ \mathbf{h}_i, & \text{if } \mathbf{h}_i >= 0 \end{cases} \tag{1.2.7}$$

$\mathbf{h}_i$ indicates the i'th element in the vector $\mathbf{h}$. Several such layers can be chained, i.e. the output of one layer serving as input to the next layer. Let us call the first layer's input $\mathbf{x}$ and the last layer's output $\tilde{\mathbf{y}} = \text{Net}(\mathbf{x})$.

Objective Functions:

Once the output of a neural network is computed, a loss or utility function is defined to indicate which criteria the model should minimize (in the case of a loss function, also known as cost) or maximized (e.g. in the case of a utility function such as log likelihood or reward). This is usually done by taking the output of a neural network and defining a loss, or utility function with it. In the rest of this chapter, I assume a loss function is used. However, the argument can be similarly extended to a utility function. In the supervised learning case, the loss is defined by comparing the network's output with the ground truth labels which act as a teacher to indicate how much the network's output is off the ground truth value.

Neural networks are utilized for supervised tasks such as regression or classification. In regression, the model is trained to bring the estimated output $\tilde{\mathbf{y}}$ close to the ground truth output $\mathbf{y}$ by training the network with the following loss function:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x},\mathbf{y} \in \mathcal{D}_{\text{train}}} \mathcal{L}(\mathbf{y}, \tilde{\mathbf{y}}; \boldsymbol{\theta}) \tag{1.2.8}$$

$$\tilde{\mathbf{y}} = \text{Net}(\mathbf{x}) \tag{1.2.9}$$

If $\mathcal{L}$ is a L2 loss, $\tilde{\mathbf{y}}$ predicts the mean of a Gaussian distribution. The loss can then be written as:

$$\mathcal{L}(\mathbf{y}, \tilde{\mathbf{y}}) = \|\mathbf{y} - \tilde{\mathbf{y}}\|^2 \tag{1.2.10}$$

If $\mathcal{L}$ is a L1 loss, $\tilde{\mathbf{y}}$ predicts the mean of a Laplacian distribution. The loss can then be written as:

$$\mathcal{L}(\mathbf{y}, \tilde{\mathbf{y}}) = \sum_{j \in J} |\mathbf{y}_j - \tilde{\mathbf{y}}_j|. \tag{1.2.11}$$

Other loss functions are also used.

In classification, we want the final output layer to generate the probabilities for K different classes. To ensure this, usually a softmax layer is used as the output of a classification network. Let $\tilde{\mathbf{y}}$ be the vector of pre-activations of the last layer before the non-linearity, i.e. it is an affine transform of the previous layer's output. Let its size K correspond to the number of classes, so that $\tilde{\mathbf{y}}_i$ is the score for class number i. To convert these scores into proper probabilities we use the softmax non-linearity defined as:

$$f(\tilde{\mathbf{y}})_i = \frac{\exp\left(\tilde{\mathbf{y}}_i\right)}{\sum_{k=1}^{K} \exp\left(\tilde{\mathbf{y}}_k\right)} \tag{1.2.12}$$

$f(\tilde{\mathbf{y}})_i$ is the probability assigned to the i^{th} class. Note that similar to any other probability distribution we have $\sum_{i=1}^{K} f(\tilde{\mathbf{y}})_i = 1$. The model is then trained with the cross-entropy loss:

$$\mathcal{L}(\mathbf{p}, \mathbf{q}) = -\sum_{k=1}^{K} \mathbf{p}_k \ \log(\mathbf{q}_k) = -\sum_{k=1}^{K} \mathbf{p}_k \ \log(f(\tilde{\mathbf{y}})_k) \tag{1.2.13}$$

where $\mathbf{p}_k$ is the ground truth probability and $\mathbf{q}_k$ is the network's estimated probability for k^{th} class. Since $\mathbf{p}_k$ is equal to one for only one class and zero for other classes meaning $\mathbf{p}_k \in \{0, 1\}$ with $\sum_{k \in K} \mathbf{p}_k = 1$, the objective over the entire training set can be written as

$$\mathcal{L}(\mathbf{x}, y) = -\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{\mathbf{x}, y \in \mathcal{D}_{\text{train}}} \sum_{k=1}^{K} 1(y = k) \ \log(f(\tilde{\mathbf{y}})_k) \tag{1.2.14}$$

where 1 is the indicator function, y is the ground truth label, and $\boldsymbol{\theta}$ indicates the model parameters containing the set of weights and biases of the neural network.

Gradient Backpropagation:

After measuring the loss, the gradient of the loss $\mathcal{L}$ with respect to (w.r.t) each parameter in the network is measured. The chain rule is used to obtain the gradients of the loss $\mathcal{L}$ w.r.t the parameters, which is computed from the output of the network all the way back to the first parameterized layer that needs to be updated, effectively updating all learnable parameters. This is known as backpropagation. Let us consider $\frac{\partial \mathcal{L}}{\partial \mathbf{o}_j^{l+1}}$ as the gradient of the loss $\mathcal{L}$ with respect to (w.r.t) unit j in layer $l+1$ and $\frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{o}_i^{l}}$ as the gradient of unit j in layer $l+1$ w.r.t unit i in layer l. The gradient of loss $\mathcal{L}$ with respect to unit i in layer l is then measured by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{o}_i^{l}} = \sum_{j \in output_i} \frac{\partial \mathcal{L}}{\partial \mathbf{o}_j^{l+1}} \times \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{o}_i^{l}} \tag{1.2.15}$$

where $\mathbf{o}_j^{l+1}$ is j'th unit in layer $l+1$ that takes input from unit $\mathbf{o}_i^{l}$ in layer l.

If $\mathbf{o}_j^{l+1}$ is measured by applying weights and a bias and then non-linearity f as in

$$\mathbf{o}_j^{l+1} = f(\mathbf{z}_j) \tag{1.2.16}$$

$$\mathbf{z}_j = \mathbf{W}^{l+1} \times \mathbf{o}^{l} + \mathbf{b}^{l+1} \tag{1.2.17}$$

then we can apply the chain rule to the internal components of $\mathbf{z}$ as well:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l+1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{o}_j^{l+1}} \times \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{W}^{l+1}} \tag{1.2.18}$$

$$\frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{W}^{l+1}} = \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{z}_j} \times \frac{\partial \mathbf{z}_j}{\partial \mathbf{W}^{l+1}} \tag{1.2.19}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{l+1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{o}_j^{l+1}} \times \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{b}^{l+1}} \tag{1.2.20}$$

$$\frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{b}^{l+1}} = \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{z}_j} \times \frac{\partial \mathbf{z}_j}{\partial \mathbf{b}^{l+1}} \tag{1.2.21}$$

The expanded version of 1.2.15 then becomes

$$\frac{\partial \mathcal{L}}{\partial \mathbf{o}_i^l} = \sum_{j \in output_i} \frac{\partial \mathcal{L}}{\partial \mathbf{o}_j^{l+1}} \times \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{o}_i^l} \tag{1.2.22}$$

$$\frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{o}_i^l} = \frac{\partial \mathbf{o}_j^{l+1}}{\partial \mathbf{z}_j} \times \frac{\partial \mathbf{z}_j}{\partial \mathbf{o}_i^l} \tag{1.2.23}$$

Optimization Algorithms:

The final step in training is updating the parameters of the neural networks. Optimization algorithms are used the to update the value of the parameters such that they produce lower error on similar data. The simplest optimization algorithm is stochastic gradient decent (SGD). This is stochastic because the parameters are updated using one example instead of the entire batch, which allows the network to converge faster, since otherwise the entire training data should be passed through the network before updating the parameters. SGD is an unbiased estimator of the direction of the gradient. In practice a mini-batch containing more than one example is used instead of one example which is known as mini-batch stochastic gradient decent. The following definitions are the same regardless of which of the two variants are used. For simplicity we use the term SGD. The general updating rule of SGD for a parameters $\mathbf{W}_i$ is

$$\mathbf{W}_i^{t+1} = \mathbf{W}_i^t - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_i^t} \tag{1.2.24}$$

with $\mathbf{W}_i^t$ and $\mathbf{W}_i^{t+1}$ being the values of parameter $\mathbf{W}_i$ before and after the SGD update, η being the learning rate and $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_i^t}$ being the gradient of loss w.r.t $\mathbf{W}_i^t$.

Commonly with SGD, momentum is used which takes the average direction of the previous updates and enforces the next update in that direction. This has the advantage of letting the model converge faster, but also allowing the model to continue moving the parameters in plateaus (regions of very low or almost no direction of gradient for optimization). The negative aspect of momentum is that it can continue updating the parameters even after reaching the optima. In practice, it has

been mostly advantageous when used jointly with SGD. The updating rule is as follows:

$$\mathbf{m}_i^{t+1} = \alpha \mathbf{m}_i^t - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_i^t} \tag{1.2.25}$$

$$\mathbf{W}_i^{t+1} = \mathbf{W}_i^t + \mathbf{m}_i^{t+1} \tag{1.2.26}$$

with $\mathbf{m}_i^t$ and $\mathbf{m}_i^{t+1}$ being the values of momentum before and after the SGD update, η being the learning rate, α being the momentum coefficient, and $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_i^t}$ being the gradient of loss w.r.t $\mathbf{W}_i^t$. A value of $\alpha = 0.9$ enforces a strong usage of the past gradients, while a value of $\alpha = 0.1$ enforces an update more based on the recent values of the gradients. α and η are hyper-parameters, meaning they should be tuned manually by using the validation set.

Recently many other optimization algorithms have been proposed, such as Adam (Kingma and Ba, 2014), Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012), RMSProp (Tieleman and Hinton, 2012), which track gradient updates in the network and update parameters accordingly. While the default version of these algorithms perform well most of the time, they also introduce some hyper-parameters which should be tuned to enhance the performance on a specific task.

Training Steps:

The whole training process of a neural network can be described as the following: first a mini-batch, which is a subset of the training data, is taken and passed through the network in a *forward* pass. Once the output of the network is calculated, the loss of the output is measured and then gradient of the loss w.r.t every parameters of the network is computed, which includes weights and biases of the network. Finally, the parameters are updated by an optimization algorithm. The goal is to reduce the estimated loss by measuring how much each parameter had contributed to the loss and redistribute it between parameters based on their contribution to the loss. These training steps are repeated multiple times until the model finally converges, meaning the error on the validation set cannot be further improved. At that point, training is stopped and the parameters of the neural network are taken as the model's parameters. Several such models are trained using different hyper-parameters, such as the learning rate, the number of parameters in the network, and the architecture of the network. The hyper-parameters that yield the best performance on the validation set are selected

and the model trained with those hyper-parameters becomes the final trained model. The error on the test-set is reported using the trained parameters of this network.

In this section I introduced artificial neural networks and their most basic model, namely multi-layer perceptron. I also described general training steps of artificial neural networks. In the next sections I describe some other commonly used deep learning algorithms.

1.2.2. Convolutional Neural Networks

Convolutional neural networks (CNN) (LeCun et al., 1989) are feed-forward neural networks where neurons are connected only to a local sub-set of neurons in the previous layer which is known as the neuron's receptive field. CNNs aim at exploiting strong local correlations at each layer, while capturing high level correlations in the CNN's input space through deeper layers of the network. Inspired by animal cortex systems (Hubel and Wiesel, 1959, 1967, 1968, 1960), CNNs have demonstrated outstanding performance on vision tasks including classification (He et al., 2016; Krizhevsky et al., 2012), face verification (Taigman et al., 2014), object detection and segmentation (Girshick et al., 2014; He et al., 2016, 2017) and also pose estimation (Toshev and Szegedy, 2014) among other tasks.

Each layer in CNNs is usually composed of multiple feature maps, where each feature map is a $2D$ arrangement of units. In two dimensional (2D) convolutions, which are usually applied to images, this forms a three dimensional (3D) tensor in each layer expanding across rows, columns and feature maps. When having time-series data (as in videos), 3D convolutions are used, where each layer has a four dimensional (4D) tensor expanding across rows, columns, time, and feature maps. Figure 1.2 shows an example of a CNN. CNNs are commonly composed of convolution and subsampling (pooling) layers, as shown in Figure 1.2.

Convolutional Layer: In a 2D convolutional or conv layer, each neuron is connected to a local region of neurons in the previous layer, usually including a local set of neurons across rows and columns but including all feature maps of the previous layer. Each neuron is connected to neurons in the previous layer through parameters (or filters, also known as kernels) that are shared with all other neurons in the same layer. This enforces capturing a feature regardless of its position in the input space. For example a tree can be detected in an image regardless of its location.

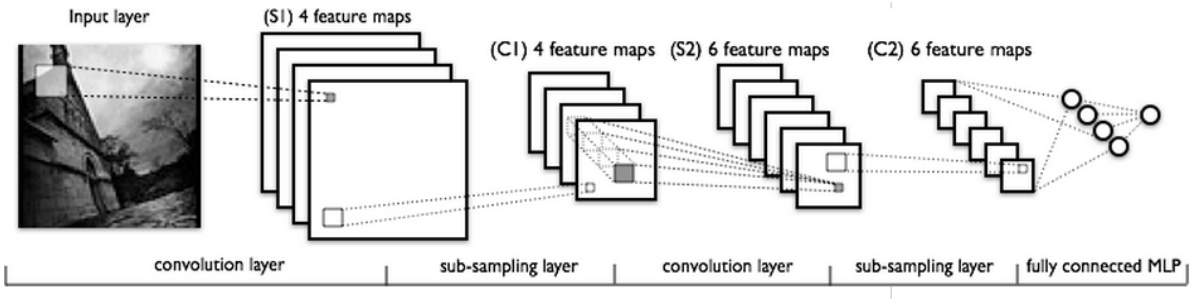


Fig. 1.2. Example of a convolutional neural network composed of two convolutional layers followed by pooling (sub-sampling) layers and two fully connected layers. In this example the output layer has one unit. Image is taken from (DeepLearning.net).

Due to applying the same convolutional parameters to different spacial locations of a layer, the output of the convolutional layer contains activation responses of the convolutional parameters as applied to different input locations. If the location of a feature changes in the input layer, in the output of the convolutional layer it changes to the same extent, meaning the features extracted by a convolutional layer in its output change equivariantly with respect to the position of features in its input. Therefore, the convolutional layers aim at finding *equivariant* features.

Sharing parameters also acts as a strong regularizer that enforces learning meaningful features instead of extracting unimportant details or overfitting to the training data.

In a convolutional layer, weights are multiplied into the neurons (or units) of the previous layer as

$$\mathbf{x}_{k,i}^{l+1} = \mathbf{W}_k^{l+1} \cdot \mathbf{x}_i^l + \mathbf{b}_k \tag{1.2.27}$$

where $\mathbf{x}_{k,i}^{l+1}$ is the i^{th} unit in the k^{th} feature map of layer $l+1$. $\mathbf{W}_k^{l+1}$ is a 3D weight tensor used to compute all units in the k^{th} feature map of layer $l+1$, which are computed by multiplying $\mathbf{W}_k^{l+1}$ into different 3D sub-tensor of $\mathbf{x}^l$. The symbol $\cdot$ indicates a dot product between two vectors after flattening its two operands and $\mathbf{x}_i^l$, which has the same shape as $\mathbf{W}_k^{l+1}$, indicates a 3D tensor which is in the receptive field of $\mathbf{x}_{k,i}^{l+1}$ unit. $\mathbf{b}_k$ is the bias of the feature map k.

The units in the feature map k are computed by multiplying the weights $\mathbf{W}_k^{l+1}$ into different 3D sub-tensors of $\mathbf{x}^l$, which is commonly done by shifting across rows and columns of $\mathbf{x}^l$. The amount by which $\mathbf{W}_k^{l+1}$ is shifted across rows or columns of layer l to compute the adjacent units in layer $l+1$, is known as the stride size of the conv layer. See an illustration of a conv layer in Figure 1.3.

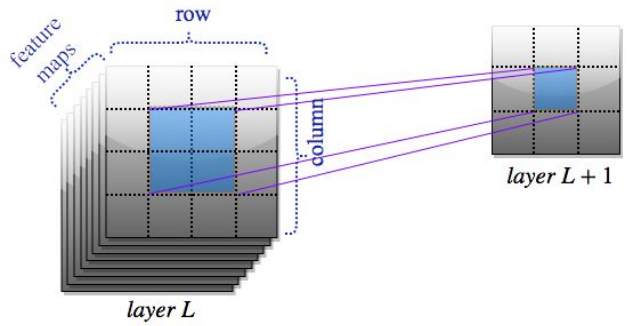


Fig. 1.3. An example a convolution operation from layer l to layer $l+1$. Here a 2×2 kernel is used for convolution across each feature map and considering F^l feature maps in layer l this yields a 3D weight tensor of shape $F^l \times 2 \times 2$ for the conv operation. The receptive field of the unit in blue in layer $l+1$ is shown in blue in layer l. In this example the stride size is one, and only one feature map in layer $l+1$ is generated. Using zero padding in layer l, this yields an output of 3×3 for an input of size $4 \times 4 \times F^l$.

Since each layer usually has multiple feature maps, we have $\mathbf{W}_k^{l+1}$, with $f \in \{1, \ldots, F^{l+1}\}$ and F^{l+1} being the number of feature maps of layer $l+1$. A nonlinear function such as ReLU (1.2.5), hyperbolic tangent (1.2.4), or sigmoid (1.2.3) is then applied to each output $\mathbf{x}_{k,i}^{l+1}$.

Pooling Layer: After a convolutional layer, some networks apply a pooling or subsampling layer, e.g. in classification tasks. Different pooling layers can be used such as average or max pooling layers. A stride of more than one unit in the convolutional layer also acts as subsampling. In a max pooling layer the maximum of a local subset of units is taken, usually across both row and column indices:

$$p = \max_{i,j} \mathbf{x}_{i,j} \tag{1.2.28}$$

where p is the max-pooled output and $\mathbf{x}_{i,j}$ indicates the unit in the receptive field of p with row index i and column index j, with i and j usually including a small number of indices (e.g. in a max-pooling over a 2×2 region, i and j each take two different values).

In an average pooling layer we have:

$$p = \text{mean}_{i,j} \mathbf{x}_{i,j} \tag{1.2.29}$$

The receptive field of unit p is from a sub-region of a feature map, or in the case of cross channel pooling, a sub-region across multiple feature maps. Similar to conv layers, pooling layers have a

stride size, which indicates the elements $\mathbf{x}_{i,j}^{l+1}$ and $\mathbf{x}_{i+1,j}^{l+1}$ (or $\mathbf{x}_{i,j}^{l+1}$ and $\mathbf{x}_{i,j+1}^{l+1}$) of pooling operation output in layer $l+1$ are computed by which amount of shift in row i (or column j) index locations of the pooling operation input in layer l. For example, a stride of 2 indicates after applying pooling operation on a sub-tensor in layer l, the i index is shifted by 2 to compute the next pooling operation output in layer $l+1$ in the same row and the j index is shifted by 2 to compute the next pooling operation output in layer $l+1$ in the same column. Usually, the stride size is more than 1, resulting in an output being smaller than the pooling layer's input, e.g. in the case of 2×2 pooling with stride of 2, the output is $2\times$ smaller than the pooling layer's input, across both row or column.

The pooling layers aims at finding features regardless of their position in the input space, since the max or average functions only take the resulting output and ignore the location where the feature came from. Therefore the pooling layer aims at finding *invariant* features. Note that the exact location of a feature is lost in the pooling operation due to maximization or averaging. However, the discarded information is considered of insignificant importance for the task under study, e.g. classification.

We use convolutional neural networks in the building block of the Recombinator Networks, presented in Chapter 6, and also in our proposed semi-supervised model architecture, presented in Chapter 8, for the landmark localization task. We also leverage convolutional neural networks in the DepthNet architecture, presented in Chapter 10, to estimate the depth of landmarks.

1.2.3. Autoencoders

Autoencoders (Ackley et al., 1985; Elman and Zipser, 1988; Hinton and Salakhutdinov, 2006), which was initially introduced as auto-association techniques (Ackley et al., 1985; Elman and Zipser, 1988), are a family of neural network algorithms where the networks are trained in an unsupervised style by reconstructing their input. Autoencoders can extract useful features from data without requiring any labels. They are also used as an unsupervised pre-training technique (Erhan et al., 2010; LeCun et al., 2015). Such pre-training acts as a strong regularizer that puts the parameters of the network in an space that eases the optimization for other tasks of interest. For example, when a model extracts useful features from images, those features can be also used for classification tasks and the model can be easily fine-tuned to do classification. It can also be used in

semi-supervised techniques where few labeled data are available and where without unsupervised learning techniques the network could easily overfit on the labelled data.

An autoencoder contains an encoder and a decoder. The encoder reduces the input feature dimension in multiple steps until reaching the bottleneck of the network (also known as the code). The features are then expanded back to the original input dimension by the decoder. The goal of using autoencoders is to extract codes that reduce the dimensionality of the input data but contain information that explain the most salient features in the data. This is done by getting rid of the noise (or unimportant features) in the data, while maintaining only the important features, to reconstruct the original input. If learned properly, the code can be used for a low dimensional data representation, classification, visualization, communication or storage of high dimensional data. Unlike PCA that learns a linear extraction of the most important directions of variations in the data, autoencoders learn a non-linear generalization of PCA (Hinton and Salakhutdinov, 2006). Figure 1.4 represents an example of an autoencoder.

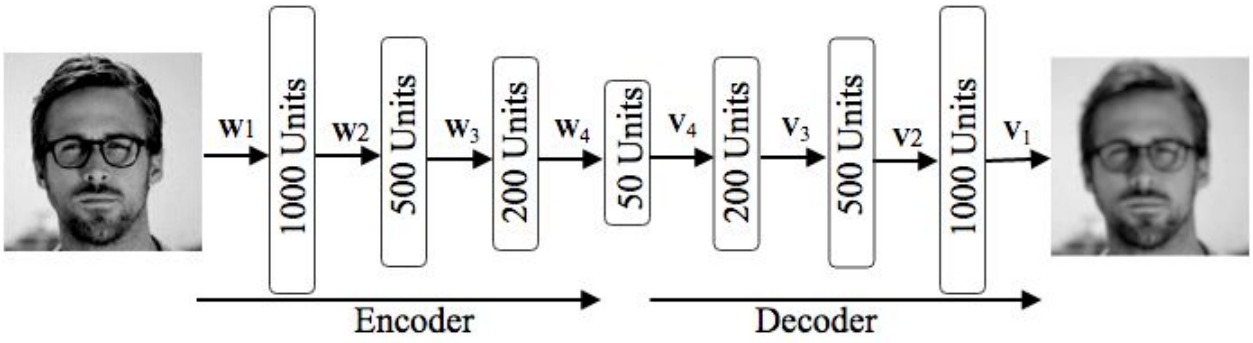


Fig. 1.4. Example of an autoencoder that reduces the input dimension in four layers by using $\mathbf{W}_1$ to $\mathbf{W}_4$ and then reconstruct the input using $\mathbf{V}_4$ to $\mathbf{V}_1$. The contractive part of the network is known as *encoder* and the reconstructing part of the network is known as *decoder*. Encoder output is referred to as the code. In this example the code has 50 units.

An autoencoder with one hidden layer can be formulated by minimizing the loss between $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$ by the network:

$$\mathcal{L}(\mathbf{x},\tilde{\mathbf{x}}) = \mathcal{L}(\mathbf{x}, f_2(\mathbf{W}' f_1(\mathbf{W}\,\mathbf{x}))) \tag{1.2.30}$$

$\mathcal{L}(\mathbf{x},\tilde{\mathbf{x}})$ represents the loss between input $\mathbf{x}$ and autoencoder's reconstruction $\tilde{\mathbf{x}}$. Commonly a MSE loss (Eq. 1.2.10) or a L1 loss (Eq. 1.2.11) is used for $\mathcal{L}$. f represents a non-linear function such as

ReLU (Eq. 1.2.5), tanh (Eq. 1.2.4) or sigmoid (Eq. 1.2.3) or an identity function (commonly used for output layers when output can take any range of values). Depending on the design $\mathbf{W}'$ can be the transpose of $\mathbf{W}$, in which case the encoder and the decoder parameters are tied, or they can be separate (untied) parameters. The formulation in Eq. (1.2.30) can include more than one hidden layer to contains further set of non-linearities and a deeper architecture.

The code of an autoencoder has a smaller dimension than the input, known as an undercomplete representation. This bottleneck acts as a regularizer. When the code of the autoencoder has equal or higher dimension than the input (known as the overcomplete case), if no regularization is used, the model could copy the input to output without learning any useful latent features. The bottleneck in the network enforces the model to learn meaningful features to reconstruct the input. Other regularization techniques have also been proposed such as sparse autoencoders (Ng, 2011), denoising autoencoders (Vincent et al., 2008, 2010), and contractive autoencorders (Rifai et al., 2011).

Sparse Autoencoders: In sparse autoencoders (Ng et al., 2011) the average activation of each unit in the network is pushed to be close to a small probability. In the following notation, if a_i indicates the activation of unit i, then the average activation of unit a_i on N training data can be represented as

$$\tilde{p} = \frac{1}{N} \sum_{n=1}^{N} a_i(\mathbf{x}_n) \tag{1.2.31}$$

where $\tilde{p}$ is set to be close to a probability target, e.g. 0.05, to enforce sparsity of the units in the network. While each unit is responsible for explaining a different feature in the data, sparsity encourages activation of only the units that explain the most salient features in the data.

Denoising Autoencoders: In denoising autoencoders (Vincent et al., 2008, 2010), the data samples $\mathbf{x}$ are corrupted yielding $\bar{\mathbf{x}}$. The autoencoder is then trained to reconstruct $\mathbf{x}$ from $\tilde{\mathbf{x}}$. The denoising autoencoder can be represented as

$$\mathcal{L}(\mathbf{x},\tilde{\mathbf{x}}) = \mathcal{L}(\mathbf{x}, f_2(\mathbf{W}' f_1(\mathbf{W}\ \bar{\mathbf{x}}))). \tag{1.2.32}$$

The difference between Eq. (1.2.32) and Eq. (1.2.30) is in passing $\bar{\mathbf{x}}$ instead of $\mathbf{x}$ to the input of the network. The denoising autoencoder can be seen as learning a manifold where the model learns to maximize $P(\mathbf{x}|\bar{\mathbf{x}})$. In doing so, the model learns to map lower probability data $\bar{\mathbf{x}}$ to higher probability data $\mathbf{x}$. Since in denoising autoencoders usually a small percentage of the input components are corrupted (e.g. 10% of pixels in an image), by reconstructing $\mathbf{x}$ from $\bar{\mathbf{x}}$, the model learns to infer the components in the corrupted locations given the information that is not corrupted. This means the model internally learns to reconstruct the corrupted components given the clean components without knowing which components are clean or corrupted, indicating the model implicitly learns $P(\mathbf{x}_j = \text{clean}|\mathbf{x}_{\neg j})$. In simple terms this means the model learns the internal correlation in between the data elements.

Generative variants of autoencoders, such as variational autoencoder (Kingma and Welling, 2013) and generalized denoising autoencoders (Bengio et al., 2013b) also have been proposed.

We use a convolutional variant of denoising auto-encoders in the building block of the denoising keypoint model, presented in Section 6.4, to learn a joint distribution over landmarks and hence reduce the noise that can appear in the predicted landmarks of the Recombinator Networks. By leveraging the denoising keypoint model, the predicted landmarks have a higher probability under the joint distribution of the landmarks and consequently they are better aligned (less noisy) with respect to each other.

1.2.4. Generative Adversarial Networks

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a family of generative deep neural networks that are composed of two sub-networks: a generator and a discriminator. The discriminator is trained to distinguish between the ground truth (GT) data distribution samples and the generated (fake) samples. The discriminator loss can be written as

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\in\text{GT}}[\log(D_{\boldsymbol{\theta}}(\mathbf{x}))] + \mathbb{E}_{\tilde{\mathbf{x}}\in\text{fake}}[\log(1 - D_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}))] \tag{1.2.33}$$

with D being the discriminator. The discriminator loss can be also written as

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\in\text{GT}}[\log(D_{\boldsymbol{\theta}}(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\in\text{P}_\mathbf{z}}[\log(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})))] \tag{1.2.34}$$

where G is the generator and $\mathbf{z}$, which is the input to the generator, is a sampled noise from a prior distribution, usually a Gaussian or a uniform distribution. The discriminator maximizes the probability of the GT samples and minimizes the probability of the fake samples.

The generator is trained to generate samples that can fool the discriminator in the sense that the discriminator cannot distinguish between GT and generated samples. To do so the generator is trained by maximizing the discriminator's probability on the generated samples such that the discriminator believes they are real. The generator then gets the gradient of this loss from the discriminator and updates its parameters without updating the discriminator parameters. In simple terms, generator gets discriminator's estimated probability on its generated samples and tries to update its generated samples slightly to make the discriminator believe that they are real samples. The generator objective is

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{z}\in \mathrm{P}_{\mathbf{z}}}[\log(1 - D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})))] \tag{1.2.35}$$

The above loss is known for being unstable. The following version has been used in practice, which is more stable

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{z}\in \mathrm{P}_{\mathbf{z}}}[-\log(D_{\boldsymbol{\theta}}(G_{\boldsymbol{\phi}}(\mathbf{z})))]. \tag{1.2.36}$$

Training GANs was initially a very difficult task since the generator and the discriminator have contradictory objectives and it is difficult to balance the training of these two networks. If the balance is lost one gets stronger than the other and it diverges the training process. Usually the discriminator has an easier task compared to the generator since it only needs to distinguish real from fake samples while the generator has to generate realistic looking samples. So, usually the discriminator gets too certain on real versus fake samples and the training diverges. In the original formulation of GANs (Goodfellow et al., 2014), when the discriminator gets too certain about fake samples, the generator gets almost no gradients. Therefore, the gradient for the generator diminishes and it does not converge. DCGAN (Radford et al., 2015) was one of the first models that proposed a stable training of GANs. Especially using batch norm (Ioffe and Szegedy, 2015) was important in making their training stable. Recent methods have focused on regularizing the discriminator to stabalize the training.

Wasserstein GAN (WGAN) (Arjovsky et al., 2017) minimizes an earth-mover distance between the distribution of real and fake data and enforces the discriminator to be within 1-Lipschitz functions. It uses weight clipping in the discriminator (a strong *regularizer*) to enforce the Lipschitz constraint. Improved training of Wasserstein GANs (Gulrajani et al., 2017) proposes a different way to enforce the Lipschitz constraint. They enforce having gradient of one almost everywhere to enforce having differentiable 1-Lipschitz functions. To do so, they *regularize* the discriminator by pushing the gradients of the data points in between real and fake values to be one. Spectral Norm (Miyato et al., 2018) is another strong regularization used in the discriminators to stabilize training, which gets very decent results. In general these regularization techniques in discriminators, such as weight clipping, gradient penalty, batch normalization, layer normalization (Ba et al., 2016), weight normalization (Salimans and Kingma, 2016), or spectral normalization have been crucial in stabilizing the training of GANs.

We leverage a conditional variant of GANs in the building block of the DepthNet+GAN model presented in Section 10.4.2 to train a model that generates depth of the landmarks conditioned on the 2D landmark locations.

Cycle Consistent Generative Adversarial Networks: Cycle Consistent Generative Adversarial Networks (CycleGAN) (Zhu et al., 2017) are generative adversarial networks proposed for unpaired image-to-image translation. Unpaired means that the images in the two domains, such as camera images and segmentation images, are not paired in the sense that for each image in the one domain the training data does not identify (nor usually has) a corresponding image in the other domain. This is useful since in many applications it is difficult to get paired data, e.g. in semantic segmentation where labeling data is laborious, or in day-night, winter-summer image pairs, where it is almost impossible to get paired data. CycleGAN was one of the first proposed models to do unpaired image to image translation. DualGAN (Yi et al., 2017) was a concurrent work that proposes a similar solution.

CycleGAN is composed of two generators; the first generator translates images from the first domain (A) to the second domain (B) and the second generator translates images from domain B to A. The two GANs are trained to generate realistic looking images when translating images from

one domain to another. However, using only GAN losses are not sufficient to get equivalent images in the other domain. To enforce semantically meaningful translation, a cycle consistency loss has been added to the generator's objective. The cycle consistency loss is

$$\mathcal{L}_{cycle} = \min_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{a}\in \mathrm{P}_{\mathbf{a}}}[\,|G_{\boldsymbol{\phi}}^{B,A}\left(G_{\boldsymbol{\phi}}^{A,B}\left(\mathbf{a}\right)\right) - \mathbf{a}|\,] + \mathbb{E}_{\mathbf{b}\in \mathrm{P}_{\mathbf{b}}}[\,|G_{\boldsymbol{\phi}}^{A,B}\left(G_{\boldsymbol{\phi}}^{B,A}\left(\mathbf{b}\right)\right) - \mathbf{b}|\,] \tag{1.2.37}$$

with $G^{B,A}$ and $G^{A,B}$ being generators translating images from domain B to A and A to B, respectively. By using the above loss, sample $\mathbf{a} \in A$ is translated to $\tilde{\mathbf{b}}$ using $G^{A,B}$ and then translated back to $\tilde{\mathbf{a}}$ using $G^{B,A}$. The model enforces consistency by generating $\tilde{\mathbf{a}}$ to be close to $\mathbf{a}$.

The total generator's objective can be then formulated as

$$\mathcal{L}_{total} = \mathcal{L}_{GAN} + \lambda \mathcal{L}_{cycle} \tag{1.2.38}$$

with $\mathcal{L}_{GAN}$ being

$$\min_{\boldsymbol{\phi}} \mathbb{E}_{\mathbf{a}\in \mathrm{P}_{\mathbf{a}}, \mathbf{b}\in \mathrm{P}_{\mathbf{b}}} - [\,\log\left(D_{\boldsymbol{\theta}}^{B}\left(G_{\boldsymbol{\phi}}^{A,B}\left(\mathbf{a}\right)\right)\right) + \log\left(D_{\boldsymbol{\theta}}^{A}\left(G_{\boldsymbol{\phi}}^{B,A}\left(\mathbf{b}\right)\right)\right)\,] \tag{1.2.39}$$

The two discriminators $D_{\boldsymbol{\theta}}^{A}$ and $D_{\boldsymbol{\theta}}^{B}$ are trained using Eq. (1.2.34).

CycleGAN has worked well in practice when the changes between domains A and B are mostly in textures and the data in domains A and B almost cover the same distribution on the underlying semantic diversity, without requiring to learn to transform the geometry or the structure in the data. For example, the model can do a reasonable job when transforming apples to oranges, or horses to zebras. There are issues with CycleGAN, however, most notably when the underlying distribution of data in domains A and B do not have the same semantics, in which case CycleGAN translates images from A to B without maintaining critical information in A. For example, as shown in (Cohen et al., 2018), when translating T1 to Flair images (two types of MRI images), CycleGAN can remove cancer indicators if the data distribution in the target domain only contains healthy images or CycleGAN can add tumors if the target domain only contains cancer images. This is due to having the generator match the distribution of the target data, which is learned by the discriminator in the target domain, without maintaining vital information from the source domain. So, if the two sets of data distributions do not have the same underlying semantics, the model generates uncorresponding images when translating images from one domain to another. This phenomenon can also happen

when one domain is more expressive that another. For example, if data in domain A contains rotation across ‘yaw’ and ‘pitch’ axes and data in domain B contains rotation only across ‘yaw’ axis, then when translating from A to B, the generated samples cannot transform well the rotation across ‘pitch’ axis. One question would be how CycleGAN can reconstruct the original image using cycle consistency loss, when the translated image in the second domain does not correspond to the original image in the first domain. CycleGAN can hide the source image in the generated target image, which is known as steganography (Chu et al., 2017), and uses the hidden image in the backward reconstruction. CycleGAN should be used while considering its limitations and taking into account the underlying data distribution of the two domains.

We leverage CycleGANs in Section 10.4.3 to illustrate how to use the DepthNet model for the full-head face frontalization task (by synthesizing the background in addition to the central part of the face) and also for the face replacement task.

Chapter 2

Machine Learning Models for Faces

In this chapter I discuss some of the face related applications that are relevant to this thesis and review some of the well known and also recently emerging machine learning and deep learning approaches used to address them. Due to relevance to this thesis I review face detection, landmark localization, depth estimation, and face rotation. There are other face related tasks such as emotion recognition, face tracking, and face identification that I either do not review or do not delve too much into details since they are out of the scope of this thesis.

2.1. Face Detection

Face Detection is the process of finding human faces, usually in form of a bounding box detected around a face. A common application of face detection algorithms is in digital cameras, where human faces are detected to better focus on them. Its applications, however, do not stop there. Most of the computer vision tasks on faces require face detection, as the first step of their pipeline, in order to only process the face region. These tasks include face recognition and verification, face tracking, face rotation, face swap, face morphing and reconstruction, landmark detection, and emotion recognition, among other tasks. Although the first models proposed on face detection date back to 1960s (Chan and Bledsoe, 1965), it wasn't until (Viola and Jones, 2004) that the face detection algorithms were capable of handling in-the-wild images and became applicable to cameras. The Viola and Jones method has three steps:

(1) Extract features from an image.

(2) Learn an Adaboost classifier on the features. The Adaboost algorithm uses a weak classifier. In this task the classifier decides whether there is a face in an extracted feature or not. Adaboost starts by giving a weight to each example in the data, weighting positive and

negative examples to have them equally represented in the data. In each iteration Adaboost reassigns the weights by giving a higher weight to the wrongly labeled examples and training another classifier on the re-weighted data and continuing this process until it reaches a high accuracy on the difficult examples. The Adaboost aggregated model then takes a weighted average of all classifiers leading to a model that performs well on both easy and difficult examples.

(3) Learn a cascade model that rejects false positive faces. The cascade model is composed of K classifiers, with the j-th classifier being trained to reject false positive samples that have passed the previous $j-1$ classifiers. Therefore starting from the 1-st to K-th classifier, each subsequent classifier is trained on more difficult false positive examples, by using the Adaboost algorithm described in part (2). The final region proposal is the output of the $K-th$ model in the cascade architecture.

Viola and Jones model is capable of reaching a face detection accuracy with the minimum detection rate and the maximum false positive rate specified by the user.

There have been some neural network models proposed for the face detection task, such as (Féraud et al., 2001; Garcia and Delakis, 2004; Rowley et al., 1998; Vaillant et al., 1994). The model in (Rowley et al., 1998), shown in Figure 2.1, takes as input a pyramid of image patches of a fixed resolution size but taken from subsampled image resolutions at different subsampling rates. The model classifies for each patch whether it contains a face or not. It uses then a batch of such networks, each trained with different initial parameters. Each network generates an independent set of proposal regions. The output of all these networks are passed to another network to reduce false positive proposals.

Deep learning variants for object detection have also been proposed. The most famous work is Regions with Convolutional Neural Networks features (Girshick et al., 2014), known as *R-CNN*, and its later variants *Fast R-CNN* (Girshick, 2015) and *Faster R-CNN* (Ren et al., 2015).

R-CNN first uses *selective search* (Uijlings et al., 2013) to get region proposals, an algorithm that merges nearby image patches based on their similarity in color, texture, size and region of intersection in a hierarchical style. R-CNN then takes the proposed regions by selective search and passes them through a CNN network, which is trained on positive and negative object proposal

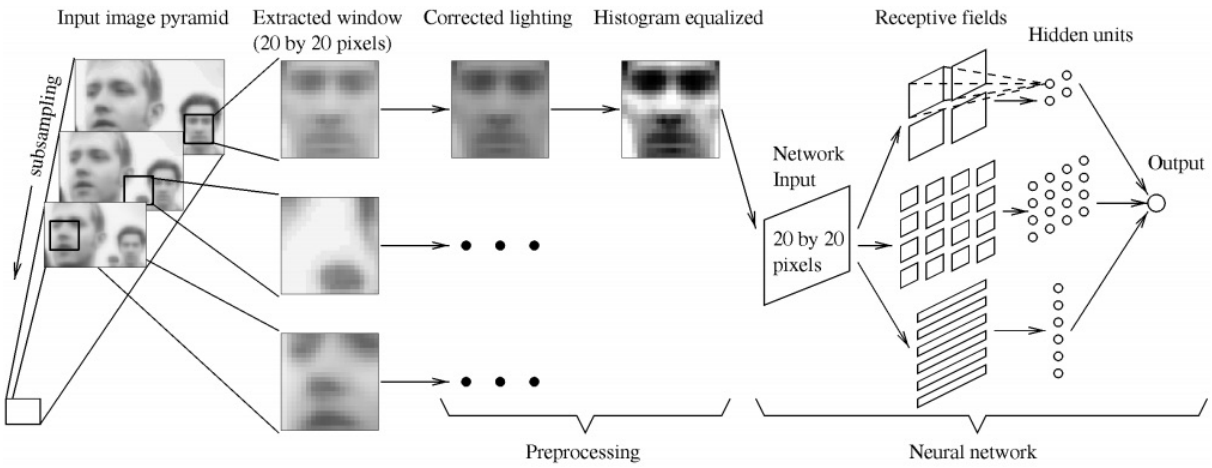


Fig. 2.1. Neural network based face detection framework. Image taken from (Rowley et al., 1998).

regions. The final classification layer (FC_out) is then dropped and the features right at the input of (FC_out) are then passed to class specific linear Support Vector Machines (SVM)s. They obtained better results by using SVMs compared to directly taking the output of the final CNN classification layer. Each SVM is trained on an object class with positive and negative examples of that specific class and indicates whether the passed features belongs to that class or not. Figure 2.2 shows the components of the R-CNN framework. R-CNN was also the first model to pre-train its network in a *supervised* style on a large auxiliary dataset (ILSVRC), showing a model pre-trained on ImageNet can be later fine-tuned on a specific task. This approach was further used in many other vision tasks to boost their performance, indicating training parameters of a neural network on images, even on different classes of objects, can extract useful features that can be transferred to other visual tasks.

While R-CNN and its follow-up works are applied to the general object detection task, they can be used also on faces. Some recent methods have adapted R-CNN variants to the face detection task, as in (Farfade et al., 2015) that replaces the CNN network in R-CNN with a fine-tuned model of AlexNet (Krizhevsky and Hinton, 2009), or the model in (Jiang and Learned-Miller, 2017) that gets state-of-the-art on WiderFace dataset (Yang et al., 2016) using Faster R-CNN framework. Hyperface (Ranjan et al., 2017) also adapts R-CNN by replacing its CNN component with Fully Convolutional

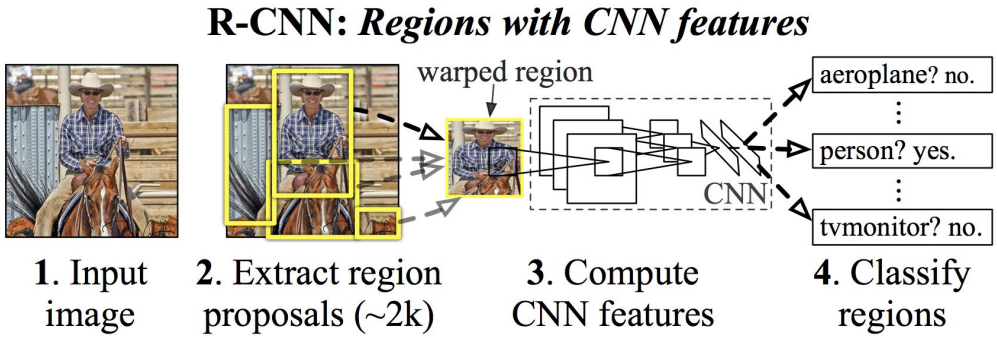


Fig. 2.2. R-CNN framework. The model extracts region proposals from an input image and passes the extracted regions to a Convolutional Neural Network (CNN). The output of CNN classifies for each category (e.g. aeroplane, person) whether the extracted region contains an object of that category or not. Image taken from (Girshick et al., 2014).

Networks (FCN) (Long et al., 2015) and detects faces in a multi-tasking framework while doing other tasks such as landmark localization and pose estimation.

2.2. Landmark Localization

Fiducial Landmark localization is the problem of localizing important features on faces such as eye centers, mouth corners, and the nose tip. Landmark localization is used to improve the performance on other tasks, such as face rotation and verification (Taigman et al., 2014), identification (Sun et al., 2014), pose estimation (Zhu and Ramanan, 2012), gaze estimation (Gou et al., 2017; Wang and Ji, 2017), and emotion recognition (Kahou et al., 2016).

While landmarks indicate fiducial features, such as eyes, nose, eyebrows, mouth, and face-contour, there is not a common consensus on the number and locations of the landmarks. Each dataset defines its own sets of landmarks. While some introduce very few landmarks, as 5 landmarks in MTFL dataset (Zhang et al., 2016) pinpointing eye centers, nose-tip and mouth corners, some others datasets localize a very comprehensive set of landmarks, e.g. 194 landmarks in Helen (Le et al., 2012), where it is hard to assign each landmark to a specific feature on the face. Popular

datasets are AFLW (Köstinger et al., 2011) with 25 and 300W (Sagonas et al., 2013) with 68 landmarks.

There are many old models on landmark localization such as (Yuille et al., 1992) and Active Shape Models (Cootes et al., 1995). One of the best known models is the active appearance model (Cootes et al., 2001).

Active Appearance Model (AAM) (Cootes et al., 2001):
AAM is proposed to match a template model to a new image using both shape (geometry represented by landmarks) and appearance (the image texture represented by pixel values). AAM assumes during training a set of images with labelled landmarks are provided and the model uses principal component analysis (PCA) to learn over all training examples an average representation and an orthogonal basis for both shape and appearance that maps the latent representation of each training example to the observed variables of shape and appearance. It also learns a linear transformation between *the difference between the input image and the AAM estimated appearance* and *the residual that should be added to the latent representation to fix this appearance difference*. At test time when a new query image is given, the model uses this learned linear transformation to update the latent representation in order to map the template model to the query image and minimize the appearance error.

Formally, the following steps are taken in AAM:
First a PCA model is learned over the template shape model by applying PCA to the provided landmarks in the training set, where each set of landmarks $\mathbf{y}$ belonging to a face can be represented as

$$\mathbf{y} = \bar{\mathbf{y}} + \mathbf{P}_s \mathbf{b}_s, \tag{2.2.1}$$

with $\mathbf{y}$ representing the set of landmarks per training example, $\bar{\mathbf{y}}$ being a vector of mean landmark locations over the entire training set, $\mathbf{P}_s$ being the PCA orthogonal basis for shapes, and $\mathbf{b}_s$ being the latent representation corresponding to the training example $\mathbf{y}$ that when used in the above equation reconstructs back $\mathbf{y}$.

To learn a statistical model over appearance, which is over pixel values, each face image is warped such that the set of landmarks of each face maps to the mean shape $\bar{\mathbf{y}}$. A PCA model is then learned on images such that each appearance example $\mathbf{g}$ is represented as

$$\mathbf{g} = \bar{\mathbf{g}} + \mathbf{P}_g \mathbf{b}_g, \tag{2.2.2}$$

where $\bar{\mathbf{g}}$ is a vector of mean appearance of the pixel intensities, $\mathbf{P}_g$ is the PCA orthogonal basis for appearance, and $\mathbf{b}_g$ is the latent representation for example $\mathbf{g}$ that can reconstruct back $\mathbf{g}$ when used in the above equation. Given the above PCA models, the shape and appearance of each example can be represented using $\mathbf{b}_s$ and $\mathbf{b}_g$. However, there might be still correlation between latent representations $\mathbf{b}_s$ and $\mathbf{b}_g$. To decorrelate these features another PCA is learned to map $\mathbf{b}_g$ and $\mathbf{b}_s$ to an uncorrelated latent representation. To do so, first for each example a latent vector $\mathbf{b}$ is measured by

$$\mathbf{b} = \begin{pmatrix} \mathbf{W}_s \mathbf{b}_s \\ \mathbf{b}_g \end{pmatrix} = \begin{pmatrix} \mathbf{W}_s \mathbf{P}_s^T (\mathbf{y} - \bar{\mathbf{y}}) \\ \mathbf{P}_g^T (\mathbf{g} - \bar{\mathbf{g}}) \end{pmatrix} \tag{2.2.3}$$

where $\mathbf{W}_s$ is a diagonal matrix allowing to adapt for differences in units between shape and appearance. A PCA model on latent representation $\mathbf{b}$ is then learned to map $\mathbf{b}$ to an uncorrelated latent representation $\mathbf{c}$

$$\mathbf{b} = \mathbf{Q}\mathbf{c}, \tag{2.2.4}$$

where $\mathbf{Q}$ is the orthogonal PCA basis for both shape and appearance. If we represent

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_s \\ \mathbf{Q}_g \end{pmatrix}, \tag{2.2.5}$$

then using a shared latent representation $\mathbf{c}$, we can reconstruct both shape and appearance by

$$\mathbf{y} = \bar{\mathbf{y}} + \mathbf{P}_s \mathbf{W}_s \mathbf{Q}_s \mathbf{c} \tag{2.2.6}$$

$$\mathbf{g} = \bar{\mathbf{g}} + \mathbf{P}_g \mathbf{Q}_g \mathbf{c}. \tag{2.2.7}$$

The Equations (2.2.6) and (2.2.7) provide a generative model for shape and appearance by sampling different values from the latent representation $\mathbf{c}$. Note that the highest eigenvalues corresponding to the eigenvectors in $\mathbf{Q}$ indicate the most important directions of variations in $\mathbf{c}$. These salient directions of variations can be changed to observe a variety of shapes and appearances.

The provided model so far can learn a shape and appearance model on the training data which allows generating new samples, however we also need a procedure to map the AAM model to a new image. To reduce test time adaptation of AAM to new examples, the model learns how to adapt the latent representation $\mathbf{c}$ to a new query example by learning a linear model on the training data. To do so, it first defines a difference vector $\delta\mathbf{g}$ between a query image appearance $\mathbf{g}_n$ and the model estimation of the appearance model $\mathbf{g}$

$$\delta\mathbf{g} = \mathbf{g}_n - \mathbf{g}. \tag{2.2.8}$$

Then, matrix $\mathbf{A}$ is learned that maps $\delta\mathbf{g}$ to $\delta\mathbf{c}$ by using

$$\delta\mathbf{c} = \mathbf{A}\ \delta\mathbf{g} \tag{2.2.9}$$

where $\delta\mathbf{c}$ is the residual between target and current value of $\mathbf{c}$, meaning if we change the current latent representation $\mathbf{c}_0$ by $\delta\mathbf{c}$, we get new latent representation $\mathbf{c}_1 = \mathbf{c}_0 - \delta\mathbf{c}$ that eliminates the appearance error $\delta\mathbf{g}$.

The latent representation $\mathbf{c}$ is augmented to contain also scale, translation and rotation. To train $\mathbf{A}$, displacement of known training latent representations $\mathbf{c}$ and the corresponding image residual $\mathbf{g}_n$ is used. Once $\mathbf{A}$ is trained, Algorithm 1 is used to fit AAM onto a new query image.

The procedure in Algorithm 1 maps both shape and appearance of AAM to a query image. Once the algorithm has converged, the final c is taken and passed to Equation (2.2.6) to measure AAM shape for the query image. The estimated shape corresponds to the landmark prediction of AAM that is fit to the query image.

Algorithm 1 AAM Fitting Algorithm

1: Start with an initial latent representation $\mathbf{c}_0$
2: Using Equations (2.2.6) and (2.2.7) measure shape and appearance of model $\mathbf{y},\ \mathbf{g}$
3: Obtain normalized apprearance for the query image $\mathbf{g}_n$
4: Measure error vector $\delta\mathbf{g}_0 = \mathbf{g}_n - \mathbf{g}$
5: Set current error as $E_0 = |\delta\mathbf{g}_0|^2$
6: Using matrix $\mathbf{A}$ compute the latent representation displacement $\delta\mathbf{c} = \mathbf{A}\ \delta\mathbf{g}_0$
7: set $h = 1$ and measure $\mathbf{c}_1 = \mathbf{c}_0 - h\ \delta\mathbf{c}$
8: Using $\mathbf{c}_1$ sample image to get $\mathbf{g}_n$ and also estimate $\mathbf{g}$ using Equation (2.2.7)
9: measure $\mathbf{g}_1 = \mathbf{g}_n - \mathbf{g}$
10: If $|\delta\mathbf{g}_1|^2 < E_0$ then accept $\mathbf{c}_1$, otherwise try $h = 1.5,\ h = .5\ h = .25$
11: If $|\delta\mathbf{g}_1|^2$ has reduced goto step 2, otherwise stop

Constrained Local Model (CLM) (Cristinacce and Cootes, 2008, 2006):

Another well-known model for landmark localization is CLM. Similar to AAM, CLM uses a shape and appearance representation as in Equations (2.2.1) and (2.2.2). By applying a unified latent representation, they can be further represented as:

$$\mathbf{y} = \bar{\mathbf{y}} + \mathbf{P}_s\mathbf{W}_s\mathbf{Q}_s\mathbf{c} \tag{2.2.10}$$

$$\mathbf{g} = \bar{\mathbf{g}} + \mathbf{P}_g\mathbf{Q}_g\mathbf{c}. \tag{2.2.11}$$

However, unlike AAM, CLM uses a prior distribution over the joint distribution of landmarks. If $p(\mathbf{y}|\boldsymbol{\theta})$ represents shape given the parameters $\boldsymbol{\theta}$ and $p(\mathbf{x}|\mathbf{y},\boldsymbol{\theta})$ represents the probability of image $\mathbf{x}$ given the shape and parameters, then using the Bayes rule, one can represent shape distribution using

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}) \propto p(\mathbf{x}|\mathbf{y},\boldsymbol{\theta})\ p(\mathbf{y}|\boldsymbol{\theta}) \tag{2.2.12}$$

In the CLM, $p(\mathbf{x}|\mathbf{y},\boldsymbol{\theta})$ is modeled by using a matching score between the image $\mathbf{x}$ and the appearance estimated by the model using

$$p(\mathbf{x}|\mathbf{y},\boldsymbol{\theta}) \propto \prod_{k=1}^{K} e^{-\alpha\mathbf{q}_k}, \tag{2.2.13}$$

where K is the number of landmarks and $\mathbf{q}_k$ is the matching quality between the input image and the CLM appearance representation. Assuming shape latent representations in $\mathbf{b}_s$ are independent and Gaussian distributed, $p(\mathbf{y}|\boldsymbol{\theta})$ can be represented by

$$p(\mathbf{y}|\boldsymbol{\theta}) \propto \prod_{j=1}^{Z} e^{-b_j^2/\lambda_j}, \tag{2.2.14}$$

where Z is the number of shape parameters, b_j is the j-th element of $\mathbf{b}_s$, and λ_j are the corresponding eigenvalues of the shape model. By putting Equation (2.2.14) and Equation (2.2.13) back into Equation (2.2.12) we get

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \sum_{k=1}^{K} -\alpha q_k - \sum_{j=1}^{Z} \frac{b_j^2}{\lambda_j}. \tag{2.2.15}$$

Let's use $q_k = -R_k(w_k, h_k)$ as the normalized correlation response for the k'th feature template (with w_k and h_k as the width and height coordinates of the k-th landmark), then Equation (2.2.15) can be written as

$$f(\mathbf{b}_s) = \alpha \sum_{k=1}^{K} R_k(w_k, h_k) - \sum_{j=1}^{Z} \frac{b_j^2}{\lambda_j}. \tag{2.2.16}$$

The first term acts as an independent classifier per landmark and the second term puts a joint distribution over landmarks. CLM model is then optimized using Algorithm 2.

Algorithm 2 CLM Fitting Algorithm

1: Start with an initial set of landmarks.
2: Using Equations (2.2.6) and (2.2.7) get the current appearance representation g.
3: Measure the normalized response R_k per landmark k, which indicates how far is the current appearance representation from the image.
4: Use Equation (2.2.16) to find the new latent representations $\mathbf{b}_s$ that gives more accurate shape representation, by considering both the current appearance response maps in the first term and the shape prior in the second term.
5: Go to step 2 until convergence.

The Nelder–Mead simplex algorithm is used to Optimize Equation (2.2.16) in Algorithm 2. It starts from an initial set of landmarks and stops when changes to shape are less than a small constant. One difference between CLM and AAM is that in AAM the appearance is a representation over the entire face, while in CLM the appearance is represented by extracting a patch around each landmark and concatenating them.

Mixture of Trees (Zhu and Ramanan, 2012):

Another famous landmark localization approach before the era of deep learning is the joint model of face detection, pose estimation, and landmark localization in (Zhu and Ramanan, 2012). They use a mixture of trees, where each element in the mixture corresponds to a viewpoint and for each viewpoint m they construct a tree of $\mathbf{T}_m = (\mathbf{V}_m, \mathbf{E}_m)$ with $\mathbf{V}_m$ representing the Vertices (landmarks) and $\mathbf{E}_m$ representing the edges connecting the landmarks to build a tree. This mixture is shown in Figure 2.3.

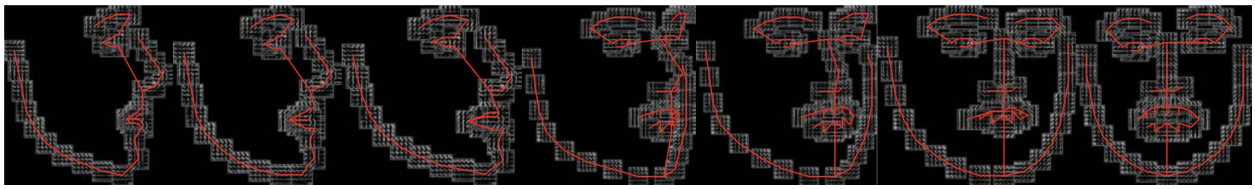


Fig. 2.3. Mixture of trees over landmarks. For each viewpoint a different tree is constructed. Each tree, which connects the landmarks of a face, corresponds to a viewpoint in the mixture. Image is taken from (Zhu and Ramanan, 2012).

Given an input image $\mathbf{x}$ and a set of landmarks $\mathbf{y} = \{\mathbf{y}_k, k \in \mathbf{V}_m\}$, the model over keypoints for view m is represented as

$$S(\mathbf{x},\mathbf{y}, m) = \text{App}_m(\mathbf{x}, \mathbf{y}) + \text{Shape}_m(\mathbf{y}) + \boldsymbol{\alpha}_m \tag{2.2.17}$$

where $\text{App}_m(\mathbf{x}, \mathbf{y})$ is the appearance model, $\text{Shape}_m(\mathbf{y})$ is the shape model, and $\boldsymbol{\alpha}_m$ is a scalar bias or prior for viewpoint m. The appearance model learns for each view m a per landmark set of

weights $\mathbf{w}_k^m, k \in \mathbf{V}_m$ by

$$\text{App}_m(\mathbf{x}, \mathbf{y}) = \sum_{k \in \mathbf{V}_m} \mathbf{w}_k^m . \phi(\mathbf{x}, \mathbf{y}_k) \tag{2.2.18}$$

where $\mathbf{y}_k = (w_k, h_k)$ is the k-th landmark and $\phi(\mathbf{x}, \mathbf{y}_k)$ corresponds to the HOG features extracted from image $\mathbf{x}$ at location $\mathbf{y}_k$.

The Shape model is presented by:

$$\text{Shape}_m(\mathbf{y}) = \sum_{i,j \in \mathbf{E}_m} \mathbf{a}_{ij}^m dw^2 + \mathbf{b}_{ij}^m dw + \mathbf{c}_{ij}^m dh^2 + \mathbf{d}_{ij}^m dh \tag{2.2.19}$$

with dw and dh representing changes across width w and height h dimensions between landmarks i and j that are connected by an edge

$$dw = w_i - w_j \tag{2.2.20}$$

$$dh = h_i - h_j \tag{2.2.21}$$

The parameters $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ in Equation (2.2.19) can be considered as springs in between landmarks i and j that put constraints between the neighbouring landmarks to restrict their movements. Using re-parameterizations of $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$, Equation (2.2.19) can be written as

$$\text{Shape}_m(\mathbf{y}) = -(\mathbf{y} - \boldsymbol{\mu}_m)^T \boldsymbol{\Lambda} (\mathbf{y} - \boldsymbol{\mu}_m) + \text{constant} \tag{2.2.22}$$

$\boldsymbol{\Lambda}$ is a sparse precision matrix with nonzero entries for elements i,j in the set $\mathbf{E}_m$. The above shape formulation corresponds to a model that penalizes configurations away from $\boldsymbol{\mu}_m$.

The model in Equation (2.2.17) is optimized by running a dynamic programming (DM) algorithm that maximizes the following objective

$$S^*(\mathbf{x}) = \max_m [\max_{\mathbf{y}} S(\mathbf{x}, \mathbf{y}, m)]. \tag{2.2.23}$$

The DM algorithm searches across all m viewpoints and in each mixture it searches across landmarks $\mathbf{y}$ to find the best configuration that maximizes the above objective.

Deep Learning approaches on landmark localization:

One of the first deep learning approaches on facial landmark localization after the renaissance of deep learning is the model proposed in (Sun et al., 2013). The diagram of this model is shown in Figure 2.4. The model uses three levels of landmark localization with the first level predicting the initial location of the landmarks. The subsequent two levels crop patches from the input face image around the landmarks predicted by the previous level and process that region further to improve the landmark prediction accuracy.

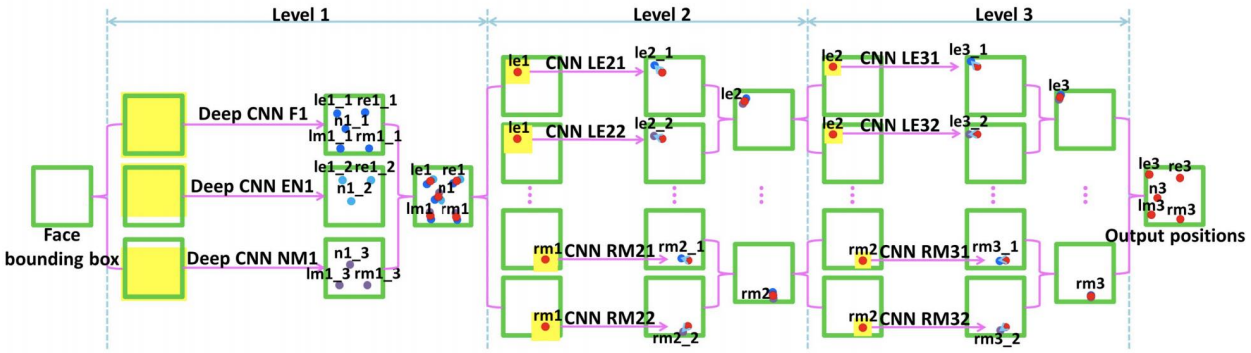


Fig. 2.4. The cascade architecture for landmark localization. The first level has three streams for landmark localization, where some landmarks are predicted in multiple streams. The second and third levels crop image patches (shown in yellow) from the input image by using the location of landmarks predicted in the previous level. Again each landmark is predicted in multiple streams. Image is taken from (Sun et al., 2013).

In each level there are multiple streams, each using a convolutional neural network (CNN), which takes an image patch as input and predicts the landmark locations using a regression loss. For example, the CNN for stream F1 in 2.4 is shown in Figure 2.5.

The final location of each landmark is predicted using the following equation

$$\mathbf{y} = \frac{\mathbf{y}_1^{(1)} + \mathbf{y}_2^{(1)} + \cdots + \mathbf{y}_{l_1}^{(1)}}{l_1} + \sum_{i=2}^{s} \frac{\Delta\mathbf{y}_1^{(i)} + \Delta\mathbf{y}_2^{(i)} + \cdots + \Delta\mathbf{y}_{l_i}^{(i)}}{l_i}, \tag{2.2.24}$$

where l_i indicates the number of streams in level i that predicts the same landmark and s indicates the total number of levels. The outputs of the first level is taken as initial prediction and the outputs

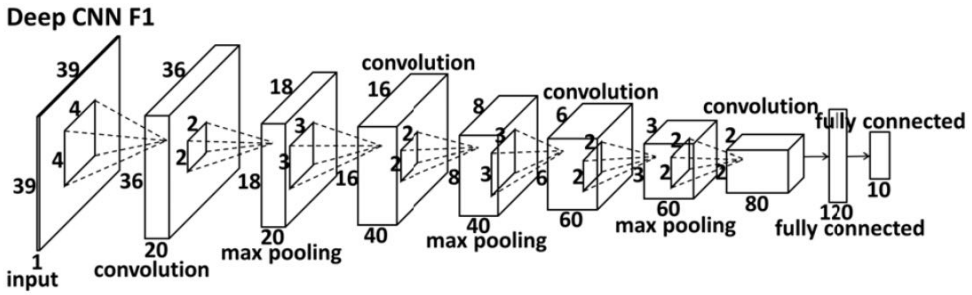


Fig. 2.5. The convolutional neural network (CNN) used in stream F1 of the architecture shown in Figure 2.4. F1 is composed of convolution, max-pooling, and fully connected layers. The feature map resolution in each layer and the kernel/filter size of convolution and max-pooling layers are shown in the image. Image is taken from (Sun et al., 2013).

of the following levels are used as residual modifications to the initial prediction. Since each level predicts a given landmark in multiple streams, the prediction of all streams are averaged.

There has been many follow-up works on landmark localization using deep learning including the multi-tasking approach in (Zhang et al., 2016) and the stacked hourglass network in (Newell et al., 2016) that uses a recursive model to improve its prediction in multiple iterations. Other recent works are reviewed and contrasted within the following chapters when our proposed models on landmark localization are presented.

2.3. Depth Estimation

Depth estimation on faces has been used to build 3D face models from 2D images, which facilitate image manipulation such as face rotation and animation. There has been many models that estimate a 3D face models, such as (Jiang et al., 2005; Kemelmacher-Shlizerman and Basri, 2011; Paysan et al., 2009). One of the most known works is the 3D morphable model (3DMM) (Blanz and Vetter, 1999). 3DMM constructs a 3D face mesh, where each vertex contains a 3D location and a texture. It then renders the 3D face using computer graphics rendering techniques to a two dimensional (2D) image. By comparing the rendered face to a given image of a face, it optimizes the 3D face model and brings it as close as possible to the face in the given 2D input

image in terms of both 3D geometry and also texture.

3D Morphable Model (3DMM) (Blanz and Vetter, 1999):

Formally, 3DMM is composed of a set of K vertices, each containing a location (w, h, d), and a texture (R, G, B). The vertices' locations form the face shape and the vertices' textures form the face appearance. Let's represent the average of shape and texture, measured over all samples in the training set, by $\bar{\mathbf{S}}$ and $\bar{\mathbf{T}}$. Then covariance matrices for shape and texture differences from the mean, namely $\Delta\mathbf{S}_i = \mathbf{S}_i - \bar{\mathbf{S}}$ and $\Delta\mathbf{T}_i = \mathbf{T}_i - \bar{\mathbf{T}}$, are measured. Finally, a PCA is learned on each of these covariance matrices to find the m most salient directions of variations for shape and texture. Having done so, the shape and appearance of each sample can be represented by

$$\mathbf{S}_{model} = \bar{\mathbf{S}} + \sum_{i=1}^{m} \alpha_i \mathbf{s}_i, \tag{2.3.1}$$

$$\mathbf{T}_{model} = \bar{\mathbf{T}} + \sum_{i=1}^{m} \beta_i \mathbf{t}_i \tag{2.3.2}$$

where $\mathbf{s}_i$ and $\mathbf{t}_i$ are the eigenvectors of the covariance matrices found by PCA. α_i and β_i are the coefficients assigned to eigenvector i. Apart from $\vec{\boldsymbol{\alpha}}$ and $\vec{\boldsymbol{\beta}}$ parameters, 3DMM also uses a set of rendering parameters $\vec{\boldsymbol{\rho}}$ such as camera position, object scale, image plane rotation and translation, intensity of ambient light, intensity of directed light, and the color contrast among others. The total set of parameters of 3DMM is therefore represented by $(\vec{\boldsymbol{\alpha}}, \vec{\boldsymbol{\beta}}, \vec{\boldsymbol{\rho}})$.

The 3D face model is rendered using Phong illumination and the parameters are optimized to minimize the difference between the input image $\mathbf{x}_{input}$ and the rendered image $\mathbf{x}_{model}$ using perspective projection. The objective function is therefore

$$\mathcal{L} = \sum_{w,h} \|\mathbf{x}_{input}(w,h) - \mathbf{x}_{model}(w,h)\|^2 \tag{2.3.3}$$

which is optimized over w and h locations in the image. Without any regularization, the above optimization is an ill-posed problem since the 3DMM has multiple free parameters to minimize the above loss. To start optimizing this loss from a good initial configuration, the template 3D face is manually aligned with the input image regarding pose, position, size, orientation and illumination.

This is due to the fact that 3DMM has many free parameters and without some constraints it can end up with 3D face models away from the face in the input image. After the manual initialization, the model is optimized using the following procedure. For each triangle k in the 3D face mesh, the average of the values at the corners (regarding both position and texture) are measured to form an estimate in the center of the triangle and by applying perspective projection the triangle center is rendered to the image location $(\bar{p}_{w,k}, \bar{p}_{h,k})$, yielding $\mathbf{x}_{model,k}$. Considering all K vertices in the 3D model, the total loss can be written as

$$\mathcal{L} = \sum_{k=1}^{K} \|\mathbf{x}_{input}(\bar{p}_{w,k}, \bar{p}_{h,k}) - \mathbf{x}_{model,k}\|^2 \tag{2.3.4}$$

where using Phong illumination, each color component of RGB values, namely $\mathbf{x}_{r,model}, \mathbf{x}_{g,model}, \mathbf{x}_{b,model}$, is measured by

$$\mathbf{x}_{r,model,k} = (i_{r,amb} + i_{r,dir} \cdot (\mathbf{n}_k \mathbf{I}))R_k + i_{r,dir} s \cdot (\mathbf{r}_k \mathbf{v}_k)^\gamma \tag{2.3.5}$$

where $i_{r,amb}$ is intensity of ambient light, $i_{r,dir}$ is intensity of directed light for the red channel, $\mathbf{n}_k$ is the surface normal at vertex k, $\mathbf{I}$ is the direction of illumination, R_k is the red channel texture value at vertex k, s is the face shininess, $\mathbf{r}_k$ is the direction of the reflected ray at vertex k, $\mathbf{v}_k$ is the normalized difference of camera position and the position of the triangle's center at vertex k, and γ controls the angular distribution of the specular reflection. The loss in Equation (2.3.4) is optimized until convergence, which optimizes both on location and appearance of the 3DMM model. The final output is taken as the 3D face representation of the provided 2D face in the input image.

Jiang et al. (2005) automate some of the manual initialization procedures of 3DMM such as head position and orientation, focal length of the camera, and illumination direction. Since 3DMM extracts latent parameters of shape and texture, which are separated from camera parameters, it has also been used for face recognition (Jiang et al., 2005; Paysan et al., 2009) and identification (Blanz et al., 2002).

There have been variants of 3DMM using deep neural networks. Tran and Liu (2018) build a 3DMM model using neural networks where instead of using PCA to to learn shape and texture basis

linearly, they use deep neural networks to measure the latent representation of shape and texture non-linearly.

MOFA (Tewari et al., 2017) also applies a deep learning approach to 3DMM. Similar to 3DMM, it measures the shape and texture basis using PCA, however the model parameters such as shape and texture coefficients and camera rendering parameters are measured using an encoder that encodes an image into a latent representation that estimates the model parameters (shape, texture, and camera parameters). Then, it uses a rendering decoder (without any learnable parameters) that applies computer graphics techniques to render the image. To allow backpropagation, they implement a backward path to measure the gradients. Similar to 3DMM, this model also has many free parameters. To constrain the model, it applies some regularization on the shape, texture and camera parameters. Also it uses optional 2D landmarks to bring the 3D face model close to the pose of the face in the input image. Figure 2.6 shows the MOFA pipeline.

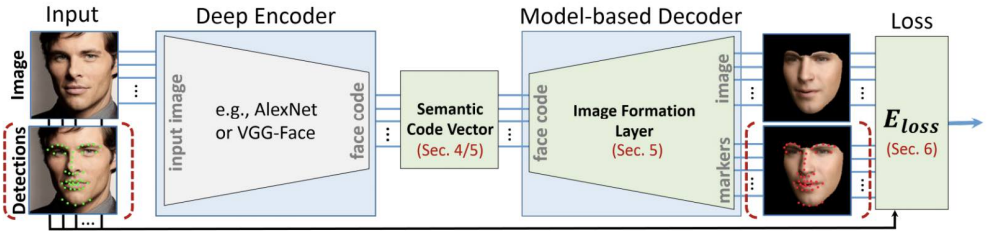


Fig. 2.6. The architecture of MOFA (Tewari et al., 2017). A deep convolutional encoder takes an input image and estimates the parameters of 3DMM such as shape, texture coefficients, and camera parameters. The estimated parameters are then used to render a 2D image via a non-parametric decoder. The model reconstructs the input image and can optionally use landmarks to guide the reconstruction. Image is taken from (Tewari et al., 2017).

Adversarial Inverse Graphics Networks (AIGN) (Tung et al., 2017b) uses a similar approach to 3DMM, however instead of constructing both 3D shape and texture, it only aims at estimating a 3D shape model given an input image. Similar to 3DMM, it uses a PCA to learn the most salient directions of variation in the 3D shape. However, unlike vanilla 3DMM, it uses a convolutional neural network to estimate the coefficients of the shape PCA as well as the camera parameters

that are used to render the 3D shape model to a 2D projection of the landmarks. The projected landmarks are then compared with the ground truth 2D landmarks of the input image. So, AIGN does not try to match the input image in its appearance (pixel intensity), it only matches to the 2D pose of the landmarks, that are provided as ground truth labels. AIGN minimizes the following loss

$$\min_{G} \max_{D} \|P(G(\mathbf{x})) - \mathbf{y}_{2D}\| + \beta [\sum_{i=1}^{K} \log D_i(\mathbf{y}_{3D}) + \log (1 - D_i(G_i(\mathbf{x})_{\mathbf{y}_{3D}}))] \tag{2.3.6}$$

where G is a generator that given an input image $\mathbf{x}$ estimates the parameters of the 3D model, such as PCA shape coefficients and camera parameters, P is the projection function of the 3D model to 2D, and $\mathbf{y}_{2D}$ is the ground truth 2D landmark locations in image $\mathbf{x}$. The first component in the loss is therefore mapping the ground truth 2D landmarks to the 2D projection of the 3D model. The second component is an adversarial loss, where discriminator D is trained to maximize the probability of the ground truth 3D shape parameters $\mathbf{y}_{3D}$ and minimize the probability of the 3D shape parameters generated by G, namely $G_i(\mathbf{x})_{\mathbf{y}_{3D}}$. Note that G generates both the 3D shape parameters (corresponding to the geometry) as well as the camera parameters. The discriminator is trained to only distinguish the shape parameters of real and generated ones (not camera parameters). The ground truth data used to train the discriminator can be taken from a prior distribution, that is different from the training dataset. As shown in Equation (2.3.6), the discriminator maximizes the loss, while the generator minimizes the loss. The second term acts as a regularization to constrain the freedom of parameters in the 3D model and make the estimated shape more realistic.

Jackson et al. (2017) propose a direct estimation of a 3D face, via depth supervision, from a 2D input image. It does not apply any computer graphics rendering techniques or uses any camera parameters. This approach also has a variant where the 3D face estimation is guided by landmark heatmaps that are estimated at an intermediate point through the network.

2.4. Face Rotation

Face rotation is the task of providing other views of a face given a single 2D view of a face in an image. It has been mainly used in the face recognition and verification tasks, where the goal is to identify the person in the image. Face rotation has been leveraged either to provide frontal views of

the person in the image for easier recognition, as in (Gross, 2015; Huang et al., 2017; Taigman et al., 2014; Yin et al., 2017), or rotate frontal faces in the dataset towards more profile views to balance the dataset and enhance recognition on non-frontal faces, as in (Masi et al., 2016; Zhao et al., 2017). Some methods first frontalize faces and then extract features for identity verification, as in (Hassner et al., 2015; Huang et al., 2017; Shen et al., 2018; Taigman et al., 2014; Yin et al., 2017; Zhao et al., 2017, 2018), some other methods rotate faces but use the encoded latent features instead of the rotated face images for identity verification, as in (Shen et al., 2018; Tran et al., 2017).

3D morphable models (Blanz and Vetter, 1999; Blanz et al., 2004) have been used for face rotation, since they construct a 3D face model and the face can be easily rotated to render different views of the face. However, due to mapping the texture of the 3D face to the 2D images, they lack texture on the occluded side of the face. Hence, these methods are not capable of handing occlusion, e.g. when a profile face is frontalized.

One of the known models in face verification is Deepface (Taigman et al., 2014), which applies face frontalization as part of its pipeline. To frontalize faces, it uses a 3D frontal face template and maps the 2D image to the 3D face template to frontalize the face in the image. Specifically, it first does a 2D alignment of the image by applying scale, rotation, and translation to the image such that the 2D landmarks in the image are mapped to the 2D landmarks of the template face. Then it uses an affine camera to map the 2D aligned face image to the 3D face template by minimizing the following loss

$$\mathcal{L} = (\mathbf{y}_{2D} - \mathbf{y}_{3D}\vec{\mathbf{P}})^T \mathbf{\Sigma}^{-1} (\mathbf{y}_{2D} - \mathbf{y}_{3D}\vec{\mathbf{P}}) \tag{2.4.1}$$

where $\mathbf{\Sigma}$ is a known covariance matrix, $\vec{\mathbf{P}}$ is an affine 3D-to-2D camera whose parameters are learned. $\mathbf{y}_{2D}$ are the 2D landmarks on the aligned image, and $\mathbf{y}_{3D}$ are the 3D landmarks on the face template. Once the camera parameters $\vec{\mathbf{P}}$ are learned, the frontal 3D face is constructed by a piece-wise affine transformation guided by the Delaunay triangulation (Okabe et al., 2009). The model then uses the frontalized face for identity verification in a deep convolutional neural network.

Hassner et al. (2015) make the observation that even if a single 3D face template is used for all faces in the dataset, the difference in the shape and geometry between the 3D template and the input query image has little impact on facial appearance features and the person remains easily

identifiable in the frontalized projection of the 3D template model. They exploit this observation and put their effort on frontalizing the face in the query image rather than mapping the 3D face template to the geometry and shape of the face in the query image. They argue that the latter takes more effort than the former. To frontalize the face in a query image they use a template 3D face (used for all query images), and learn a projection matrix between the 3D face template and the landmarks detected on the query image. They then learn another projection matrix between the 3D face template and a reference frontal view, which is the projection of the 3D face template to a 2D frontal face image. By learning these two projection matrices they can map pixels from the query image to the 3D template face and then onto the frontal reference image. They also observe that the pixels in the more occluded region of the query image get copied to more regions of the reference frontal face and use this observation to replace the occluded regions of the frontalized face with the unoccluded symmetric regions of the reference face. While the frontalized faces have some artifacts they boost the face verification results.

Masi et al. (2016) make the observation that on many face recognition datasets, the majority of faces are frontal. Hence, they synthesize different poses, shapes, and expressions for a given input image to augment the dataset, especially for non-frontal views, and boost the recognition performance. In particular, they map the 3D landmark face template $\mathbf{y}_{3D}$ to 2D landmarks $\mathbf{y}_{2D}$ detected in the query image and apply a PnP model (Hartley and Zisserman, 2003) that yields perspective projection parameters $\mathbf{M}$ by using $\mathbf{y}_{3D} = \mathbf{M}\ \mathbf{y}_{2D}$. They then change the rotation parameters in $\mathbf{M}$ to synthesize new faces using a rendering engine. Using this approach they rotate frontal faces to profile views.

Recently there have been adversarial approaches for face rotation. TP-GAN (Huang et al., 2017) uses a two-path network for face frontalization, where one path frontalizes the whole face and another path frontalizes different face components such as eye, mouth, and nose before putting them back together onto the image. The two paths are finally merged together to generate the final frontalized image. The model is trained using paired data, meaning ground truth frontal images are provided for a per-pixel loss. In addition, other losses are introduced such as a symmetric loss (to generate a symmetric face), adversarial loss (to generate realistic images), identity preserving loss

(to maintain the identity in the image), and a variation regularization loss (to maintain a smooth transition in pixel intensities of the image).

PIM (Zhao et al., 2018) also uses a similar two-path approach for face frontalization. Similar to TP-GAN it also leverages paired data and applies a per-pixel loss, adversarial loss, symmetric loss, and variation regularization loss. However, it introduces a domain adaptation loss (to adapt the latent features of the encoded training image to the encoded representation of test images), and an identity preserving loss (by using gradients from an identity discriminator). Both TP-GAN (Huang et al., 2017) and PIM (Zhao et al., 2018) only do face-frontalization with the goal of enhancing face verification.

DR-GAN (Tran et al., 2017) is one of the first adversarial models that does complete face rotation, yielding visually appealing results. The model architecture is shown in Figure 2.7. Given an input image $\mathbf{x}$ and its labels $\mathbf{L} = \{l^d, l^p\}$ for identity l^d and pose l^p, the discriminator D is trained to distinguish between N^d real identities in the training set and a fake identity. The discriminator is also trained on the target pose class l^p. Considering $G(\mathbf{x}, \mathbf{c}, \mathbf{z})$ as a generated image using the input image $\mathbf{x}$, a one-hot pose vector $\mathbf{c}$, and noise $\mathbf{z}$, the discriminator is trained by

$$\max_D \left(\mathbb{E}_{\mathbf{x},\mathbf{L}\in p(\mathbf{x},\mathbf{L})}[\log(D^d_{l^d}(\mathbf{x})) + \log(D^p_{l^p}(\mathbf{x}))] + \right.$$
$$\left. \mathbb{E}_{\mathbf{x}\in p(\mathbf{x}),\ \mathbf{c}\in p(\mathbf{c}),\ \mathbf{z}\in p(\mathbf{z})}[\log(D^d_{N^d+1}(G(\mathbf{x}, \mathbf{c}, \mathbf{z})))]\right) \tag{2.4.2}$$

where D^d is the discriminator output for identity classification with $N^d + 1$ elements (N^d real identities and one fake identity). D^p is the discriminator output for pose classification. D^d_i and D^p_i are the i-th elements in D^d and D^p. The discriminator is therefore trained to distinguish real from fake identities while also to classify the pose in real images.

Given an input image $\mathbf{x}$ the generator first gets the encoded latent representation for identity $f(\mathbf{x})$ and then by adding noise $\mathbf{z}$ and a one-hot pose vector $\mathbf{c}$, it generates the decoded image $G(\mathbf{x}, \mathbf{c}, \mathbf{z})$. The generator is trained to generate the image $\hat{\mathbf{x}}$ with pose l^t encoded in $\mathbf{c}$, and identity l^d in image $\mathbf{x}$. The generator is therefore trained by

$$min_G\ \mathbb{E}_{\mathbf{x},l^d\in p(\mathbf{x},l^d),\ \mathbf{c}\in p(\mathbf{c}),\ \mathbf{z}\in p(\mathbf{z})}[\log(D^d_{l^d}(G(\mathbf{x}, \mathbf{c}, \mathbf{z}))) + \log(D^p_{l^t}(G(\mathbf{x}, \mathbf{c}, \mathbf{z})))]. \tag{2.4.3}$$

DR-GAN learns an identity representation $f(\mathbf{x})$, which is used for face verification. It can leverage a target pose vector $\mathbf{c}$ to rotate the input face $\mathbf{x}$ to this target pose.

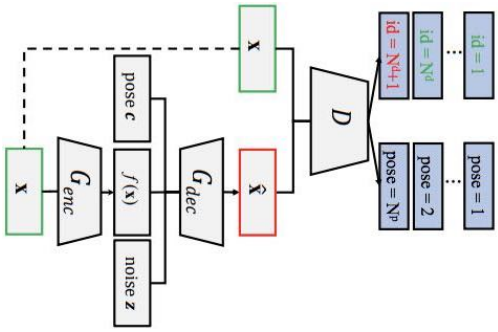


Fig. 2.7. The architecture of DR-GAN (Tran et al., 2017). It takes an input image $\mathbf{x}$ and uses an encoder G_{enc} to get latent features $f(x)$. The encoded features together with noise $\mathbf{z}$ and a one-hot pose vector $\mathbf{c}$ are decoded into $\hat{\mathbf{x}}$. A discriminator D is trained to distinguish N^d real from a fake identity class and also to estimate the pose of the image. Image is taken from (Tran et al., 2017).

There have been recently GAN-based face rotation approaches (Shen et al., 2018; Yin et al., 2017; Zhao et al., 2017) that leverage a 3D morphable model (3DMM). FF-GAN (Yin et al., 2017) does face frontalization by estimating parameters of a 3DMM. The model is shown in Figure 2.8. It takes an input image and first estimates the parameters of the 3DMM. Then it merges the upsampled 3DMM features with the features extracted from the image to frontalize the input face. To train this model, an adversarial loss on the generated image, a symmetry loss, a pixel-wise frontalization loss (due to using paired data), a variation regularization loss, and an identity preserving loss is applied. To enforce learning meaningful latent parameters such as identity, expression, and texture by 3DMM, they first pre-train it with a dataset that provides such labels. FF-GAN only does face-frontalization. The frontalized images are used for face recognition, 3D reconstruction, and landmark localization.

DA-GAN (Zhao et al., 2017) rotates frontal faces to profile poses in order to increase the percentage of data in the profile poses and leverages that for face verification and recognition tasks. To do so, DA-GAN first detects the face and the landmarks and estimates the parameters of a transformation matrix that maps the estimated 2D landmarks to the landmarks of a 3DMM. Once the

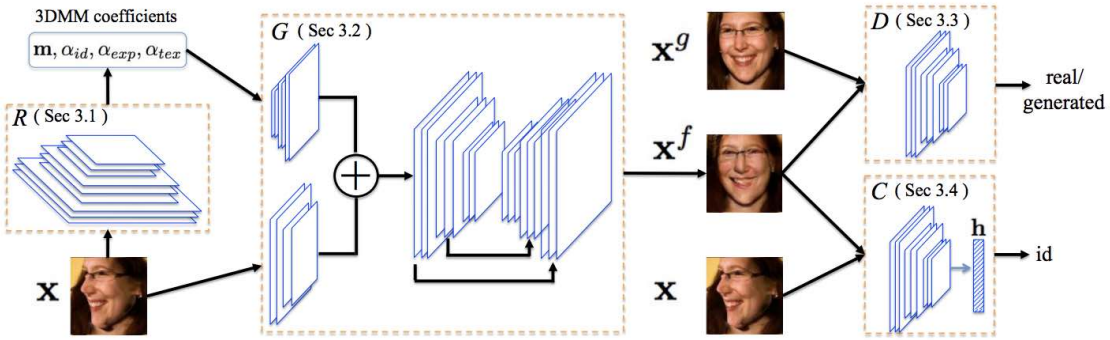


Fig. 2.8. The architecture of FF-GAN (Yin et al., 2017). It takes an input image $\mathbf{x}$ and uses a network R to get 3DMM parameter coefficients, such as identity, expression, and texture. It then merges the upsampled 3DMM features with the image features to frontalize the face. A discriminator D and an identity classifier C are used to guide this process. The discriminator D is trained to distinguish between the ground truth frontal image $\mathbf{x}^g$ and the frontalized image $\mathbf{x}^f$. The identity classifier produces identity features $\mathbf{h}$ for an image passed to it. The generator network is trained to bring the feature representation $\mathbf{h}$ of the frontalized image $\mathbf{x}^f$ close to the feature representation of the original non-frontalized image $\mathbf{x}$ to maintain the identity in the frontalized image. Image is taken from (Yin et al., 2017).

parameters of 3DMM are estimated and the 2D image is mapped to the template, the face is rotated to more profile views. This yields a synthetic face. In the second phase an adversarial network is trained to make these faces realistic. This approach boosts the performance on face verification due to synthesizing more extreme pose faces. However, it maps every face to a generic face template, which can change the input face geometry.

FaceID-GAN (Shen et al., 2018) is another adversarial approach that does complete face rotation. Similar to FF-GAN and DA-GAN it uses a 3DMM to estimate face parameters given an input image. 3DMM yields parameters for identity, expression and pose, which is pre-trained to extract meaningful parameters. Then, they leverage two discriminators, one trained to distinguish real from fake images, and another as an identity discriminator trained to distinguish N real from N fake identities. The generator takes identity, noise, and 3DMM parameters and generates images at the pose specified by the 3DMM parameters that belongs to the same identity. The generator is trained by fooling the two discriminators such that the images are realistic and belong the specified identity.

It also minimizes a mean squared error between the 3DMM parameters passed to the generator and the ones extracted from the generated image. This model is shown in Figure 2.9. Both DR-GAN and FaceID-GAN can rotate faces to different poses.

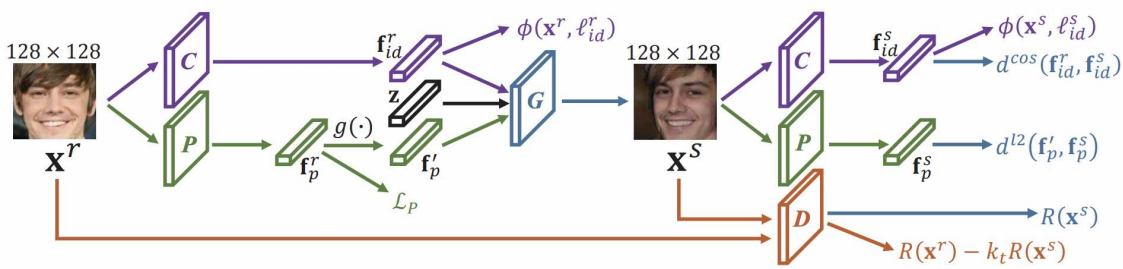


Fig. 2.9. The architecture of FaceID-GAN (Shen et al., 2018). The left part is the generator. The identity features $\mathbf{f}_{id}^r$ (extracted by network C) together with the 3DMM parameters (extracted by network P) and the noise $\mathbf{z}$ are passed to the generator to generate a new image $\mathbf{X}^S$ with the same identity as $\mathbf{X}^r$ but with a pose specified by $\mathbf{f}_p'$ (after changing the pose parameters of 3DMM in $\mathbf{f}_p^r$). The generated image $\mathbf{X}^S$ is passed to three networks: C to extract the identity features $\mathbf{f}_{id}^s$, P to extract the 3DMM parameters $\mathbf{f}_p^s$, and a discriminator D to verify if the image is realistic or not. The generator is trained by taking gradients from these three networks. In particular, $d^{cos}(\mathbf{f}_{id}^r \mathbf{f}_{id}^s)$ compares the extracted id features $\mathbf{f}_{id}^s$ of the new image with the id features $\mathbf{f}_{id}^r$ passed to the generator and $d^{l2}(\mathbf{f}_p', \mathbf{f}_p^s)$ compares 3DMM parameters $\mathbf{f}_p^s$ of the new image with the 3DMM parameters $\mathbf{f}_p'$ passed to the generator. Finally, $R(\mathbf{X}^s)$ increases the probability of the generated image $\mathbf{X}^S$ being realistic. Image is taken from (Shen et al., 2018).

In this chapter I reviewed some of the known and also recently emerging models on faces for the tasks of face detection, landmark localization, depth estimation, and face rotation. In the following chapters I present the models I worked on during the course of my PhD.

Chapter 3

Prologue to First Article

3.1. Article Details

Improving Facial Analysis and Performance Driven Animation through Disentangling Identity and Expression. Sina Honari*, David Rim*, Md Kamrul Hasan, and Christopher Pal (2016). *Image and Vision Computing Journal*, 52, 125-140.

Christopher Pal asked me to take over this project, that he had started with David Rim, at the point where it required major modifications, additions, and new results to be accepted by the Image and Vision Computing Journal. I first recomputed the math formulations and fixed some of the notations. Then, I implemented the identity and expression models. Kamrul also helped on the constrained local model (CLM). I carried out the experiments regarding landmark localization and the analysis of the results. I also re-wrote the sections of the paper regarding the identity-expression model and wrote the section regarding landmark localization. I jointly first authored this work with David Rim, who had started the project but did not pursue it afterwards. While the complete journal paper contains sections on animation control and emotion recognition, I dropped these sections in the thesis to focus only the sections on which I worked, namely the identity-expression formulations and the section on landmark localization, which is the central part of this thesis. This being said, some parts in the contributions and introduction briefly mention also these other tasks. The variable notations in this chapter have been changed compared to the journal version to unify the notations of the repeating concepts in the thesis.

3.2. Context

The goal of this work is to explain variations that are observed on faces through usage of disentangled latent representations of identity and expression. Our model builds on top of (Prince et al., 2011) which uses identity to explain the variations observed over the images. In their work, however, they are interested in extracting variations only across identity. In our work, we explain variations in images through both identity and expression and exploit the expression information through tasks that provide this information and can benefit from it, such as facial expression analysis, animation control and landmark localization.

3.3. Contributions

We present techniques for improving performance driven facial animation, emotion recognition, and facial key-point or landmark prediction using learned identity invariant representations. Established approaches to these problems can work well if sufficient examples and labels for a particular identity are available and factors of variation are highly controlled. However, labeled examples of facial expressions, emotions and key-points for new individuals are difficult and costly to obtain. In this work we improve the ability of techniques to generalize to new and unseen individuals by explicitly modeling previously seen variations related to identity and expression. We show how to extend the widely used techniques of constrained local models through replacing the underlying point distribution models, which are typically constructed using principal component analysis, with identity-expression factorized representations.

3.4. Recent Developments

Non deep learning approaches have fallen out of favor for landmark localization methods and currently the state-of-the-art performing models use deep learning approaches in their formulations. Known examples of deep learning approaches include (Toshev and Szegedy, 2014), which is one of the first models that uses deep learning for landmark localization, (Ranjan et al., 2017; Zhang et al., 2014c), which use multi-tasking approaches for landmark localization in addition to solving other tasks on faces, and (Newell et al., 2016; Xiao et al., 2016), which use a recursive refinement framework for improving landmark localization in multiple steps.

Chapter 4

Improving Facial Analysis and Performance Driven Animation through Disentangling Identity and Expression

4.1. Introduction

One of the primary sources of variation in facial images is identity. Although this is an obvious statement, many approaches to vision tasks other than facial recognition do not directly account for the interaction between identity-related variation and other sources. However, many facial image datasets are subdivided by subject identity and this provides additional information that is often unused. This paper deals with the natural question of how to effectively use identity information in order to improve tasks other than identity recognition. In this work we show how it can be used together with emotion information to improve landmark localization models.

In (Prince et al., 2011) identity is separated from other sources of variation in 2D image data in a fully probabilistic way with the goal of improving face recognition tasks. In their model, the latent features of face (such as identity) are assumed to be additive and independent. This procedure can be interpreted as a probabilistic version of Canonical Correlation Analysis (CCA) presented in (Bach and Jordan, 2005), or as a standard factor analysis with a particular structure in the factors.

In this paper, we investigate and extend the use of this probabilistic approach to separate sources of variation, but unlike prior work which focuses on inferences on identity, we focus on inference on facial expressions. Our goal is to use learned representations so as to create automated techniques for expression analysis that better generalize across identities. We show here how disentangling factors of variation related to identity and expression can be passed to discriminative classification methods to enhance results.

In our experiments, we apply this learning technique to improve the facial keypoint prediction performance of constrained local models (CLMs) through our identity-expression factorization extensions to these widely used techniques.

The rest of this manuscript is structured as follows: In Section 4.2, we review the relevant works. The various methods we present here build in particular on the work of (Prince et al., 2011) in which a linear Gaussian probabilistic model was proposed to explicitly separate factors of variation due to identity. In Section 4.3, we extend this model as a way to disentangle factors of variation arising from identity and expression variations. While (Prince et al., 2011) used an identity-based analysis to make inferences about identity, our work here focuses on how disentangling such factors can be used to make inferences about facial expression. In Section 4.4, we go on to extend the underlying identity and expression analysis technique to show that constrained local models (CLMs) can also be reformulated, extended and improved through using an underlying identity-expression analysis model. Our reformulation also provides a novel energy function and minimization formulation for CLMs in general.

4.2. Relevant Prior Work

Active Appearance Models (AAMs) (Cootes et al., 1995; Edwards et al., 1998) and Constrained Local Models (CLMs) (Saragih et al., 2010) are widely used techinques for keypoint tracking. Both of these methods rely on so called point distribution models (PDMs) (Cootes et al., 1995) which are constructed using principal component analysis (PCA).

AAMs and CLMs usually suffer from a degree of identity-dependence. That is, a model trained on a sample of subjects does not necessarily perform well on an unseen subject. AAMs in particular suffer greatly from this effect, performing much better when samples of an individual are used for both training and testing. View-based approaches (Pentland et al., 1994), and multi-stage solutions (Liao et al., 2004; Tistarelli and Nixon, 2009) address this issue by using multiple subspaces for each identity. Gaussian mixture models, (Frey and Jojic, 1999) indirectly deal with identity variation by learning clusters of training data. (Gross et al., 2005) described reduced fitting robustness of AAMs on unseen subjects, suggesting simultaneous appearance and shape fitting improve generalization. Our method approaches identity variation directly and probabilistically, learning both factors of

variation simultaneously. Some other notable recent work (Jeni et al., 2012) has extended CLMs to account for 3D shape. Another work has used a structured max-margin approach to keypoint placement (Zhu and Ramanan, 2012) yielding state of the art results for facial keypoint placement. We compare with this approach in Section 4.4.

Deep learning has recently emerged as a method capable of yielding state of the art results for keypoint placement (Sun et al., 2013). Deep networks are however known to overfit when data set sizes are small, which is the case for many of the problems that we are addressing in our work here (e.g. markerless facial performance capture imagery from helmet mounted cameras).

Multi-linear and bilinear analysis of facial images (Freeman and Tenenbaum, 1997; Vasilescu and Terzopoulos, 2002; Vlasic et al., 2005) can model the interaction of different kinds of variation. However, with these models, a full image tensor is often needed, which can be difficult to obtain. Some approaches overcome this restriction by treating the required tensor labels as missing (Del Bue et al., 2012). However, this leads to discarding data.

The problem we address in this paper is that of learning an expression representation which leverages identity information, then using the technique to improve a wide variety of expression and emotion related tasks. This is similar in spirit to the expression synthesis approaches used by (Du and Lin, 2003) and (Zhou and Lin, 2005), which generate identity-independent expression factors. Our approach generalizes this work as a framework for unsupervised learning of expression factors, building in particular on the identity-expression factorization technique of (Prince et al., 2011). In the next section we present their probabilistic approach, adapted in this work for expression related tasks. Subsequently, in Section 4.4 we go on to extend the widely used formulation of CLMs so as to use identity-expression factorized representations. We show how keypoint localization performance can be improved over the traditional techniques which use principal component analysis as their underlying point distribution models.

4.3. A Model for Disentangling Identity and Expression

Face image datasets have a rich diversity due to many contributing factors of variation such as age, illumination, identity, pose, facial expression and emotion. In this literature, we consider two

main sources of variation. The primary variation is due to the identity factors and the second source of variation is due to the facial expressions as well as some degree of pose deviation.

A graphical model of this approach is shown in Figure 4.1, where for each face image a set of keypoints $\mathbf{y}_{ij} \in \mathbf{Y}$ is generated by $\mathbf{u}_i$, representing the identity i, and $\mathbf{v}_{ij}$, representing the j^{th} expression of identity i. $\mathbf{y}_{ij}$ represents the set of keypoints of an identity i for a given expression j observed in an image. The set of observed data is $\mathbf{Y} = \{\mathbf{y}_{ij}\}_{i=1,\ldots,N^{id}\ ;\ j=1,\ldots,N_i^{exp}}$ which contains all keypoint configurations of N^{id} identities, where for each each identity i, a total of N_i^{exp} expressions exist.

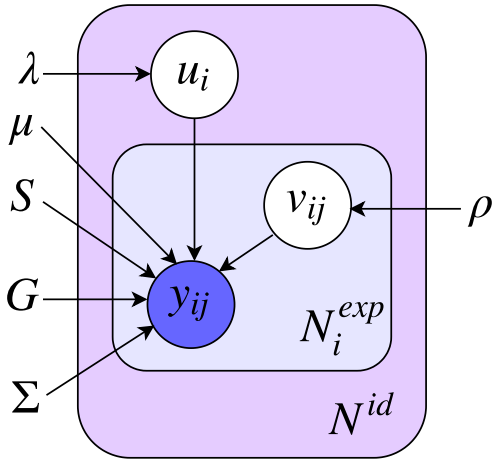


Fig. 4.1. Graphical model of facial landmark generation. $\mathbf{y}_{ij}$ is generated from $p(\mathbf{y}_{ij} \mid \mathbf{v}_{ij}, \mathbf{u}_i)$, after sampling $\mathbf{u}_i$ from an identity and $\mathbf{v}_{ij}$ from an expression-related distribution respectively.

Each $\mathbf{y}_{ij}$ is generated by sampling $\mathbf{u}_i$ and $\mathbf{v}_{ij}$ from Gaussian distributions corresponding to identity i and its j^{th} expression distributions $p(\mathbf{u}_i)$ and $p(\mathbf{v}_{ij})$, and then combining these by sampling keypoints $\mathbf{y}_{ij}$ according to $p(\mathbf{y}_{ij} \mid \mathbf{u}_i, \mathbf{v}_{ij})$. We use zero-mean independent Gaussian distributions for $p(\mathbf{u}_i)$ and $p(\mathbf{v}_{ij})$.

$$p(\mathbf{u}_i) = \mathcal{N}(\mathbf{u}_i;\ \mathbf{0},\ \lambda\mathbf{I}), \tag{4.3.1}$$

$$p(\mathbf{v}_{ij}) = \mathcal{N}(\mathbf{v}_{ij};\ \mathbf{0},\ \rho\mathbf{I}). \tag{4.3.2}$$

The observation $\mathbf{y}_{ij}$ is then sampled from a multivariate Gaussian conditional distribution parameterized by the mean $\boldsymbol{\mu}$, matrices $\mathbf{S}$, $\mathbf{G}$ and diagonal covariance $\boldsymbol{\Sigma}$, such that

$$p(\mathbf{y}_{ij} \mid \mathbf{u}_i, \mathbf{v}_{ij}) = \mathcal{N}(\mathbf{y}_{ij};\ \boldsymbol{\mu} + \mathbf{S}\mathbf{u}_i + \mathbf{G}\mathbf{v}_{ij},\ \boldsymbol{\Sigma}). \tag{4.3.3}$$

This corresponds to a conditional distribution given the variable $\mathbf{u}_i$, which is identical for all images of a unique identity, and the variable $\mathbf{v}_{ij}$, which varies across different expressions of a particular identity. The loading matrices $\mathbf{S}$ and $\mathbf{G}$ correspond respectively to identity and expression and are shared across all observations. The joint probability can be written as

$$\begin{aligned} p(\mathbf{Y}, \mathbf{U}, \mathbf{V} \mid \mathbf{S}, \mathbf{G}, \lambda, \rho, \boldsymbol{\mu}) = \prod_{i}^{N^{id}} \mathcal{N}(\mathbf{u}_i;\ \mathbf{0},\ \lambda\mathbf{I}) \\ \prod_{j}^{N_i^{exp}} \mathcal{N}(\mathbf{y}_{ij};\ \boldsymbol{\mu} + \mathbf{S}\mathbf{u}_i + \mathbf{G}\mathbf{v}_{ij},\ \boldsymbol{\Sigma}) \mathcal{N}(\mathbf{v_{ij}};\ \mathbf{0},\ \rho\mathbf{I}). \end{aligned} \tag{4.3.4}$$

where $\mathbf{Y}$ is the entire set of keypoints and $\mathbf{U} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{N^{id}}\}$ is the set of all latent identity representations and $\mathbf{V} = \{\mathbf{v}_{ij}\}_{i=1,\ldots,N^{id}\ ;\ j=1,\ldots,N_i^{exp}}$ is the set of all latent expression representations for all identities.

Learning

Since model parameters $\boldsymbol{\theta}$ depend on the latent variables $\{\mathbf{V}, \mathbf{U}\}$ and the latent variables depend on the model parameters, expectation maximization (EM) approach is used to learn the parameters of the model $\boldsymbol{\theta} = \{\mathbf{S}, \mathbf{G}, \boldsymbol{\Sigma}, \boldsymbol{\mu}, \lambda, \rho\}$. The goal is to maximize the joint distribution

$$\max \mathbb{E}[\log p(\mathbf{Y}, \mathbf{U}, \mathbf{V} \mid \boldsymbol{\theta})], \tag{4.3.5}$$

where in the E-step the parameters $\boldsymbol{\theta}$ are kept fixed and the expectation is taken with respect to the posterior distribution to update $\mathbf{U}$ and $\mathbf{V}$

$$p(\mathbf{U}, \mathbf{V} \mid \mathbf{Y}, \boldsymbol{\theta}^{\text{old}}) \tag{4.3.6}$$

and then in the M-step the latent variables $\mathbf{U}, \mathbf{V}$ are kept fixed and the parameters in $\boldsymbol{\theta}$ are updated by maximizing the log of the joint distribution

$$\max_{\boldsymbol{\theta}} \mathbb{E}[\log p(\mathbf{Y}, \mathbf{U}, \mathbf{V} \mid \boldsymbol{\theta})]. \tag{4.3.7}$$

Since the priors in Eqs. (4.3.1) and (4.3.2) and the likelihood distribution in Equation (4.3.3) are all Gaussian, the resulting joint distribution is also Gaussian.

In this part we explain how to do E and M steps in detail. For a given identity i, the set of observed variables $\{\mathbf{y}_{ij}\}_{j=1,\ldots,N_i^{exp}}$ with N_i^{exp} expressions can be written as a single feature vector $\mathbf{y}_i = (\mathbf{y}_{i1}^T, \mathbf{y}_{i2}^T, \ldots, \mathbf{y}_{iN_i^{exp}}^T)^T$. Similarly the factors can be combined into a single loading matrix

$$\mathbf{A}_i = \begin{pmatrix} \mathbf{S} & \mathbf{G} & 0 & \ldots & 0 \\ \mathbf{S} & 0 & \mathbf{G} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \mathbf{S} & 0 & 0 & \ldots & \mathbf{G} \end{pmatrix}, \tag{4.3.8}$$

where the number of rows and columns in $\mathbf{A}_i$ are N_i^{exp} and $N_i^{exp}+1$. The latent space representation corresponding to identity i becomes

$$\mathbf{b}_i = (\mathbf{u}_i^T, \mathbf{v}_{i1}^T, \mathbf{v}_{i2}^T, \ldots, \mathbf{v}_{iN_i^{exp}}^T)^T. \tag{4.3.9}$$

$\mathbf{b}_i$ is a vector of dimensionality $dim_{\mathbf{u}_i} + dim_{\mathbf{v}_{ij}} \times N_i^{exp}$, where $dim_{\mathbf{u}_i}$ is the dimensionality of $\mathbf{u}_i$ and $dim_{\mathbf{v}_{ij}}$ is the dimensionality of $\mathbf{v}_{ij}$. Since $\mathbf{b}_i$ is composed of two sets of vectors with zero-mean Gaussian distributions, it is also distributed as a zero mean Gaussian:

$$p(\mathbf{b}_i) = \mathcal{N}(\mathbf{b}_i;\, \mathbf{0},\, \boldsymbol{\Phi}_\mathbf{i}), \tag{4.3.10}$$

$$\boldsymbol{\Phi}_i = \text{diag}\left(\lambda_1,\, \ldots,\, \lambda_{dim_{\mathbf{u}_i}},\, \rho_1,\, \ldots,\, \rho_{N^{exp_i} \times dim_{\mathbf{v}_{ij}}}\right), \tag{4.3.11}$$

where the term $\boldsymbol{\Phi}_i$ is a diagonal covariance matrix, in which the first $dim_{\mathbf{u}_i}$ elements of the diagonal are extracted from the diagonal elements of the covariance matrix $\lambda\mathbf{I}$. The diagonal of $\boldsymbol{\Phi}_i$ is then composed of N_i^{exp} blocks of size $dim_{\mathbf{v}_{ij}}$ that are extracted from the diagonal of covariance matrix $\rho\mathbf{I}$ repeated N_i^{exp} times. Intuitively the first set of $dim_{\mathbf{u}_i}$ elements represent identity variations, while each of the N_i^{exp} blocks of size $dim_{\mathbf{v}_{ij}}$ represents an expression variation

of an identity. Given this construction, the probability for identity i can be rewritten as a Gaussian $p(\mathbf{y}_i \mid \mathbf{b}_i) = \mathcal{N}(\mathbf{y}_i;\ \boldsymbol{m}_i + \mathbf{A}_i\mathbf{b}_i,\ \boldsymbol{\Psi}_i)$, where $\boldsymbol{\Psi}_i$ is constructed as a diagonal matrix by concatenating N_i^{exp} times the diagonal of $\boldsymbol{\Sigma}$. The term $\boldsymbol{m}_i$ is N_i^{exp} blocks of $\boldsymbol{\mu}$ being concatenated.

The posterior probability of $\mathbf{b}_i$ is also Gaussian with moments

$$\mathbb{E}[\mathbf{b}_i] = (\boldsymbol{\Phi}_i^{-1} + \mathbf{A}_i^T {\boldsymbol{\Psi}_i}^{-1}\mathbf{A}_i)^{-1}\mathbf{A}_i^T {\boldsymbol{\Psi}_i}^{-1}(\mathbf{y}_i - \boldsymbol{m}_i), \tag{4.3.12}$$

$$\mathbb{E}[\mathbf{b}_i\mathbf{b}_i^T] = (\boldsymbol{\Phi}_i^{-1} + \mathbf{A}_i^T {\boldsymbol{\Psi}_i}^{-1}\mathbf{A}_i)^{-1} - \mathbb{E}[\mathbf{b}_i]\mathbb{E}[\mathbf{b}_i^T]. \tag{4.3.13}$$

In the E-step, the mean and covariance matrices of the posterior distribution $p(\mathbf{b}_i \mid \mathbf{y}_i)$ are taken, which are measured over all of the expressions corresponding to the same identity. This representation style assures that all of the expressions of the same person have the same representation for identity. Note that the expectations in Eqs. (4.3.12) and (4.3.13) should be taken separately for all of the identities in the training set. In the M-step, however, we can disentangle the $\mathbf{b}_i$ into a set of $\mathbf{b}_{ij}$, in which each $\mathbf{b}_{ij}$ contains one sample of identity and one sample of expression. This is due to the fact that the parameters of the model should be updated with respect to all of the data in the training set and having done the E-step, we have already obtained the same identity representation for all expression observations of the same person. Therefore, disentangling $\mathbf{b}_i$ into a set of $\mathbf{b}_{ij}$ can preserve that information while at the same time it encodes a simpler latent representation for updating the parameters of the joint distribution.

We define $\mathbf{b}_{ij} = \begin{bmatrix} \mathbf{u}_i \\ \mathbf{v}_{ij} \end{bmatrix}$ and $\mathbf{C} = [\ \mathbf{S} \quad \mathbf{G}\ ]$, such that $\mathbf{b}_{ij}$ is distributed as

$$p(\mathbf{b}_{ij}) = \mathcal{N}(\mathbf{b}_{ij}; \mathbf{0}, \boldsymbol{\Phi}), \tag{4.3.14}$$

$$\boldsymbol{\Phi} = \text{diag}\left(\lambda_1, \ldots, \lambda_{dim_{\mathbf{u}_i}}, \rho_1, \ldots, \rho_{dim_{\mathbf{v}_{ij}}}\right). \tag{4.3.15}$$

Given this notation, the conditional distribution can now be written as

$$p(\mathbf{y}_{ij}|\mathbf{b}_{ij}) = \mathcal{N}(\mathbf{y}_{ij}; \boldsymbol{\mu} + \mathbf{C}\mathbf{b}_{ij}, \boldsymbol{\Sigma}). \tag{4.3.16}$$

We can further simplify this notation by setting $\tilde{\mathbf{b}}_{ij} = \begin{bmatrix} \mathbf{b}_{ij} \\ 1 \end{bmatrix}$ and $\tilde{\mathbf{C}} = [\ \mathbf{C} \quad \boldsymbol{\mu}\ ]$, which in turn gives the following conditional and prior distributions

$$p(\mathbf{y}_{ij}|\tilde{\mathbf{b}}_{ij}) = \mathcal{N}(\mathbf{y}_{ij}; \tilde{\mathbf{C}}\tilde{\mathbf{b}}_{ij}, \boldsymbol{\Sigma}), \tag{4.3.17}$$

$$p(\tilde{\mathbf{b}}_{ij}) = \mathcal{N}(\tilde{\mathbf{b}}_{ij}; \mathbf{0}, \tilde{\boldsymbol{\Phi}}) \tag{4.3.18}$$

with $\tilde{\boldsymbol{\Phi}}$ being equal to

$$\tilde{\boldsymbol{\Phi}} = \begin{bmatrix} \boldsymbol{\Phi} & \mathbf{0}_{dim_{\mathbf{u}_i}+dim_{\mathbf{v}_{ij}},1} \\ \mathbf{0}_{1,dim_{\mathbf{u}_i}+dim_{\mathbf{v}_{ij}}} & 0_{1,1} \end{bmatrix}, \tag{4.3.19}$$

where two zero vectors with row or column size of $dim_{\mathbf{u}_i} + dim_{\mathbf{v}_{ij}}$ are concatenated with $\boldsymbol{\Phi}$ and another zero element to build a square matrix of size $dim_{\mathbf{u}_i} + dim_{\mathbf{v}_{ij}} + 1$. The joint distribution is then

$$p(\mathbf{Y}, \mathbf{B}|\boldsymbol{\Sigma}, \tilde{\mathbf{C}}, \tilde{\boldsymbol{\Phi}}) = \prod_{\mathbf{ij}} \mathcal{N}(\mathbf{y_{ij}}; \tilde{\mathbf{C}}\tilde{\mathbf{b}}_{\mathbf{ij}}, \boldsymbol{\Sigma})\mathcal{N}(\tilde{\mathbf{b}}_{\mathbf{ij}}; \mathbf{0}, \tilde{\boldsymbol{\Phi}}). \tag{4.3.20}$$

with $\mathbf{Y}$ representing the whole set of observed keypoints and $\mathbf{B}$ representing the entire set of latent variables $\tilde{\mathbf{b}}_{ij}$. This simplification is preferred since the variables $\mathbf{S}, \mathbf{G}, \boldsymbol{\mu}$ are mutually dependent. Updating each one requires the other two. In the new notation, all of them can be updated simultaneously by updating $\tilde{\mathbf{C}}$. Maximizing with respect to $\tilde{\mathbf{C}}, \boldsymbol{\Sigma}, \tilde{\boldsymbol{\Phi}}$, yields the following updates in the M-step

$$\tilde{\boldsymbol{\Phi}} = \frac{1}{N} \sum_{ij} \text{diag} \left\{ \mathbb{E} \left[\tilde{\mathbf{b}}_{ij} \tilde{\mathbf{b}}_{ij}^T \right] \right\}, \tag{4.3.21}$$

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{ij} \text{diag} \left\{ \tilde{\mathbf{C}} \big(\mathbb{E} \left[\tilde{\mathbf{b}}_{ij} \tilde{\mathbf{b}}_{ij}^T \right] \big) \tilde{\mathbf{C}}^T + \mathbf{y}_{ij} \mathbf{y}_{ij}^T - 2 \mathbf{y}_{ij} \big(\mathbb{E} \left[\tilde{\mathbf{b}}_{ij}^T \right] \big) \tilde{\mathbf{C}}^T \right\}, \tag{4.3.22}$$

$$\tilde{\mathbf{C}} = \sum_{ij} \left\{ \mathbf{y}_{ij} \big(\mathbb{E} \left[\tilde{\mathbf{b}}_{ij}^T \right] \big) \right\} \left\{ \sum_{ij} \mathbb{E} \left[\tilde{\mathbf{b}}_{ij} \tilde{\mathbf{b}}_{ij}^T \right] \right\}^{-1}. \tag{4.3.23}$$

Note that the parameters are updated with respect to all expressions of all identities. As for inference at test time, the procedure for determining the optimal $\mathbf{b}_{ij}$ vector is straight-forward, using the

following posterior distribution:

$$p(\mathbf{b}_{ij}|\mathbf{y}_{ij}) = \mathcal{N}(\mathbf{b}_{ij}; (\mathbf{\Phi}^{-1} + \mathbf{C}^T\mathbf{\Sigma}^{-1}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{\Sigma}^{-1}(\mathbf{y}_{ij} - \boldsymbol{\mu}), (\mathbf{\Phi}^{-1} + \mathbf{C}^T\mathbf{\Sigma}^{-1}\mathbf{C})^{-1}). \quad (4.3.24)$$

4.4. Identity-Expression Constrained Local Models (IE-CLMs)

The term Constrained Local Models (CLMs) has evolved from the original work of (Cristinacce and Cootes, 2006), which can be viewed as a particular instance of a CLM (Saragih et al., 2010). Nowadays the term CLM has come to refer to a number of methods which involve finding the landmarks of an image $\mathbf{x}$ through assigning a cost $\mathcal{L}$ to candidate landmark positions $\{\mathbf{y}^1, ..., \mathbf{y}^K\} \in \mathbf{y}$ on the image and the parameters of the model $\mathbf{p}$. The corresponding objective function can be written as:

$$\mathcal{L}(\mathbf{y}, \mathbf{p}) = \sum_{k=1}^{K} \mathcal{D}^k(\mathbf{y}^k; \mathbf{x}) + Reg(\mathbf{y},\mathbf{p}), \quad (4.4.1)$$

where $\mathcal{D}^k$ encodes the image dependent suitability measure for the k^{th} landmark being located at position $\mathbf{y}^k$ in the image and K indicates the number of landmarks of a face. The Reg term can be interpreted as a regularization term that encodes preferences for certain spatial configurations of landmark positions[1]. This set-up leads to what some refer to as a deformable model fitting problem. Further, it is common to use a linear approximation for how the shape of non-rigid objects deform and a common variant of such a modeling technique is known as a point distribution model or PDM (Cootes et al., 1995). The PDM of (Cootes et al., 1995) models non-rigid shape variations linearly and composes them with a global rigid transformation such that the 2D location of the PDM's k^{th} landmark is given by:

$$\mathbf{y}^k = s\mathbf{R}(\boldsymbol{\mu}^k + \mathbf{\Phi}^k\mathbf{q}) + \mathbf{t}, \quad (4.4.2)$$

where the PDM parameters are defined by $\mathbf{p} = \{s,\mathbf{R},\mathbf{t},\boldsymbol{\mu},\mathbf{\Phi}\}$. These parameters consist of a global scale s, rotation $\mathbf{R}$, and translation $\mathbf{t}$ (forming a similarity transformation), as well as global non-rigid deformations though a sub-matrix $\mathbf{\Phi}^k$, which is part of a larger matrix $\mathbf{\Phi}$. $\boldsymbol{\mu}$ is the

[1]We have reversed the order of terms compared to the notation in (Saragih et al., 2010) and indicated an explicit dependence on $\mathbf{y}_i$ for the underlying objective and Reg.

concatenation of all $\boldsymbol{\mu}^k$ values for $k \in \{1, \ldots, K\}$, with $\boldsymbol{\mu}^k$ indicating the average location of landmark k. Using a probabilistic notation, the probability over the landmark position $\mathbf{y}^k$ can be represented as a normal distribution with mean $s\mathbf{R}(\boldsymbol{\mu}^k + \boldsymbol{\Phi}^k\mathbf{q}) + \mathbf{t}$ and a covariance of σ^2, such that

$$p(\mathbf{y}^k) = \mathcal{N}(s\mathbf{R}(\boldsymbol{\mu}^k + \boldsymbol{\Phi}^k\mathbf{q}) + \mathbf{t}, \sigma^{\mathbf{2}}). \tag{4.4.3}$$

Consequently, the distribution over the whole set of landmarks $\mathbf{y}$ takes the form of a normal distribution which corresponds to

$$p(\mathbf{y}) = \mathcal{N}(s\mathbf{R}(\boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{q}) + \mathbf{t}, \sigma^{\mathbf{2}}\mathbf{I}), \tag{4.4.4}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Phi}$ represent respectively a concatenation of the terms $\boldsymbol{\mu}^k$ and $\boldsymbol{\Phi}^k$ for $k \in \{1, \ldots, K\}$. The covariance $\sigma^2\mathbf{I}$ is a diagonal matrix having the values in its diagonal being repeated K times. Equation (4.4.4) gives a principled probabilistic representation for the regularization term Reg in Equation (4.4.1). It is common to use a joint distribution over landmarks as a prior for the regularization term Reg.

4.4.1. A Deeper Probabilistic View of PDMs

When viewed through the lens of modern graphical modeling techniques and probabilistic principal component analysis, the classical formulation of PDMs can be re-written more formally as a probabilistic generative model over all K landmarks in $\mathbf{y}$, where

$$\begin{aligned} p(\mathbf{y}) &= P(\mathbf{y}|\mathbf{z})\mathbf{P}(\mathbf{z}|\mathbf{q})\mathbf{P}(\mathbf{q}) \\ &= \int_{\mathbf{q}}\int_{\mathbf{z}} \mathcal{N}(\mathbf{y};\, s\mathbf{R}\mathbf{z} + \mathbf{t},\, \alpha\mathbf{I})\mathcal{N}(\mathbf{z};\, \boldsymbol{\mu} + \boldsymbol{\Phi}\mathbf{q}, \sigma^{\mathbf{2}}\mathbf{I})\mathcal{N}(\mathbf{q};\, \mathbf{0},\, \mathbf{I})\, \mathbf{dq}\, \mathbf{dz}. \end{aligned} \tag{4.4.5}$$

Variable $\mathbf{q}$ acts as the prior of the $p(\mathbf{y})$ distribution with mean zero and unit covariance matrix. Given $\mathbf{q}$, the conditional $P(\mathbf{z}|\mathbf{q})$ gets its mean by applying the non-rigid deformation $\boldsymbol{\Phi}$ and adding the $\boldsymbol{\mu}$ term. The mean of the conditional $P(\mathbf{z}|\mathbf{q})$ sets the intermediate value in the mean of Equation (4.4.4) before applying the rigid transformations. Finally, the mean of the conditional $P(\mathbf{y}|\mathbf{z})$ is measured by applying affine transformations of rotation $\mathbf{R}$, scaling s, and translation $\mathbf{t}$ on the variable $\mathbf{z}$.

Having integrated out the variable $\mathbf{q}$, the distribution over all landmarks $\mathbf{y}$ can be simplified to

$$p(\mathbf{y}) = \int_{\mathbf{z}} \mathcal{N}(\mathbf{y};\, s\mathbf{R}\mathbf{z} + \mathbf{t},\, \alpha\mathbf{I})\mathcal{N}(\mathbf{z};\, \boldsymbol{\mu},\, \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathbf{T}} + \sigma^{\mathbf{2}}\mathbf{I})\, \mathbf{dz}. \tag{4.4.6}$$

Note that in this formulation the variable $\mathbf{z}$ represents the set of all landmarks in a transformed space where they are represented in a standardized coordinate space. That is different from their representation in $\mathbf{y}$ where each landmark is undergone an affine transformation and therefore can be observed in more volatile regions of the image. This difference guides us to a better representation in the $\mathbf{z}$ space to formulate the distribution of the keypoints. Therefore, we transform the landmarks in each image from the $\mathbf{y}$ space to $\mathbf{z}$ space through a set of affine transformation, whose parameters are trained to be robust against translation, rotation, and scaling transformations. Note that Eq. (4.4.6) provides a probabilistic PCA (PPCA) representation for the regularization term Reg in Equation (4.4.1). We use this PPCA notation in Section 4.4.2 for a PPCA-CLM formulation over the keypoints, however, as discussed above we operate in the $\mathbf{z}$ space instead of $\mathbf{y}$ space. From this point forward, we discuss our model over the landmarks in the $\mathbf{z}$ space, both when using the Probabilistic PCA model in Section 4.4.2 and identity-expression model in Section 4.4.3.

4.4.2. Probabilistic PCA based CLMs

Previous work (Hasan et al., 2013) has formulated a CLM in the following way: for a given image $\mathbf{x}$, we wish to combine the outputs of the local classifiers for the keypoints with a spatial model of global keypoint configuration. The local classifiers acts as discriminative predictors for the keypoints, while the spatial model provides a prior on how the keypoints are distributed. The local classifier is the term $\mathcal{D}^k$ for keypoint k in Equation (4.4.1) and the global model, as the joint distribution over keypoints, is the term Reg in the same equation.

To obtain a local classifier, for each keypoint k a local SVM classifier is trained. At test time, the local classifier for keypoint k, generates a response image map $\mathbf{f}^k$, which is a 2D array with a probability prediction for each pixel position in the image being the keypoint k. The set $\mathbf{f} = \{\mathbf{f}^1, \mathbf{f}^2 \cdots, \mathbf{f}^K\}$ represents the set of all generated response images for a given image, where K is the number of keypoints. Figure 4.2 provides a visualization of the response image probability values. Note that the probabilities are scaled by a factor of 255 (8 bit gray-scale images).

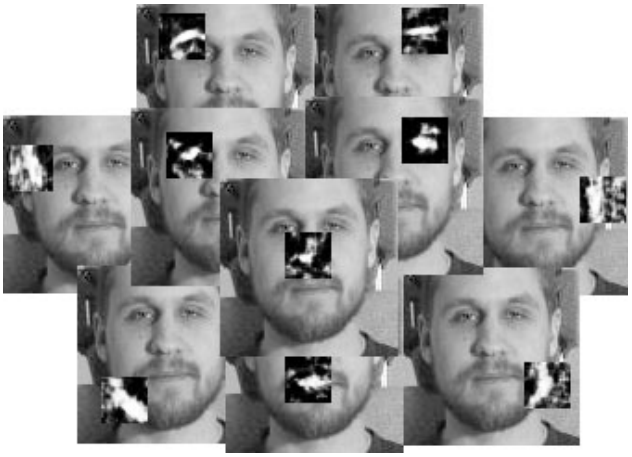


Fig. 4.2. Sample SVM response maps for an image generated by trained SVMs. Each response map is generated by one local classifier trained for a particular keypoint.

Let $\tilde{\mathbf{z}}^k \in \mathbf{f}^k$ indicate the coordinates of a gridpoint location on the 2D response image for keypoint k, then log of the score for the positive prediction of the local classifier at this location can be defined as $sc(\tilde{\mathbf{z}}^k)$. We use a probabilistic PCA over the keypoints to model the global keypoint configurations (the Reg term) and use the log of its Gaussian distribution with a factorized covariance matrix as its energy term and couple it with the log score of the local classifier predictions in a spatial interaction energy function as follows:

$$\begin{aligned} E(\mathbf{z}_{ij}) &= \sum_{k=1}^{K} \mathcal{D}^k(\mathbf{z}_{ij}^k; \mathbf{x}) + Reg(\mathbf{z}_{ij}, \mathbf{p}) \\ &= -\sum_{k=1}^{K} \sum_{\tilde{\mathbf{z}}_{ij}^k \in \mathbf{f}_{ij}^k} sc(\tilde{\mathbf{z}}_{ij}^k)\delta(\mathbf{z}_{ij}^k - \tilde{\mathbf{z}}_{ij}^k) \\ &\quad + \frac{1}{2}\left(\mathbf{z}_{ij} - \boldsymbol{\mu}\right)^T \left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathbf{T}} + \sigma^{\mathbf{2}}\mathbf{I}\right)^{-\mathbf{1}} \left(\mathbf{z}_{\mathbf{ij}} - \boldsymbol{\mu}\right), \end{aligned} \tag{4.4.7}$$

where $\mathbf{z}_{ij} = [\mathbf{z}_{ij}^1, \mathbf{z}_{ij}^2, \cdots \mathbf{z}_{ij}^K]^T$ gives the coordinates of the K candidate locations for the keypoints of the j-th expression of the i-th identity, whose energy is measured. The energy function E is composed of two terms: The first term consists of the local response maps contribution where the first sum is over the K keypoints and the second sum gets the log score $sc(\tilde{\mathbf{z}}_{ij}^k)$ for each point $\tilde{\mathbf{z}}_{ij}^k$ in the set of all grid locations of the response image map $\mathbf{f}_{ij}^k$. The term δ is the Dirac delta function whose output is one only if the grid location $\tilde{\mathbf{z}}_{ij}^k$ equals the queried location $\mathbf{z}_{ij}^k$. The second term

in Equation (4.4.7) is log of the probabilistic PCA (PPCA) where $\boldsymbol{\mu}$ is simply the mean of the keypoints after RANSAC similarity registration. The terms $\boldsymbol{\Phi}$ and σ^2 in the covariance matrix of PPCA equal to:

$$\boldsymbol{\Phi} = \mathbf{U}_p(\boldsymbol{\Lambda}_p - \sigma^2\mathbf{I})^{\mathbf{1/2}}\boldsymbol{\Omega}, \tag{4.4.8}$$

$$\sigma^2 = \frac{1}{K-p}\sum_{k=p+1}^{K}\lambda^k, \tag{4.4.9}$$

where $\mathbf{U}_p$ is a matrix of the p principle eigenvectors of the keypoints in $\mathbf{z}$ space, $\boldsymbol{\Lambda}_p$ is a diagonal matrix whose diagonal is composed of eigenvalues $\lambda^1, \ldots, \lambda^p$, knowing that eigenvalue λ^k corresponds to the eigenvector in column k of the matrix $\mathbf{U_p}$, and $\boldsymbol{\Omega}$ is an arbitrary orthogonal matrix. The term σ^2 is the average of the remaining eigenvalues which explain the least significant direction of variation in data. Note that the $\mathbf{z}$ term used in Eq. (4.4.7) is the same as the $\mathbf{z}$ term presented in Eq. (4.4.6). The keypoints are transformed from the $\mathbf{y}$ space to the $\mathbf{z}$ space through a set of learned affine transformation parameters, namely s, R, and t, where face keypoints are mapped from the original more dynamic $\mathbf{y}$ space to a more unified representation in the $\mathbf{z}$ space.

To minimize E we perform a search over the candidate keypoint locations $\mathbf{z}$. This is done by iterating over the keypoints, where in each iteration all keypoints are fixed except one. The optimum value for that keypoint is found through a comprehensive search over the energy of the local response map grid locations. The iterations continue until the maximum keypoint location change over all keypoints is less than a small threshold. Note that the term $E(\mathbf{z}_{ij})$ in Eq. (4.4.7) is minimized for each expression j of identity i separately in this formulation. We refer to the model described in this section as probabilistic PCA constrained local model (PPCA-CLM).

4.4.3. Identity-Expression Factorized CLMs

Once the parameters of the identity expression model are learned using the EM procedure, as explained in Section 4.3, this model is used in conjunction with the local response classifiers derived from the previously trained SVMs. Combining the energy of the local classifier with the energy term of the Identity Expression Factorized model, we minimize the energy of the following function:

$$E(\boldsymbol{w}_i) = -\sum_{\mathbf{z_{ij}}\in\boldsymbol{w}_i}\sum_{k=1}^{K}\sum_{\tilde{\mathbf{z}}_{ij}^k\in\mathbf{f}_{ij}^k} sc(\tilde{\mathbf{z}}_{ij}^k)\delta(\mathbf{z}_{ij}^k - \tilde{\mathbf{z}}_{ij}^k) \tag{4.4.10}$$
$$+ (\boldsymbol{w}_i - \boldsymbol{m})^T \left(\boldsymbol{\psi} + \mathbf{A}\boldsymbol{\Phi}\mathbf{A}^T\right)^{-1} (\boldsymbol{w}_i - \boldsymbol{m}),$$

where $\boldsymbol{w}_i$ represents a set of candidate keypoints $\mathbf{z}_{ij}$ for all expressions j belonging to the same identity i whose energy is measured. Note that as in the previous section $\mathbf{z}_{ij}$ represents the set of all candidate keypoint locations on a single query image of identity i and expression j. Assuming there are a total of N_i^{exp} images in $\boldsymbol{w}_i$ representing different expressions of a given identity, the term $\boldsymbol{m}$ is simply N_i^{exp} times concatenation of $\boldsymbol{\mu}$. Compared to the first term in Equation (4.4.7), there is an added summation which iterates over all images belonging to the same identity. The second term in Equation (4.4.10), is equivalent to the log of the marginal distribution $P(\boldsymbol{w}_i)$ of the identity expression factorized model, explained in Section 4.3, which gets the probability of the joint set of keypoints $\boldsymbol{w}_i$ in all images of the same identity being valid keypoint locations under the joint distribution. The term $\boldsymbol{\Psi}$ is constructed as a diagonal matrix by concatenating N_i^{exp} times the diagonal of $\boldsymbol{\Sigma}$ and the matrix definitions of $\mathbf{A}$ and $\boldsymbol{\Phi}$ are given respectively in Equation (4.3.8) and Equation (4.3.11).

The optimization procedure is done as follows: for each keypoint $k \in K$ in each expression $j \in N_i^{exp}$ of the identity i a SVM response map $\mathbf{f}_{ij}^k$ is generated. In each response map the position with the highest probability is chosen as the keypoint location. These set of selected locations give the initial value of $\boldsymbol{w}_i$. Then, using Iterated Conditional Modes (ICM) (Besag, 1986), the energy of (4.4.10) is minimized by updating the position of each keypoint to its minimum energy position while having all other keypoints fixed, iterating over all keypoints multiple times until the maximum change in the keypoint locations are smaller than a threshold. We refer to the model described in this section as identity expression constrained local model (IE-CLM).

4.4.4. Keypoint Localization Experiments with the MultiPIE Dataset and IE-CLMs

The CMU MultiPIE face dataset (Gross et al., 2010) captures a subset of expressions, such as surprise, smile, neutral, squint, disgust, and scream, of 337 identities. The images are taken in four

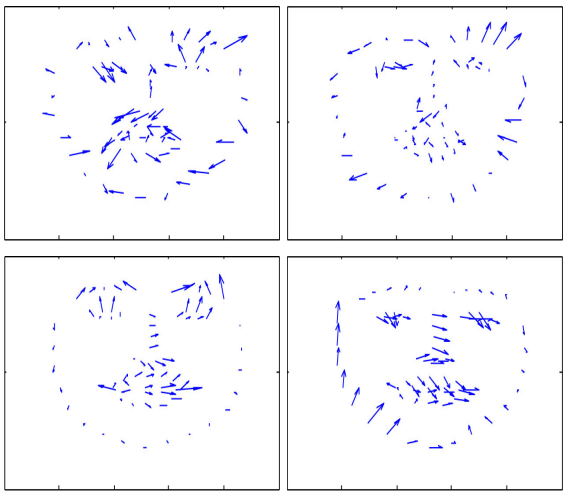


Fig. 4.3. Top row: the 1st (left) and 2nd (right) directions of variation for identity. Second row: 1st (left) and 2nd (right) directions of variation for expression.

sessions and in each session a subset of expressions is captured. Therefore, not all expressions are registered per identity. The availability of multiple expressions per identity provides the right dataset to evaluate our identity expression model. We perform the same experiment as in (Zhu and Ramanan, 2012) and compare our model with other models on the frontal face images with 68 keypoints. Following the same split of the dataset as in (Zhu and Ramanan, 2012), the first 300 images are used for training and the next 300 images are used for testing.

Figure 4.3 shows the main directions of variation for the identity and expression features captured by our model on the MultiPIE training set. The directions of variation for expression captures how faces change in between different emotions. On the other hand, the directions of variation for identity mainly deals with different face sizes as well as some 2D in-plane rotations which is due to the fact that the faces are not completely frontal in some of the expressions of an identity.

Using the training set of MultiPIE, we train a binary SVM classifier on HOG feature representations for each keypoint. Indeed, for training the SVM classifier of each keypoint, positive patches are generated from the training set centered at the keypoint and negative patches are generated within a bounding box around the keypoint. We compare our model on the MultiPIE test-set with the independent model of (Zhu and Ramanan, 2012) and other models reported in that paper as well

Tab. 4.1. Percentage of faces with an average localization error less than the given fraction of face size on MultiPIE dataset. This metric is used in (Zhu and Ramanan, 2012) and we leveraged it in order to compare our model with other models in the literature.

Fraction of face size	0.03	0.04	0.05	0.06
(Zhu and Ramanan, 2012)	0.86	0.97	1.00	1.00
(Hasan et al., 2013)	0.78	0.94	1.00	1.00
Oxford (Everingham et al., 2006)	0.28	0.77	0.94	0.98
Star Model (Felzenszwalb et al., 2010)	0.29	0.80	0.92	0.92
Multi-AAM (Kroon, 2012)	0.64	0.87	0.91	0.92
CLM of (Saragih et al., 2010)	0.68	0.85	0.90	0.93
Face.com (fac)	0.31	0.61	0.79	0.87
Our PPCA-CLM	0.13	0.54	0.79	0.92
Our IE-CLM	0.52	0.89	0.98	1.00

as the more recent model of (Hasan et al., 2013). The error is the average difference in Euclidean space between the true pixel locations and the predicted ones normalized by the face size, which is the average of height and width. The results are reported in Table 4.1. Our model catches up with Zhu's model at 0.06 of face fraction size, though at a face fraction size of 0.05 the difference is marginal at 0.016. The values of the CLM model of (Saragih et al., 2010) are reported from (Zhu and Ramanan, 2012) where the latter source evaluates the previously trained CLM model on the MultiPIE test-set. However, it is not clear on which data the CLM model was trained on and which feature representation was used for training them. In order to provide a CLM model trained on precisely the same training set and using precisely the same features and local classifiers, we use the PPCA-CLMs described in Section 4.4.2, which are trained on HOG features. This allows us to establish a more highly controlled baseline for the PCA versus IE-CLM comparison. The performance of our PPCA-CLM is different from the PCA-CLM of (Saragih et al., 2010). These differences could be due to the difference in the training set used, or due to the difference in the feature representation. Other variations could arise from our slightly different underlying CLM formulation or our use of ICM as the inference procedure. We see that for small face fraction size accuracy the difference is significant; however, at the 0.06 face fraction threshold, the performance of the two PCA based CLM methods are comparable.

Fig. 4.4. Sample of keypoint localization for multiple expressions of different identities by the IE-CLM model.

Turning to the IE-CLM we see that it lags behind Saragih's CLM only at the very small face fraction accuracy level. We speculate that the lower performance of IE-CLM in this range is due to the difference in the training data for the local classifiers. The IE-CLM model catches up rapidly to both Saragih's CLM and the Multi-AAM (Kroon, 2012) models from the 0.04 face fraction level forward, and yields predictions for 100% of the points with an error of less than 0.06 of the face size. This level of performance is on par with the state of the art methods of (Zhu and Ramanan, 2012) and (Hasan et al., 2013).

Figures 4.5 and 4.6 illustrate our model's performance on different expressions of two sample identities in the test-set. Our model exploits the fact that all these images have the same identity and therefore, the same identity representation is used when the model searches for the optimal keypoints. Figure 4.4 shows some sample keypoint localization by our model on different identities in the test set, which includes data of different expression, ethnicity, age, gender, illumination, and face masks such as trimming style and eye-glasses.

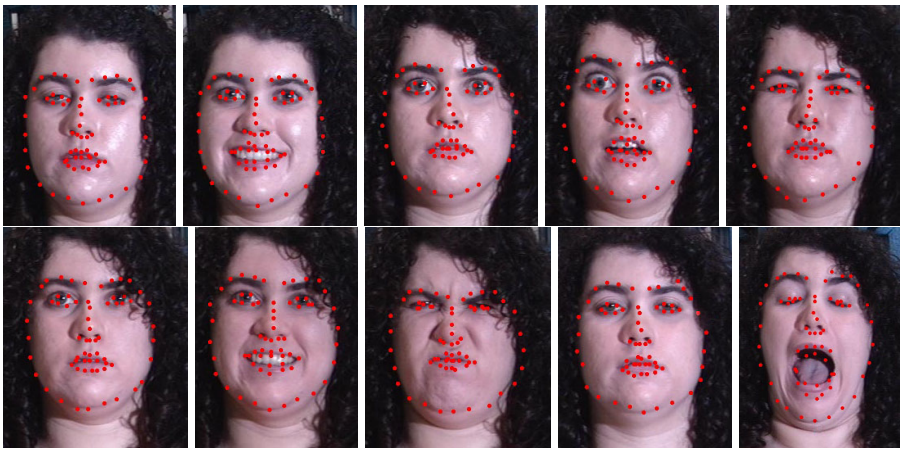


Fig. 4.5. Images labeled by Identity Expression Factorized CLM model for different expressions of identity sample 1.

Fig. 4.6. Images labeled by Identity Expression Factorized CLM model for different expressions of identity sample 2.

In terms of running time, the PPCA-CLM model takes 32 seconds for the local response image generation and 6 seconds for the optimization procedure. However, we have used none of the obvious acceleration techniques that could be applied to speed the local response map generation procedure, such as GPU acceleration or optimizing operations that are convolutional in nature. The IE-CLM model's time changes linearly based on the number of images of the identity being

evaluated. If intended for an interactive or real-time procedure where the images are to be processed sequentially, the timing could be made much more comparable to a PCA-CLM for which various real-time implementations are available. To accelerate the IE-CLM computations one could adapt the joint inference procedure over all images in a batch so as to *incrementally* use estimated factors from images in previous time steps. This could be formulated as a form of incremental inference or probability propagation, an approach which is fairly well understood in the literature.

4.5. Conclusions

We have shown how the identity-expression factorization approach can be integrated into a CLM for keypoint localization. We believe there are a number of possible directions that could lead to further improvements. Firstly, we believe that replacing the binary SVM classifier with a multi-class SVM might yield increased performance with minimal changes required to the underlying model formulation. However, once formulated in this way, it would also be more straight-forward to train the model to make structured predictions using a fully discriminative training procedure, for example, along the lines of (Zhu and Ramanan, 2012). Another direction is to replace the Iterated Conditional Modes (ICM) energy minimization step with a temperature based annealing procedure. The ICM algorithm is known to be quite fast at finding local minima, but it may be the case that better solutions could be found with a stochastic annealing procedure.

The use of multi-scale intensity response information in conjunction with alternative optimization techniques could be a promising direction for future research. It is also likely possible to speed up response image generation and energy minimization when multiple images are processed per identity through various means. Finally, the local SVM classifiers and hand engineered features that we have used here could be replaced with a convolutional neural network (CNN), such as the architecture proposed in (Honari et al., 2016). At the time this work was performed, a CNN approach was not feasible in the context of our primary goal of animation control (one of the tasks aimed for by our model). The reason is that CNNs typically require a considerable amount of training data to avoid overfitting and our face camera dataset was quite small compared to the data sets that have been used to train CNNs. However, for other application settings and datasets

research combining identity-expression disentanglement techniques with CNNs may be a promising direction of future research.

Acknowledgements

We would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support under the CRD and Discovery grant programs and the Fonds de recherche du Québec - Nature et technologies (FRQNT) for a doctoral research scholarships (B2) grant to Sina Honari. We thank Yoshua Bengio for his comments and suggestions regarding this work.

Chapter 5

Prologue to Second Article

5.1. Article Details

Recombinator Networks: Learning Coarse-to-Fine Feature Aggregation. Sina Honari, Jason Yosinski, Pascal Vincent, and Christopher Pal. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)*.

I started this project by first implementing the TCDCN model (Zhang et al., 2014c). Jason and I then evaluated the results and tried to improve them, given the shortcomings of the TCDCN model. The idea of passing the encoded features in the decoding path emerged in one of these meetings when we wanted to find a solution to the information lost due to max pooling layers. This is how the ReCombinator Networks (RCN) idea emerged. We came up with the idea of this model independently and parallel to the U-Net model (Ronneberger et al., 2015), a model with a very similar architecture which was published in *Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015)*. Our model got published a bit later in *CVPR 2016* since it was initially rejected from *Neural Information Processing Systems (NIPS 2015)* and to maintain anonymity we did not put it on arXiv earlier. NIPS and MICCAI had almost the same submission deadline time in 2015. Using our RCN architecture idea, we were able to significantly improve the results over then the state-of-the-art baseline. Later, we observed noise in the keypoint prediction of the RCN model and wanted to find a solution for the noisy keypoint predictions of the model, where some of the keypoints appeared completely unaligned, i.e. inconsistent with respect to the predicted positions of the other keypoints. I discussed the issue at length with Pascal and together we came up with the convolutional denoising model that we use for post-processing, as a framework to learn and leverage the joint distribution over the keypoints. This helped further improve the results. The

implementation and experimental runs was done completely by myself. Jason helped on discussing the problems that emerged in this project to curb many obstacles that hindered our progress, both on the implementational details and the ideas aspects. The variable notations in this chapter have been changed compared to the published article version to unify the notations of the repeating concepts in the thesis.

5.2. Context

At the time this article was written, the state-of-the-art models for landmark localization either used fully connected output layers for landmark localization, such as in (Zhang et al., 2014c), and lost precision due to usage of earlier max pooling layers or used a coarse feature extractor on the entire image and made a hard decision on where to crop the image and passed the cropped patch to a localization model that processed features at full resolution, as in (Sun et al., 2013; Tompson et al., 2015). Some other approaches, such as Fully Convolutional Networks (FCN) (Long et al., 2015) or Hypercolumns (Hariharan et al., 2015) used features from the entire image without requiring a hard crop, however, they merged features of different resolutions by upsampling them back to the image resolution and merging them right before predicting the outputs. Our proposed model instead uses an encoder-decoder approach, where the extracted coarse features are re-introduced in the decoding path using skip connections to progressively recover the information lost due to pooling layers. In this process instead of upsampling all encoded resolutions back to the image resolution at once, as in FCN or Hypercolumns, we only upsample features to one resolution higher at each step, allowing the finer features to be computed using the coarser features. This gradual resolution increase in reconstruction acts as a conditional computation, where the features of higher resolution are computed using the lower resolution features. This, in turn, allows the low resolution coarse features, which have a big receptive field over the input image, to inform higher resolution fine features of global information, such as occlusion in part of the image, when higher resolution features are constructed in the decoding path. Our proposed approach yielded more accurate results and also faster convergence.

5.3. Contributions

Deep neural networks with alternating convolutional, max-pooling and decimation layers are widely used in state of the art architectures for computer vision. Max-pooling purposefully discards precise spatial information in order to create features that are more robust, and typically organized as lower resolution spatial feature maps. On some tasks, such as whole-image classification, max-pooling derived features are well suited; however, for tasks requiring precise localization, such as pixel level prediction and segmentation, max-pooling destroys exactly the information required to perform well. Precise localization may be preserved by shallow convnets without pooling but at the expense of robustness. Can we have our max-pooled multi-layered cake and eat it too? Several papers have proposed summation and concatenation based methods for combining upsampled coarse, abstract features with finer features to produce robust pixel level predictions. Here we introduce another model — dubbed Recombinator Networks — where coarse features inform finer features early in their formation such that finer features can make use of several layers of computation in deciding how to use coarse features. The model is trained once, end-to-end and performs better than summation-based architectures, reducing the error from the previous state of the art on two facial keypoint datasets, AFW and AFLW, by 30% and beating the current state-of-the-art on 300W without using extra data. We improve performance even further by adding a denoising prediction model based on a novel convnet formulation.

5.4. Recent Developments

The proposed approach here has been highly used in many models that require pixel level accuracy, as in landmark localization (Newell et al., 2016), or image segmentation (Çiçek et al., 2016; Milletari et al., 2016). Also, it has been used for models that require full image reconstruction that changes the modality of the input. Examples of such models include image to image translation models, as in (Isola et al., 2017; Yi et al., 2017). Although the credit is generally attributed to U-Net (Ronneberger et al., 2015) due to its earlier date of publication, it is the same fundamental idea, and has since been successfully leveraged in many models. As of time of writing this thesis the idea has been cited more than 3500 times.

Regarding landmark localization, some recent successful models used recursive processing of the outputs in order to better capture the joint distribution over keypoints. Examples include stacked hourglass network (Newell et al., 2016) and also (Xiao et al., 2016). We had also tried the idea of multiple refinement steps by using the convnet refinement network multiple times, however, it yielded only a tiny improvement on our datasets, so we did not further explore that direction.

Chapter 6

Recombinator Networks: Learning Coarse-to-Fine Feature Aggregation

6.1. Introduction

Recent progress in computer vision has been driven by the use of large convolutional neural networks. Such networks benefit from alternating convolution and pooling layers (Krizhevsky et al., 2012; Sermanet and LeCun, 2011; Sermanet et al., 2013; Simonyan and Zisserman, 2014; Sun et al., 2014; Szegedy et al., 2015; Zhang et al., 2016) where the pooling layers serve to summarize small regions of the layer below. The operations of convolution, followed by max-pooling, then decimation cause features in subsequent layers of the network to be increasingly translation invariant, more robust, and to more coarsely summarize progressively larger regions of the input image. As a result, features in the fourth or fifth convolutional layer serve as more robust detectors of more global, but spatially imprecise high level patterns like text or human faces (Yosinski et al., 2015). In practice these properties are critical for many visual tasks, and they have been particularly successful at enabling whole image classification (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015). However, for other types of vision tasks these architectural elements are not as well suited. For example on tasks requiring pixel-precise localization or labeling, features arising from max-pooling and decimation operations can only provide approximate localization, as in the process of creating them, the network has already thrown out precise spatial information by design. If we wish to generate features that preserve accurate localization, we may do so using shallow networks without max-pooling, but shallow networks without pooling cannot learn robust, invariant features. What we would like is to have our cake and eat it too: to combine the best of

both worlds, merging finely-localized information from shallow, non-pooled networks with robust, coarsely-localized features computed by deep, pooled networks.

Several recently proposed approaches (Hariharan et al., 2015; Long et al., 2015; Tompson et al., 2015) address this by adding or concatenating the features obtained across multiple levels. We use this approach in our baseline model termed *SumNet* for our task of interest: facial keypoint localization. To the best of our knowledge this is the first time this general approach has been applied to the problem of facial keypoint localization and even our baseline is capable of yielding state of the art results. A possible weakness of these approaches however is that all detection paths, from coarsely to finely localized features, only become aggregated at the very end of the feature processing pipeline. As a thought experiment to illustrate this approach's weakness, imagine that we have a photo of a boat floating in the ocean and would like to train a convnet to predict with single pixel accuracy a keypoint corresponding to the tip of the boat's bow. Coarsely localized features[1] could highlight the rough region of the bow of the boat, and finely localized features could be tuned to find generic boat edges, but the fine features must remain generic, being forced to learn boat edge detectors for all possible ocean and boat color combinations. This would be difficult, because boat and ocean pixels could take similar colors and textures. Instead, we would like a way for the coarse features which contain information about the global scene structure (perhaps that the water is dark blue and the boat is bright blue) to provide information to the fine feature detectors earlier in their processing pipeline. Without such information, the fine feature detectors would be unable to tell which half of a light blue/dark blue edge was ocean and which was boat. In the *Recombinator Networks* proposed in this paper, the finely localized features are conditioned on higher level more coarsely localized information. It results in a model which is deeper but – interestingly – trains faster than the summation baseline and yields more precise localization predictions. In summary, this work makes the following contributions:

(1) We propose a novel architecture — the Recombinator Networks — for combining information over different spatial localization resolutions (Section 6.3).

(2) We show how a simple denoising model may be used to enhance model predictions (Section 6.4).

[1]From now on we use the shorthand fine/coarse features to mean finely/coarsely localized features.

(3) We provide an in-depth empirical evaluation of a wide variety of relevant architectural variants (Section 6.5.1).

(4) We show state of the art performance on two widely used and competitive evaluations for facial keypoint localization (Section 6.5.2).

6.2. Related work

Keypoint localization methods: Our task of interest is the well studied problem of facial keypoint localization (Asthana et al., 2013; Cao et al., 2014; Ren et al., 2014; Tzimiropoulos and Pantic, 2014; Xiong and De la Torre, 2013; Yu et al., 2013; Zhang et al., 2014a, 2016; Zhu et al., 2015; Zhu and Ramanan, 2012) illustrated in Figure 6.1. Precise facial keypoint localization is often an essential preprocessing step for face recognition (Gross, 2015) and detection (Zhu and Ramanan, 2012). Recent face verification models like DeepFace (Taigman et al., 2014) and DeepID2 (Sun et al., 2014) also include keypoint localization as the first step. There have been many other approaches to general keypoint localization, including active appearance models (Cootes et al., 2001; Zhao et al., 2012), constrained local models (Asthana et al., 2013; Cristinacce and Cootes, 2008, 2006; Saragih et al., 2009), active shape models (Cristinacce and Cootes, 2007), point distribution models (Cootes et al., 1995), structured model prediction (Baltrušaitis et al., 2014; Tompson et al., 2015), tree structured face models (Zhu and Ramanan, 2012), group sparse learning based methods (Yu et al., 2013), shape regularization models that combine multiple datasets (Smith and Zhang, 2014), feature voting based landmark localization (Smith et al., 2014; Yang and Patras, 2013) and convolutional neural network based models (Sun et al., 2013; Zhang et al., 2014c, 2016). Two closely related models to our approach are (Tompson et al., 2015), where a multi-resolution model is proposed with dual coarse/fine paths and tied filters, and (Sun et al., 2013), which uses a cascaded architecture to refine predictions over several stages. Both of these latter models make hard decisions using coarse information halfway through the model.

Approaches that combine features across multiple levels: Several recent models — including the fully convolutional networks (FCNs) in (Long et al., 2015), the Hypercolumn model (Hariharan et al., 2015), and the localization model of Tompson et al. (Tompson et al., 2015) — generate features or predictions at multiple resolutions, upsample the coarse features to the fine resolution,

and then add or concatenate the features or predictions together. This approach has generally worked well, improving on previous state of the art results in detection, segmentation, and human-body pose estimation (Hariharan et al., 2015; Long et al., 2015; Tompson et al., 2015). In this paper we create a baseline model similar to these approaches that we refer to as *SumNet* in which we use a network that aggregates information from features across different levels in the hierarchy of a conv-pool-decimate network using concatenation followed by a weighted sum over feature maps prior to final layer softmax predictions. Our goal in this paper is to improve upon this architecture. Differences between the Recombinator Networks and related architectures are summarized in Table 6.5. U-Net (Ronneberger et al., 2015) is another model that merges features across multiple levels and has a very similar architecture to Recombinator Networks. The two models have been developed independently and were designed for different problems[2]. Note that none of these models use a learned denoising post-processing as we do (see section 6.4).

6.3. Summation versus Recombinator Networks

In this section we describe our baseline SumNet model based on a common architectural design where information from different levels of granularity are merged just prior to predictions being made. We contrast this with the Recombinator Networks architecture.

6.3.1. Summation based Networks

The SumNet architecture, shown in Figure 6.1(top), adds to the usual bottom to top convolution and spatial pooling, or "trunk", a horizontal left-to-right "branch" at each resolution level. While spatial pooling progressively reduces the resolution as we move "up" the network along the trunk, the horizontal branches only contains full convolutions and element-wise non-linearities, with no spatial pooling, so that they can preserve the spatial resolution at that level while doing further processing. The output of the finest resolution branch only goes through convolutional layers. The finest resolution layers keep positional information and use it to guide the coarser layers within the patch that they cannot have any preference, while the coarser resolution layers help finer layers to get rid of false positives. The architecture then combines the rightmost low resolution output of

[2]For keypoint localization, we apply the softmax *spatially* i.e. across possible *spatial locations*, whereas for segmentation (Hariharan et al., 2015; Long et al., 2015; Ronneberger et al., 2015) it is applied across all possible *classes* for each pixel.

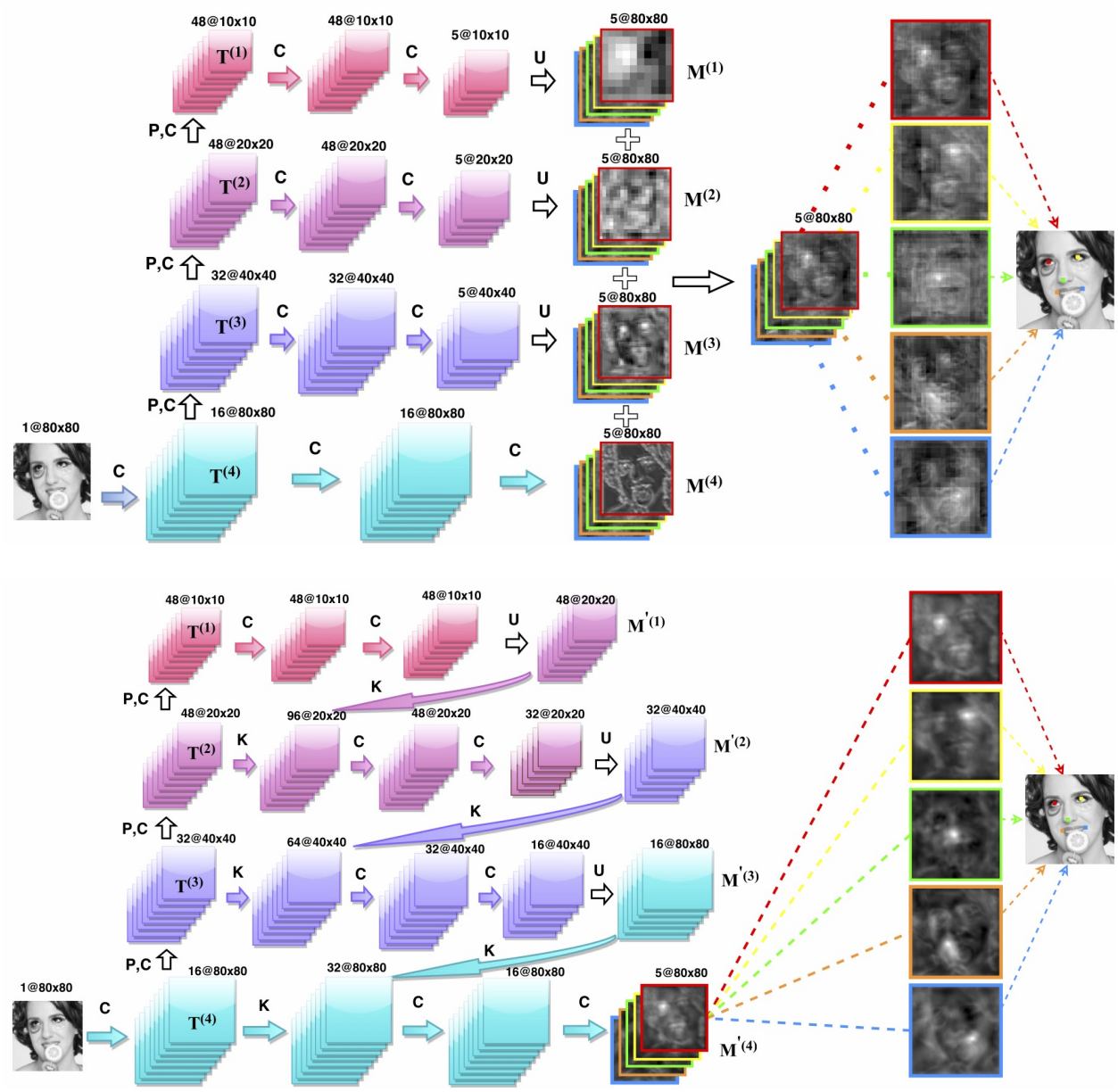


Fig. 6.1. (*top*) Architecture of summation based coarse-fine network (SumNet). C is a convolutional layer. P,C represents a pooling layer followed by a convolutional layer. All convolutions are 3×3 and all poolings are 2×2. All convolutional layers are followed by ReLU non-linearity except the last convolutional layer in each branch. U represents an upsampling layer. Each branch's output is 5 feature maps of size 80×80. FCN/Hypercolumn models use this architecture. (*bottom*) Architecture of the Recombinator Networks (RCN). All convolutions are 3×3 and all poolings are 2×2. All upsamplings are by a factor of 2. K represents concatenation of two sets of feature maps along the feature map dimension. All convolutional layers are followed by ReLU non-linearity except the one right before the softmax. In the Recombinator Networks model with skip connections (not shown), each branch takes upsampled features not only from one coarser branch, but from all coarser branches. Please note that the notations C and K used in this figure are independent from the similar notation used in this Chapter.

all horizontal branches, into a single high resolution prediction, by first up-sampling[3] them all to the model's input image resolution (80×80 for our experiments) and then taking a weighted sum to yield the pre-softmax values. Finally, a softmax function is applied to yield the final location probability map for each keypoint. Formally, given an input image $\mathbf{x}$, we define the *trunk* of the network as a sequence of blocks of traditional groups of convolution, pooling and decimation operations. Starting from the layer yielding the coarsest scale feature maps we call the outputs of Br such blocks $\mathbf{T}^{(1)}, \ldots, \mathbf{T}^{(Br)}$. At each level br of the trunk we have a horizontal *branch* that takes $\mathbf{T}^{(br)}$ as its input and consists of a sequence of convolutional layers with no subsampling. The output of such a branch is a stack of K feature maps, one for each of the K target keypoints, at the same resolution as its input $\mathbf{T}^{(br)}$, and we denote this output as $\mathrm{branch}(\mathbf{T}^{(br)})$. It is then upsampled $\mathrm{up}_{[\times F]}$ by some factor F which returns the feature map to the original resolution of the input image. Let these upsampled maps be $\mathbf{M}_1^{(1)}, \ldots, \mathbf{M}_K^{(Br)}$ where $\mathbf{M}_k^{(br)}$ is the score map given by the br^{th} branch to the k^{th} keypoint (left eye, right eye, ...). Each such map $\mathbf{M}_k^{(br)}$ is a matrix of the same resolution as the image fed as input (i.e. 80×80). The score ascribed by branch br for keypoint k being at coordinate r,c is given by $\mathbf{M}_{k,r,c}^{(br)}$. The final *probability map* for the location $\mathcal{Y}_k$ of keypoint k is given by a softmax over all possible locations. We can therefore write the model as

$$
\begin{aligned}
&\mathbf{M}^{(1)} = \mathrm{up}_{[\times 2^{Br-1}]}(\mathrm{branch}(\mathbf{T}^{(1)})) \\
&\mathbf{M}^{(2)} = \mathrm{up}_{[\times 2^{Br-2}]}(\mathrm{branch}(\mathbf{T}^{(2)})) \\
&\ldots \\
&\mathbf{M}^{(Br)} = \mathrm{branch}(\mathbf{T}^{(Br)}) \\
&P(\mathcal{Y}_k|\mathcal{X} = \mathbf{x}) = \mathrm{softmax}\Big(\sum_{br=1}^{Br} \boldsymbol{\alpha}_k^{(br)} \mathbf{M}_k^{(br)}\Big),
\end{aligned}
\tag{6.3.1}
$$

where $\boldsymbol{\alpha}_k^{(br)}$ is a 2D matrix that gives a weight to every pixel location r,c of keypoint k in branch br. The weighted sum of features over all branches taken here is equivalent to concatenating the features of all branches and multiplying them in a set of weights, which results in one feature map per keypoint. This architecture is trained globally using gradient backpropagation to minimize the sum of negated conditional log probabilities of all N training (input-image, keypoint-locations)

[3]Upsampling can be performed either by tiling values or by using bilinear interpolation. We found bilinear interpolation degraded performance in some cases, so we instead used the simpler tiling approach. By tiling we mean the value of each pixel is repeated in a local region of $u \times u$, where u is the upsampling ratio.

pairs, for all K keypoints $(\mathbf{x}^n,\mathbf{y}_k^n)$, with an additional regularization term for the weights ; i.e. we search for network parameters $\boldsymbol{\theta}$ that minimize [4]

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} -\log P(\mathcal{Y}_k = \mathbf{y}_k^n | \mathcal{X} = \mathbf{x}^n) + \lambda\|\boldsymbol{\theta}\|^2. \tag{6.3.2}$$

6.3.2. The Recombinator Networks

In the SumNet model, different branches can only communicate through the updates received from the output layer and the features are merged linearly through summation. In the Recombinator Networks (RCN) architecture, as shown in Figure 6.1(bottom), instead of taking a weighted sum of the upsampled feature maps in each branch and then passing them to a softmax, the output of each branch is upsampled, then concatenated with the next level branch with one degree of finer resolution. In contrast to the SumNet model, each branch does not end in K feature maps. The information stays in the form of a keypoint independent feature map. It is only at the end of the Br^{th} branch that feature maps are converted into a per-keypoint scoring representation that has the same resolution as the input image, on which a softmax is then applied. As a result of RCN's different architecture, branches pass more information to each other during training, such that convolutional layers in the finer branches get inputs from both coarse and fine layers, letting the network learn how to combine them *non-linearly* to maximize the log likelihood of the keypoints given the input images. The whole network is trained end-to-end by backprop. Following the previous conventions and by defining the concatenation operator on feature maps $\mathbf{A}$, $\mathbf{B}$ as $\mathrm{concat}(\mathbf{A},\mathbf{B})$, we can write the model as

$$\begin{aligned}
&\mathbf{M}'^{(1)} = \mathrm{up}_{[\times 2]}(\mathrm{branch}(\mathbf{T}^{(1)}))\\
&\mathbf{M}'^{(2)} = \mathrm{up}_{[\times 2]}(\mathrm{branch}(\mathrm{concat}(\mathbf{T}^{(2)},\mathbf{M}'^{(1)})))\\
&\dots\\
&\mathbf{M}'^{(Br)} = \mathrm{branch}(\mathrm{concat}(\mathbf{T}^{(Br)},\mathbf{M}'^{(Br-1)}))\\
&P(\mathcal{Y}_k|\mathcal{X} = \mathbf{x}) = \mathrm{softmax}(\mathbf{M}_k'^{(Br)}).
\end{aligned} \tag{6.3.3}$$

[4] We also tried L2 distance cost between true and estimated keypoints (as a regression problem) and got worse results. This may be due to the fact that a softmax probability map can be multimodal , while L2 distance implicitly corresponds to likelihood of a *unimodal* isotropic Gaussian.

We also explore RCN with skip connections, where the features of each branch are concatenated with upsampled features of not only one-level coarser branch, but all previous coarser branches and, therefore, the last branch computes $\mathbf{M'}^{(Br)} = \mathrm{branch}(\mathrm{concat}(\mathbf{T}^{(Br)}, \mathbf{M'}^{(Br-1)}, \mathbf{M'}^{(Br-2)}, \ldots, \mathbf{M'}^{(1)})$. In practice, the information flow between different branches makes RCN converge faster and also perform better compared to the SumNet model.

6.4. Denoising keypoint model

Convolutional networks are excellent edge detectors. If there are few samples with occlusion in the training sets, convnets have problem detecting occluded keypoints and instead select nearby edges (see some samples in Figures 6.5, 6.8). Moreover, the convnet predictions, especially on datasets with many keypoints, do not always correspond to a plausible keypoint distribution and some keypoints jump off the curve (e.g. on the face contour or eye-brows) irrespective of other keypoints' position (see some samples in Figure 6.10). This type of error can be addressed by using a structured output predictor on top of the convnet, that takes into account how likely the location of a keypoint is relative to other keypoints. Our approach is to train another convolutional network that captures useful aspects of the prior keypoint distribution (not conditioned on the image). We train it to predict the position of a random subsets of keypoints, given the position of the other keypoints. More specifically, we train the convolutional network as a denoising model, similar to the denoising auto-encoder (Vincent et al., 2008) by completely corrupting the location of a randomly chosen subset of the keypoints and learning to accurately predict their correct location given that of the other keypoints. This network receives as input, not the image, but only keypoint locations represented as one-hot 2D maps (one 2D map per keypoint, with a 1 at the position of the keypoint and zeros elsewhere). It is composed of convolutional layers with large receptive fields (to get to see nearby keypoints), ReLU nonlinearities and no subsampling (see Figure 6.4). The network outputs probability maps for the location of all keypoints, however, its training criterion uses only prediction errors of the corrupted ones. The cost being optimized is similar to Eq.(6.3.2) but includes only the corrupted keypoints.

Once, this denoising model is trained, the output of RCN (the predicted most likely location in one-hot binary location 2D map format) is fed to the denoising model. We then simply sum

the pre-softmax values of both RCN and denoising models and pass them through a softmax to generate the final output probability maps. The joint model is depicted in Figure 6.4. The joint model combines the RCN's predicted conditional distribution for keypoint k given the image $P(\mathcal{Y}_k|\mathcal{X} = \mathbf{x})$ with the denoising model's distribution of the location of that keypoint given other keypoints $P(\mathcal{Y}_k|\mathcal{Y}_{\neg k})$, to yield an estimation of keypoint k's location given both image and other keypoint locations $P(\mathcal{Y}_k|\mathcal{Y}_{\neg k}, \mathcal{X} = \mathbf{x})$. The choice of convolutional networks for the denoising model allows it to be easily combined with RCN in a unified deep convolutional architecture.

6.5. Experimental setup and results

We evaluate our model[5] on the following datasets with evaluation protocols defined by the previous literature:

AFLW and AFW datasets: Similar to TCDCN (Zhang et al., 2014c), we trained our models on the MTFL dataset,[6] which we split into 9,000 images for training and 1,000 for validation. We evaluate our models on the same subsets of AFLW (Köstinger et al., 2011) and AFW (Zhu and Ramanan, 2012) used by (Zhang et al., 2014c), consisting of 2995 and 377 images, respectively, each labeled with 5 facial keypoints.

300W dataset: 300W (Sagonas et al., 2013) standardizes multiple datasets into one common dataset with 68 keypoints. The training set is composed of 3148 images (337 AFW, 2000 Helen, and 811 LFPW). The test set is composed of 689 images (135 IBUG, 224 LFPW, and 330 Helen). The IBUG is referred to as the challenging subset, and the union of LFPW and Helen test sets is referred to as the common subset. We shuffle the training set and split it into 90% train-set (2834 images) and 10% valid-set (314 images).

One challenging issue in these datasets is that the test set examples are significantly different and more difficult compared to the training sets. In other words the train and test set images are not from the same distribution. In particular, the AFLW and AFW test sets contain many samples with occlusion and more extreme rotation and expression cases than the training set. The IBUG subset of 300W contains more extreme pose and expressions than other subsets.

[5]Our models and code are publicly available at https://github.com/SinaHonari/RCN

[6]MTFL consists of 10,000 training images: 4151 images from LFW (Huang et al., 2007) and 5849 images from the web.

Error Metric: The Euclidean distance between the true and estimated landmark positions normalized by the distance between the eyes (interocular distance) is used:

$$\text{error} = \frac{1}{KN} \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{\sqrt{(w_k^n - \tilde{w}_k^n)^2 + (h_k^n - \tilde{h}_k^n)^2}}{D^n}, \tag{6.5.1}$$

where K is the number of keypoints, N is the total number of images, D^n is the inter-ocular distance in image n. (w_k^n, h_k^n) and $(\tilde{w}_k^n, \tilde{h}_k^n)$ represent the true and estimated coordinates for keypoint k in image n, respectively.

6.5.1. Evaluation on SumNet and RCN

We evaluate RCN on the 5-keypoint test sets. To avoid overfitting and improve performance, we applied online data augmentation to the 9,000 MTFL training set using random scale, rotation, and translation jittering[7]. We preprocessed images by making them gray-scale and applying local contrast normalization [8]. In Figure 6.2, we show a visualization of the contribution of each branch of the SumNet to the final predictions: the coarsest layer provides robust but blurry keypoint locations, while the finest layer gives detailed face information but suffers from many false positives. However, the sum of branches in SumNet and the finest branch in RCN make precise predictions.

[7]We jittered data separately in each epoch, whose parameters were uniformly sampled in the following ranges (selected based on the validation set performance): Translation and Scaling: [-10%, +10%] of face bounding box size; Rotation: [-40, +40] degrees.

[8]RGB images performed worse in our experiments.

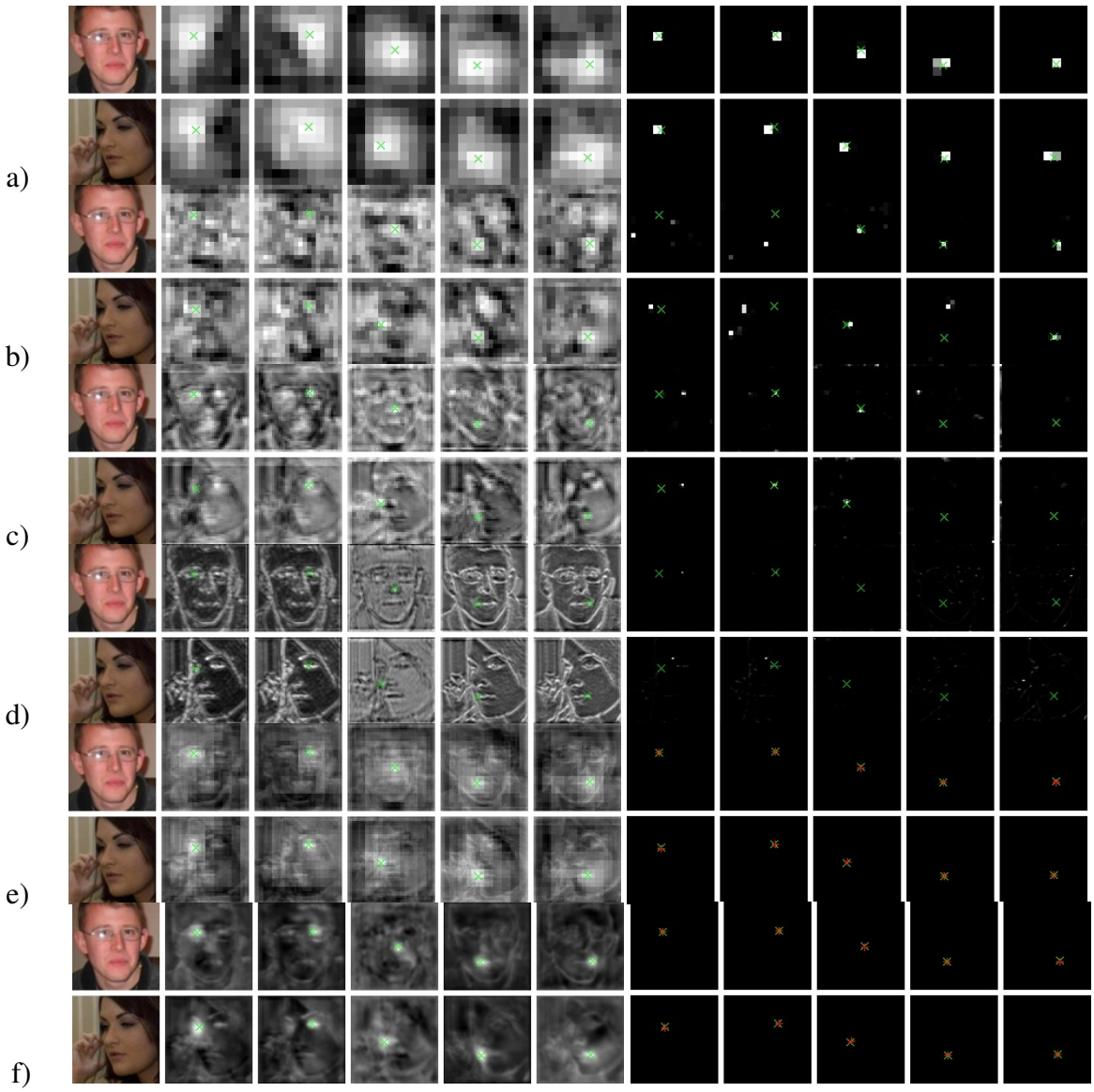


Fig. 6.2. Sub-figures a), b), c), d) show pre-sum (left) and softmax (right) of the coarsest to finest branches in a 4-branch SumNet model. The softmax used in these branches are only for illustration purposes and is not part of the trained model. Sub-figure e) (left) shows the sum of branches in the SumNet model and Sub-figure f) (left) depicts the pre-softmax values in RCN. The true keypoint locations are shown by green cross in all figures to show their relative correspondence with the branch activations. SumNet and RCN's predictions are shown by red plus on the post-softmax maps in Sub-figures e) (right) and f) (right), respectively. In each row the images correspond to the keypoints in this order from left to right: left-eye, right-eye, nose, left-mouth, right-mouth. Best viewed electronically with zoom.

Since the test sets contain more extreme occlusion and lighting conditions compared to the training set, we applied preprocessing to the training set to bring it closer to the test set distribution. In addition to jittering, we found it helpful to occlude images in the training set with randomly

placed black rectangles[9] at each training iteration. This trick forced the convnet models to use more global facial components to localize the keypoints and not rely as much on the features around the keypoints, which in turn, made it more robust against occlusion and lighting contrast in the test set. Figure 6.5 shows the effects of this occlusion when used to train the SumNet and RCN models on randomly drawn samples. The samples show for most of the test set examples the models predict well. Figure 6.6 shows the performance of the four models in Figure 6.5 as the difficulty of the examples increase. This plot shows that on difficult examples, RCN performs better that the SumNet model and on those examples the occlusion preprocessing improves further the results in the RCN model compared to the SumNet model. Figure 6.7 shows some hand-picked examples from the test sets, to show extreme expression, occlusion and contrast that are not captured in the random samples of Figure 6.5. Figure 6.8 similarly uses some manually selected examples to show the benefits of using occlusion.

To evaluate how much each branch contributes to the overall performance of the model, we trained models excluding some branches and report the results in Table 6.1. The finest layer on its own does a poor job due to many false positives, while the coarsest layer on its own does a reasonable job, but still lacks high accuracy. One notable result is that using only the coarsest and finest branches together produces reasonable performance. However, the best performance is achieved by using all branches, merging four resolutions of coarse, medium, and fine information.

We also experimented with adding extra branches, getting to a coarser resolution of 5×5 in the 5 branch model, 2×2 in the 6 branch model and 1×1 in the 7 branch model. In each branch, the same number of convolutional layers with the same kernel size is applied,[10] and all new layers have 48 channels. The best performing model, as shown in Table 6.2, is RCN with 6 branches. Comparing RCN and SumNet training, RCN converges faster. Using early stopping and without occlusion pre-processing, RCN requires on average 200 epochs to converge (about 4 hours on a NVidia Tesla K20 GPU), while SumNet needs on average more than 800 epochs (almost 14 hours). RCN's error on both test sets drops below 7% on average after only 15 epochs (about 20 minutes),

[9]Each image was occluded with one black (zeros) rectangle, whose size was drawn uniformly in the range [20, 50] pixels. Its location was drawn uniformly over the entire image.

[10]A single exception is that when the 5×5 resolution map is reduced to 2×2, we apply 3×3 pooling with stride 2 instead of the usual 2×2 pooling, to keep the resulting map left-right symmetric.

	SumNet		RCN	
Mask	AFLW	AFW	AFLW	AFW
1, 0, 0, 0	10.54	10.63	10.61	10.89
0, 1, 0, 0	11.28	11.43	11.56	11.87
1, 1, 0, 0	9.47	9.65	9.31	9.44
0, 0, 1, 0	16.14	16.35	15.78	15.91
0, 0, 0, 1	45.39	47.97	46.87	48.61
0, 0, 1, 1	13.90	14.14	12.67	13.53
0, 1, 1, 1	7.91	8.22	7.62	7.95
1, 0, 0, 1	6.91	7.51	6.79	7.27
1, 1, 1, 1	**6.44**	**6.78**	**6.37**	**6.43**

Tab. 6.1. The performance of SumNet and RCN trained with masks applied to different branches. A mask value of 1 indicates the branch is included in the model and 0 indicates it is omitted (as a percent; lower is better). In SumNet model mask 0 indicates no contribution from that branch to the summation of all branches, while in RCN, if a branch is omitted, the previous coarse branch is upsampled to the following fine branch. The mask numbers are ordered from the coarsest branch to the finest branch. See Figure 6.3 for visualizing how branches are masked.

while SumNet needs on average 110 epochs (almost 2 hours) to get to this error. Using occlusion preprocessing increases these times slightly but results in lower test error. At test time, a feedforward pass on a K20 GPU takes 2.2ms for SumNet and 2.5ms for RCN per image in Theano (Al-Rfou et al., 2016). Table 6.2 shows occlusion pre-processing significantly helps boost the accuracy of RCN, while slightly helping SumNet. We believe this is due to global information flow from coarser to finer branches in RCN. In the RCN model with skip connections, each branch takes upsampled features not only from one coarser branch, but from all coarser branches. This model variant is shown in the last row of Table 6.2. However, it did not improve the performance compared to the same model without skip connections (row in bold). We believe the concatenation in between successive branches, which is done in the RCN model, is already passing global information from coarser branches to finer branches of the network and leveraging extra flow of information was not necessary in our case.

6.5.2. Comparison with other models

AFLW and AFW datasets: We first re-implemented the TCDCN model (Zhang et al., 2014c), which is the current state of the art model on 5 keypoint AFLW (Köstinger et al., 2011) and AFW

[10]SumNet and RCN models are trained using occlusion preprocessing.

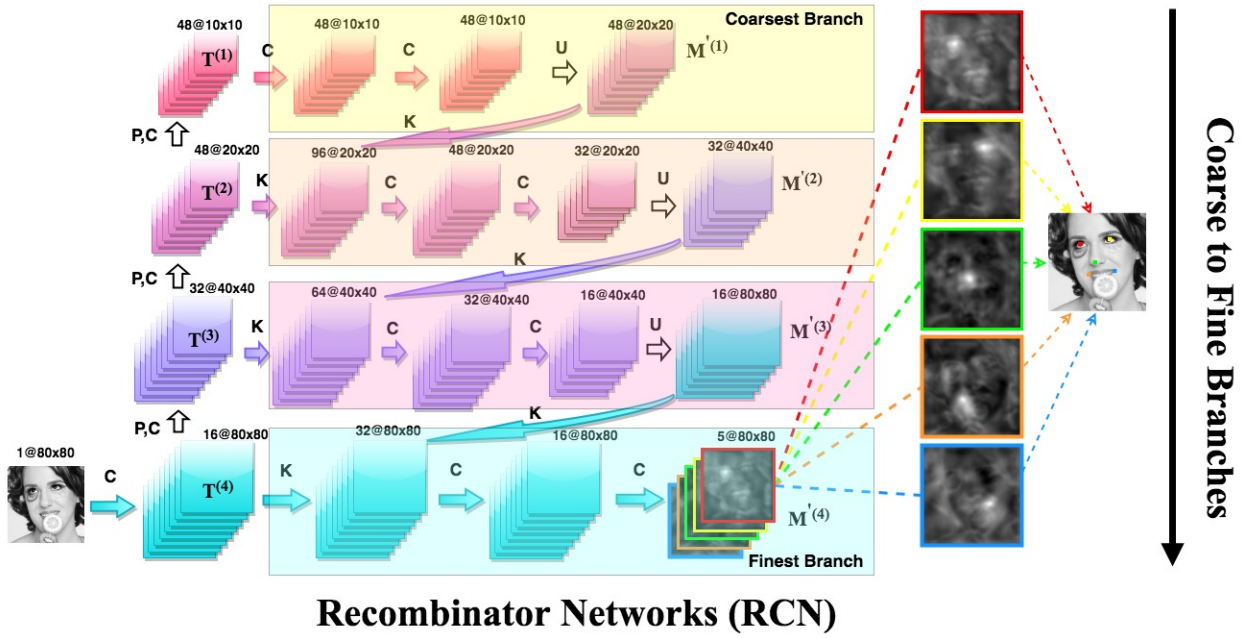


Fig. 6.3. Each series of convolution and upsampling layers in a row (which are covered with a colored rectangle) form a branch. The branch on the top (in yellow) is the coarsest and the branch on the bottom (in blue) is the finest branch. In the case of masking (used in Table 6.1), the 4 numbers indicate the presence or absence of the 4 branches. The numbers are ordered from the coarsest to the finest branch. For example, '1,1,0,0' uses the top two branches and then upsamples the output of the second (finer) branch to the input image resolution to be passed to the output softmax layer. In the case of '0,0,1,1' only the bottom two branches are used. In this case, the two convolutional layers to the left of the two coarsest branches are not needed and hence not used in the model. In the case of '1,0,0,1' only the coarsest and the finest branches are used and the output of the coarsest branch is upsampled to the finest branch resolution to be passed to that branch.

Model	AFLW	AFW
SumNet (4 branch)	6.44	6.78
SumNet (5 branch)	6.42	6.53
SumNet (6 branch)	6.34	6.48
SumNet (5 branch - occlusion)	6.29	6.34
SumNet (6 branch - occlusion)	6.27	6.33
RCN (4 branch)	6.37	6.43
RCN (5 branch)	6.11	6.05
RCN (6 branch)	6.00	5.98
RCN (7 branch)	6.17	6.12
RCN (5 branch - occlusion)	5.65	5.44
RCN (6 branch - occlusion)	**5.60**	**5.36**
RCN (7 branch - occlusion)	5.76	5.55
RCN (6 branch - occlusion - skip)	5.63	5.56

Tab. 6.2. SumNet and RCN performance with different number of branches, occlusion preprocessing and skip connections.

Model	AFLW	AFW
TSPM (Zhu and Ramanan, 2012)	15.9	14.3
CDM (Yu et al., 2013)	13.1	11.1
ESR (Cao et al., 2014)	12.4	10.4
RCPR (Burgos-Artizzu et al., 2013)	11.6	9.3
SDM (Xiong and De la Torre, 2013)	8.5	8.8
TCDCN (Zhang et al., 2014c)	8.0	8.2
TCDCN baseline (our implementation)	7.60	7.87
SumNet (FCN/HC) baseline (this)	6.27	6.33
RCN (this)	**5.60**	**5.36**

Tab. 6.3. Facial landmark mean error normalized by interocular distance on AFW and AFLW sets (as a percent; lower is better).[10]

(Zhu and Ramanan, 2012) sets, and applied the same pre-processing as our other experiments. Through hyper-parameter search, we even improved upon the AFLW and AFW results reported in (Zhang et al., 2014c). Table 6.3 compares RCN with other models. Especially, it improves the SumNet baseline, which is equivalent to FCN and Hypercolumn models, and it also converges faster. The SumNet baseline is also provided by this paper and to the best of our knowledge this is the first application of any such coarse-to-fine convolutional architecture to the facial keypoint problem. Figure 6.9 compares TCDCN with SumNet and RCN models, on some difficult samples reported in (Zhang et al., 2014c).

300W dataset (Sagonas et al., 2013): The RCN model that achieved the best result on the validation set, contains 5 branches with 64 channels for all layers (higher capacity is needed to extract features for more keypoints) and 2 extra convolutional layers with 1×1 kernel size in the finest branch right before applying the softmax. Table 6.4 compares different models on all keypoints (68) and a subset of keypoints (49) reported in (Tzimiropoulos, 2015). The denoising model is trained by randomly choosing 35 keypoints in each image and jittering them (changing their location uniformly to any place in the 2D map). It improves the RCN's prediction by considering how locations of different keypoints are inter-dependent. Figure 6.10 compares the output of RCN, the denoising model and the joint model, showing how the keypoint distribution modeling can reduce the error. We only trained RCN on the 2834 images in the train-set. No extra data is taken

Model	#keypoints	Common	IBUG	Fullset
PO-CR (Tzimiropoulos, 2015)		4.00	6.82	4.56
RCN (this)	49	2.64	5.10	3.88
RCN + denoising keypoint model (this)		**2.59**	**4.81**	**3.76**
CDM (Yu et al., 2013)		10.10	19.54	11.94
DRMF (Asthana et al., 2013)		6.65	19.79	9.22
RCPR (Burgos-Artizzu et al., 2013)		6.18	17.26	8.35
GN-DPM (Tzimiropoulos and Pantic, 2014)		5.78	-	-
CFAN (Zhang et al., 2014a)		5.50	16.78	7.69
ESR (Cao et al., 2014)		5.28	17.00	7.58
SDM (Xiong and De la Torre, 2013)	68	5.57	15.40	7.50
ERT (Cao et al., 2014)		-	-	6.40
LBF (Ren et al., 2014)		4.95	11.98	6.32
CFSS(Zhu et al., 2015)		4.73	9.98	5.76
TCDCN † (Zhang et al., 2016)		4.80	8.60	5.54
RCN (this)		4.70	9.00	5.54
RCN + denoising keypoint model (this)		**4.67**	**8.44**	**5.41**

Tab. 6.4. Facial landmark mean error normalized by interocular distance on 300W test sets (as a percent; lower is better). [10]

to pre-train or fine-tune the model [11]. The current state-of-the-art model without any extra data† is CFSS(Zhu et al., 2015). We reduce the error by 15% on the IBUG subset compared to CFSS.

Features \ Models	Efficient Localization (Tompson et al., 2015)	Deep Cascade (Sun et al., 2013)	Hyper-columns (Hariharan et al., 2015)	FCN (Long et al., 2015)	RCN (this)
Coarse features: hard crop or soft combination?	Hard	Hard	Soft	Soft	Soft
Learned coarse features fed into finer branches?	No	No	No	No	Yes

Tab. 6.5. Comparison of multi-resolution architectures. The Efficient Localization and Deep Cascade models use coarse features to crop images (or fine layer features), which are then fed into fine models. This process saves computation when dealing with high-resolution images but at the expense of making a greedy decision halfway through the model. Soft models merge local and global features of the entire image and do not require a greedy decision. The Hypercolumn and FCN models propagate all coarse information to the final layer but merge information via addition instead of conditioning fine features on coarse features. The Recombinator Networks (RCN), in contrast, inject coarse features directly into finer branches, allowing the fine computation to be tuned by (conditioned on) the coarse information. The model is trained end-to-end and results in *learned* coarse features which are tuned directly to support the eventual fine predictions.

[11] We only jittered the train-set images by random scaling, translation and rotation similar to the 5 keypoint dataset.
† TCDCN (Zhang et al., 2016) uses 20,000 extra dataset for pre-training.

6.6. Conclusion

In this chapter we have introduced the Recombinator Networks architecture for combining coarse maps of pooled features with fine non-pooled features in convolutional neural networks. The model improves upon previous summation-based approaches by feeding coarser branches into finer branches, allowing the finer resolutions to learn upon the features extracted by coarser branches. We find that this new architecture leads to both reduced training time and increased facial keypoint prediction accuracy. We have also proposed a denoising model for keypoints which involves explicit modeling of valid spatial configurations of keypoints. This allows our complete approach to deal with more complex cases such as those with occlusions.

Acknowledgements

We would like to thank the Theano developers, particularly F. Bastien and P. Lamblin, for their help throughout this project. We appreciate Y. Bengio feedbacks and also L. Yao, F. Ahmed and M. Pezeshki's helps in this project. We also thank Compute Canada, and Calcul Quebec for providing computational resources. Finally, we would like to thank Fonds de Recherche du Québec – Nature et Technologies (FRQNT) for a doctoral research scholarship (B2) grant during 2014 and 2015 (Sina Honari) and the NASA Space Technology Research Fellowship (Jason Yosinski).

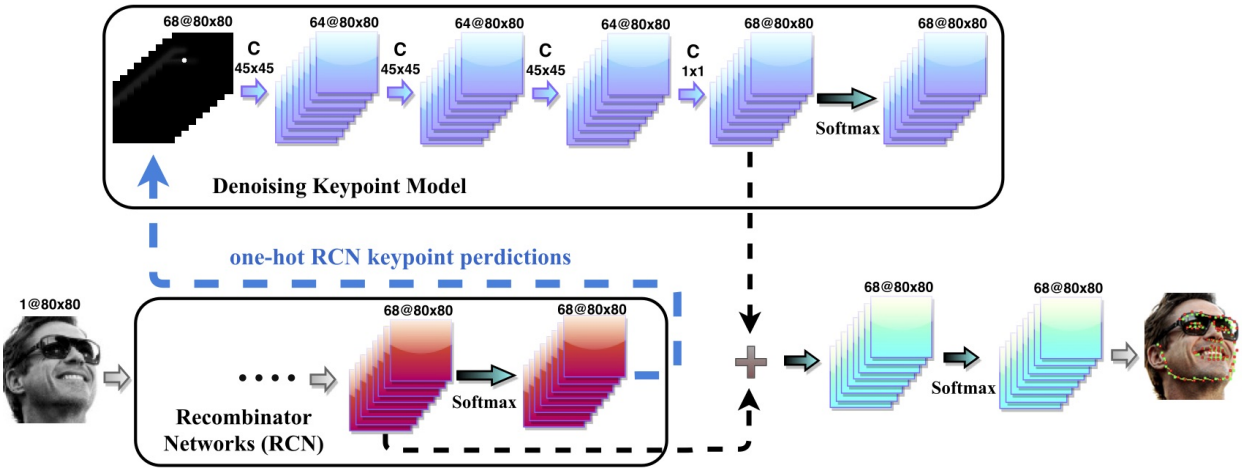


Fig. 6.4. Denoising / joint keypoint model. The Recombinator Networks (RCN) and the keypoint location denoising models are trained separately. At test time, the keypoint hard prediction of RCN is first injected into the denoising model as one-hot maps. Then the pre-softmax values computed by the RCN and the denoising models are summed and pass through a final softmax to predict keypoint locations.

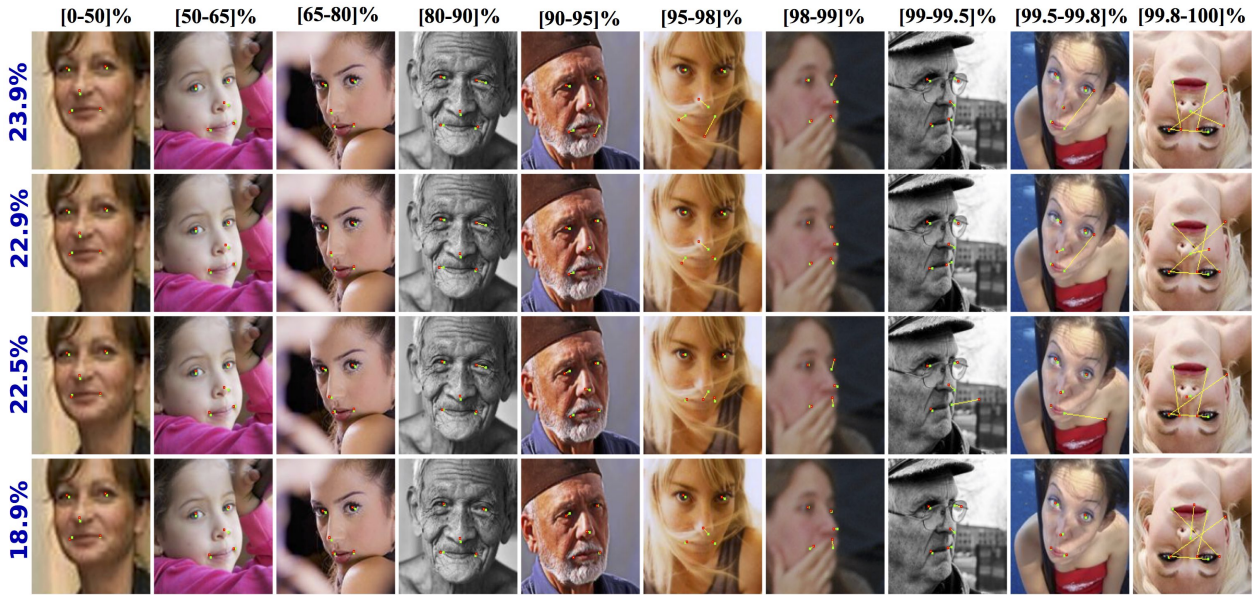


Fig. 6.5. Keypoint predictions on random test set images from easy (left) to hard (right). Each column shows predictions of following models from top to bottom: SumNet, SumNet with occlusion, RCN, RCN with occlusion (all models have 5 branches). We note for each test set image (including both AFLW and AFW) the average error over the four models and use this as a notion of that image's difficulty. We then sort all images by difficulty and draw a random image from percentile bins, using the bin boundaries noted above the images. To showcase the models' differing performance, we show only a few easier images on the left side and focus more on the hardest couple percent of images toward the right side. The value on the left is the average error of these samples per model (much higher than the results reported in Table 6.3 because of the skew toward difficult images). The yellow line connects the true keypoint location (green) to the model's prediction (red). Dots are small to avoid covering large regions of the image. Best viewed with zoom in color. Figure 6.6 shows the performance of these four models as the difficulty of the examples increase.

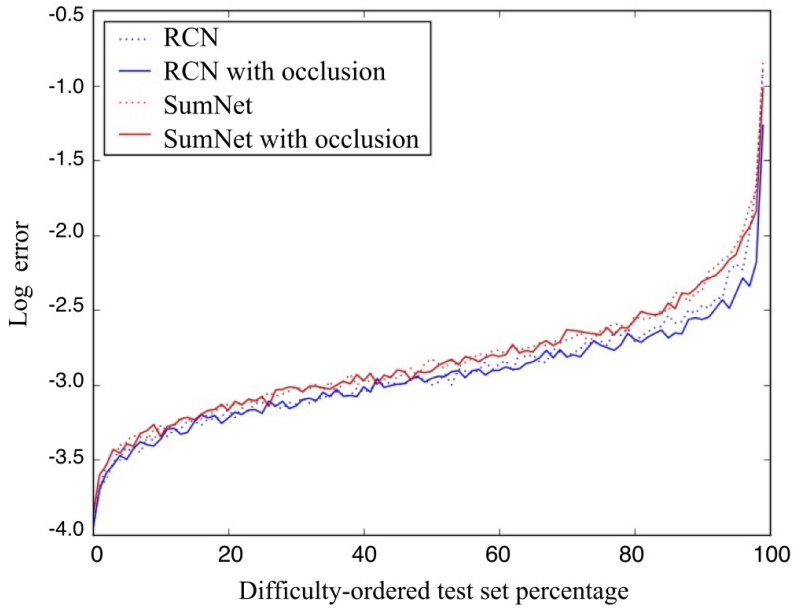


Fig. 6.6. The performance of SumNet and RCN models with and without occlusion pre-processing on the merged AFW and AFLW test sets as the difficulty of the examples increase (lower is better). To get this plot, we note for each test set image (including both AFLW and AFW) the average error over the four models and use this as a notion of that image's difficulty. We then sort all images by difficulty and get each model's log error (using Eq. 6.5.1) on each test example. Finally, we plot each model's performance on the sorted test set examples from the easiest (0% difficulty) to the most difficult (100% difficulty) percentage of the test set examples. The plot shows RCN performs better than SumNet, especially on the harder examples. The occlusion pre-processing helps RCN on the most difficult examples (difficulty $> 65\%$), while it slightly helps SumNet.

Fig. 6.7. Samples with different expressions (green border), contrast and illuminations (red border) and occlusions (blue border) from AFLW and AFW sets. In each box, top row depicts samples from SumNet and bottom row shows samples from RCN, both with occlusion pre-processing.

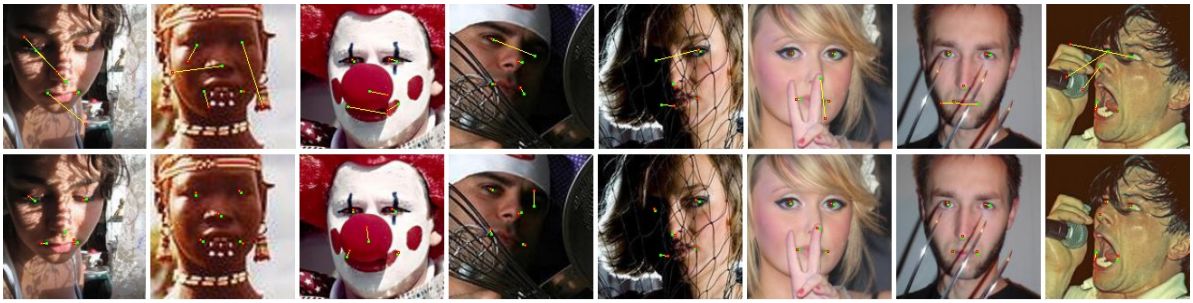


Fig. 6.8. Samples from AFLW and AFW test sets showing keypoint detection accuracy without (top row) and with (bottom row) occlusion pre-processing using RCN.

Fig. 6.9. Samples from TCDCN (Zhang et al., 2014c) (yellow border with green predicted points) versus SumNet (orange border) and RCN (blue border). In the latter two models, red and green dots show predicted and true keypoints. TCDCN samples are taken directly from (Zhang et al., 2014c).

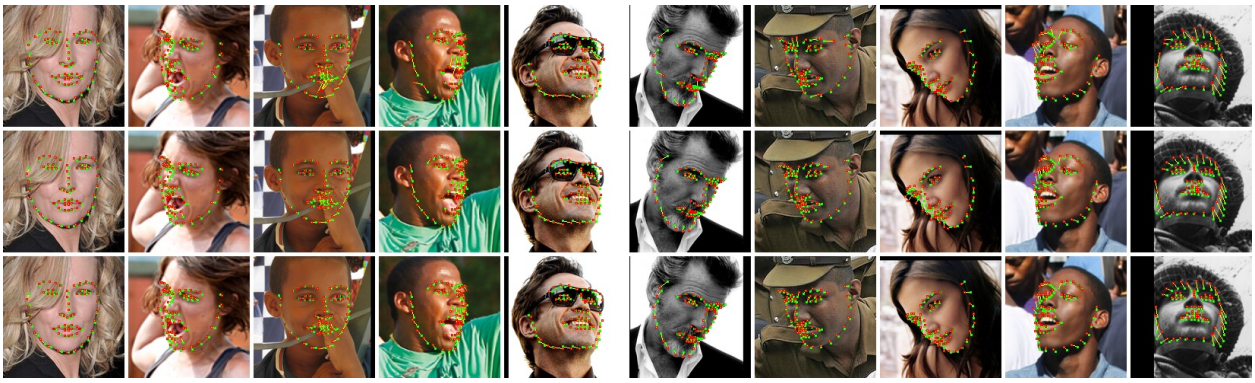


Fig. 6.10. Samples from 300W test sets. Each column shows samples in this order (top to bottom): RCN, keypoint denoising model and the joint model. The first two columns show extreme expression and occlusion samples where RCN's prediction is highly accurate. The next 5 columns show samples where the denoising model improves the RCN's predictions. In the 8th column the structured model find a reasonable keypoint distribution but deteriorates the RCN's predictions. Finally, the last two columns show cases where the denoising model generates plausible keypoint distributions but far from the true keypoints.

Chapter 7

Prologue to Third Article

7.1. Article Details

Improving Landmark Localization with Semi-Supervised Learning. Sina Honari, Pavlo Molchanov, Stephen Tyree, Pascal Vincent, Christopher Pal, and Jan Kautz. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2018)*.

This project started during my internship at Nvidia. Our goal in this work was to leverage weaker labelled data, such as class labels to improve landmark locations and also use unsupervised techniques over images to enhance landmark localization. Pavlo and I discussed many possible ways to do it and tried many different ideas. The idea of the two proposed approaches in this work, which are leveraging class labels and also the equivariant landmark transformation emerged throughout this process. We also tried many other ideas that did not work. Most of the implementation and experimental runs were done by myself. Pavlo also helped on some part of the implementations and running the models. The writing of the manuscript was mostly done by myself. Please note that the variable notations in this chapter have been changed compared to the published article version to unify the notations of the repeating concepts in the thesis.

7.2. Context

At the time of writing this article landmark localization approaches leveraged fully labelled datasets on faces and hands for accurate localization. However, this is quite demanding in terms of the manual effort required for labelling, since landmark localization requires pinpointing many landmarks on a given image, such as 68 landmarks in 300W dataset (Sagonas et al., 2013) or 21 landmarks in AFLW dataset (Köstinger et al., 2011). Most of the labelled datasets are limited

in terms of the variations they present. If a model is likely to be used at test time on data with new variations, it might not work well. Addressing this may require labelling a new dataset under these additional variations and then training a model on it. The goal of this paper is to build robust landmark localization models when few labelled landmarks are available. In particular, we aim at leveraging weaker labelled data, such as class labels, that are more abundant or more easily obtainable as only a single categorical label is needed per image, or using unsupervised approaches to enhance landmark localization networks in the few labelled landmarks data-settings.

7.3. Contributions

We present two techniques to improve landmark localization in images from partially annotated datasets. Our primary goal is to leverage the common situation where precise landmark locations are only provided for a small data subset, but where class labels for classification or regression tasks related to the landmarks are more abundantly available. First, we propose the framework of sequential multitasking and explore it here through an architecture for landmark localization where training with class labels acts as an auxiliary signal to guide the landmark localization on unlabeled data. A key aspect of our approach is that errors can be backpropagated through a complete landmark localization model. Second, we propose and explore an unsupervised learning technique for landmark localization based on having a model predict equivariant landmarks with respect to transformations applied to the image. We show that these techniques, improve landmark prediction considerably and can learn effective detectors even when only a small fraction of the dataset has landmark labels. We present results on two toy datasets and four real datasets, with hands and faces, and report new state-of-the-art on two datasets in the wild, e.g. with only 5% of labeled images we outperform previous state-of-the-art trained on the AFLW dataset.

7.4. Recent Developments

Recently Thewlis et al. (2017) also explored unsupervised learning of landmarks by matching corresponding landmarks under different transformations. Another unsupervised learning approach for landmarks has been proposed in (Jakab et al., 2018), where consecutive frames of a video clip with someone doing an action is taken and the model extracts only landmarks from frame a and

adds them to visual features of frame b to reconstruct back frame a. The idea is that if landmark information is taken from a frame and passed to the visual features of another frame, assuming the two frames are visually similar such as someone doing an action, the model should use be able to reconstruct the person with locations specified by the landmarks. This idea is also applied to still images that are manually warped. DeTone et al. (2018) applied our proposed equivariant landmark transformations to match landmarks across different homographies and show they can get improved performance compared to features built by Scale Invariant Feature Transform (SIFT) (Lowe, 1999) and Learned Invariant Feature Transform (LIFT) (Yi et al., 2016). Our sequential multi-tasking pipeline has been used in (Park et al., 2018) for gaze landmark localization. Laine and Aila (2017) also explored invariant feature extraction over time. Due to the recent publication of our work more time is needed to evaluate its impact and observe how this problem is addressed in the future.

Chapter 8

Improving Landmark Localization with Semi-Supervised Learning

8.1. Introduction

Landmark localization – finding the precise location of specific parts in an image – is a central step in many complex vision problems. Examples include hand tracking (Datcu and Lukosch, 2013; Hu et al., 2010), gesture recognition (Dardas et al., 2010), facial expression recognition (Kahou et al., 2013), face identity verification (Sun et al., 2014; Taigman et al., 2014), and eye gaze tracking (Mora and Odobez, 2012; Zhang et al., 2015). Reliable landmark estimation is often part of the pipeline for sophisticated, robust vision tools. Neural networks have yielded state-of-the art results on numerous landmark estimation problems (Honari et al., 2016; Tompson et al., 2015; Wang et al., 2016; Xiao et al., 2016; Yu et al., 2016). However, neural networks generally need to be trained on a large set of labeled data to be robust to the variations in natural images. Landmark labeling is a tedious manual work where precision is important; as a result, few landmark datasets are large enough to train reliable deep neural networks. On the other hand it is much easier to label an image with a single class label rather than the entire set of precise landmarks, and datasets with labels related to—but distinct from—landmark detection are far more abundant.

The key elements of our approach are illustrated in Figure 8.1. The top diagram illustrates a traditional convolutional neural network (CNN) based landmark localization network. The first key element of our work – illustrated in the second diagram of Figure 8.1, is that we use the indirect supervision of class labels to guide classifiers trained to localize landmarks. The class label can be considered a weak label that sends indirect signals about landmarks. For example, a photo of a hand gesture with the label "waving" likely indicates that the hand is posed with an open palm

and spread fingers, signaling a set of reasonable locations for landmarks on the hand. We leverage class labels that are more abundant or more easily obtainable than landmark labels, putting our proposed method in the category of multi-task learning. A common approach (Devries et al., 2014; Zhang and Zhang, 2014; Zhang et al., 2014c, 2016) to multi-task learning uses a traditional CNN, in which a final common fully-connected (FC) layer feeds into separate branches, each dedicated to the output for a different task. This approach learns shared low-level features across the set of tasks and acts as a regularizer, particularly when the individual tasks have few labeled samples.

There is a fundamental caveat to applying such an approach directly to simultaneous classification and landmark localization tasks, because the two have opposing requirements: classification output needs to be insensitive (invariant) to small deformations such as translations, whereas landmark localization needs to be equivariant to them, i.e., follow them precisely with high sensitivity. To build in invariance, traditional convolutional neural networks for classification problems rely on pooling layers to integrate signals across the input image. However, tasks such as landmark localization or image segmentation require both the global integration of information as well as an ability to retain local, pixel-level details for precise localization. The goal of producing precise landmark localization has thus led to the development of new layers and network architectures such as dilated convolutions (Yu and Koltun, 2016), stacked what-where auto-encoders (Zhao et al., 2016), recombinator-networks (Honari et al., 2016), fully-convolutional networks (Long et al., 2015), and hyper-columns (Hariharan et al., 2015), each preserving pixel-level information. These models have however not been developed with multi-tasking in mind.

Current multi-task architectures (Devries et al., 2014; Ranjan et al., 2017; Zhang and Zhang, 2014; Zhang et al., 2014c, 2016) predict landmark locations and auxiliary tasks as separate branches, i.e., *in parallel*. In this scenario the auxiliary task is used for partial supervision of landmark localization model. We propose a novel class of neural architectures which force classification predictions to flow through the intermediate step of landmark localization to provide complete supervision during backpropagation.

One of the contributions of our model is to leverage auxiliary classification tasks and data, enhancing landmark localization by backpropagating classification errors through the landmark localization layers of the model. Specifically, we propose a sequential architecture

in which the first part of the network predicts landmarks via pixel-level heatmaps, maintaining high-resolution feature maps by omitting pooling layers and strided convolutions. The second part of the network computes class labels using predicted landmark locations. To make the whole network differentiable, we use soft-argmax for extracting landmark locations from pixel-level predictions. Under this model, learning the landmark localizer is more directly influenced by the task of predicting class labels, allowing the classification task to enhance landmark localization learning.

Semi-supervised learning techniques (Rasmus et al., 2015; Salimans et al., 2016; Wan et al., 2017; Weston et al., 2012) have been used in deep learning to improve classification accuracy with a limited amount of labeled training data. A recently proposed method (Laine and Aila, 2017) enforces invariance in class predictions over time and across a variety of data augmentations applied to unlabeled training data. **Our second contribution is to propose and explore an unsupervised learning technique for landmark localization where the model is asked to produce landmark localizations equivariant with respect to a set of transformations applied to the image.** In other words, we transform an image during training and ask the model to produce landmarks that are similarly transformed. Importantly, this technique does not require the true landmark locations, and thus can be applied during semi-supervised training to leverage images with unlabeled landmarks. This element of our work is illustrated in the third diagram of Figure 8.1. Independently from our work, (Thewlis et al., 2017) proposed an unsupervised technique for landmark localization, however, the question if it can be used to improve supervised training remains open.

To summarize, in this chapter we make the following contributions: 1) We propose a novel multi-tasking neural architecture, which a) predicts landmarks as an intermediate step before classification in order to use the class labels to improve landmark localization, b) uses soft-argmax for a fully-differentiable model in which end-to-end training can be performed, even from examples that do not provide labeled landmarks. 2) We propose an unsupervised learning technique to learn features that are equivariant with respect to transformations applied to the input image. Combining contributions 1) and 2), we propose a robust landmark estimation technique which learns effective landmark predictors while requiring fewer labeled landmarks compared to current approaches.

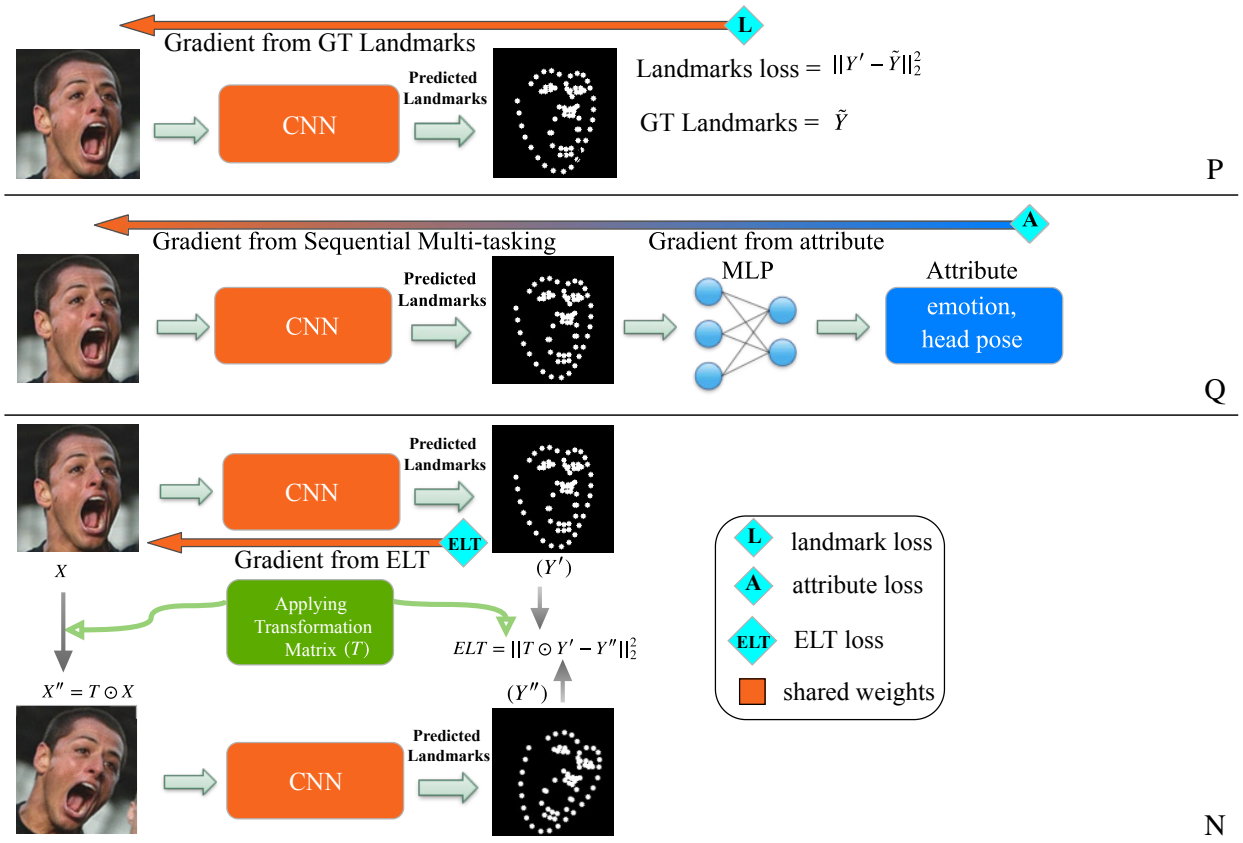


Fig. 8.1. In our approach three sources of gradients are used for learning a landmark localization network, from top to bottom: 1) The gradient from P labeled image-landmark pairs. 2) The gradient from Q attribute examples, obtained through sequential multitasking. The first part of the network (a CNN) predicts landmarks with a soft-argmax output layer to make the entire network fully differentiable. The predicted landmarks (as w, h pairs) are then fed into a multi-layer perceptron (MLP) for attribute regression/classification. 3) The gradient received from an unsupervised component of the composite loss which we refer to as an equivariant landmark transformation (ELT) (applied to N images). This loss encourages the model to output landmarks that are equivariant to transformations applied to the image. Importantly, MLP regression and the ELT are applied to the model's predictions and not the ground truth (GT) landmarks, so they can be applied on images that are not labelled with landmarks. Our proposed approach allows efficient training even when $P \ll Q \leq N$.

3) We report state-of-the-art on 300W (Sagonas et al., 2013) and AFLW (Köstinger et al., 2011) without leveraging any external data.

8.2. Sequential Multi-Tasking

We refer to the new architecture that we propose for leveraging the attributes to guide the learning of landmark locations as *sequential multi-tasking*. This architecture first predicts the landmark locations and then uses the predicted landmarks as the input to the second part of the network, which performs classification (see Fig. 8.1-middle). In doing so, we create a bottleneck in the network, forcing it to solve the classification task only through the landmarks. If the goal were to enhance classification, this architecture would have been harmful since such bottlenecks (He et al., 2016) would hurt the flow of information for classification. However, since our goal is landmark localization, this architecture enforces receiving signal from class labels through back-propagation to enhance landmark locations. This architecture benefits from auxiliary tasks that can be efficiently solved relying only on extracted landmark locations without observing the input image.

In order to make the whole pipeline trainable end-to-end, even on examples that do not provide any landmarks, we apply soft-argmax (Chapelle and Wu, 2010) on the output of the last convolutional layer in the landmark prediction model. Specifically, let $\mathbf{M}(\mathbf{x})$ be the stack of K two-dimensional output maps produced by the last convolutional layer for a given network input image $\mathbf{x}$. The map associated to the k^{th} landmark will be denoted $\mathbf{M}_k(\mathbf{x})$. To obtain a single 2d location $\mathbf{y}_k = (w,h)$ for the landmark from $\mathbf{M}_k(\mathbf{x})$, we use the following soft-argmax operation:

$$\begin{aligned} \mathbf{y}_k(\mathbf{x}) &= \text{soft-argmax}(\beta \mathbf{M}_k(\mathbf{x})) \\ &= \sum_{r,c} \text{softmax}(\beta \mathbf{M}_k(\mathbf{x}))_{r,c}(r,c) \end{aligned} \tag{8.2.1}$$

where softmax denotes a spatial softmax of the map, i.e.

$$\text{softmax}(\mathbf{A})_{r,c} = \exp(\mathbf{A}_{r,c}) / \sum_{r',c'} \exp(\mathbf{A}_{r',c'}), \tag{8.2.2}$$

β controls the temperature of the resulting probability map, and (r,c) iterate over pixel coordinates. In short, soft-argmax computes landmark coordinates $\mathbf{y}_k = (w, h)$ as a weighted average of all pixel coordinate pairs (r,c) where the weights are given by a softmax of landmark map $\mathbf{M}_k(\mathbf{x})$.

Predicted landmark coordinates are then fed into the second part of the network for attribute estimation. Having either classification or regression task, the model optimizes

$$Cost\_attr(\mathbf{x}, \tilde{a}) = \begin{cases} -\log P(A = \tilde{a}|\mathcal{X} = \mathbf{x}) & \text{, if classification} \\ |\tilde{a} - atr(\mathbf{x})| & \text{, if regression} \end{cases}$$

$P(A = \tilde{a}|\mathcal{X} = \mathbf{x})$ denotes the probability ascribed by the model to the class $\tilde{a}$ given input image $\mathbf{x}$, as computed by the final classification softmax layer. $\tilde{a}$ denotes the ground truth (GT) and $atr(\mathbf{x})$ the predicted attributes in the regression task. Using soft-argmax, as opposed to a simple softmax, the model is fully differentiable through its landmark locations and is trainable end-to-end.

8.3. Equivariant Landmark Transformation

We propose the following unsupervised learning technique to make the model's prediction consistent with respect to different transformations that are applied to the image. Consider an input image $\mathbf{x}$ and the corresponding landmarks $\mathbf{y}(\mathbf{x})$ predicted by the network given an input image $\mathbf{x}$. Now consider a small affine coordinate transformation T. We will use $T \odot \ldots$ to denote the application of such a transformation in coordinate space, whether it is applied to deform a bitmap image or to transform actual coordinates. If we apply this transformation to produce a deformed image $\mathbf{x}' = T \odot \mathbf{x}$ and compute the resulting landmark coordinates $\mathbf{y}(\mathbf{x}')$ predicted by the network, they should be very close to the result of applying the transformation on landmark coordinates $\mathbf{y}(\mathbf{x})$, i.e., we expect to have $\mathbf{y}(T \odot \mathbf{x}) \approx T \odot \mathbf{y}(\mathbf{x})$. The architecture for this technique, which we call *equivariant landmark transformation (ELT)*, is illustrated in Fig. 8.1-bottom. Multiple instances of C_T can thus be added to the overall training cost, each corresponding to a different transformation T.

Our entire model is trained end-to-end to minimize the following cost

$$\begin{aligned} \mathcal{L} = \quad & \tfrac{1}{N} \textstyle\sum_{(\mathbf{x},\tilde{a})\in\mathcal{D}} \{Cost\_attr(\mathbf{x},\tilde{a}) \; + \\ & \tfrac{\tau}{K} \textstyle\sum_{k=1}^{K} \| T \odot \underbrace{\mathbf{y}_k(\mathbf{x})}_{\mathbf{y}'_k} - \underbrace{\mathbf{y}_k(T \odot \mathbf{x})}_{\mathbf{y}''_k} \|_2^2 \} \; + \\ & \tfrac{\omega}{PK} \textstyle\sum_{\mathbf{x},\tilde{\mathbf{y}}_k} \sum_{k=1}^{K} ||\tilde{\mathbf{y}}_k - \mathbf{y}_k(\mathbf{x})||_2^2 + \gamma ||\boldsymbol{\theta}||_2^2, \end{aligned} \tag{8.3.1}$$

where $\mathcal{D}$ is the training set containing N pairs $(\mathbf{x}, \tilde{a})$ of input image and GT attribute. K is the number of landmarks per image. $\tilde{\mathbf{y}}_k$, $\mathbf{y}_k(\mathbf{x})$ and P respectively correspond to the GT, predicted landmarks and the number of images in the train set with labelled landmarks. $\boldsymbol{\theta}$ represents the parameters of the model. τ, ω, and γ are weights for losses. The first part of the cost is attribute classification or regression and affects the entire network. The second part is the ELT cost and can be applied to any training image, regardless of whether or not it is labeled with landmarks. This cost only affects the first part of the network (Landmark Localization). The third part is the squared Euclidean distance between GT and estimated landmark locations and is used only when landmark labels are provided. This cost only affects the first part of the network. The last cost is ℓ_2-norm on the model's parameters.

8.4. Experiments

To validate our proposed model, we begin with two toy datasets in Sections 8.4.1 and 8.4.2, in order to verify to what extent the class labels can be used to guide the landmark localization regardless of the complexity of the dataset. Later, we evaluate the proposed network on four real datasets: Polish sign-language dataset (Kawulok et al., 2014) in Section 8.4.3, Multi-PIE (Gross et al., 2010) in Section 8.4.4, and two datasets in the wild; 300W (Sagonas et al., 2013) and AFLW (Köstinger et al., 2011) in Sections 8.4.5 and 8.4.6. All the models are implemented in Theano (Al-Rfou et al., 2016).

8.4.1. Shapes dataset

To begin, we use a simple dataset to demonstrate our method's ability to learn consistent landmarks without direct supervision. Images in our Shapes dataset (see Fig. 8.3 top row for examples) consist of a white triangle and a white square on black background, with randomly sampled size, location, and orientation. The classification task is to identify which shape (triangle or square) is positioned closer to the upper-left corner of the image. We trained a model (as illustrated in Fig. 8.2) with six convolutional layers using 7×7 kernels, followed by two convolutional layers with 1×1 kernels, then the soft-argmax layer for landmark localization. Predicted landmarks input

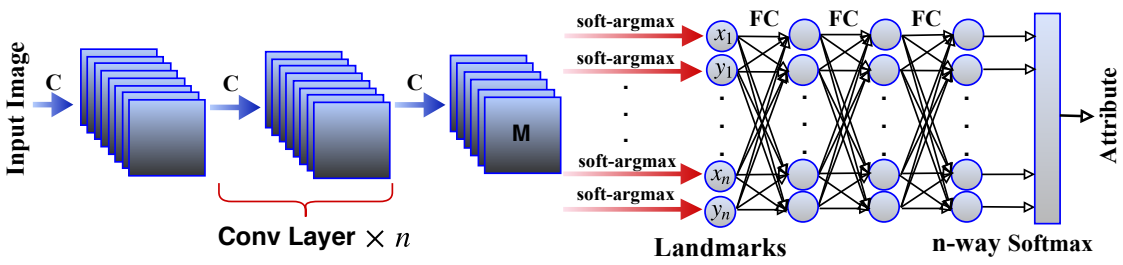


Fig. 8.2. Our basic implementation of the sequential multi-tasking architecture. The landmark localization model is composed of a series of conv (C) layers (with no pooling) and a soft-argmax output layer to detect landmarks. $\times n$ indicates repeating conv layer n times without parameter sharing. The detected landmarks are then fed to FC layers for attribute classification.

to two fully connected (FC) layers of size 40 and 2, respectively. The model is trained with *only* the cross-entropy cost on the class label *without* labeled landmarks or the unsupervised ELT cost.

Figure 8.3 shows the predictions of the trained model on a few samples from the dataset. In the second row, the green shape corresponds to the shape predicted to be the nearest to the upper-left corner, which was learned with 99% accuracy. The red and blue crosses correspond to the first soft-argmax and second soft-argmax landmark localizations, respectively. We observe that the red cross is consistently placed adjacent to the triangle, while the blue cross is near the square. This experiment shows the sequential architecture proposed here properly guides the first part of the network to find meaningful landmarks on this dataset, based solely on the supervision of the related classification task.

8.4.2. Blocks dataset

Our second toy dataset, Blocks, presents additional difficulty: each image depicts a figure composed of a sequence of five white squares with one white triangle at the head. See Fig. 8.4-top for all fifteen classes of Block dataset. We split the dataset into train, validation, and test sets, each having 3200 images.

Initially we trained the model with only cross-entropy on the class labels and evaluated the quality of the resulting landmark assignments. Ideally, the model would consistently assign each landmark to a particular block in the sequence from head (the triangle) to tail (the final square). However, in this more complex setting, the model did not predict landmarks consistently across

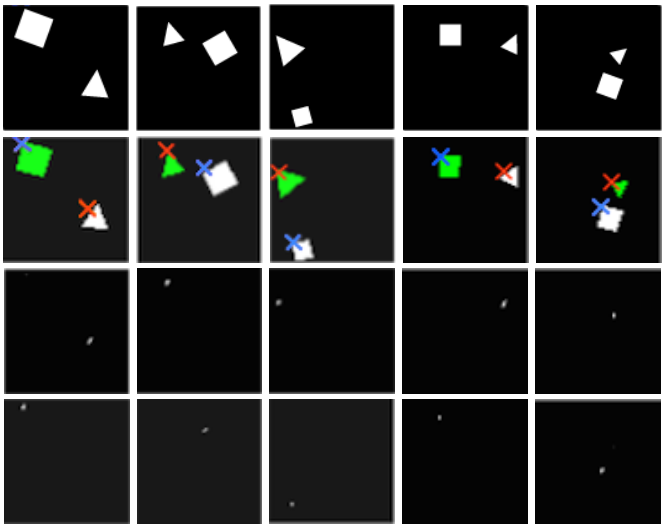


Fig. 8.3. Top row: Sample images from the Shapes dataset. Each 60×60 image contains one square and one triangle with randomly sampled location, size, and orientation. Second row: The two predicted landmarks and the object (in green) closest to the top-left corner classified by network. The *third and fourth* show the first and second landmark feature maps, corresponding closely with the location of triangle and square. (Best viewed in color with zoom.)

Tab. 8.1. Error of different architectures on Blocks dataset. The error is reported in pixel space. An error of 1 indicates 1 pixel distance to the target landmark location. The first 4 rows show the results of Seq-MT architecture, as shown in Fig. 8.2. The 5th and 6th rows show results of Comm-MT, depicted in Fig. 8.5. The last two rows show the results of Heatmap-MT, depicted in Fig. 8.6. The results are averaged over five seeds.

	Percentage of Images with Labeled Landmarks					
Model	1%	5%	10%	20%	50%	100%
Seq-MT (L)	8.33	3.95	3.35	1.98	1.19	0.44
Seq-MT (L+A)	8.02	3.45	3.20	1.67	1.05	**0.38**
Seq-MT (L+ELT)	6.42	1.94	1.37	1.16	0.85	
Seq-MT (L+ELT+A)	**6.25**	**1.70**	**1.26**	**1.07**	**0.74**	
Comm-MT (L)	12.89	11.56	10.72	9.39	5.04	3.41
Comm-MT (L+A)	12.28	11.19	10.36	9.01	4.21	2.97
Heatmap-MT (L)	10.09	6.59	5.27	3.82	2.78	2.01
Heatmap-MT (L+A)	9.27	6.35	5.62	3.75	3.14	2.23

examples. With the addition of the ELT cost, the model learns relatively consistent landmarks between examples *from the same class*, but this consistency does not extend between different classes. Unlike the Shapes dataset—where there was a consistent, if indirect, mapping between landmarks and the classification task—the correspondence among the classification task and landmark identities is more tenuous in the Blocks dataset. Hence, we introduce a labeled set of ground truth (GT)

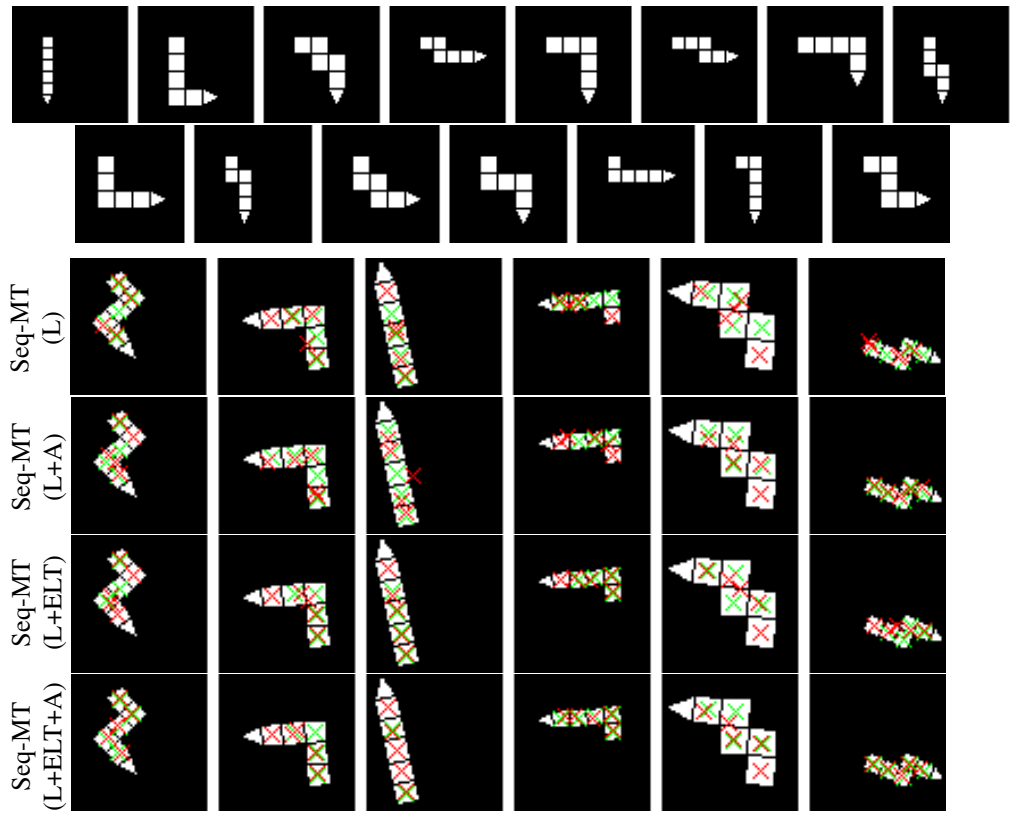


Fig. 8.4. *(top)* The fifteen classes of the Blocks dataset. Each class is composed of five squares and one triangle. To create each 60×60 image in the dataset, a random scale, translation, and rotation (up to 360 degrees) is applied to one of the base classes. *(bottom)* Sample landmark prediction on Blocks dataset using sequential multi-tasking models when only 5% of data is labeled with landmarks. Green and red cross show in order GT and predicted landmarks. (Best viewed in color with zoom.)

landmark locations and evaluate the landmark localization accuracy by having different percentages of the training set being labelled with landmarks.

Table 8.1 compares the results using the sequential multi-tasking model in the following scenarios: 1) using only the landmarks (Seq-MT (L)), which is equivalent to training only the first part of the network, 2) using landmarks and attribute labels (Seq-MT (L+A)), which trains the whole network on class labels and the first part of the network on landmarks, 3) using landmarks and the ELT cost (Seq-MT (L+ELT)), which trains only the first part of the network, and 4) using

three costs together (Seq-MT (L+ELT+A)).[1] When using the ELT cost (scenarios 3 & 4), we only apply it to images that do not provide GT landmarks to simulate semi-supervised learning [2].

As shown in Table 8.1, the *Seq-MT (L+A)* improves upon *Seq-MT (L)*, indicating that class labels can be used to guide the landmark locations. By adding the ELT cost, we can improve the results considerably. With *Seq-MT (L+ELT)* better performance is obtained compared to *Seq-MT (L+A)* showing that the unsupervised learning technique can substantially enhance performance. However, the best results are obtained with all costs when using class labels, the ELT and landmark costs. See Fig. 8.4-bottom for prediction samples when only 5% of the data are labeled with landmarks.

Since our model can be considered as a multi-tasking network, we contrast it with other multi-tasking architectures in the literature. We compare with two architectures: 1) The "common" multi-tasking architecture (Comm-MT) (Devries et al., 2014; Zhang and Zhang, 2014; Zhang et al., 2014c, 2016) where sub-networks for each task share a common set of initial layers ending in a common fully-connected layer (see Fig. 8.5).[3] We train two variants of this model, one with only landmarks (*Comm-MT (L)*) and another with landmarks and class labels (*Comm-MT (L+A)*) to see whether the class labels improve landmark localization. 2) Heat-map multi-tasking (Heatmap-MT), where – to avoid pooling layers – we follow the recent trend of maintaining the resolution of feature maps (Hariharan et al., 2015; Honari et al., 2016; Long et al., 2015; Tompson et al., 2015) and features detected for landmark localization do not pass through a FC layer. See Fig. 8.6 for an illustration of this architecture. The heatmaps right before the softmax layer are taken as input to the classification model. Note that this model doesn't have a landmark bottle-neck such as *Seq-MT (L+A)*.

As shown in Table 8.1, the Comm-MT approach is performing much worse than our Seq-MT architecture for landmark estimation. A drawback of this architecture is its use of pooling

[1]The set of examples with labeled landmarks is class-balanced.

[2]This is done to avoid unfair advantage of our model compared to other models on examples that provide landmarks. However, the ELT technique can be applied to any image, both with and without labeled landmarks.

[3]We tried other variants such as 1) a model that goes directly from the feature maps that have the same size as input image to the FC layer without any pooling layers and 2) a model that has more pooling layers and goes to a lower resolution before feeding the features to FC layers. Both models achieved worse results. Model 1 suffers from over-parameterization when going to FC layer. Model 2 suffers from loosing track of pooled features' locations since more pooling layers are used.

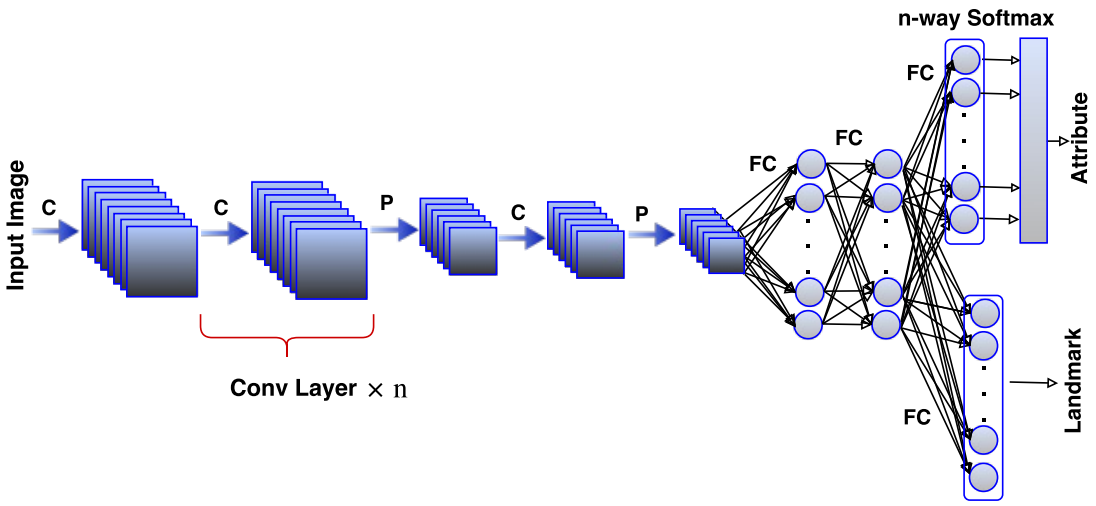


Fig. 8.5. Our implementation of the common multi-tasking (Comm-MT) architecture used in the literature (Devries et al., 2014; Zhang and Zhang, 2014; Zhang et al., 2014c, 2016). The model takes an image and applies a series of conv (C) and pooling (P) layers which are then passed to few common (shared) FC layers. The last common FC layer is then connected to two branches (each for a task), one for the classification task and another for the landmark localization.

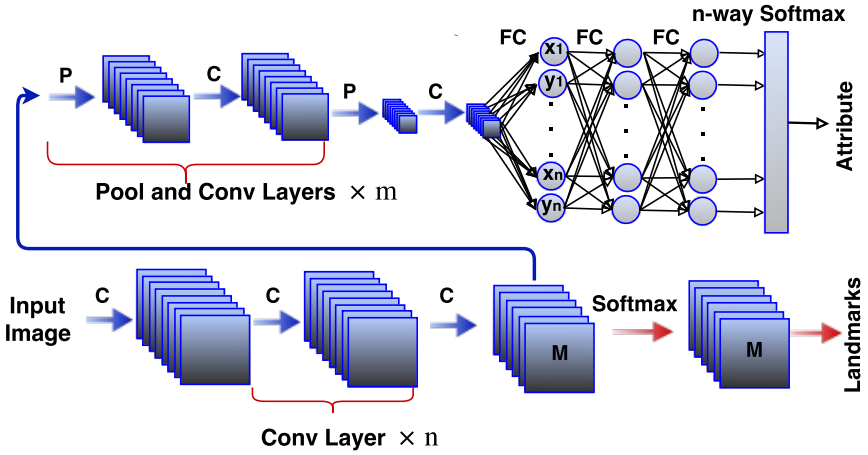


Fig. 8.6. Our implementation of multi-tasking architecture using heatmaps (Heatmap-MT). Landmarks are detected using conv (C) layers without sub-sampling, pooling, or FC layers. A softmax layer is used for landmark prediction in the output layer. Landmark heatmaps right before softmax layer are fed to a series of pool (P) and conv (C) layers which are then passed to FC layers. The last FC layer is fed to softmax for attribute classification.

layers which leads to sub-optimal results for landmark estimation. The model trained with extra class information performs better than the model trained only on landmarks. Heatmap-MT also performs worse than Seq-MT. This is likely due in part to Heatmap-MT using softmax log-likelihood

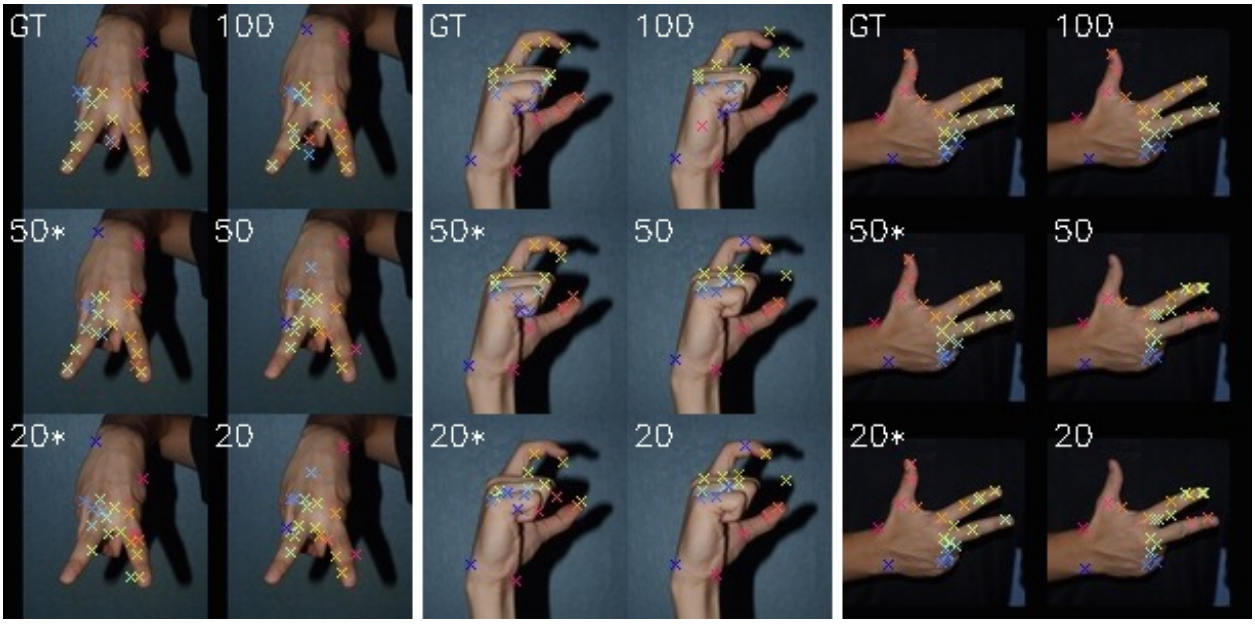


Fig. 8.7. Examples of our model predictions on the test set of the HGR1 dataset (Kawulok et al., 2014; Nalepa and Kawulok, 2014). GT represents ground-trust annotations, while numbers 100, 50, and 20 indicate which percentage of the training set with labeled landmarks used for training. Results are computed with Seq-MT (L+ELT+A) model (denoted *) and Seq-MT (L). Best viewed in color with zoom.

training (which cannot be more accurate than the discretization grid), while Seq-MT uses soft-argmax training based on real number coordinates. Moreover, in Heatmap-MT the class label is mostly helping when using a low percentage of labeled data, but in Seq-MT it is helping for all percentages of labeled data. We believe this is due to creating a bottle-neck of landmarks before class label prediction, which causes the class labels to impact landmarks more directly through back-propagation.

8.4.3. Hand pose estimation

Our first experiment on real data is on images of hands captured with color sensors. Most common image datasets with landmarks on hands such as NYU (Tompson et al., 2014) and ICVL (Tang et al., 2014) do not provide class labels. Also, most of the prior works in landmark estimation for hands are based on depth data (Sinha et al., 2016; Sridhar et al., 2013; Sun et al., 2015; Tang et al., 2014; Tompson et al., 2014; Yuan et al., 2018) whereas estimating from color data is more challenging. We use the Polish hand dataset (HGR1) (Kawulok et al., 2014; Nalepa and Kawulok,

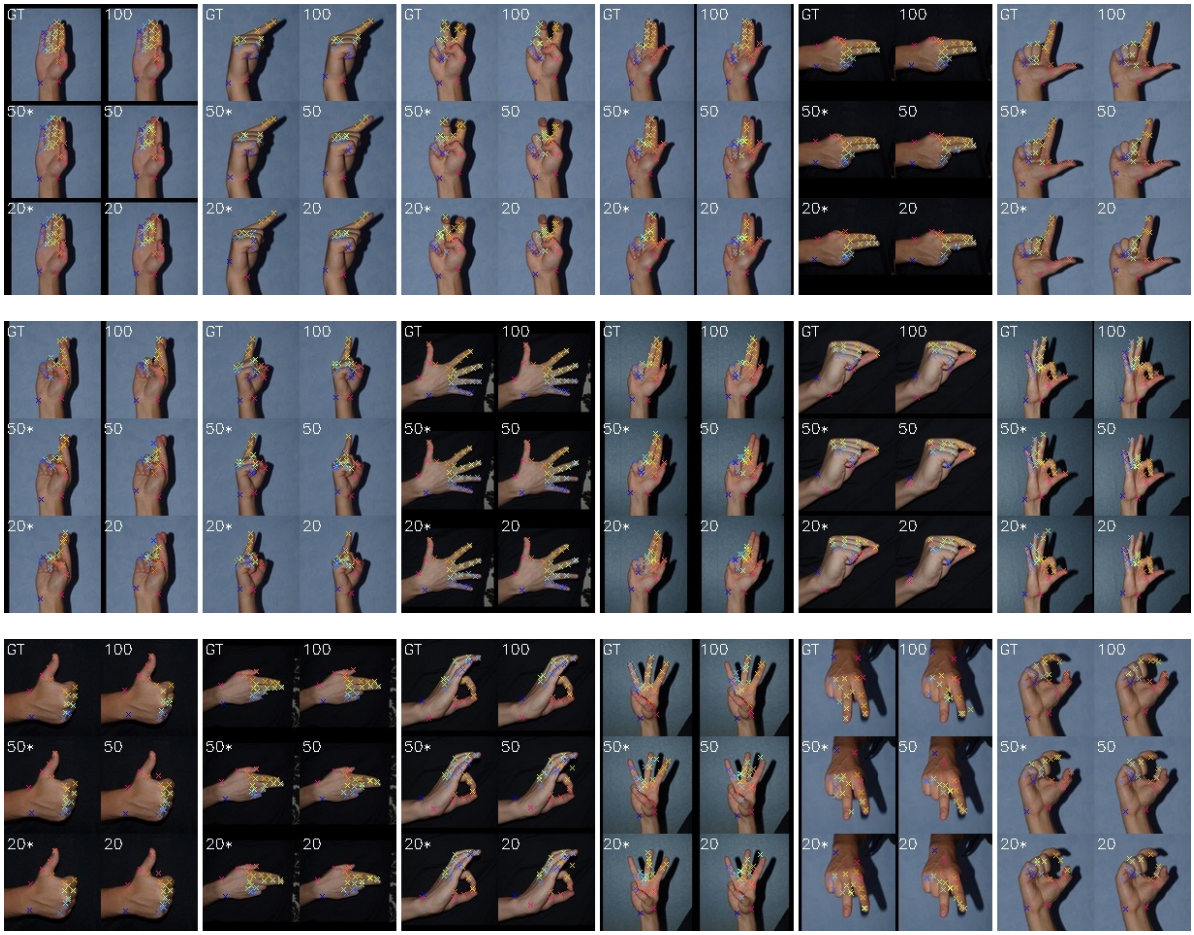


Fig. 8.8. Extra examples of our model predictions on the HGR1 (Kawulok et al., 2014; Nalepa and Kawulok, 2014) test set. GT represents ground-trust annotations, while numbers 100, 50, and 20 indicate the percentage of the training set with labeled landmarks. Results are computed with Seq-MT (L+ELT+A) model (denoted *) and Seq-MT (L). Examples illustrate improvement of the landmark prediction by using the class label and the ELT cost in addition to the labeled landmarks. The last three examples on the bottom row show examples with high errors. Best viewed in color with zoom.

2014), which provides 898 RGB images with 25 landmarks and 27 gestures from Polish sign language captured with uncontrolled lightning and uncontrolled background from 12 subjects. We divide images by id (with no overlap in subjects between sets) into training set (ids 1 to 8), validation (ids 11 and 12), and test (ids 9 and 10). We end up with 573, 163, and 162 images for training, validation and test sets, respectively. Accuracy of landmark detection on the HGR1 dataset is measured by computing the average RMSE metric in the image domain for every landmark and

Tab. 8.2. Performance of architectures on the HGR1 hands dataset. The error is Euclidean distance normalized by wrist width. Results are shown as percent; lower is better.

	Percentage of Images with Labeled Landmarks				
Model	5%	10%	20%	50%	100%
Seq-MT (L)	57.6	41.1	32.0	21.4	**15.8**
Seq-MT (L+A)	50.0	38.1	28.1	19.8	16.9
Seq-MT (L+ELT)	43.7	31.5	25.1	**17.7**	
Seq-MT (L+ELT+A)	**38.5**	**30.3**	**24.0**	19.1	
Comm-MT (L)	77.1	62.8	52.7	41.8	35.7
Comm-MT (L+A)	53.4	39.3	35.5	26.9	24.1
Heatmap-MT (L)	66.5	51.9	42.4	30.9	25.5
Heatmap-MT (L+A)	64.8	54.9	43.2	30.5	26.7

normalizing it by wrist width (the Euclidean distance between landmarks #1 and #25). We apply the ELT cost only on the images that do not have GT landmarks. Table 8.2 shows results for landmark localization on the HGR1 test set. All results are averaged over 5 seeds. We observe: 1) sequential multitasking improves results for most experiments compared to using only landmarks (Seq-MT(L)) or other multi-tasking approaches, 2) the ELT cost significantly improves results for all experiments, and 3) *Seq-MT (L+ELT+A)* compared to *Seq-MT (L)* can achieve the same performance with only half provided landmark labels (see 5%, 10%, 20%). We show examples of landmark prediction with different models in Figures 8.7 and 8.8. ELT and attribute classification (A) losses significantly improve results with a smaller fraction of annotated landmarks.

8.4.4. Multi-PIE dataset

We next evaluate our model on facial landmark datasets. Similar to Hands, most common face datasets including Helen (Le et al., 2012), LFPW (Belhumeur et al., 2013), AFW (Zhu and Ramanan, 2012), and 300W (Sagonas et al., 2013), only provide landmark locations and no classes. We start with Multi-PIE (Gross et al., 2010) since it provides, in addition to 68 landmarks, 6 emotions and 15 camera locations. We use these as class labels to guide landmark prediction.[4] We use images from 5 cameras (1 frontal, 2 with $\pm$15 degrees, and 2 with $\pm$30 degrees) and in each case a random

[4]Our research does not involve face recognition, and emotion classes are used only to improve the precision of landmark localization.

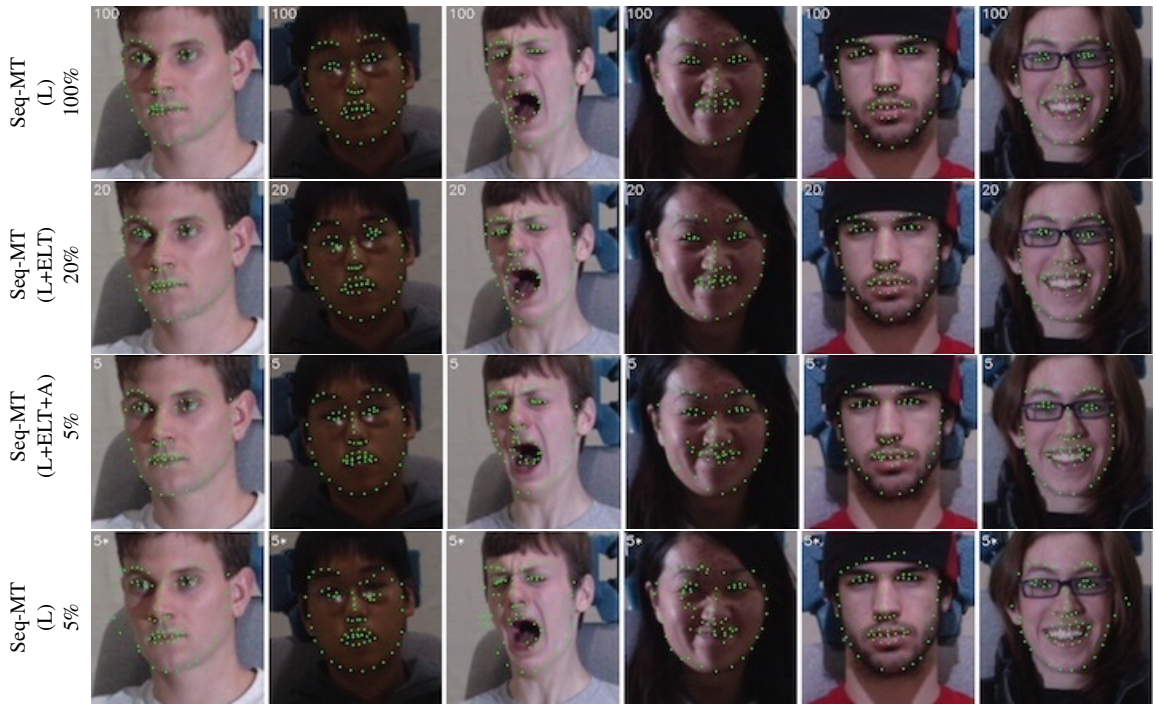


Fig. 8.9. Examples of our model predictions on Multi-PIE (Gross et al., 2010). On left is the percentage of labelled data. We observe close predictions between the top two rows indicating the effectiveness of the proposed ELT cost. Comparison between the last two rows shows the effectiveness of our method with only a small amount of labeled landmarks. Best viewed in color with zoom.

illumination is selected. The images are then divided into subsets by id[5], with ids 1-150 in the training set, ids 151-200 in the valid set, and ids 201-337 in the test set. We end up with 1875, 579, and 1054 images in training, validation, and test sets.

Due to using camera and emotion classes, our classification network has two branches, one for emotion and one for camera, with each branch receiving landmarks as inputs (see Supp. for architecture details). We compare our model with Comm-MT, Heatmap-MT architectures with and without class labels in Table 8.3. Comparing models, we make the same observation as in Section 8.4.3 and the best performance is obtained when ELT and classification costs are used jointly, indicating both techniques are affective to get the least error. See some sample predictions in Fig. 8.9.

Tab. 8.3. Performance of different architectures on Multi-PIE dataset. The error is Euclidean distance normalized by eye-centers (as a percent; lower is better). We do not apply ELT cost on the examples that provide GT landmarks.

Model	Percentage of Images with Labeled Landmarks				
	5%	10%	20%	50%	100%
Seq-MT (L)	7.98	7.02	6.28	5.50	**5.09**
Seq-MT (L+A)	7.71	6.91	6.20	5.49	5.12
Seq-MT (L+ELT)	6.69	6.24	5.78	5.27	
Seq-MT (L+ELT+A)	**6.57**	**6.16**	**5.73**	**5.23**	
Comm-MT (L)	9.22	7.93	7.02	6.27	5.71
Comm-MT (L+A)	9.11	8.00	6.92	6.20	5.68
Heatmap-MT (L)	10.83	9.18	8.13	7.00	6.63
Heatmap-MT (L+A)	11.03	9.03	8.15	7.11	6.65

Tab. 8.4. Comparison of recent models on their training conditions. RAR and (Lv et al., 2017) initialize their models by pre-trained parameters. TCDCN uses 20,000 extra labelled data. Finally, RAR adds manual samples by occluding images with sunglasses, medical masks, phones, etc to make them robust to occlusion. Similar to RCN, Seq-MT and RCN$^+$ both have an explicit validation set for HP selection and therefore use a smaller training set. Neither use any extra data, either through pre-trained models or explicit external data.

Feature	Model						
	RAR	(Lv et al., 2017)	TCDCN	CFSS	RCN	RCN$^+$ / Seq-MT	RCN$^+$ (L+ELT) (all-train)
Hyper-parameter selection dataset	Test-set	Test-set	Test-set	Test-set	Valid-set	Valid-set	Test-set
Training on entire training set?	Yes	Yes	Yes	Yes	No	No	Yes
Uses extra dataset?	Yes	Yes	Yes	No	No	No	No
Manually augmenting the training set?	Yes	No	No	No	No	No	No
FPS (GPU)	4	83	667	-	545	487 / 545	545

8.4.5. 300W dataset

In order to evaluate our architecture on natural images in the wild we use 300W (Sagonas et al., 2013) dataset. This dataset provides 68 landmarks and is composed of 3,148 (337 AFW, 2,000 Helen, and 811 LFPW) and 689 (135 IBUG, 224 LFPW, and 330 Helen) images in the training and test sets, respectively. Similar to RCN (Honari et al., 2016), we split the training set into 90%

[5]These ids are not personally identifiable information.

Fig. 8.10. Landmark localization samples on 300W (Sagonas et al., 2013) test-set. The green and red dots show GT and model predictions, respectively. The yellow lines show the error. These examples illustrate the improved accuracy obtained by using the ELT cost. The rectangles show the regions that landmarks are mostly improved.

(2,834 images) train-set and 10% (314 images) valid-set. Since this dataset does not provide any class label, we can evaluate our model in L and L+ELT cases.

In Table 8.5-left we compare Seq-MT with other models in the literature. Seq-MT model outperforms many models including CDM, DRMF, RCPR, CFAN, ESR, SDM, ERT, LBF and CFSS, and is only doing worse than few recent models with complicated architectures, e.g., RCN (Honari et al., 2016) with multiple branches, RAR (Xiao et al., 2016) with multiple refinement procedure and Lv et. al. (Lv et al., 2017) with multiple steps. Note that the originality of Seq-MT is not in the specific architecture used for the first part of the network that localizes landmarks, but rather in its multi-tasking architecture (specifically in its usage of the class labels to enhance landmark localization) and also leveraging ELT cost. The landmark localization part of Seq-MT can be replaced with more complex models. To verify this, we use the RCN model (Honari et al., 2016), with publicly available code, and replace the original softmax layer with a soft-argmax layer in order to apply the ELT cost. We refer to this model as RCN$^+$ and it is trained with these hyperparameters:

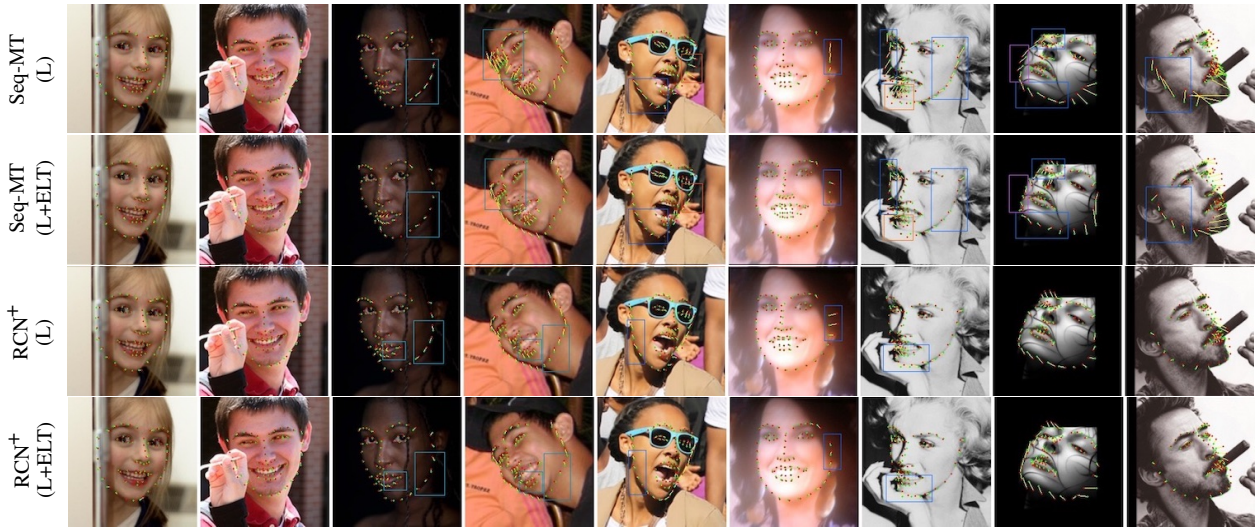


Fig. 8.11. Extra examples of our model predictions on 300W (Sagonas et al., 2013) test-set. The first two columns depict examples where all models get accurate predictions, The next 5 columns illustrate the improved accuracy obtained by using ELT loss in two different architectures (Seq-MT and RCN). The last two columns show difficult examples where error is high. The rectangles indicate the regions that landmarks are mostly affected. The green and red dots show ground truth (GT) and model predictions (MP), respectively. The yellow lines show the error by connecting GT and MP. Note that the ELT loss improves predictions in both architectures. Best viewed in color with zoom.

$\beta = 1.0, \tau = 0.5, \gamma = 0, \omega = 1.0$. In Figure 8.12 we show the architecture of *RCN* $^+$. The result is shown as RCN$^+$(L) when using only landmark cost and RCN$^+$(L+ELT) when using landmark plus ELT cost. On the 300W dataset we apply the ELT cost to samples with or without labelled landmarks to observer how much improvement can be obtained when used on all data. We can further reduce RCN error from 5.54 to 5.1 by applying the ELT cost and soft-argmax. This is a new state of the art without any data-augmentation. Also we evaluate accuracy of RCN$^+$(L+ELT) trained without validation set and with early stopping on test set and achieve error of 4.9 - the overall state-of-the-art on this dataset.

In Table 8.6 we compare Seq-MT with Heatmap-MT and Common-MT on different percentage of labelled landmarks. We also demonstrate the improvement that can be obtained by using RCN$^+$. Note that the ELT cost improves the results when applied to two different landmark localization architectures (Seq-MT, RCN). Moreover, it considerably improves the results on IBUG test-set that

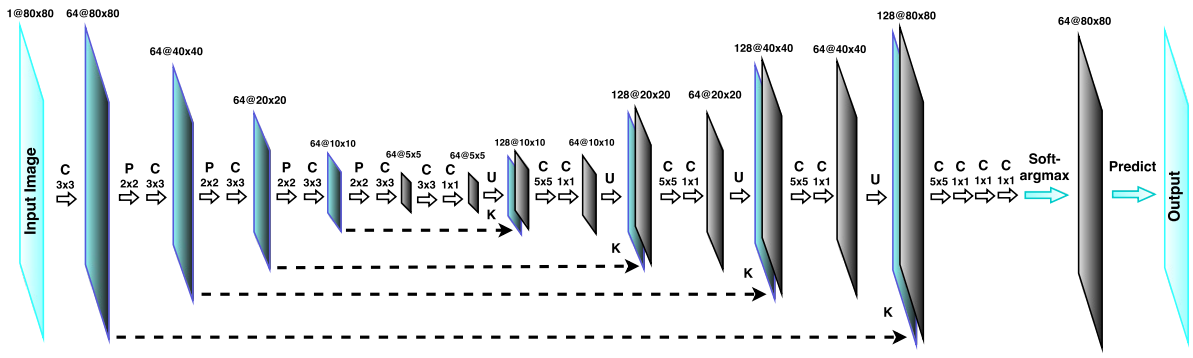


Fig. 8.12. The ReCombinator Networks (RCN) (Honari et al., 2016) architecture used for experiments on 300W dataset. P indicates a pooling layer. All pooling layers have stride of 2. C indicates a convolutional layer. The number written below C indicates the convolution kernel size. All convolutions have stride of 1. U indicates an upsampling layer, where each feature map is upsampled to the next (bigger) feature map resolution. K indicates concatenation, where the upsampled features are concatenated with features of the same resolution before a pooling is applied to them. The dashed arrows indicate the feature maps are carried forward for concatenation. The solid arrows following each other, e.g. P, C, indicate the order of independent operations that are applied. The number written above feature maps in $num@width \times height$ format indicate number of feature maps and the width and height of the feature maps. On AFLW, we use 70 feature maps per layer (instead of 64) and we get two levels coarser to get to 1×1 resolution (instead of 5×5). On both datasets we shoud $\beta = 100$ for soft-argmax layer. Please note that the notations C and K used in this figure are independent from the similar notation used in this Chapter.

contains more difficult examples than the training set. Figures 8.10 and 8.11 shows the improvement obtained by using ELT cost on some test set samples.

8.4.6. AFLW dataset

AFLW (Köstinger et al., 2011) contains images of 24,386 faces with 19 fiducial landmarks and 3D real-valued head pose information. We use pose as auxiliary task. We split dataset into training, testing sets, with 20,000 and 4,386 images, respectively. Furthermore, we allocate 2,000 images from training set for validation set. We use the same splits as in previous work (Ren et al., 2014), (Lv et al., 2017) for direct comparison. We normalize RMSE by face size as in (Lv et al., 2017). We evaluate our method on RCN$^+$ trained with ELT cost and head pose regression cost and obtain a new state of the art of 1.59 with 27% relative improvement. See comparison with other models in Table 8.5-righ-bottom. We also evaluate our method with only 180 (1%) or 900 (5%) images

Tab. 8.5. Comparison with other SOTA models (as a percent; lower is better). *(left)* Performance of different architectures on 300W test-set using 100% labeled landmarks. The error is Euclidean distance normalized by ocular distance. *(right-top)* Comparison with four other multi-tasking approaches and RCN. For these comparisons, we have implemented the specific architectures proposed in those papers. Error is as in Sections 8.4.3 and 8.4.4. *(right-bottom)* Comparison of different architectures on AFLW test set. The error is Euclidean distance normalized by face size.

300W Dataset

Model	Common	IBUG	Fullset
CDM (Yu et al., 2013)	10.10	19.54	11.94
DRMF (Asthana et al., 2013)	6.65	19.79	9.22
RCPR (Burgos-Artizzu et al., 2013)	6.18	17.26	8.35
CFAN (Zhang et al., 2014a)	5.50	16.78	7.69
ESR (Cao et al., 2014)	5.28	17.00	7.58
SDM (Xiong and De la Torre, 2013)	5.57	15.40	7.50
ERT (Cao et al., 2014)			6.40
LBF (Ren et al., 2014)	4.95	11.98	6.32
CFSS (Zhu et al., 2015)	4.73	9.98	5.76
TCDCN* (Zhang et al., 2016)	4.80	8.60	5.54
RCN (Honari et al., 2016)	4.70	9.00	5.54
RCN +\ denoising (Honari et al., 2016)	4.67	8.44	5.41
RAR (Xiao et al., 2016)	**4.12**	8.35	4.94
(Lv et al., 2017)	4.36	**7.56**	4.99
Heatmap-MT (L)	6.18	13.56	7.62
Comm-MT (L)	5.68	11.04	6.73
Seq-MT (L)	4.93	10.24	5.95
Seq-MT (L+ELT)	4.84	9.53	5.74
RCN$^+$ (L)	4.47	8.47	5.26
RCN$^+$ (L+ELT)	4.34	8.20	5.10
RCN$^+$ (L+ELT) (all-train)	4.20	7.78	**4.90**

Model	Multi-PIE		HGR1
Percent Labelled	5%	100%	100%
MT-DCNN (Zhang and Zhang, 2014)(L+A)	11.13	7.60	20.87
TCDCN (Zhang et al., 2014c)(L+A)	18.46	10.59	25.85
TCDCN-2 (Zhang et al., 2016)(L+A)	10.75	5.83	18.81
MT-Conv (Devries et al., 2014)(L+A)	9.99	8.08	19.20
RCN (Honari et al., 2016) (L)	7.53	5.78	13.65
RCN+ (L)	6.89	5.04	11.02
RCN+ (L+A)	**6.82**	**4.97**	**10.88**

AFLW Dataset

Model	Labeled Images		
	1%	5%	100%
CDM (Yu et al., 2013)	_	_	5.43
ERT (Cao et al., 2014)	_	_	4.35
LBF (Ren et al., 2014)	_	_	4.25
SDM (Xiong and De la Torre, 2013)	_	_	4.05
CFSS (Zhu et al., 2015)	_	_	3.92
RCPR (Burgos-Artizzu et al., 2013)	_	_	3.73
CCL (Zhu et al., 2016)	_	_	2.72
(Lv et al., 2017)	_	_	2.17
RCN$^+$ (L)	2.88	2.17	1.61
RCN$^+$ (L+A)	2.52	2.08	1.60
RCN$^+$ (L+ELT+A)	2.46	**2.03**	**1.59**

Tab. 8.6. Performance of different architectures on 300W test-set. The error is Euclidean distance normalized by ocular distance (eye-centers). Error is shown as a percent; lower is better.

		Percentage of Images with Labeled Landmarks				
	Model	5%	10%	20%	50%	100%
Fullset	Heatmap-MT (L)	13.47	11.68	9.85	8.18	7.62
	Comm-MT (L)	16.73	9.66	8.61	7.39	6.73
	Seq-MT (L)	9.82	8.30	7.26	6.28	5.95
	Seq-MT (L+ELT)	8.23	7.28	6.62	6.10	5.74
	RCN$^+$ (L)	7.26	6.48	5.91	5.52	5.26
	RCN$^+$ (L+ELT)	**7.22**	**6.32**	**5.88**	**5.45**	**5.10**
IBUG	Heatmap-MT (L)	26.36	22.77	18.46	14.94	13.56
	Comm-MT (L)	28.64	16.17	14.56	12.16	11.04
	Seq-MT (L)	18.74	16.21	13.41	11.20	10.24
	Seq-MT (L+ELT)	14.68	12.73	11.39	10.37	9.53
	RCN$^+$ (L)	15.36	12.74	11.82	10.12	8.47
	RCN$^+$ (L+ELT)	**12.54**	**10.35**	**9.56**	**8.67**	**8.20**

of labeled landmarks. Under these settings we get significant improvement with semi-supervised

learning. With only 5% of labeled data our method outperforms the previous state of the art methods. Figure 8.13 shows some samples on AFLW test set.

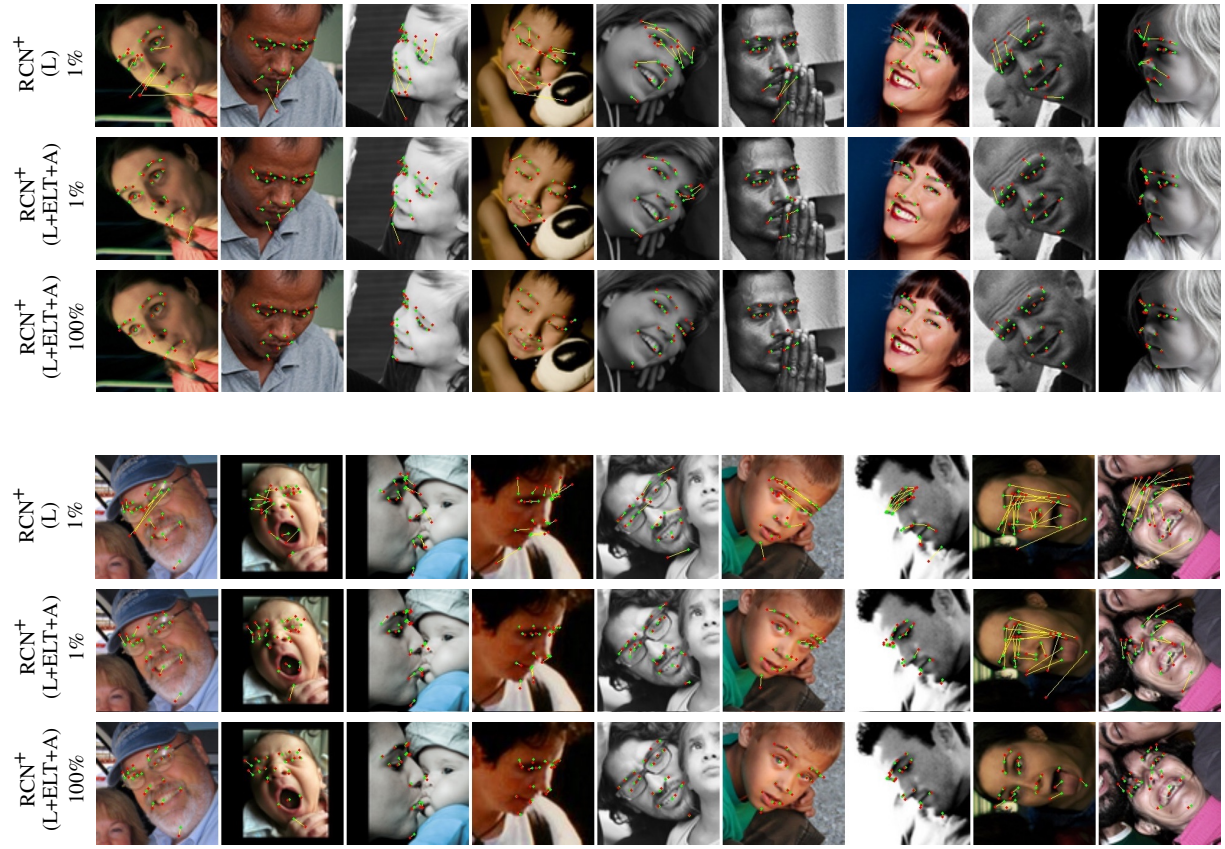


Fig. 8.13. Examples of our model predictions on the AFLW test set. Comparing the first and second rows shows the improvement obtained by using ELT+A with only 1% of labelled landmarks. Note the model trained using ELT+A preserves better the distribution over the landmarks. The last two columns in the bottom row show samples with high error on small percentage of labelled landmaks, which is due to extreme rotation. The bottom row shows the prediction using L+ELT+A on the entire set of labelled landmarks, which gets the best results. The green and red dots show ground truth (GT) and model predictions (MP), respectively. The yellow lines show the error by connecting GT and MP. Best viewed in color with zoom.

8.4.7. MTFL dataset

In table 8.7 we compare with other models on MTFL (Zhang et al., 2014c) dataset which provides 5 landmarks on facial images: eye-centers, nose tip, mouth corners. We follow the same

protocol as (Honari et al., 2016) for comparison, where we use train and valid sets of 9,000 and 1,000 images, respectively. We test our model on AFLW and AFW subsets, with 29,995 and 337 images, that were re-annotated with 5 landmarks. For the $L + A$ case we use the head-pose which is categorized into one of the five cases: right profile, right, frontal, left, left profile. Other attribute labels, e.g. gender and wearing glasses, cannot be determined from such few landmarks and therefore are not useful in our proposed semi-supervised learning of landmarks.

Tab. 8.7. Results on MTFL test sets for 100% labelled data

	Model					Our	
	ESR	RCPR	SDM	TCDCN	RCN	RCN+(L)	RCN+(L+A)
AFLW	12.4	11.6	8.5	8.0	5.6	5.22	**5.02**
AFW	10.4	9.3	8.8	8.2	5.36	5.13	**5.08**

8.4.8. Comparison with other techniques

In Table 8.5 we compare with recent models proposed for landmark localization and in Table 8.4 we evaluate their training conditions. RAR, TCDCN, Lv et. al., and CFSS do not use an explicit validation set. This makes comparison with these models more difficult for two reasons: 1) These models do hyper-parameter (HP) selection on the test set, which makes them overfit on the test set; and 2) Their effective training size is bigger. When we use the entire training set (row *RCN$^+$ (L+ELT) (all-train)* in Table 8.5-left we report new SOTA on the 300W dataset. The first three models use extra datasets, either through pre-trained models (RAR, Lv et. al.) or additional labeled data (TCDCN), while we do not leverage any extra data. Finally, our method is 136 and 6.5 times faster than RAR and Lv et. al methods.

8.5. Analysis

In this section, we provide some analysis such as which auxiliary labels can be leveraged in the proposed semi-supervised setting with landmarks, or how soft-argmax compares with the soft-max layer for landmark localization. We also report classification performance for different multi-tasking models studied in this chapter.

8.5.1. Selecting auxiliary labels for semi-supervised learning

The impact of an attribute on the landmark in sequential training depends on the amount of informational overlap between the attribute and the landmarks. We suggest to measure the normalized mutual information adjusted to randomness (Adjusted Mutual Information (AMI)), as a selection heuristic, prior to applying our method. AMI ranges from 0 to 1 and indicates the fraction of statistical overlap. We compute for each attribute its AMI with all landmark coordinates.

On Multi-PIE we got $AMI(w;h)$ = 0.045, indicating a low mutual information between the width w and the height h landmark coordinates. We therefore compute AMI for attribute (A) and every landmark (as w,h pair) by discretizing every variable uniformly under assumption of coordinate independence: $AMI(A;w,h) = AMI(A;w)+AMI(A;h)$. Every variable is uniformly discretized to have 20 levels at most. Finally we measure averaged mutual information between an attribute and the set of landmarks as

$$\frac{1}{N \times K} \sum_{n=1}^{N} \sum_{k=1}^{K} AMI(A_n; w_n^k) + AMI(A_n; h_n^k)$$

where N and K indicate the number of samples and landmarks. The second sum is over the width $\mathbf{w}_n^k$ and height h_n^k for each landmark $k \in K$ in sample n. In Table 8.8 we observe that hand gesture labels and head pose regression are among the most effective attributes for our method. There is little mutual information between wearing glasses and landmarks, indicating lack of usefulness of this attribute for our semi-supervised setting.

Tab. 8.8. Mutual Information between all landmarks and each attribute

Dataset		MultiPIE			HGR1
Attribute	Random	Emotion	Camera	Identity	Gesture Label
AMI, mean	.000	.098	.229	.049	.559
AMI, max	.006	.229	.493	.088	.669

Dataset		AFLW	MTFL	
Attribute	Random	Pose Regression	Glasses	Pose Classification
AMI, mean	.000	.536	.002	.069
AMI, max	.006	.576	.003	.222

The attributes that are mostly useful yield a high accuracy, or low error, if we just train a neural network that takes only ground truth landmarks as input and predicts the attribute. This indicates that by relying only on landmarks we can get high accuracy for those attributes. In Table 8.9 we

compare the attribute prediction accuracy from the proposed Seq-MT model with a case when we do such prediction from GT landmarks. Prediction from GT landmarks always outperforms the one of Seq-MT. This indicates that in our semi-supervised setting, where we have few labelled landmarks, by improving the predicted locations of landmarks, both attribute and landmarks error would reduce.

Tab. 8.9. Attribute classification accuracy (MultiPIE, HGR1)—higher is better—or prediction error (AFLW)—lower is better—from GT & estimated landmarks.

	MultiPIE		HGR1	AFLW
Attribute	Camera	Emotion	Label	Pose Error
From GT Landmarks	99.54 ↑	88.21 ↑	91.7 ↑	4.98 ↓
Best Seq-MT Attr. Predict.	98.96	86.48	79.1	5.10

8.5.2. Comparison of softmax and soft-argmax

Heatmap-MT(L) and Seq-MT(L) have the same architectures but use different loss functions (softmax vs. soft-argmax). RCN(L) and RCN+(L) also only differ in their loss function. When comparing these models in Tables 8.1, 8.2, 8.3, 8.5, and 8.6 soft-argmax outperforms soft-max. To further examine these two losses we replace soft-max with soft-argmax in Heatmap-MT and show the results in Table 8.10. Comparing the results in Table 8.10 with Tables 8.2 and 8.3, we observe improved performance of landmark localization using soft-armgax. In soft-max the model cannot be more accurate than the number of elements in the grid, since soft-max does a classification over the pixels. However, in soft-argmax the model can regress to any real number and hence can get more accurate results. We believe this is the reason behind its better performance.

Tab. 8.10. Results on Heatmap-MT (L+A) comparing soft-max with soft-argmax.

Dataset		5%	10%	20%	50%	100%
Multi-PIE	softmax	11.03	9.03	8.15	7.11	6.65
	soft-argmax	**8.00**	**7.06**	**6.29**	**5.49**	**5.14**
HGR1	softmax	64.8	54.9	43.2	30.5	26.7
	soft-argmax	**56.88**	**42.79**	**33.07**	**22.5**	**18.8**

8.5.3. Attribute classification accuracy

Although the focus of this paper is on improving landmark localization, in order to observe the impact of each multi-tasking approach on the attribute classification accuracy, we report the

classification results for MultiPIE dataset on emotion in Table 8.11 and on camera in Table 8.12. Results show that the classification accuracy improves by providing more labeled landmarks, despite having the number of *(image, class label)* pairs unchanged. It indicates that improving landmark localization can directly impact the classification accuracy. Landmarks are especially more helpful in emotion classification. On camera classification, the improvement is small and all models are getting high accuracy. Another observation is that Heatmap-MT performs better on classification tasks compared to the other two multi-tasking approaches. We believe this is due to passing more high-level features from the image to the attribute classification network compared to Seq-MT. However, this model is performing worse than Seq-MT on landmark localization. The Seq-MT model benefits from the landmark bottleneck to improve its landmark localization accuracy. In Tables 8.11 and 8.12 by adding the ELT cost the classification accuracy improves (in addition to landmarks) indicating the improved performance in landmark localization can enhance classification performance.

In Table 8.13 we show classification accuracy obtained using different multi-tasking techniques for the HGR1 hands dataset. Similar to the Multi-PIE dataset, we observe increased accuracy by providing more labeled landmarks, showing the classification would benefit directly from landmarks. Also similar to Multi-PIE, we observe better classification accuracy with Heatmap-MT. Comparing Seq-MT models, we observe improved classification accuracy by using the ELT cost. It demonstrates the impact of this component on both landmark localization and classification accuracy.

In Table 8.14 we show pose estimation error on AFLW dataset using different percentage of labelled data for RCN$^+$ (L+ELT+A) model and compare the results to a model trained to estimate pose from GT landmarks. All models get close results compared to GT model indicating RCN$^+$ (L+ELT+A) can do a reliable pose estimation using a small set of labelled landmarks.

8.6. Architectural details

The architectural details of Seq-MT model on different datasets can be seen in Tables 8.17, 8.18 and 8.19. Architectural details of Comm-MT and Heatmap-MT for Blocks dataset are shown in Tables 8.15 and 8.16. For other dataset, the kernel size and the number of feature maps for conv layers and the number of units for FC layers change similar to Seq-MT model on those datasets.

Tab. 8.11. Emotion classification accuracy on Multi-PIE test set. In percent; higher is better.

	Percentage of Images with Labeled Landmarks				
Model	5%	10%	20%	50%	100%
Comm-MT (L+A)	74.67	79.90	83.76	86.37	86.83
Heatmap-MT (L+A)	**85.14**	**87.50**	**86.93**	**88.16**	**87.29**
Seq-MT (L+A)	78.78	82.62	84.69	84.03	84.86
Seq-MT (L+A+ELT)	82.90	84.57	84.85	86.48	

Tab. 8.12. Camera classification accuracy on Multi-PIE test set. In percent; higher is better.

	Percentage of Images with Labeled Landmarks				
Model	5%	10%	20%	50%	100%
Comm-MT (L+A)	96.98	97.53	98.30	98.63	98.80
Heatmap-MT (L+A)	**98.46**	**98.99**	**98.99**	**98.98**	**98.98**
Seq-MT (L+A)	97.97	98.31	98.50	98.96	98.92
eq-MT (L+A+ELT)	98.41	98.53	98.47	98.43	

Tab. 8.13. Classification accuracy on hands test set. In percent; higher is better.

	Percentage of Images with Labeled Landmarks				
Model	5%	10%	20%	50%	100%
Comm-MT (L+A)	60.86	69.64	69.20	76.03	73.42
Heatmap-MT (L+A)	**83.74**	**87.86**	**87.55**	**90.29**	**89.27**
Seq-MT (L+A)	69.08	70.14	72.26	77.07	75.92
Seq-MT (L+A+ELT)	74.64	75.01	73.90	79.10	

Tab. 8.14. Pose degree estimation error on AFLW test set, as average of yaw, pitch, roll values. lower is better.

	Percentage of Images with Labeled Landmarks		
Model	1%	5%	100%
RCN^+(L+ELT+A)	5.05	5.01	5.12
GT			4.98

8.7. Conclusion

We presented a new architecture and training procedure for semi-supervised landmark localization. Our contributions are twofold; We first proposed an unsupervised technique that leverages equivariant landmark transformation without requiring labeled landmarks. In addition we developed

Tab. 8.15. Architectural details for Comm-MT Model on Blocks dataset.

Input = $60 \times 60 \times 1$	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Pool 2×2, stride 2	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Pool 2×2, stride 2	
Conv $1 \times 1 \times 8$, ReLU, stride 1, SAME	
Conv $1 \times 1 \times 8$, ReLU, stride 1, SAME	
FC $\#units = 256$, ReLU, dropout-prob=.25	
FC $\#units = 256$, ReLU, dropout-prob=.25	
Classification branch	Landmark localization branch
FC $\#units = 15$, Linear softmax(dim=15)	FC $\#units = 10$, Linear

an architecture to improve landmark estimation using auxiliary attributes such as class labels by backpropagating errors through the landmark localization components of the model. Experiments show that these achieve high accuracy with far fewer labeled landmark training data in tasks of landmark location for hands and faces. We achieve new state of the art performance on public benchmark datasets for fiducial points in the wild, 300W and AFLW.

Acknowledgements

We would like to thank Compute Canada and Calcul Quebec for providing computational resources. This work was partially funded by NVIDIA's NVAIL program.

Tab. 8.16. Architectural details for Heatmap-MT Model on Blocks datasets.

Input = $60 \times 60 \times 1$	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	
Conv $1 \times 1 \times 8$, ReLU, stride 1, SAME	
Conv $1 \times 1 \times 5$, ReLU, stride 1, SAME	
classification branch	landmark localization branch
Pool 2×2, stride 2	—
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	—
Pool 2×2, stride 2	—
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	—
Pool 2×2, stride 2	—
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	—
Pool 2×2, stride 2	—
Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME	—
FC $\#units = 256$, ReLU, dropout-prob=.25	—
FC $\#units = 256$, ReLU, dropout-prob=.25	—
FC $\#units = 15$, Linear	—
softmax(dim=15)	softmax(dim=60×60)

Tab. 8.17. Architectural details of Seq-MT model used for Shapes and Blocks datasets. Each conv layer has three values as $width \times height \times num$ indicating width, height of kernel and the number of feature maps of the convolutional layer. SAME indicates the input map is padded with zeros such that input and output maps have the same resolution.

Shapes Dataset	**Blocks Dataset**
Model HP: $\omega = 0, \tau = 0, \gamma = 0, \beta = 1$, ADAM	Model HP: $\omega = 1, \tau = 1, \beta = 1$, ADAM
Landmark Localization Network	Landmark Localization Network
Input = $60 \times 60 \times 1$	Input = $60 \times 60 \times 1$
Conv $7 \times 7 \times 16$, ReLU, stride 1, SAME	Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME
Conv $7 \times 7 \times 16$, ReLU, stride 1, SAME	Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME
Conv $7 \times 7 \times 16$, ReLU, stride 1, SAME	Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME
Conv $7 \times 7 \times 16$, ReLU, stride 1, SAME	Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME
Conv $7 \times 7 \times 16$, ReLU, stride 1, SAME	Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME
Conv $7 \times 7 \times 16$, ReLU, stride 1, SAME	Conv $9 \times 9 \times 8$, ReLU, stride 1, SAME
Conv $1 \times 1 \times 16$, ReLU, stride 1, SAME	Conv $1 \times 1 \times 8$, ReLU, stride 1, SAME
Conv $1 \times 1 \times 2$, ReLU, stride 1, SAME	Conv $1 \times 1 \times 5$, ReLU, stride 1, SAME
soft-argmax(num_channels=2)	soft-argmax(num_channels=5)
Classification Network	Classification Network
FC $\#units = 40$, ReLU	FC $\#units = 256$, ReLU, dropout-prob=.25
FC $\#units = 2$, Linear	FC $\#units = 256$, ReLU, dropout-prob=.25
	FC $\#units = 15$, Linear
softmax(dim=2)	softmax(dim=15)

Tab. 8.18. Architectural details of Seq-MT model used for Hands and Multi-PIE datasets.

Hands Dataset	**Multi-PIE Dataset**	
Model HP: $\omega = 0.5, \tau = 0.3, \gamma = 10^{-5}, \beta = 0.001$, ADAM	Model HP: $\omega = 2, \tau = 0.3, \gamma = 10^{-5}, \beta = 0.001$, ADAM	
Preprocessing: scale and translation [-10%, 10%] of face bounding box, rotation [-20, 20] applied randomly to every epoch.		
Landmark Localization Network	Landmark Localization Network	
Input = $64 \times 64 \times 1$ Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 25$, ReLU, stride 1, SAME soft-argmax(num_channels=25)	Input = $64 \times 64 \times 1$ Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 64$, ReLU, stride 1, SAME Conv $9 \times 9 \times 68$, ReLU, stride 1, SAME soft-argmax(num_channels=68)	
Classification Network	Emotion Classification Branch	Camera Classification Branch
FC $\#units = 256$, ReLU, dropout-prob=.5 FC $\#units = 256$, ReLU, dropout-prob=.5 FC $\#units = 27$, Linear softmax(dim=27)	FC $\#units = 256$, ReLU, dropout-prob=.25 FC $\#units = 256$, ReLU, dropout-prob=.25 FC $\#units = 6$, Linear softmax(dim=6)	FC $\#units = 256$, ReLU, dropout-prob=.25 FC $\#units = 256$, ReLU, dropout-prob=.25 FC $\#units = 5$, Linear softmax(dim=5)

Tab. 8.19. Architectural details of Seq-MT model used for 300W datasets.

300W Dataset
Model HP: $\omega = 2.0, \tau = 2.0, \gamma = 10^{-5}, \beta = 0.01$, ADAM
Preprocessing: scale and translation [-10%, 10%] of face bounding box, rotation [-30, 30] applied randomly to every epoch.
Landmark Localization Network
Input = $64 \times 64 \times 1$ Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 32$, ReLU, stride 1, SAME Conv $9 \times 9 \times 68$, ReLU, stride 1, SAME soft-argmax(num_channels=68)

Chapter 9

Prologue to Fourth Article

9.1. Article Details

Unsupervised Depth Estimation, 3D Face Rotation and Replacement. Joel Ruben Antony Moniz, Christopher Beckham, Simon Rajotte, Sina Honari, Christopher Pal. *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018).*

The idea of estimating depth without supervision by matching source and target keypoints was initially proposed by Christopher Pal and Joel Moniz. Initially I helped Joel by providing my ReCombinator Networks model to extract two dimensional (2D) keypoints. The keypoints were used to train the DepthNet model. While my initial help was limited to this technical help, I then, at Chris Pal's request, took on a more senior leading role in this project. The work lacked comparison with the models in the literature due to its different approach compared to existing models, which made comparisons difficult. I did a literature review on depth estimation papers and proposed a comparison platform with the current models, which provided a baseline for model comparison. I also helped on the design and experimental setup of the applications of DepthNet such as face rotation, background synthesis and face replacement. Most of the coding and running of the experiments was done by Christopher Beckham, Joel Moniz and Simon Rajotte. I had mostly a mentorship role helping Christopher Beckham, Joel Moniz and Simon Rajotte on experiment setup, evaluations, debugging and analysis of the results. I also did most of the writing of the manuscript which got accepted at NeurIPS. Please note that the variable notations in this chapter have been changed compared to the published article version to unify the notations of the repeating concepts in the thesis.

9.2. Context

At the time of working on this research, most of the models on depth estimation required ground truth depth values in their formulations, as in (Eigen et al., 2014; Jackson et al., 2017; Liu et al., 2015). Some other models estimated depth in an unsupervised style, by either requiring left-right images from a stereo camera, as in (Garg et al., 2016; Godard et al., 2017) or using nearby frames of monocular video (Zhou et al., 2017). Since these model require pixel level reconstruction from one image to another, they need very similar frames for this reconstruction, either taken from a stereo camera with images captured at the same time from the same scene with a small disparity, or nearby frames of a video containing small changes from one frame to another. Our proposed approach, on the other hand, maps keypoints on pairs of images and since it does not require pixel reconstruction from one image to another, it can take two faces from different identities or pixel intensities (e.g. different skin colors). Some other frameworks, as in (Atapour-Abarghouei and Breckon, 2018; Tung et al., 2017a), used synthetic data to pre-train their model on ground truth 3D values, and then on real data they use projection of their estimated 3D model onto a 2D representation or leverage domain adaptation for further tuning the model. Adversarial Inverse Graphics Network (Tung et al., 2017b) takes 2D keypoint heatmaps to estimate 3D poses and uses a reconstruction loss from projecting 3D back to 2D. To estimate the depth properly, they pass the 3D data to a discriminator which is trained to distinguish real 3D from generated 3D data. Although they do not use the ground truth depth value corresponding to each training example, they have a set of ground truth depth values that they use to train the discriminator. Therefore all methods in (Atapour-Abarghouei and Breckon, 2018; Tung et al., 2017a,b) use depth values either through synthetic data or a set of ground truth values. MOFA (Tewari et al., 2017) uses an unsupervised encoder-decoder architecture on RGB face images, where the encoder estimates underlying face parameters such as 3D shape, expression, skin reflectance, and camera parameters, and the decoder is a differentiable face renderer that reconstructs the input image from the estimated latent parameters. This model is unsupervised and the parameters estimated by the encoder are not trained using ground truth data. However, due to lack of supervision on the latent parameters and having many free parameters such as skin reflectance, pose, illumination and expression, their parameters are not constrained and, as we also show in our experiments, their model is not capable of adapting well to different

poses. Our approach estimates depth of keypoints without depth supervision by mapping source face keypoints onto a target face keypoints using an affine transformation matrix. We show that the affine parameters can be formulated as a function of keypoints and estimated depth, and unlike MOFA which has many unknown free parameters – and therefore their setting is not well constrained – in our formulation the only unknown parameter is depth. This provides a more constrained setting and therefore the estimated depth is more robust. Another advantage of our model is mapping source to target keypoints instead of pixel intensities. Therefore, the pair passed to the model does not need to belong to the same person.

9.3. Contributions

We present an unsupervised approach for learning to estimate three dimensional (3D) facial structure from a single image while also predicting 3D viewpoint transformations that match a desired pose and facial geometry. We achieve this by inferring the depth of facial keypoints of an input image in an unsupervised manner, without using any form of ground-truth depth information. We show how it is possible to use these depths as intermediate computations within a new backpropable loss to predict the parameters of a 3D affine transformation matrix that maps inferred 3D keypoints of an input face to the corresponding 2D keypoints on a desired target facial geometry or pose. Our resulting approach, called DepthNets, can therefore be used to infer plausible 3D transformations from one face pose to another, allowing faces to be frontalized, transformed into 3D models or even warped to another pose and facial geometry. Lastly, we identify certain shortcomings with our formulation, and explore adversarial image translation techniques as a post-processing step to re-synthesize complete head shots for faces re-targeted to different poses or identities.

9.4. Recent Developments

Due to our work just being published in NeurIPs 2018, further time is required to observe how this line of research evolves from here. An interesting recent work is (Srinivasan et al., 2018), which takes one all-in-focus RGB image and by changing aperture size of the image they estimate depth of the pixels. Another recent work is (Mahjourian et al., 2018), which estimates depth by aligning the

3D geometry using point clouds of adjacent frames. In their approach they consider both geometry of the scene and also pixel consistency between adjacent frames. A depth estimation using images from the Internet is done in (Li and Snavely, 2018), which takes picture of known landmarks (e.g. Colosseum, Big Ben) uploaded by random people all over the web and extracts depth by using overlapping viewpoints through applying structure-from-motion (SfM) and multi-view stereo (MVS) methods. Although their approach takes pairs of corresponding patches during training, they show it can predict reliable depth using random images from the Internet and when applied to other datasets it can estimate decent depth values.

Chapter 10

Unsupervised Depth Estimation, 3D Face Rotation and Replacement

10.1. Introduction

Face rotation is an important task in computer vision. It has been used to frontalize faces for verification (Hassner et al., 2015; Taigman et al., 2014; Yin et al., 2017; Zhao et al., 2018) or to generate faces of arbitrary poses (Shen et al., 2018; Tran et al., 2017). In this paper we present a novel unsupervised learning technique for face rotation and warping from a 2D *source* image – whose facial appearance will be used in the rotation – to a *target* face – to which the facial pose and geometry inferred from the source image is mapped. A use case is when we have an image of someone in a particular target pose and we want to put a given source face into that pose, without knowing the exact target face pose. This can be leveraged, for example, in the advertisement industry, when putting someone in a particular location can be costly or unfeasible, or in the movie industry when the main actor's limited time or high cost can enforce using another actor whose face can be later replaced by the main actor's. This is achieved through estimating the source face depth and the 3D affine parameters that warp the source to the target face using neural networks. These neural networks use a novel loss formulation for the structured prediction of keypoint depths. Once the 3D affine transformation matrix is estimated, it can be used to warp the source image onto the target face geometry using a textured triangular mesh. The use of a 3D affine transform means that we can capture both a 3D rotation of the face to a new viewpoint as well as a global non-Euclidean warping of the geometry to match a target face. We call these neural networks Depth Estimation-Pose Transformation Hybrid Networks, or DepthNets in short.

Our first contribution is to propose a neural architecture that predicts both the depth of source keypoints as well as the parameters of a 3D geometric affine transformation which constitute the explicit outputs of the DepthNet model. The predicted depth and affine transformation could be then used to map a source face to a target face for object orientation, distortion and viewpoint changes.

Our second contribution consists of making the observation that given 3D source and 2D target keypoints, closed form least squares solutions exist for estimating geometric affine transformation models between these sets of keypoint correspondences, and we can therefore develop a model that captures the dependency between depth and the affine transformation parameters. More specifically, we express the affine transformation as a function of the pseudoinverse transformation of 2D keypoints in a source image – augmented by inferred depths – and the target keypoints. Thus, the second and major contribution in this work is capturing the relationship between an estimated affine transformation and the inferred depth as a deterministic relationship. In this formulation, DepthNet only predicts depth values explicitly and the affine parameters are inferred through a pseudoinverse transformation of source and target keypoints. Here, one can directly optimize through the solutions of what might otherwise be formulated as a secondary minimization step.

Our proposed DepthNet can map the central region of the source face to the target geometry. This leads to background mismatch when warping one face to another. Finally, *our third contribution* is to use an adversarial unpaired image-to-image transformation approach to repair the appearance of 3D models inferred from DepthNet. Together these contributions allow 3D models of faces that construct realistic images in the target pose. Our proposed method can be used for pose normalization or face swaps with no manually specified 3D face model. To the best of our knowledge, this is the first such neural network based model that estimates a 3D affine transformation model for face rotation which neither requires ground-truth 3D images nor any ground truth 3D face information such as depth.

10.2. Related Work

In this section, we review the related works on 3D face models, generative adversarial models on face rotation, and depth estimation models.

10.2.1. 3D Transformation on Faces

While there is a large body of literature on 3D facial analysis, many standard techniques are not applicable to our setting here. As an example, morphable models (Blanz and Vetter, 1999) cover a wide variety of approaches which are capable of high quality 3D reconstructions, but such methods usually require 3D face scans or reconstructions from multi-view stereo to be assembled so as to learn complex parametric distributions over face shapes. A close approach to our own is that of (Hassner, 2013) on viewing real world faces in 3D. Similar to our work, this approach does not require aligned 3D face scans, highly engineered models or manual interventions. They make the observation that if 2D keypoints can be obtained from a single input image of a face and these keypoints are matched to an arbitrary 3D target geometry, then standard camera calibration techniques can be used to estimate plausible intrinsics and extrinsics of the camera. This allows the estimated camera matrix, 3D rotation matrix and 3D translation vector to be used to transform the target 3D model to the pose of the query image from which an approximate depth can be obtained. Hassner et. al (Hassner et al., 2015) explore the use of a single unmodified 3D surface as an approximation to the shape of all input faces. In contrast, our approach only requires 2D keypoints from the source and target faces as input. It then estimates the depth of the source face keypoints, thereby inferring an image specific 3D model of the face.

DeepFace (Taigman et al., 2014) uses face frontalization to improve the performance of a face verification system. It uses a 3D mask composed of facial keypoints, detects the corresponding locations of these keypoints in the image, and maps the 2D keypoints onto a 3D face model to frontalize it. DeepFace, however, maps to a template 3D face, therefore always mapping to a specific pose and geometry. DepthNet, on the other hand, can map to any pose and geometry, giving it more expressive flexibility.

10.2.2. Generative Adversarial Networks on Face Rotation

Recently, adversarial models in (Huang et al., 2017; Shen et al., 2018; Tran et al., 2017; Yin et al., 2017; Zhao et al., 2017, 2018) have explored face rotation. TP-GAN (Huang et al., 2017) performs face frontalization through introducing several losses to preserve identity and symmetry

of the frontalized faces. PIM (Zhao et al., 2018) frontalizes faces in a composed adversarial loss and then extracts pose invariant features for face recognition. These models are mainly aimed for face verification, where they can only do face-frontalization. Another limitation of these models is in requiring ground truth frontal images of the same identity during training. DR-GAN (Tran et al., 2017) rotates faces to any target pose by using a discriminator that also does identity classification in addition to pose prediction, to preserve id and pose. While these models do pure face rotation of a 2D face, our model can warp the input face to any other target face, allowing warping the input face to any other identity, with a different geometry and pose. Moreover, our model also estimates the 3D geometric affine transformation parameters explicitly, allowing these parameters to be used later, e.g., for face texture swap.

FF-GAN (Yin et al., 2017), DA-GAN (Zhao et al., 2017), and FaceID-GAN (Shen et al., 2018) estimate parameters of either a 3D Morphable Model (3DMM), as in (Shen et al., 2018; Yin et al., 2017), or source to target pose transformation, as in (Zhao et al., 2017). FF-GAN uses 3DMM parameters to frontalize faces in an adversarial approach, while FaceID-GAN uses the 3DMM parameters to generate any target pose. These models, however, train 3DMM on ground truth labels such as identity, expression and pose. DepthNet, on the other hand, estimates depth and affine transformation parameters without requiring ground truth affine or depth labels or pre-training. Similar to DepthNet, DA-GAN (Zhao et al., 2017) estimates parameters of an affine transformation model that maps a 2D face to a 3D face. Unlike DepthNet that estimates depth on the source face, DA-GAN uses depth in a template target face. While their approach eliminates the need for depth estimation, it only allows the source face to be mapped to the target template geometry, while DepthNet can map the source face to any target geometry, provided by a target image, or its keypoints. We demonstrate the application of this flexibility for the face replacement task.

The aforementioned adversarial models use an identity preserving loss to maintain identity. The core DepthNet model does not need identity labels and preserves well the identity (as shown in Figure 10.1 (right)). However, the identity information can be used by the proposed adversarial components, as in background synthesis, to further improve the results. Unlike some of these models that take target pose as input, DepthNet uses the target keypoints to estimate the target geometry and does not require the target pose. This has several advantages; 1) DepthNet can map to

the geometry of the target face in addition to the pose, and 2) in the face replacement task, DepthNet can replace the target face with the warped source face directly onto the target face location. Its application is shown in the face swap experiment in Section 10.4.3.

10.2.3. Depth Estimation

Thewlis et. al (Thewlis et al., 2017) propose a mapping technique to learn a proxy of 2D landmarks in an unsupervised way. A semi-supervised technique has been also proposed in (Honari et al., 2018) that improves landmark localization by using weaker class labels (e.g. emotion or pose) and also by making the model predict equivariant variations of landmarks when such transformations are applied to the image. Similar to these approaches, DepthNet also maps a source to a target to learn its parameters. However, unlike these two approaches that estimate 2D landmarks, DepthNet estimates the depth of the landmarks using 2D matching of keypoints, by formulating affine parameters as a function of depth augmentated keypoints in a closed form solution.

While several models (Eigen et al., 2014; Jackson et al., 2017; Liu et al., 2015) estimate depth with direct supervision, there have been recent models (Garg et al., 2016; Godard et al., 2017; Zhou et al., 2017) that estimate depth in an unsupervised training procedure. These models rely on pixel reconstruction by using frames that are captured from very similar scenes, e.g. nearby frames of a video (Zhou et al., 2017) or left-right frames captures by stereo cameras (Garg et al., 2016; Godard et al., 2017). These models estimate depth on one frame and then by using the disparity map, measure how pixel values of nearby frames compare to each other. To do this, they also require camera intrinsic parameters, e.g. focal length or distance between cameras. Unlike these models, our approach does not require source to target pixel mapping. This allows mapping faces from different people with completely different skin colors, without knowing camera parameters or how they are positioned with respect to each other. Therefore, DepthNet is not susceptible to variations in illuminations or lighting between source and target faces.

(Tung et al., 2017a) estimates 3D human pose in videos, where it uses synthetic data to pre-train internal parameters of the model and fine-tunes them by keypoint, segmentation and motion loss. Adversarial Inverse Graphics Networks (AIGN) (Tung et al., 2017b) estimates 3D human pose from 2D keypoint heatmaps in a semi-supervised manner with a similar formulation to that of CycleGAN.

It applies an adversarial loss on the 3D pose to make them look realistic. These models leverage the depth values either through synthetic data (Tung et al., 2017a), or by adversarial usage of ground truth depth values (Tung et al., 2017b). Unlike these models, DepthNet does not rely on any depth signal, either directly or indirectly. MOFA (Tewari et al., 2017) builds a 3D face mesh using a single image, where the 3D face parameters such as 3D shape and skin reflectance are estimated by an encoder and then using a differentiable model they are rendered back to the image by the decoder. This model requires manual initialization to map the input image to the 3D mesh, since otherwise it is doing an unconstrained optimization by adapting both the face pose and the skin reflectance. Our model, however, does not require any manual initialization.

10.3. Our Approach

As we have outlined above, our approach uses neural networks for inferring depth and geometric transformation – referred to as DepthNets; and, an adversarial image-to-image transformation network which improves the quality of the appearance of a 3D model inferred from a DepthNet.

DepthNets

We propose three DepthNet formulations, described in Sections 10.3.1, 10.3.2, and 10.3.3. For each of the three models we explore two architectural scenarios: (A) a Siamese-like architecture that uses the source and target images themselves as well as keypoints extracted from these images, and (B) a fully-connected neural network variant which uses only facial keypoints in the source and target images. See Figure 10.1 (left) for details.

It is interesting to note that if DepthNets are used to register a set of images of objects to the same common viewpoint, the same image and geometry can be used as the target. This is the case for the frontalization of faces, for example. While the DepthNet framework is sufficiently general to be applied to any object type where 2D keypoint detections have been made, our experiments here focus on faces. We describe the three variants of DepthNets below.

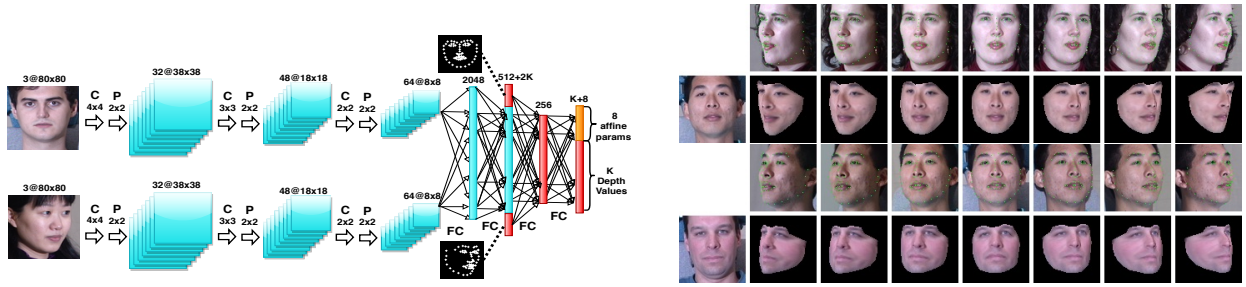


Fig. 10.1. (Left) DepthNet architecture. The blue region is only used in case (A) and the red part is used in both cases (A) and (B), described in Section 10.3. The orange output (the 8 affine transformation parameters) is predicted only by model variations described in Sections 10.3.1 and 10.3.2, and not the model described in section 10.3.3. All three models predict the K depth values of the source keypoints. C, P, and FC correspond to valid conv, pool and fully-connected layers. The two paths of Siamese network share parameters and the black dots indicate concatenating keypoint values to FC units. (Right) Visualizing face rotation by re-projecting a frontal face (far left) to a range of other poses defined by the faces in the row above (in each pair of rows). In this experiment, we only use keypoints from the top-row in the DepthNet model (Model 7 in Table 10.1).

10.3.1. Predicting Depth and Viewpoint Separately

In this variant of DepthNets, the model predicts both depths and viewpoint geometry, but as separate *explicit* outputs of a neural network. The input is comprised of only the geometry and pose of the source and target faces (encoded in the form of a 2D keypoint template), in case (B), or both keypoints and images of the source and target faces, in case (A). The key phases of this stage are described by the sequence of steps given below:

(1) *Keypoint extraction*: Raw width w and height h values for each keypoint in each image, are extracted using a Recombinator Network (RCN) (Honari et al., 2016) architecture, and then concatenated before being passed into the keypoint processing step.

(2) *(Optional) Image Feature Extraction*: DepthNets can be conditioned on only keypoints, case (B), or on keypoints and the original images, case (A). We can therefore optionally subject the source and target images to alternating conv-maxpool layers. If this component of the architecture is used, the last spatial feature maps in the Siamese architecture are concatenated before being given to a set of densely connected hidden layers.

(3) *Keypoint processing*: In this step keypoints are passed through a set of hidden layers. If the *Image Feature Extraction* stage is used, the keypoints are concatenated to image features, the output

of which is in turn fed to densely connected layers. The output layer of this phase will be of size $K + 8$, where K is the number of keypoints. The first K points represent the depth proxy, and the last 8 points form a 4×2 matrix representing the learned parameters of the affine transform. See Figure 10.1.

(4) *Geometric Affine Transformation Normalizer*: This phase applies the predicted affine transform on each (depth augmented) source keypoint to estimate its target location. Let (w_s^k, h_s^k) represent the k$^{\text{th}}$ source keypoint, (w_e^k, h_e^k) the corresponding estimated keypoint by applying the affine transformation matrix, (w_t^k, h_t^k) the k$^{\text{th}}$ target keypoint (as ground truth (GT)), and $\mathbf{x}_s$ and $\mathbf{x}_t$ represent the source and target images respectively. Depending on which underlying architectural variant we use, two cases arise: one that utilizes only the keypoints (B), and another utilizing both the keypoints and the images (A). Since the keypoints are generated using RCNs, they are technically functions of the input images: $[\mathbf{w}_s, \mathbf{h}_s] = L(\mathbf{x}_s)$, and $[\mathbf{w}_t, \mathbf{h}_t] = L(\mathbf{x}_t)$. Depending on the (A) or (B) variant, the k$^{\text{th}}$ source keypoint's estimated depth proxy d_s^k is inferred as a function of the input keypoints, or both input keypoints and input images. In both cases the keypoints are derived from the images, so $d_s^k = d_s^k(\mathbf{x}_s, \mathbf{x}_t)$. Similarly, the 3D-2D affine transform $\mathbf{F}$ is a function of the images, such that $\mathbf{F} = F(\mathbf{x}_s, \mathbf{x}_t)$, where the 8 predicted parameters are: $\mathbf{F} = \{m_1, m_2, m_3, tr_w, m_4, m_5, m_6, tr_h\}$. These constitute the 3D-2D affine transform which is used by all keypoints. In other words, each of the k points is transformed using $\mathbf{y}_e^k = F(\mathbf{x}_s, \mathbf{x}_t)\ \mathbf{y}_s^k$, or:

$$\begin{bmatrix} w_e^k \\ h_e^k \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 & tr_w \\ m_4 & m_5 & m_6 & tr_h \end{bmatrix} \begin{bmatrix} w_s^k \\ h_s^k \\ d_s^k(\mathbf{x}_s, \mathbf{x}_t) \\ 1 \end{bmatrix}$$

The loss function of a DepthNet is obtained by transforming the source face to match the target face using the simple squared error of the corresponding target object's keypoint vector $\mathbf{y}_t = [\mathbf{w}_t, \mathbf{h}_t]^T$, as GT values, and the estimated keypoint vector $\mathbf{y}_e = [\mathbf{w}_e, \mathbf{h}_e]^T$. The loss for one example where we predict depth and affine viewpoint geometry can therefore be expressed as:

$$\mathcal{L} = \sum_{k=1}^{K} \left\| \mathbf{y}_t^k - F(\mathbf{x}_s, \mathbf{x}_t)\ [w_s^k\ \ h_s^k\ \ d_s^k(\mathbf{x}_s, \mathbf{x}_t)]^T \right\|^2 \tag{10.3.1}$$

(5) *Image Warper*: This phase consists of using the depth proxy and affine transform matrix generated to actually warp the face from its source pose to be matched to the target object geometry. The final projection to 2D is achieved by simply dropping the transformed d coordinate (which corresponds to an orthographic projection model). In the case of DepthNets, this orthographic projection is effectively embedded in the *Geometric Affine Transformation Normalizer* step, since the affine corresponding to the d coordinate is not predicted, essentially dropping it.

As we operate on keypoints, the actual warping of pixels can be performed with a high quality OpenGL pipeline that performs the warp separately from the rest of the architecture. Source image, keypoints augmented with depth, and the affine matrix are passed to OpenGL pipeline to warp the source image towards the target pose. This OpenGL warping is not needed during DepthNet training, which means we do not have to do feedforward or backprop through OpenGL. In Summary, for step 1 the RCN model (Honari et al., 2016) is used, for steps 2 to 4 the DepthNet model, shown in Figure 10.1 (left), is trained, and for step 5 an OpenGL pipeline is used. No data or parameters are needed to train the OpenGL pipeline. It warps images by directly using the provided data.

10.3.2. Estimating Viewpoint Geometry as a Second Step

In this model variant, training is similar to Section 10.3.1 and the model outputs depth and 3D affine transformation parameters. However, at test time, rather than using the predicted 3D affine transformation for pairs of faces, we use only the predicted depths and estimate the affine geometry parameters as a second estimation step. More precisely, given 3D points for a scene and the corresponding 2D points for a target geometry it is possible to formulate the estimation of a 3D affine transformation as a linear least squares estimation problem. An overdetermined system of the

form $\mathbf{AF} = \mathbf{y}_t$ for this problem can be constructed as shown in (10.3.2).

$$\begin{bmatrix} w_s^1 & h_s^1 & d_s^1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & w_s^1 & h_s^1 & d_s^1 & 0 & 1 \\ w_s^2 & h_s^2 & d_s^2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & w_s^2 & h_s^2 & d_s^2 & 0 & 1 \\ & & & & \vdots & & & \\ w_s^K & h_s^K & d_s^K & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & w_s^K & h_s^K & d_s^K & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ tr_w \\ tr_h \end{bmatrix} = \begin{bmatrix} w_t^1 \\ h_t^1 \\ w_t^2 \\ h_t^2 \\ \vdots \\ w_t^K \\ h_t^K \end{bmatrix} \tag{10.3.2}$$

This corresponds to an affine camera model followed by an orthographic projection to 2D keypoints. This setup also leads to the following closed form solution for the affine transformation parameters:

$$\mathbf{F} = [\mathbf{A}^T\mathbf{A}]^{-1}\mathbf{A}^T\mathbf{y}_t, \tag{10.3.3}$$

where this pseudoinverse based transformation is parameterized by the reference points and their predicted depths.

10.3.3. Joint Viewpoint and Depth Prediction

Our *key observation* is that one can alternatively use the closed form analytical solution, measured in Eq. (10.3.3), for the least squares estimation problem as the underlying affine transformation matrix within the loss function. This leads to a special form of structured prediction problem for geometrically consistent depths and affine transformation matrix. For each image we have $\mathcal{L} =$

$$\sum_{k=1}^{K} \left\| \underbrace{\begin{bmatrix} w_t^k \\ h_t^k \end{bmatrix}}_{\mathbf{y}_t^k} - \underbrace{\begin{bmatrix} m_1 & m_2 & m_3 & tr_w \\ m_4 & m_5 & m_6 & tr_h \end{bmatrix}}_{\mathbf{F}} \underbrace{\begin{bmatrix} w_s^k \\ h_s^k \\ d_s^k(\mathbf{x}_s, \mathbf{x}_t) \\ 1 \end{bmatrix}}_{\mathbf{y}_s^k} \right\|^2 = \sum_{k=1}^{K} \left\| \mathbf{y}_t^k - \text{reshape}\big[[\mathbf{A}^T\mathbf{A}]^{-1}\mathbf{A}^T\mathbf{y}_t\big]\mathbf{y}_s^k \right\|^2$$

where the matrix $\boldsymbol{A}$ is parameterized as a function of $\mathbf{y}_s$ as shown in Eq. (10.3.2). In this variant, the model *explicitly* outputs only depth values during training and test time. The affine transformation matrix in the equation above is replaced by Eq. (10.3.3), which measures the affine transformation

as a pure function of source and target keypoints plus the inferred depth. The big difference of this formulation compared to Sections 10.3.1 and 10.3.2 is that geometric affine transformation parameters are no longer predicted by DepthNet during training and at both training and test time – it *solves* the least square loss through the pseudoinverse based transformation. Since $d_s^k = d_s^k(\{w_s^j, h_s^j, w_t^j, h_t^j\}_{j=1...K})$ is predicted within the analytical formulation of the solution to the least squares minimization problem, we can backpropagate through the *solution* of a minimization problem that depends on the predicted depths. While we leverage keypoints for depth estimation, the proposed approach is novel in how the depth is estimated. Note that it is unsupervised with respect to depth labels. No depth supervision either by using depth targets (as in (Eigen et al., 2014; Jackson et al., 2017; Liu et al., 2015)), or by using depth in an adversarial setting (as in (Tung et al., 2017b)), is used to estimate depth values for the base DepthNet models described in Sections 10.3.1, 10.3.2, and 10.3.3.

The depths learned for keypoints by these approaches are not necessarily true depths, but are likely to strongly correlate with the actual depth of each keypoint. This is because even though the method succeeds (as we shall see below) in aligning poses, the inferred depth and the affine transform may each be scaled by factors so as to cancel each other out (i.e., by factors which are multiplicative inverses of each other). Real world viewpoint geometry also involves perspective projection.

10.3.4. Adversarial Image-to-Image Transformation

DepthNet transforms the central region of the source face to the target pose. Inevitably, the face background will be missing, which might make the proposed method unsuitable for many application where the full face is required. To address this issue, we utilize CycleGAN (Zhu et al., 2017), an adversarial image-to-image translation technique. This serves to repair the background of faces that have undergone frontalization or face swap through the DepthNet pipeline. Importantly, the adversarial nature of CycleGAN allows one to perform image transformation between two domains without the requirement of paired data. In our work, we perform experiments translating between various domains of interest but one example is translating between the domain of images in the dataset (i.e. the ground truth) and the domain of images where the DepthNet output is pasted

onto the face region (in the case of face-swap). By doing so we clean the face background in an unsupervised manner. In the following sub-section we review the CycleGAN model and how it is leveraged for our tasks.

10.3.4.1. *CycleGAN*

Suppose we have some images belonging to one of two sets $\mathbf{v} \in \mathbf{V}$ and $\mathbf{z} \in \mathbf{Z}$, where $\mathbf{v}$ denotes a DepthNet-resulting face and $\mathbf{z}$ a ground truth face which is frontal. We wish to learn two functions $G_Z : \mathbf{V} \rightarrow \mathbf{Z}$ and $G_V : \mathbf{Z} \rightarrow \mathbf{V}$ which are able to map an image from one set to the corresponding image in the other. Correspondingly, we have two discriminators D_V and D_Z which try to detect whether the image in that particular set is real or generated. While we are only interested in the function $G_Z : \mathbf{V} \rightarrow \mathbf{Z}$ (since this is mapping to the distribution of ground truth faces) the formulation of CycleGAN requires that we learn mappings in both directions during training. We optimize the following objectives for the two generators G_Z and G_V:

$$\min_{G_V, G_Z} \mathbb{E}_{\mathbf{v},\mathbf{z}}\Big[\ell(D_V(G_V(\mathbf{z})),1)+\ell(D_Z(G_Z(\mathbf{v})),1)+\psi||\mathbf{z}-G_Z(G_V(\mathbf{z}))||_1+\psi||\mathbf{v}-G_V(G_Z(\mathbf{v}))||_1\Big] \tag{10.3.4}$$

And the following for the two discriminators D_V and D_Z:

$$\min_{D_V, D_Z} \mathbb{E}_{\mathbf{v},\mathbf{z}}\Big[\ell(D_V(\mathbf{v}),1) \ + \ \ell(D_V(G_V(\mathbf{z})),0) \ + \ \ell(D_Z(\mathbf{z}),1) \ + \ \ell(D_Z(G_Z(\mathbf{v})),0)\Big], \tag{10.3.5}$$

where 0/1 denote fake/real, $\ell()$ is the squared error loss and ψ is a coefficient for the cycle-consistency (reconstruction) loss.

In the case where we do adversarial background synthesis, $\mathbf{v}$ is a channel-wise concatenation of the DepthNet-frontalized face and the background of the original (pre-frontalized) image. For face-swap cleanup, $\mathbf{v}$ is simply a source face which has been warped to a target face and pasted on top. Once the network has been trained, we can disregard all other functions and use G_Z to clean up faces which are low quality due to artifacts from warping.

In terms of architectural details the generators and discriminators used were those described in the appendix of the CycleGAN paper (Zhu et al., 2017). In short, the generator consists of three

`conv-BN-relu` blocks which downsample the input, followed by nine ResNet blocks (which can be interpreted as iteratively performing transformations over the downsampled representation), followed by `deconv-BN-relu` blocks to upsample the representation back into the original input size. For training, we use the same hyperparameters as most CycleGAN implementations which is using the Adam optimizer with learning rate $\alpha = 2 \times 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$. However, instead of using a batch size of 1 we use the largest possible batch size, which was 16 for a 12GB GPU.

Note that in order to produce better translations, the dataset we used for all CycleGAN experiments contain both the VGG and the CelebA datasets, which has significantly more images.

10.4. Experiments

In this section we evaluate the DepthNet model on both paired images, in Section 10.4.1, and unpaired images, in Section 10.4.2. Later, in Section 10.4.3, we present applications of the DepthNet in face rotation and face replacement tasks.

The RCN, the DepthNets and the CycleGAN modules are trained separately. Each model is trained using standard techniques for the model class and has a separate objective to be optimized. DepthNet does not use the OpenGL pipeline during training and only uses it to render faces at test time, allowing DepthNet to train faster.

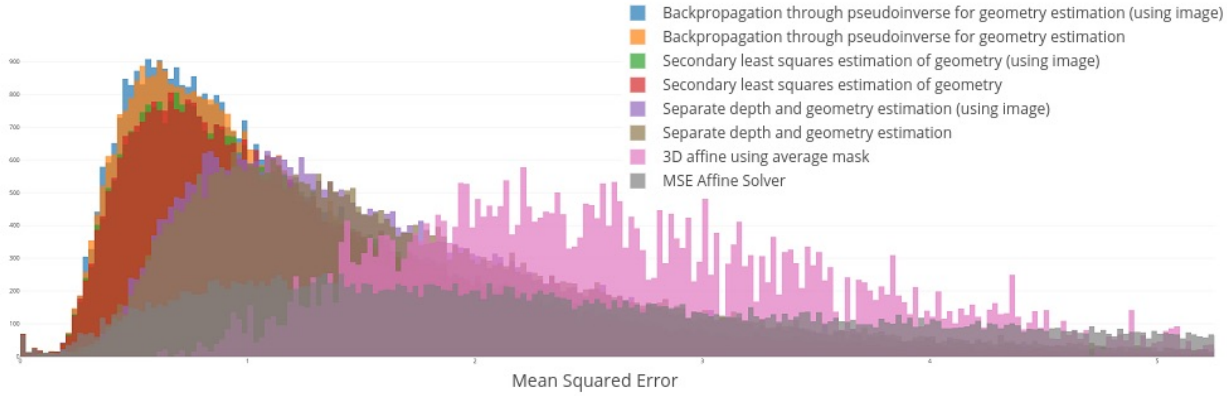
10.4.1. DepthNet Evaluation on Paired Faces

For the experiments in this section, we use a subset of the VGG dataset (Parkhi et al., 2015), with training and validating on all possible pairs of images belonging to the same identity for 2401 identities. This yields 322,227 training and 43,940 validation pairs.

We explore the three variants of DepthNets described in Sections 10.3.1, 10.3.2, and 10.3.3, each with two architectural cases (A) and (B), depending on whether image features are used in addition to keypoints or not. We also compare with a number of baselines. We measure the mean square error (MSE) between the estimated keypoints on the target face (source face normalized keypoints) and ground truth target keypoints. Results for the following models are shown in Table 10.1:

1) A baseline model registrations using a simple 2D affine transformation.

Model	Color	MSE	MSE_norm
1) A simple 2D affine registration	grey	1.562	9.547
2) A 3D affine registration model using an average 3D face template	purple	0.724	7.486
3) A DepthNet that separately estimates depth and geometry	brown	0.568	6.292
4) The model above, but with a Siamese CNN image model	violet	0.539	6.115
5) Secondary least squares estimation for visual geometry using the depths from 3)	red	0.400	5.184
6) Secondary least squares estimation for visual geometry using the depths from 4)	green	0.399	5.175
7) Backpropagation through the pseudoinverse based solution for visual geometry	orange	0.357	4.932
8) The model above, but with a Siamese CNN image model	blue	**0.349**	**4.891**



Tab. 10.1. (top) Comparing the Mean Squared Error (MSE) and MSE normalized by inter-ocular distance (MSE_norm) of different models. (bottom) Histogram of Mean Squared Errors. The second column in the Table (on top) corresponds to the color of the model in the figure (on bottom).

2) We generate a 3D average face template from the 3DFAW dataset (Gross et al., 2010; Jeni et al., 2015; Zhang et al., 2014b) by aligning the 3D keypoints of all faces in the dataset to a front-facing face using Procrustes superimposition. We report error by mapping the template face to each source face via Procrustes superimposition (to get a 3D face f) and then use an affine transformation from the 3D face f to the target face.

3, 4) We use our proposed approach to predict both depth and geometry (described in Sections 10.3.1).

5, 6) These models described in Section 10.3.2. Note that during training, these two cases are similar to models 3 and 4 in Table 10.1.

7, 8) The pseudo-inverse formulation model described in Section 10.3.3.

As observed in Table 10.1, a simple 2D affine transform (model 1) without estimating depth and a template 3D face (model 2) get high errors on mapping to the target faces. DepthNet models

get lower errors and the pseudo-inverse formulation (models 7 and 8) further reduces the error by 10%. The CNN models slightly reduce errors compared to their equivalent models that rely only on keypoints.

Our DepthNet architectures require keypoints of both source and target images to be extracted. For this, the image is first passed through the VGG-Face (Parkhi et al., 2015) face detector. The face crops are then scaled down to 80×80 and converted to greyscale, following which they are passed through RCN to obtain $K = 68$ keypoints on each image. The RCN is trained exactly as described in (Honari et al., 2016), using the 300W dataset (Sagonas et al., 2013).

The keypoint only variant of our model involves concatenating all detected keypoints and passing them through a two-layer deep fully connected network, with 256 hidden units and o output units. The size of o depends on whether we are predicting only the depth, in which case $o = K$, or both depth and affine transformation parameters, in which case $o = K + 8$.

As discussed above, it is possible to augment these models with a Siamese CNN module (case A). In the model variants that also use the image, we pass both the source and the target images through three conv-maxpool layers with shared weights of size (32, 4, 2), (48, 3, 2), (64, 2, 2), respectively for the representation (`num_filters`, `filter_size`, `pool_size`). The network's outputs for the source and target faces are then concatenated before passing them into a 4-layered fully connected network with respective output sizes of 2048, 512, 256, and o. The keypoints are concatenated to the 512-unit layer before being passed to the last two layers. See Figure 10.1 (left) for an illustration of the model. We explore these Siamese CNN augmented variants in models 4, 6 and 8 in Table 10.1.

We set the initial learning rate to 0.001 and use a Nesterov momentum optimizer (with a momentum of 0.9) in all our experiments. With the exception of the last layer, we initialize all weights with a Glorot initialization scheme (Glorot and Bengio, 2010), with the weights sampled from a uniform distribution. We use a ReLU gain (He et al., 2015), set all biases to 0, and apply a ReLU non-linearity after every layer. In the final output layer, we do not apply any non-linearity and initialize the weights to 0. The biases of units that represent depths are initialized to a random Normal distribution with $\mu = 0$ and $\sigma = 0.5$, while those that form the predicted affine transform

are initialized with the equivalent of a "flattened" identity transform. All models have been trained for 500 epochs.

We point out that except for a comparison between learning rates in the set {0.01, 0.001, 0.0001} over few (less than 10) epochs, to find a learning rate that the model seems to train well with, we have not performed a hyperparameter search, and anticipate that the performance of the model can be made *even* better by searching the hyperparameter space on a per model basis and by using deeper (or modified) architectures.

10.4.2. DepthNet Evaluation on Unpaired Faces and Comparison to other Models

In this section we train DepthNet on unpaired faces belonging to different identities and compare with other models that estimate depth. We use the 3DFAW dataset (Gross et al., 2010; Jeni et al., 2015; Zhang et al., 2014b) that contains 66 3D keypoints to facilitate comparing with ground truth (GT) depth. It provides 13,671 training and 4,500 valid images. We extract from the valid set, 75 frontal, left and right looking faces yielding a total of 225 test images, which provides a total of 50,400 source and target pairs. We train the psuedoinverse DepthNet model that relies on only keypoints (model 7 in Table 10.1). We also train a variant of DepthNet that applies an adversarial loss on the depth values (DepthNet+GAN). This model uses a conditional discriminator that is conditioned on 2D keypoints and discriminates GT from estimated depth values. The model is trained with both keypoint and adversarial losses.

Source \ Target	**DepthNet**				**DepthNet + GAN**			
	Left	Front	Right	Avg	Left	Front	Right	Avg
Left	24.67	27.71	29.70	27.36	59.78	59.67	59.63	**59.69**
Front	25.54	27.22	26.19	26.32	58.77	58.67	58.61	**58.68**
Right	21.66	21.48	23.87	22.34	59.97	59.70	59.60	**59.76**

Tab. 10.2. Comparing DepthCorr for different DepthNet models when mapping variant source to target poses. The Avg column measures the average over the three preceding columns.

We measure the correlation matrix between GT and estimated depths, where the element k in the diagonal indicates the correlation between estimated and ground truth depth values for keypoint k, yielding a value between -1 and 1. We report the sum of absolute values of the diagonal of this

matrix, indicated by DepthCorr. We compare DepthNet models on DepthCorr in Table 10.2. For this experiment we take every possible pair of source to target faces, where source and target are one of {left, front, right} looking faces. This yields a total of 5,550 pairs when the source and the target are from the same subset, and 5,625 pairs otherwise. This experiment measures the accuracy of depth estimation of the DepthNet models on different orientations of source-target faces. The baseline DepthNet model that does not leverage the depth labels performs well in different cases. DepthCorr improves more than twice for the DepthNet+GAN model, indicating a direct supervision loss using depth labels can enhance the depth estimation.

Model	**Need Depth**	**Manual Init.**	**MSE** ($\times 10^{-5}$)	**Depth Correlation Matrix Trace (DepthCorr)**		
				Left pose	**Front pose**	**Right pose**
GT Depth	Yes	-	8.86 ± 6.55	**66**	**66**	**66**
AIGN (Tung et al., 2017b)	Yes	No	9.06 ± 6.61	44.08	50.81	49.04
MOFA (Tewari et al., 2017)	No	Yes	8.75 ± 6.33	11.14	15.97	17.54
DepthNet (Ours)	No	No	$\mathbf{7.65 \pm 6.97}$	27.36	26.32	22.34
DepthNet + GAN (Ours)	Yes	No	8.74 ± 6.24	59.69	58.68	59.76

Tab. 10.3. Comparing MSE and DepthCorr for different models. A lower MSE indicates the model maps better to the target faces. A higher DepthCorr indicates more correlation between estimated and GT depths. Note that in the last three columns (related to DepthCorr), the left pose column corresponds to the case when the source face has a left pose (the same for the front and right pose columns), so the numbers for DepthNet and DepthNet+GAN correspond to the avg column in Table 10.2.

We compare our two DepthNet models with three baselines: 1) AIGN (Tung et al., 2017b), 2) MOFA (Tewari et al., 2017) and 3) GT Depth (no model trained). AIGN estimates 3D keypoints conditioned on 2D heatmaps of the keypoints. MOFA estimates a 3D mesh using only an image. We implemented the AIGN model and asked the authors of MOFA to run their model on our test-set. They provided MOFA's results for 134 images in the test set. In Table 10.3 we compare these three models with our DepthNet models on DepthCorr. We also compare them on MSE, which is measured between GT and estimated target keypoints. Since the three baselines estimate depth on a single image due to their different model formulation, we first measure $\mathbf{F}$ using closed form solution in Eq. 10.3.3 and then apply $\mathbf{F}$ to the estimated source keypoints to get the target keypoint estimations. We contrast the estimated values with the GT target keypoints. As shown in Table 10.3, GT depth has the highest DepthCorr (the maximum possible value). The depths estimated by DepthNet+GAN and AIGN have stronger correlation to GT depth compared to the baseline

DepthNet and MOFA, while baseline DepthNet performs better than MOFA. On MSE the baseline DepthNet model gets smaller MSE when mapping to target faces, indicating it is better suited for this task.

In Figure 10.2 we plot heatmaps of the estimated depth of different models (on vertical axis) and the GT depth (on horizontal axis) aggregated over all 66 keypoints on all test data. As can be seen, the depth estimated by DepthNet+GAN and AIGN models form a 45 degree rotated ellipses showing a stronger linear correspondence with respect to the GT depth compared to the the baseline DepthNet and MOFA.

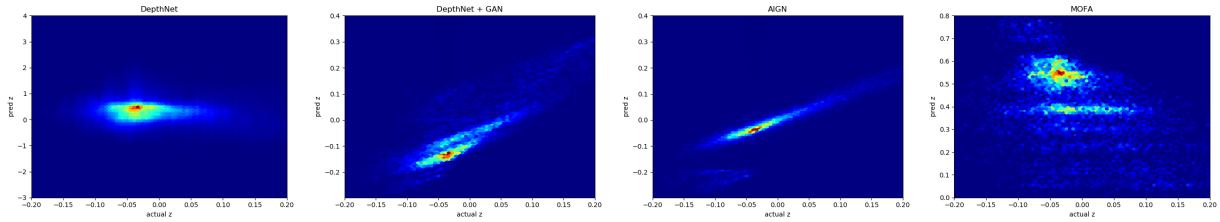


Fig. 10.2. Predicted (vertical axis) versus Ground Truth (horizontal axis) depth heatmaps for different models.

In Figure 10.3 we show some estimated depth samples for different models. AIGN and DepthNet+GAN generate more realistic results. MOFA generates very similar face templates for different poses. We believe this is due to the unconstrained reconstruction that is done by this model, since it reconstructs the input image by mapping both face pose and the texture color values to the input image. The depth estimated by a model relates to how the face is visualized. For example, for a frontal face, nose keypoints have the smallest depth (closest to the viewer) and the face contour keypoints close to the eye have the greatest depth (furthest from the viewer). Now, for a profile face looking towards left, the keypoints on the right side of the face contour have the smallest depth and the keypoints on the left side of the face contour have the furthest depth. So, the depth of keypoints affects the way the face is visualized. This explains why the depth estimated by the MOFA model makes the faces look frontal. Baseline DepthNet estimates reliable depth values in most cases, however it has some failure modes as shown in the last row.

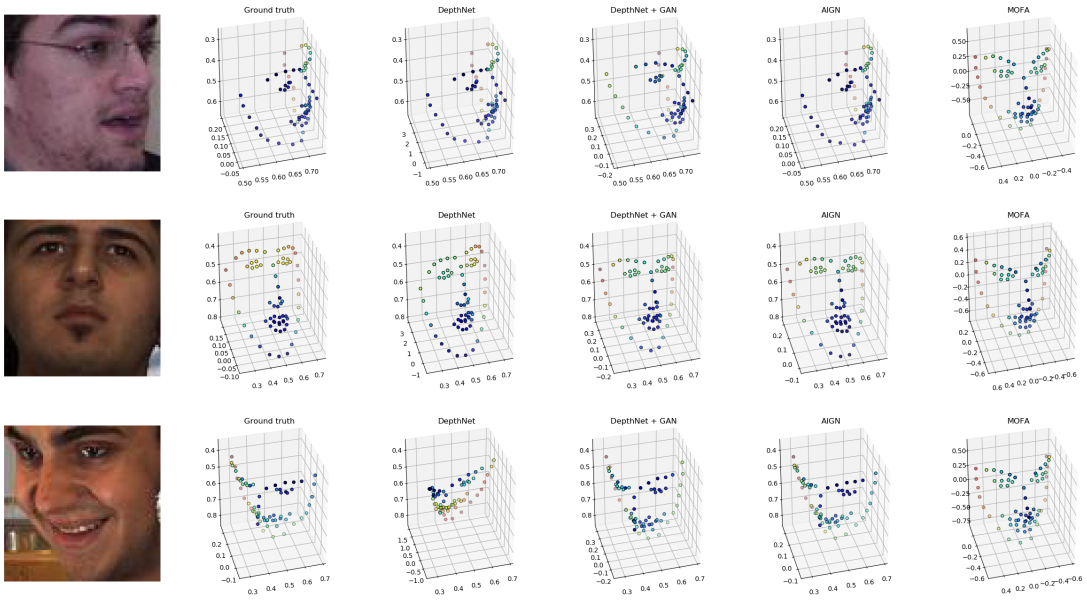


Fig. 10.3. Depth visualization for different models (color coded by depth). From left to right: RGB image, Ground Truth, DepthNet, DepthNet+GAN, AIGN and MOFA estimated depth values.

We show further estimated depth values in Figure 10.4. The baseline DepthNet model estimates reliable depth values for most cases, however it has some degree of inaccuracy, as shown in the last two rows. In DepthNet, the estimated depth indicates the position of each keypoint relative to other keypoints rather than with respect to a source and importantly it is done without any supervision.

By comparing different models in Table 10.3, MOFA requires proper initialization to map face meshes to each image. AIGN requires depth labels to train the model. Our baseline DepthNet model neither require any depth labels nor any manual tuning. The results also show DepthNet can work well on unpaired data. We would also like to emphasize that MOFA and AIGN are designed to estimate a 3D model, while DepthNet is designed to estimate the parameters that facilitate warping a face pose to another without having depth values, so these models are designed to solve different problems.

An interesting observation is that GT depth gets a higher MSE compared to DepthNet. This can be due to not having a perspective projection between source and target faces. However, since DepthNet is trained to map to the target faces, it learns the affine parameters in a way to minimize this loss.

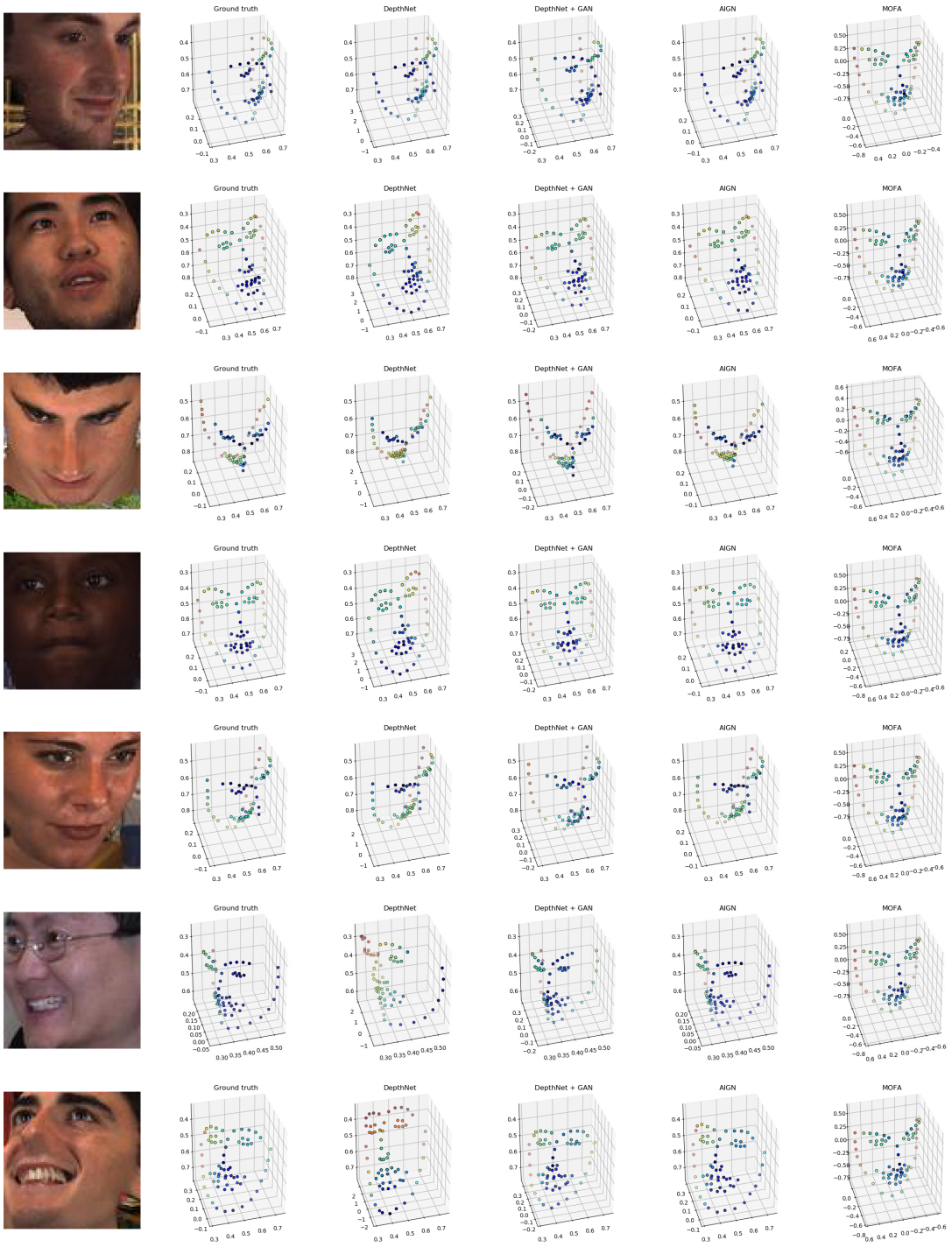


Fig. 10.4. Depth visualization for different models (color coded by depth). The Depth axis is the one pointing into the page. From left to right: RGB image, Ground Truth, DepthNet, DepthNet+GAN, AIGN and MOFA estimated depth values.

10.4.3. Face Rotation, Replacement and Adversarial Repair

In this section we show how DepthNet can be used for different applications.

10.4.3.1. *Face Rotation*

In Figure 10.1 (right) we visualize the face rotation by re-projecting a frontal face, from Multi-PIE (Gross et al., 2010), (far left) to a range of other poses defined by the faces in the row above. Since DepthNet (case B) computes transformation on keypoints rather than pixels it is robust to illumination changes between source and target faces. Note that DepthNet preserves well the identity. However, it carries forward the emotion from source to target since using a global affine transformation imparts a degree of robustness to dramatic expression changes. The views in these figures are rendered from a 3D model in OpenGL. Note the model can align well to the target face poses.

We show further camera sweeps for frontal source faces in Figures 10.5 and 10.6 and non-frontal source faces in Figure 10.7. In Figure 10.8 we use the same target identity as the source face, showing how much the generated face differs from the ground truth target. For all samples in Figures 10.5, 10.6, 10.7, and 10.8 we use the DepthNet model that relies on only key-points (model 7 in Table 10.1). Frontal faces selected from the Multi-PIE dataset (Gross et al., 2010) are re-projected to match several other poses corresponding to a person with a different identity. The DepthNets predicts reliable proxy depths, which when coupled with the analytically obtained affine transform (obtained from the least-squares pseudo-inverse-based solution described in Section 10.3.3) yields faces close to the desired target face geometry when passed through the OpenGL pipeline.

Note that for non-frontal source faces in Figures 10.7 and 10.8, the quality of images is reduced specially for frontal target faces. This is due to lack of adequate texture on the occluded side of the face to be transferred to the target pose by OpenGL pipeline, rather than inaccuracies in the affine transformation parameters. In order to reduce the side-affects, we use either of the source face or its flipped version, that is closer to the target face pose, and then warp the face to the target keypoints.

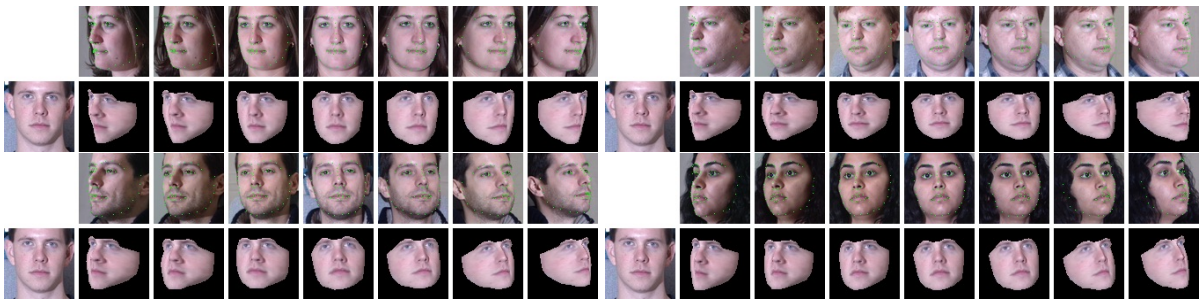


Fig. 10.5. Projecting a frontal face (far left) to a range of other poses defined by faces in the row above.

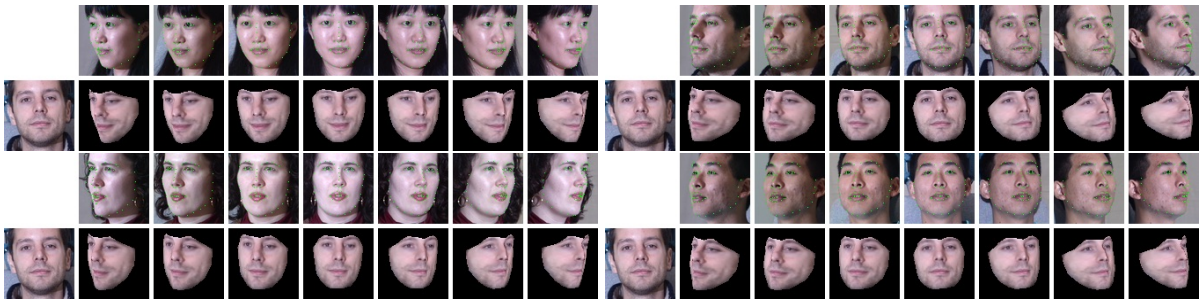


Fig. 10.6. Projecting a frontal face (far left) to a range of other poses defined by faces in the row above.

Fig. 10.7. Re-projecting a non-frontal face (far left) to a range of other poses defined by faces in the row above.

Fig. 10.8. Rotating a face (far left) to a range of other poses defined by faces of the same identity in the row above. On the top row frontal and on the bottom row non-frontal source faces are shown.

10.4.3.2. *Background Synthesis*

In another experiment, we do face frontalization with synthesized background. Here we use CycleGAN to add background detail to a face that has been frontalized with DepthNet. Referring to Figure 10.9, we perform this by conditioning the CycleGAN on the DepthNet image (column 3) and the background of column 2 (masking interior face region determined by the convex hull spanned by the keypoints). The second domain contains ground truth frontal faces. This experiment shows how to leverage DepthNet for full face generation. Note that we do not use identity information in this experiment. However, it can be used to better preserve the identity.

10.4.3.3. *Face Replacement*

Finally, we do face swaps, where we warp the face of one identity onto the geometry and background of another identity using DepthNet. To do so, we paste the rotated face by DepthNet onto the background of the target image and train a CycleGAN to map from the domain of 'swapped in faces' to the ground truth faces in our dataset, effectively learning to clean up face swaps so that the face region matches the hair and background. The examples of this procedure are shown in Figure 10.10.

10.4.3.4. *Extreme Pose Face Clean-up*

If the source image has an extreme pose, the texture will be missing on the occluded side of the face and the OpenGL pipeline cannot rotate the face without artifacts. Note that this shortcoming is

Fig. 10.9. Background synthesis with CycleGAN. Left to right: source face; keypoints overlaid; DepthNet (DN); DN + background → frontal

Fig. 10.10. Face swap experiment with CycleGAN. Left to right: source face; target face; warp to target with DepthNet; repaired result with CycleGAN. The source face is taken and warped onto the target face. The background and hairstyle is then adapted to the target face.

due to lack of texture on the occluded side of the face rather than a deficiency of the transformation parameters measured by DepthNet.

We performed an experiment using CycleGAN to fix such artifacts. For this experiment we take source images from CelebA and first frontalize it by using DepthNet. Since the frontalized faces have artifacts due to stretch of texture on the occluded side of the face by the OpenGL pipeline, we train a CycleGAN that takes DepthNet frontalizaed faces plus the background of the original non-frontal image (as two images) in one domain and the ground truth frontal faces in the other domain. The CycleGAN learns to clean-up these artifacts. Finally, we take the GAN-repaired frontalized faces and project it to different target poses using DepthNet. In Figure 10.11 we visualize camera sweep for source faces in the wild that have extreme poses. The cycleGAN reasonably cleans the face artifacts and then DepthNet projects it to different poses. This is just one approach to address the extreme pose occlusion artifacts. We see alternative methods for addressing this issue as promising directions for future research.

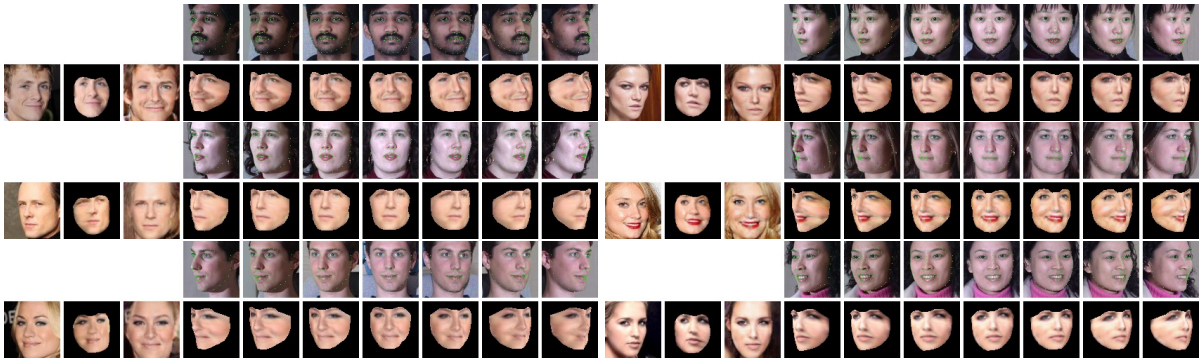


Fig. 10.11. Re-projecting a non-frontal face (far left) from CelebA to a range of other poses defined by faces in the row above. Top row (in each pair) depicts the target faces from Multi-PIE (Gross et al., 2010). The bottom row shows from left to right: source face, souce face frontalized by DepthNet, adversarial-repaired face, the repaired source face projected to the target poses (4th to 10th columns).

10.5. Conclusion

We have proposed a novel approach to 3D face model creation which enables pose normalization without using any ground truth depth data. We achieve our best quantitative keypoint registration

results using our novel formulation for predicting depth and 3D visual geometry simultaneously, learned by backpropagating through the analytic solution for the visual geometry estimation problem expressed as a function of predicted depths. We have illustrated the quality and utility of the depths and 3D transformations obtained using our method by transforming source faces to a wide variety of target poses and geometries. Our technique can be used for face rotation and replacement and when combined with adversarial repair it can blend warped faces to also synthesize the background. The proposed model, however, carries forward emotion from source to target due to learning a shared affine parameters for all keypoints. Moreover, for extreme non-frontal faces, while DepthNet can extract the transformation params (since it only relies on keypoints), OpenGL cannot extract texture due to occlusion. We show an example of how to address this problem in Section 10.4.3.4. An interesting extension to this paper can be replacing the OpenGL pipeline with a generative adversarial framework that synthesizes a face using the parameters estimated by DepthNet.

Acknowledgements

We would like to thank Samsung and Google for partially funding this project. We are also thankful to Compute Canada and Calcul Quebec for providing computational resources, and to Poonam Goyal for helpful discussions.

Chapter 11

Conclusion

The works presented in this thesis showed progress on different aspects of landmark localization. They also demonstrate an evolution from more traditional methods to deep learning approaches and also from more supervised to less supervised techniques.

The first project adopted a more traditional approach and applied a factor analysis model to disentangle features about identity and expression (IE), effectively learning a prior over landmarks. The IE model is later merged with a discriminative per-landmark classifier in a joint identity expression constrained local model (IE-CLM). Due to the emergence of deep learning models, my later works shifted towards leveraging deep neural network architectures.

In the second work, we introduced ReCombinator Networks (RCN) by proposing an architecture that preserves pixel-level feature information, effectively avoiding information loss due to pooling layers. While this approach is proposed for landmark localization, it is currently used for a wide range of models that reconstruct back the input image in a different modality, such as in image-to-image translation, or require pixel-level accuracy, such as in semantic segmentation. This work also proposed a framework for learning a joint distribution over landmarks and merged its output (as a joint distribution over landmarks) with the RCN output (as a per-landmark distribution). The similarity between this work and the first work is in merging a probabilistic model that learns a joint distribution over landmarks with another model that learns a per-landmark discriminator.

In the first two works a fully supervised setting was assumed. In the third work a semi-supervised setting was considered and the goal was to leverage weaker labelled data and unlabelled data. This line of research focuses on one of the main current limitations of deep learning approaches, namely their requiring a large set of labelled data to perform well. Meta-learning approaches and approaches that learn from very few labelled data, such as few-shot learning, are amongs the hot topics at the

time of writing of this thesis. They share a similar goal as semi-supervised techniques namely the ability to obtain high accuracy on new data while leveraging few labelled data.

Finally our last work proposed an unsupervised technique to estimate the depth of landmarks. This approach matches pairs of landmarks and creates a bottleneck that only requires estimating depth values. The novelty of this approach is in leveraging correspondences of two entities to estimate values of interest (depth in our case) and further using the estimated depth for other tasks of interest such as face rotation and replacement. Using unsupervised techniques to enhance learning algorithms is currently one of the hot topics in deep learning and computer vision communities.

The general direction of this thesis has evolved from a fully supervised setting to weakly, semi-supervised and finally unsupervised settings. Learning algorithms which have a lower dependency on abundant labels would enable further application of deep learning models to new data and facilitate learning faster on new tasks.

Bibliography

http://www.face.com.

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Akshay Asthana, Stefanos Zafeiriou, Shiyang Cheng, and Maja Pantic. Robust discriminative response map fitting with constrained local models. In *CVPR*, pages 3444–3451, 2013.

Amir Atapour-Abarghouei and Toby P Breckon. Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 18, page 1, 2018.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

F. R Bach and M. I Jordan. A probabilistic interpretation of canonical correlation analysis. Technical Report 688, Department of Statistics, University of California, Berkeley, 2005.

Pierre Baldi and Peter Sadowski. The dropout learning algorithm. *Artificial intelligence*, 210: 78–122, 2014.

Tadas Baltrušaitis, Peter Robinson, and Louis-Philippe Morency. Continuous conditional neural fields for structured regression. In *European conference on computer vision*, pages 593–608. Springer, 2014.

Peter N Belhumeur, David W Jacobs, David J Kriegman, and Neeraj Kumar. Localizing parts of faces using a consensus of exemplars. *IEEE Transactions on Pattern Analysis and Machine*

Intelligence, 35(12):2930–2940, 2013.

Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1:2012, 2012.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013a.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pages 899–907, 2013b.

Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 259–302, 1986.

Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194. ACM Press/Addison-Wesley Publishing Co., 1999.

Volker Blanz, Sami Romdhani, and Thomas Vetter. Face identification across different poses and illuminations with a 3d morphable model. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 202–207. IEEE, 2002.

Volker Blanz, Kristina Scherbaum, Thomas Vetter, and Hans-Peter Seidel. Exchanging faces in images. In *Computer Graphics Forum*, volume 23, pages 669–676. Wiley Online Library, 2004.

Xavier P Burgos-Artizzu, Pietro Perona, and Piotr Dollár. Robust face landmark estimation under occlusion. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1513–1520. IEEE, 2013.

Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. Face alignment by explicit shape regression. In *IJCV*, 107(2):177–190, 2014.

H Chan and WW Bledsoe. A man-machine facial recognition system: some preliminary results. *Panoramic Research Inc., Palo Alto, CA, USA1965*, 1965.

Olivier Chapelle and Mingrui Wu. Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval*, 13(3):216–235, 2010.

Casey Chu, Andrey Zhmoginov, and Mark Sandler. Cyclegan: a master of steganography. *arXiv preprint arXiv:1712.02950*, 2017.

Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer, 2016.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Joseph Paul Cohen, Margaux Luck, and Sina Honari. Distribution matching losses can hallucinate features in medical image translation. In *MICCAI*, 2018.

T. Cootes, G. Edwards, and C. Taylor. Active appearance models. In *PAMI*, 23(6):681–685, 2001.

T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham. Active Shape Models-Their Training and Application. *Computer Vision and Image Understanding*, 61(1):38–59, January 1995. ISSN 1077-3142.

Brian Jack Copeland. Artificial intelligence, 2018. URL `https://www.britannica.com/technology/artificial-intelligence`.

David Cristinacce and Tim Cootes. Automatic feature localisation with constrained local models. *Pattern Recognition*, 41(10):3054–3067, 2008.

David Cristinacce and Timothy F Cootes. Feature detection and tracking with constrained local models. In *BMVC*, volume 2, page 6, 2006.

David Cristinacce and Timothy F Cootes. Boosted regression active shape models. In *BMVC*, pages 1–10, 2007.

Nasser Dardas, Qing Chen, Nicolas D Georganas, and Emil M Petriu. Hand gesture recognition using bag-of-features and multi-class support vector machine. In *Haptic Audio-Visual Environments and Games (HAVE), 2010 IEEE International Symposium on*, pages 1–5. IEEE, 2010.

Dragos Datcu and Stephan Lukosch. Free-hands interaction in augmented reality. In *Proceedings of the 1st symposium on Spatial user interaction*, pages 33–40. ACM, 2013.

DeepLearning.net. http://deeplearning.net/tutorial/lenet.html.

Alessio Del Bue, João Xavier, Lourdes Agapito, and Marco Paladini. Bilinear modeling via augmented Lagrange multipliers (BALM). *IEEE transactions on pattern analysis and machine intelligence*, 34(8):1496–508, August 2012.

Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. *CVPR 2018 Deep Learning for Visual SLAM Workshop (DL4VSLAM2018)*, 2018.

Terrance Devries, Kumar Biswaranjan, and Graham W Taylor. Multi-task learning of facial landmarks and expression. In *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pages 98–103. IEEE, 2014.

Yangzhou Du and Xueyin Lin. Emotional facial expression model building. *Pattern Recognition Letters*, 24(16):2923–2934, 2003.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

G.J. Edwards, A. Lanitis, C.J. Taylor, and T.F. Cootes. Statistical models of face images . improving specificity. *Image and Vision Computing*, 16(3):203–211, 1998.

David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2366–2374, 2014.

Jeffrey L Elman and David Zipser. Learning the hidden structure of speech. *The Journal of the Acoustical Society of America*, 83(4):1615–1626, 1988.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

M. R. Everingham, J. Sivic, and A. Zisserman. Hello! my name is... buffy" – automatic naming of characters in tv video. In *Proc. BMVC*, pages 92.1–92.10, 2006. doi:10.5244/C.20.92.

Sachin Sudhakar Farfade, Mohammad J Saberian, and Li-Jia Li. Multi-view face detection using deep convolutional neural networks. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 643–650. ACM, 2015.

Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1627–1645, 2010.

Raphaël Féraud, Olivier J Bernier, Jean-Emmanuel Viallet, and Michel Collobert. A fast and accurate face detector based on neural networks. *IEEE Transactions on pattern analysis and machine intelligence*, 23(1):42–53, 2001.

William T Freeman and Joshua B Tenenbaum. Learning bilinear models for two-factor problems in vision. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 554–560. IEEE, 1997.

Brendan J Frey and Nebojsa Jojic. Estimating mixture models of images and inferring spatial transformations using the em algorithm. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, pages 416–422. IEEE, 1999.

Christophe Garcia and Manolis Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 26 (11):1408–1423, 2004.

Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–756. Springer, 2016.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010.

Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Chao Gou, Yue Wu, Kang Wang, Kunfeng Wang, Fei-Yue Wang, and Qiang Ji. A joint cascaded framework for simultaneous eye detection and eye state estimation. *Pattern Recognition*, 67: 23–31, 2017.

Ralph Gross, Iain Matthews, and Simon Baker. Generic vs. person specific active appearance models. *Image and Vision Computing*, 23(12):1080–1093, November 2005.

Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. Multi-PIE. *Image and Vision Computing*, 28(5):807–813, 2010.

Richard Gross. Fully automatic pose-invariant face recognition via 3d pose normalization. In *sychology: The Science of Mind and Behaviour 7E*, 2015.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

Md. Kamrul Hasan, Christopher Pal, and Sharon Moalem. Localizing facial keypoints with global descriptor search, neighbour alignment and locally linear models. In *Proceedings of the 2013 IEEE International Conference on Computer Vision Workshops*, ICCVW '13, pages 362–369, Washington, DC, USA, 2013. IEEE Computer Society.

Tal Hassner. Viewing real-world faces in 3d. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 3607–3614, 2013.

Tal Hassner, Shai Harel, Eran Paz, and Roee Enbar. Effective face frontalization in unconstrained images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4295–4304, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Sina Honari, Jason Yosinski, Pascal Vincent, and Christopher Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5743–5752, 2016.

Sina Honari, Pavlo Molchanov, Stephen Tyree, Pascal Vincent, Christopher Pal, and Jan Kautz. Improving landmark localization with semi-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Kaoning Hu, Shaun Canavan, and Lijun Yin. Hand pointing estimation for human computer interaction based on two orthogonal-views. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3760–3763. IEEE, 2010.

G. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, 07-49, University of Massachusetts, Amherst, 2007.

Rui Huang, Shu Zhang, Tianyu Li, and Ran He. Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

David H Hubel and Torsten N Wiesel. Cortical and callosal connections concerned with the vertical meridian of visual fields in the cat. *J. Neurophysiol*, 30(6):1561–1573, 1967.

David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

DH Hubel and TN Wiesel. Receptive fields of optic nerve fibres in the spider monkey. *The Journal of physiology*, 154(3):572–580, 1960.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint*, 2017.

Aaron S Jackson, Adrian Bulat, Vasileios Argyriou, and Georgios Tzimiropoulos. Large pose 3d face reconstruction from a single image via direct volumetric cnn regression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1031–1039. IEEE, 2017.

Tomas Jakab, Ankush Gupta, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks through conditional image generation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4019–4030. Curran Associates, Inc., 2018.

LáSzló A Jeni, András Lőrincz, Tamás Nagy, Zsolt Palotai, Judit Sebők, Zoltán Szabó, and Dániel Takács. 3d shape estimation in video sequences provides high precision evaluation of facial expressions. *Image and Vision Computing*, 30(10):785–795, 2012.

László A Jeni, Jeffrey F Cohn, and Takeo Kanade. Dense 3d face alignment from 2d videos in real-time. In *IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, volume 1, pages 1–8. IEEE, 2015.

Dalong Jiang, Yuxiao Hu, Shuicheng Yan, Lei Zhang, Hongjiang Zhang, and Wen Gao. Efficient 3d reconstruction for face recognition. *Pattern Recognition*, 38(6):787–798, 2005.

Huaizu Jiang and Erik Learned-Miller. Face detection with the faster r-cnn. In *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on*, pages 650–657. IEEE, 2017.

Samira Ebrahimi Kahou, Christopher Pal, Xavier Bouthillier, Pierre Froumenty, Çaglar Gülçehre, Roland Memisevic, Pascal Vincent, Aaron Courville, Yoshua Bengio, Raul Chandias Ferrari, et al. Combining modality specific deep neural networks for emotion recognition in video. In

Proceedings of the 15th ACM on International conference on multimodal interaction, pages 543–550. ACM, 2013.

Samira Ebrahimi Kahou, Xavier Bouthillier, Pascal Lamblin, Caglar Gulcehre, Vincent Michalski, Kishore Konda, Sébastien Jean, Pierre Froumenty, Yann Dauphin, Nicolas Boulanger-Lewandowski, et al. Emonets: Multimodal deep learning approaches for emotion recognition in video. *Journal on Multimodal User Interfaces*, 10(2):99–111, 2016.

Michal Kawulok, Jolanta Kawulok, Jakub Nalepa, and Bogdan Smolka. Self-adaptive algorithm for segmenting skin regions. *EURASIP Journal on Advances in Signal Processing*, 2014(1):170, 2014.

Ira Kemelmacher-Shlizerman and Ronen Basri. 3d face reconstruction from a single image using a single reference face shape. *IEEE transactions on pattern analysis and machine intelligence*, 33 (2):394–405, 2011.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Dirk-Jan Kroon. Active shape model and active appearance model. 2012.

Martin Köstinger, Paul Wohlhart, Peter M. Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *In: Benchmarking Facial Image Analysis Technologies (ICCV Workshop*, 2011.

Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representation*, 2017.

Vuong Le, Jonathan Brandt, Zhe Lin, Lubomir Bourdev, and Thomas S Huang. Interactive facial feature localization. In *European Conference on Computer Vision*, pages 679–692. Springer, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989. ISSN 0899-7667.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018.

P Liao, L Shen, YQ Chen, and SC Liu. Unified model in identity subspace for face recognition. *Journal of Computer Science and Technology*, 19(5):684–690, 2004.

Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5162–5170, 2015.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

Jiangjing Lv, Xiaohu Shao, Junliang Xing, Cheng Cheng, and Xi Zhou. A deep regression architecture with two-stage re-initialization for high performance facial landmark detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5667–5675, 2018.

Iacopo Masi, Anh Tuan Tran, Tal Hassner, Jatuporn Toy Leksut, and Gérard Medioni. Do we really need to collect millions of faces for effective face recognition? In *European Conference on Computer Vision*, pages 579–596. Springer, 2016.

Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 565–571. IEEE, 2016.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Kenneth Alberto Funes Mora and Jean-Marc Odobez. Gaze estimation from multimodal kinect data. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*, pages 25–30. IEEE, 2012.

Jakub Nalepa and Michal Kawulok. Fast and accurate hand shape classification. In *International Conference: Beyond Databases, Architectures and Structures*, pages 364–373. Springer, 2014.

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

Andrew Ng. Sparse autoencoder. *CS294A Lecture Notes*, page p. 72, 2011.

Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*, volume 501. John Wiley & Sons, 2009.

Seonwook Park, Xucong Zhang, Andreas Bulling, and Otmar Hilliges. Learning to find eye region landmarks for remote gaze estimation in unconstrained settings. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, ETRA '18, pages 21:1–21:10. ACM, 2018.

Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. *Proceedings of the British Machine Vision Conference (BMVC)*, 2015.

Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. In *Advanced video and signal based surveillance, 2009. AVSS'09. Sixth IEEE International Conference on*, pages 296–301. Ieee, 2009.

A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*, pages 84–91. IEEE Comput. Soc. Press, 1994.

S. Prince, P. Li, Y. Fu, U. Mohammed, and J. Elder. Probabilistic models for inference about identity. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1, 2011.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015.

Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. Face alignment at 3000 fps via regressing local binary features. In *CVPR*, pages 1685–1692, 2014.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 833–840. Omnipress, 2011.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015*, pages 234–241. Springer, 2015.

Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1):23–38, 1998.

Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *International Conference on Computer Vision, (ICCV-W), 300 Faces in-the-Wild Challenge (300-W), Sydney, Australia, 2013*, 2013.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

Jason M Saragih, Simon Lucey, and Jeffrey F Cohn. Face alignment through subspace constrained mean-shifts. In *ICCV*, pages 1034–1041, 2009.

Jason M. Saragih, Simon Lucey, and Jeffrey F. Cohn. Deformable Model Fitting by Regularized Landmark Mean-Shift. *International Journal of Computer Vision*, 91(2):200–215, September 2010.

P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *IJCNN*, pages 2809–2813, 2011.

P. Sermanet, K. Kavukcuoglu, S. Chintala, and Y. LeCun. Pedestrian detection with unsupervised multi-stage feature learning. In *CVPR*, pages 3626–3633, 2013.

Yujun Shen, Ping Luo, Junjie Yan, Xiaogang Wang, and Xiaoou Tang. Faceid-gan: Learning a symmetry three-player gan for identity-preserving face synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 821–830, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Ayan Sinha, Chiho Choi, and Karthik Ramani. Deephand: Robust hand pose estimation by completing a matrix imputed with deep features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4150–4158, 2016.

Brandon M Smith and Li Zhang. Collaborative facial landmark localization for transferring annotations across datasets. In *European Conference on Computer Vision*, pages 78–93. Springer, 2014.

Brandon M Smith, Jim Brandt, Zhe Lin, and Li Zhang. Nonparametric context modeling of local appearance for pose-and expression-robust facial landmark localization. In *CVPR*, pages 1741–1748, 2014.

Srinath Sridhar, Antti Oulasvirta, and Christian Theobalt. Interactive markerless articulated hand motion tracking using RGB and depth data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2456–2463, 2013.

Pratul P Srinivasan, Rahul Garg, Neal Wadhwa, Ren Ng, and Jonathan T Barron. Aperture supervision for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6393–6401, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Xiao Sun, Yichen Wei, Shuang Liang, Xiaoou Tang, and Jian Sun. Cascaded hand pose regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 824–832, 2015.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3476–3483. IEEE, 2013.

Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

Danhang Tang, Hyung Jin Chang, Alykhan Tejani, and Tae-Kyun Kim. Latent regression forest: Structured estimation of 3d articulated hand posture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3786–3793, 2014.

Ayush Tewari, Michael Zollhöfer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Perez, and Christian Theobalt. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, 2017.

James Thewlis, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks by factorized spatial embeddings. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 1, page 5, 2017.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Massimo Tistarelli and Mark S. Nixon, editors. *Advances in Biometrics*, volume 5558 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

Jonathan Tompson, Murphy Stein, Yann Lecun, and Ken Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (ToG)*, 33(5):169, 2014.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 648–656, 2015.

Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1653–1660, 2014.

Luan Tran and Xiaoming Liu. Nonlinear 3d face morphable model. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, June 2018.

Luan Tran, Xi Yin, and Xiaoming Liu. Disentangled representation learning gan for pose-invariant face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Hsiao-Yu Tung, Hsiao-Wei Tung, Ersin Yumer, and Katerina Fragkiadaki. Self-supervised learning of motion capture. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5242–5252, 2017a.

Hsiao-Yu Fish Tung, Adam W Harley, William Seto, and Katerina Fragkiadaki. Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, 2017b.

Georgios Tzimiropoulos. Project-out cascaded regression with an application to face alignment. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3659–3667. IEEE, 2015.

Georgios Tzimiropoulos and Maja Pantic. Gauss-newton deformable part models for face alignment in-the-wild. In *CVPR*, pages 1851–1858, 2014.

Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

Régis Vaillant, Christophe Monrocq, and Yann Le Cun. Original approach for the localisation of objects in images. *IEE Proceedings-Vision, Image and Signal Processing*, 141(4):245–250, 1994.

M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer, 2002.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11:3371–3408, 2010.

Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

Daniel Vlasic, Matthew Brand, Hanspeter Pfister, and Jovan Popović. Face transfer with multilinear models. *ACM Trans. Graph.*, 24(3):426–433, 2005. ISSN 0730-0301.

Chengde Wan, Thomas Probst, Luc Van Gool, and Angela Yao. Crossing nets: Combining gans and vaes with a shared latent space for hand pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 680–689, 2017.

Kang Wang and Qiang Ji. Real time eye gaze tracking with 3d deformable eye-face model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1003–1011, 2017.

Wei Wang, Sergey Tulyakov, and Nicu Sebe. Recurrent convolutional face alignment. In *Asian Conference on Computer Vision*, pages 104–120, 2016.

Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

Shengtao Xiao, Jiashi Feng, Junliang Xing, Hanjiang Lai, Shuicheng Yan, and Ashraf Kassim. Robust facial landmark detection via recurrent attentive-refinement networks. In *European Conference on Computer Vision*, pages 57–72. Springer, 2016.

Xuehan Xiong and Fernando De la Torre. Supervised descent method and its applications to face alignment. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 532–539. IEEE, 2013.

Heng Yang and Ioannis Patras. Sieving regression forest votes for facial feature detection in the wild. In *ICCV*. IEEE, 2013.

Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5525–5533, 2016.

Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. LIFT: Learned invariant feature transform. In *European Conference on Computer Vision*, pages 467–483. Springer, 2016.

Zili Yi, Hao (Richard) Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. In *ICCV*, pages 2868–2876, 2017.

Xi Yin, Xiang Yu, Kihyuk Sohn, Xiaoming Liu, and Manmohan Chandraker. Towards large-pose face frontalization in the wild. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1–10, 2017.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *ICML Workshop on Deep Learning*, 2015.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representation*, 2016.

X. Yu, J. Huang, S. Zhang, W. Yan, and D. Metaxas. Pose-free facial landmark fitting via optimized part mixtures and cascaded deformable shape model. In *ICCV*, pages 1944–1951, 2013.

Xiang Yu, Feng Zhou, and Manmohan Chandraker. Deep deformation network for object landmark localization. In *European Conference on Computer Vision*, pages 52–70. Springer, 2016.

Shanxin Yuan, Guillermo Garcia-Hernando, Bjorn Stenger, Tae-Kyun Kim, et al. Depth-based 3d hand pose estimation: From current achievements to future goals. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Alan L Yuille, Peter W Hallinan, and David S Cohen. Feature extraction from faces using deformable templates. *International journal of computer vision*, 8(2):99–111, 1992.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Cha Zhang and Zhengyou Zhang. Improving multiview face detection with multi-task deep convolutional neural networks. In *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*, pages 1036–1041. IEEE, 2014.

J. Zhang, S. Shan, M. Kan, and X. Chen. Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In *ECCV*, pages 1–16, 2014a.

Xing Zhang, Lijun Yin, Jeffrey F Cohn, Shaun Canavan, Michael Reale, Andy Horowitz, Peng Liu, and Jeffrey M Girard. Bp4d-spontaneous: a high-resolution spontaneous 3d dynamic facial expression database. *Image and Vision Computing*, 32(10):692–706, 2014b.

Xucong Zhang, Yusuke Sugano, Mario Fritz, and Andreas Bulling. Appearance-based gaze estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4511–4520, 2015.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *Computer Vision–ECCV 2014*, pages 94–108. Springer, 2014c.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Learning deep representation for face alignment with auxiliary attributes. *IEEE transactions on pattern analysis and machine intelligence*, 38(5):918–930, 2016.

Jian Zhao, Lin Xiong, Panasonic Karlekar Jayashree, Jianshu Li, Fang Zhao, Zhecan Wang, Panasonic Sugiri Pranata, Panasonic Shengmei Shen, Shuicheng Yan, and Jiashi Feng. Dual-agent gans for photorealistic and identity preserving profile face synthesis. In *Advances in Neural Information Processing Systems (NIPS)*, pages 66–76, 2017.

Jian Zhao, Yu Cheng, Yan Xu, Lin Xiong, Jianshu Li, Fang Zhao, Karlekar Jayashree, Sugiri Pranata, Shengmei Shen, Junliang Xing, et al. Towards pose invariant face recognition in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2207–2216, 2018.

Junbo Jake Zhao, Michaël Mathieu, Ross Goroshin, and Yann LeCun. Stacked what-where auto-encoders. In *International Conference on Learning Representation - Workshop Track*, 2016.

Xiaowei Zhao, Shiguang Shan, Xiujuan Chai, and Xilin Chen. Locality-constrained active appearance model. In *Asian Conference on Computer Vision*, pages 636–647. Springer, 2012.

Chuan Zhou and Xueyin Lin. Facial expressional image synthesis controlled by emotional parameters. *Pattern Recognition Letters*, 26(16):2611–2627, 2005.

Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

Shizhan Zhu, Cheng Li, Chen Change Loy, and Xiaoou Tang. Face alignment by coarse-to-fine shape searching. In *CVPR*, pages 4998–5006, 2015.

Shizhan Zhu, Cheng Li, Chen-Change Loy, and Xiaoou Tang. Unconstrained face alignment via cascaded compositional learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2879–2886. IEEE, 2012.