

Université de Montréal

**Leveraging noisy side information for disentangling of factors of  
variation in a supervised setting**

**par Pierre Luc Carrier**

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

Mémoire présenté à la Faculté des arts et des sciences  
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)  
en informatique

Août, 2014

© Pierre Luc Carrier, 2014.

# Résumé

Ce mémoire est composé de trois articles et présente les résultats de travaux de recherche effectués dans le but d'améliorer les techniques actuelles permettant d'utiliser des données associées à certaines tâches dans le but d'aider à l'entraînement de réseaux de neurones sur une tâche différente.

Les deux premiers articles présentent de nouveaux ensembles de données créés pour permettre une meilleure évaluation de ce type de techniques d'apprentissage machine. Le premier article introduit une suite d'ensembles de données pour la tâche de reconnaissance automatique de chiffres écrits à la main. Ces ensembles de données ont été générés à partir d'un ensemble de données déjà existant, MNIST, auquel des nouveaux facteurs de variation ont été ajoutés. Le deuxième article introduit un ensemble de données pour la tâche de reconnaissance automatique d'expressions faciales. Cet ensemble de données est composé d'images de visages qui ont été collectées automatiquement à partir du Web et ensuite étiquetées.

Le troisième et dernier article présente deux nouvelles approches, dans le contexte de l'apprentissage multi-tâches, pour tirer avantage de données pour une tâche donnée afin d'améliorer les performances d'un modèle sur une tâche différente. La première approche est une généralisation des neurones Maxout récemment proposées alors que la deuxième consiste en l'application dans un contexte supervisé d'une technique permettant d'inciter des neurones à apprendre des fonctions orthogonales, à l'origine proposée pour utilisation dans un contexte semi-supervisé.

**Mots-clés:** réseaux de neurones, apprentissage profond, apprentissage supervisé, réseaux à convolutions, vision par ordinateur, reconnaissance de caractères manuscrits, reconnaissance d'expressions faciales, apprentissage multi-tâche, invariance, démêlage des facteurs de variation

# Summary

The thesis is composed of three articles and presents the results of research done in order to improve the current methods for improving a neural network's performance on a given task by taking advantage of data from other tasks.

The two first articles present new datasets created to allow better evaluation of this type of machine learning methods. The first article introduces a dataset suite for the task of handwritten digit recognition. This dataset suite was created from the existing dataset MNIST to which new factors of variation have been added. The second article introduces a new dataset for the task of facial expression recognition. It is composed of images of faces that were automatically collected from the Web and then labelled.

The third and last article presents two new approaches to improving performance on a task of interest by leveraging labels from another task in the context of multi-task learning. The first approach is a generalization of the recently introduced Maxout Networks designed for multi-task learning. The second approach consists in the application in a fully-supervised setting of the previously introduced Contractive Discriminant Analysis penalty, originally used in the semi-supervised setting to make groups of neurons learn features orthogonal to each other.

**Keywords:** neural networks, deep learning, supervised learning, convolutional networks, computer vision, handwritten digit recognition, facial expression recognition, multi-task learning, invariance, disentangling

# Contents

<b>Résumé</b> . . . . .	<b>ii</b>
<b>Summary</b> . . . . .	<b>iii</b>
<b>Contents</b> . . . . .	<b>iv</b>
<b>List of Figures</b> . . . . .	<b>vii</b>
<b>List of Tables</b> . . . . .	<b>viii</b>
<b>1 Introduction</b> . . . . .	<b>1</b>
<b>2 Introduction to Machine learning</b> . . . . .	<b>2</b>
2.1 Machine Learning Basics . . . . .	2
2.2 Supervised Learning . . . . .	5
2.2.1 Support Vector Machine . . . . .	5
2.2.2 Multilayer Perceptron . . . . .	6
2.3 Unsupervised Learning . . . . .	8
2.3.1 Autoencoders . . . . .	8
2.3.2 Restricted Boltzmann Machine . . . . .	9
2.4 Deep learning . . . . .	10
2.4.1 Rectified Linear Units and Maxout . . . . .	12
2.4.2 Momentum . . . . .	13
2.4.3 Dropout . . . . .	13
<b>3 Disentangling factors of variation</b> . . . . .	<b>15</b>
3.1 Explicit Orthogonality Enforcing . . . . .	15
3.2 Bilinear models . . . . .	16
<b>4 Learning with Noisy Labels</b> . . . . .	<b>19</b>
4.1 Surrogate Loss Functions . . . . .	19
4.2 Optimization Methods . . . . .	20

---

4.3	Label Confidence Modelling . . . . .	21
<b>5</b>	<b>Multi-task Learning . . . . .</b>	<b>22</b>
5.1	Learning Task Relationships . . . . .	23
5.2	Learning Global Features . . . . .	24
<b>6</b>	<b>Prologue to First Article . . . . .</b>	<b>26</b>
6.1	Article Details . . . . .	26
6.2	Individual Contributions . . . . .	26
<b>7</b>	<b>MNIST+: a simple benchmark for factor disentangling . . . . .</b>	<b>27</b>
7.1	Introduction . . . . .	27
7.2	Motivation . . . . .	27
7.3	Previous Work . . . . .	28
7.4	Content and characteristics . . . . .	28
7.5	Benchmarks . . . . .	30
7.6	Dataset availability . . . . .	33
<b>8</b>	<b>Prologue to Second Article . . . . .</b>	<b>34</b>
8.1	Article Details . . . . .	34
8.2	Individual Contributions . . . . .	34
<b>9</b>	<b>Facial Expression Recognition 2014 Dataset . . . . .</b>	<b>36</b>
9.1	Introduction . . . . .	36
9.2	Previous Work and Motivation . . . . .	36
9.3	Data collection process . . . . .	37
9.4	Relationship to the Facial Expression Recognition 2013 Dataset . . . . .	39
9.5	Content and characteristics . . . . .	39
9.6	Benchmarks . . . . .	40
9.7	Discussion . . . . .	42
9.8	Obtaining the data . . . . .	43
9.9	Additional Information . . . . .	44
<b>10</b>	<b>Prologue to Third Article . . . . .</b>	<b>46</b>
10.1	Article Details . . . . .	46
10.2	Individual Contributions . . . . .	46
<b>11</b>	<b>Discriminative Disentangling via Multi-way Pooling . . . . .</b>	<b>48</b>
11.1	Introduction . . . . .	48
11.2	Previous work . . . . .	50
11.3	Maxout Networks . . . . .	52
11.3.1	Multi-task Maxout . . . . .	53
11.4	Multi-way pooling . . . . .	54


---

11.5	Discriminative CDA . . . . .	56
11.5.1	Contractive Discriminative Analysis . . . . .	56
11.5.2	Learning orthogonal representations for orthogonal tasks . .	57
11.6	Experiments . . . . .	57
11.6.1	Experiments on MNIST+ . . . . .	58
11.6.2	Experiment on FER-2014 . . . . .	59
11.6.3	Experiment with noise control . . . . .	61
11.7	Discussion . . . . .	61
11.7.1	Supervised CDA . . . . .	62
11.7.2	Tolerance to noise . . . . .	65
11.8	Conclusion . . . . .	66
	<b>References . . . . .</b>	<b>67</b>



# List of Figures

2.1	Example of hyperplane learned by an SVM . . . . .	6
2.2	Multilayer perceptron with 1 hidden layer . . . . .	7
2.3	Convolutional layer followed by a pooling layer . . . . .	12
7.1	Samples from the MNIST+ dataset suite . . . . .	31
9.1	Samples from the FER-2014 dataset . . . . .	41
11.1	Visualization of Multi-way Pooling Maxout . . . . .	55
11.2	Performance of Multi-way Pooling Maxout on MNIST+ Texture with varying levels of noisy on the labels for the side task . . . . .	62



# List of Tables

7.1	Benchmarking results on the variants of the MNIST+ dataset . . .	32
9.1	Contents of the FER-2014 dataset . . . . .	40
9.2	Benchmarking results on the FER-2014 dataset . . . . .	42
9.3	Emotion-related keywords used to assemble image search queries . .	45
11.1	Performance of our approaches on the MNIST+ Texture dataset . .	59
11.2	Performance of our approaches on the FER-2014 dataset . . . . .	60





---

# List of Abbreviations

ASO	Alternating Structure Optimization
BGD	Batch Gradient Descent
CAE	Contractive Autoencoder
CCN	Class Conditional Noise
CCN	Classification Noise Process
CDA	Contractive Discriminant Analysis
convNet	Convolutional Network
DAE	Denoising Autoencoder
disBM	Disentangling Boltzman Machine
EM	Expectation Maximization
GRBM	Gaussian Restricted Boltzmann Machine
GUI	Graphical User Interface
hossRBM	Higher Order Spike-and-Slab Restricted Boltzmann Machine
iASO	improved Alternating Structure Optimization
ICA	Independent Component Analysis
LICA	Latent Independent Component Analysis
MLP	Multilayer Perceptron
PFD	Probability Density Function
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RBM	Restricted Boltzmann Machine
SGD	Stochastic Gradient Descent
ssRBM	Spike-and-Slab Restricted Boltzmann Machine
SVM	Singular Value Decomposition
SVM	Support Vector Machines
TFD	Toronto Face Database
UI	User Interface
URL	Uniform Resource Locator

---

## Acknowledgments

I would like to thank my thesis advisors, Aaron Courville and Yoshua Bengio, for many intellectually stimulating discussions, for giving me the opportunity to work on many interesting and varied projects, and for freely sharing their knowledge with me.

I would like to thank Heng Luo, Guillaume Desjardins, Ian Goodfellow and David Wardey-Farley who taught me much during my time at LISA lab.

I would like to thank the developers of the library Theano for all the hours I did *not* have to spend computing gradients.

I would like to thank my parents, for teaching me the value of education. And my old teachers Jonathan Prud'homme and Patrice Roy, for teaching me the value of hard work. I would not have gotten this far if it wasn't for each and every one of them.

Finally, I would like to thank my friends and family, for their understanding and support throughout this academic endeavor.

# 1 Introduction

The goal behind the research presented in this thesis by articles is to explore new and more efficient ways in which we can improve the performance of neural networks on a specific task by leveraging labels associated with other tasks. We are mostly interested in the setting where the labels to these other tasks are noisy since these noisy labels are very often easier and cheaper to obtain than noise-free labels.

This thesis is heavily motivated by the current trend, in machine learning, of training larger and more complex models which, in turn, fuels the demand for larger amounts of data to train those models. Because of this and because of the costs often associated with obtaining large datasets of clean data, the ability to efficiently leverage large amounts of noisy data – which are much more often easily available – is a valuable commodity.

This thesis is composed of three articles. The two first articles introduce new datasets that were created to allow better testing of the methods explored in this thesis. The third article proposes two new approaches to leveraging labels associated with other tasks and explores their performance on the previously mentioned datasets.

The articles presented in this thesis are the results of joint work with the other authors. As such, they are narrated in the first-person plural form. On the other hand, the rest of this thesis is my own work and it is therefore narrated in the first-person singular form.

The next chapter introduces the basic notions of machine learning that prove necessary to understand the work presented in this thesis. The three chapters that follow each reviews the scientific literature regarding one of the three central themes of this thesis. Finally, the chapters that remain present the three articles that make up this thesis.

# 2

# Introduction to Machine learning

The current chapter provides a brief introduction to machine learning and serves to introduce the concepts that this thesis relies upon. The next chapter will build on this and review the machine learning literature related to three concepts central to this thesis : disentangling factors of variation, learning with noisy labels and multi-task learning.

---

## 2.1 Machine Learning Basics

Machine learning is a branch of Artificial Intelligence which specializes in learning how to perform tasks *from data*. Thus, the algorithm to solve a task is never explicitly implemented by anyone but rather automatically discovered from the data by a machine learning system. This differs from expert systems where, to obtain a system able to perform a task, a human determines the steps to adequately solve the task and implements them explicitly.

The ability to learn from data is very useful in that it allows us to obtain systems that can solve problems that we do not know how to solve. Speech recognition is such a task; humans are able to process sound waves and recognize words that have been spoken out loud. It is a task we do naturally and at which we excel but we do not have precise knowledge of *how* we do it and, as such, we are unable to explicitly implement a system to do it. Computer vision is another example of this. The ability to learn from data with machine learning allows to circumvent this difficulty and learn systems to perform these complicated tasks.

Learning can be formally defined as such : “A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ ” (?). In the context of machine learning,  $E$  involves being exposed to some

---

data representative of task  $T$  and adjusting the program's parameters.  $T$  and  $P$ , however, can vary significantly based on the problem to solve and the model that is used to do so.

With neural networks, training is often done by *gradient descent*. To do this, we define a loss function  $L$  specific to the model  $m$  and the task  $T$  that, given some data  $D$  and the model parameters  $\Theta$ , returns a scalar indicating how well the model performs the task on this data (the better  $m$  performs, the lower the return value of  $L$  is). Then, when at time  $t$ , some data  $D$  is presented to the network, the parameters  $\Theta$  are updated according to equation 2.1. The variable  $\gamma_t$ , often referred to as the *learning rate*, is a scalar that controls how much the parameters are altered by the gradient descent step. It is sometimes constant throughout training and can also vary in time.

$$\Theta_{t+1} \leftarrow \Theta_t - \gamma_t \cdot \nabla_{\Theta_t} L(D, \Theta_t) \quad (2.1)$$

Given a set of data containing multiple examples of the behavior that is expected of the model, training the model involves repeatedly exposing it to the data and adjusting  $\Theta$  through gradient descent until some condition is attained. This set of data that is used to train the model by changing its parameters is referred to as the “training set”.

It is possible to show the training examples to the model only a few at a time (Stochastic Gradient Descent or SGD) or even all at the same time (Batch Gradient Descent or BGD). An epoch refers to the model being trained once on all of the data in the training set. In BGD, since the training data is shown to the model all at the same time, the parameters are updated only once per epoch. If the examples are shown one by one to the model, the parameters will be updated, at each epoch, as many times as the number of examples in the training set.

However, because the objective of machine learning is *generalization* i.e. the model  $m$  performing well on task  $T$  and not only on the training set, the training set is not sufficient to properly train a model. At the beginning of training, the model learns general features and properties of the data that are useful on the training set: this tends to improve the model's performance on both task  $T$  and the training set. Then, the model often starts, at some point as training progresses, to learn new features that pick up on very specific properties of the training examples to keep improving its performance on the training set. This further improves the model's

---

performance on the training set, to which these new features are uniquely tuned, but degrades the model's performance on task  $T$  because these new features do not perform well on data that is not exactly like the training set. In a sense, the model starts naively *remembering* the training examples instead of extracting generally useful features from them. This phase of training where performance improves on the training set while getting worse on task  $T$  is called *overfitting*. To avoid overfitting, it is necessary to have a second set of data from task  $T$  on which the model will not be trained but that will rather be used to monitor the generalization performance of the model and decide when to stop training. This second set of data is called the validation set and it is used to test the model's performance.

It turns out that even with a validation set, it is possible to overfit. For instance, if the training is stopped when the performance is optimal on the validation set then the model is, to some degree, tuned to the validation set since the point where training is stopped might depend on some specific properties of the validation examples. Because it is tuned to the validation set, the model's performance on the validation set is now a biased estimator of the model's performance on task  $T$ . To deal with this issue, we need a "test set": a second set of data not seen during training by the model. The validation set is used to tune the parameters that can hardly be optimized by gradient descent : learning rate, number of epochs for training, size of the model, architecture of the model, etc. Once these "hyperparameters", as they are called, are tuned on the validation set, the test set is used to evaluate the model's performance. This process allows to get from the test set an unbiased estimate of the model's generalization performance on task  $T$ .

In machine learning, the tasks, or problems, we attempt to learn often fall into one of the two following categories; supervised learning and unsupervised learning. These are explained in detail in the two following sections. Some problems have the characteristics of both categories; usually solving these problems involves simultaneously solving a supervised problem and an unsupervised problem. These fall under the category of semi-supervised learning. Finally, there is another category of tasks called reinforcement learning. This category, however, is not described here because it falls outside of the scope of this thesis.

---

## 2.2 Supervised Learning

Supervised learning is a class of problems where models must learn, given the values of a set of variables, to predict the values of another set of variables. Thus, every example  $D_i$  is composed of inputs  $x_i$  and of a label  $y_i$ . Data that includes labels is said to be “labeled data” in opposition to “unlabeled data” which doesn’t.

Two distinct learning problems fall in the category of supervised learning : classification and regression. In classification, the label the model learns to predict is a discrete variable and every value that the label can take is designated as a “class”. When the label has only two possible values, the classification problem is often designated a “binary classification” problem. Otherwise, it is a “multi-class classification” problem. Predicting whether it will rain or not tomorrow from current temperature, atmospheric pressure and humidity level is an example of a binary classification problem (the two classes being “it will rain” and “it will not rain”). In a regression problem, the labels are continuous variables. Predicting a person’s salary from that person’s sex, age and education level is an example of a regression problem.

The subsections that follow describe a few common models for supervised learning.

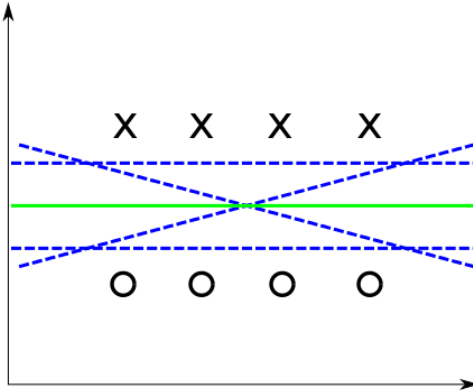
### 2.2.1 Support Vector Machine

Support Vector Machines (SVMs) are binary classifiers. An SVM learns the orientation and position of a hyperplane in the input space or a fixed feature space and uses it as a decision function to classify new examples. Data points that fall on one side of the hyperplane are classified as one class by the model and those that fall on the other side are classified as the other class. SVMs can be used to perform multi-classification with  $n$  classes by training  $n$  SVMs, one for each class, with each SVM attempting to discriminate between the examples of that class and the examples of every other class.

SVMs are shallow classifiers but they perform well because they not only attempt to learn a hyperplane that linearly separates the data but also tries to select this hyperplane as to maximize the margin between the hyperplane and the training example. Figure 2.1 illustrates this with an example.

It is possible to use SVMs to learn a non-linear decision function by projecting



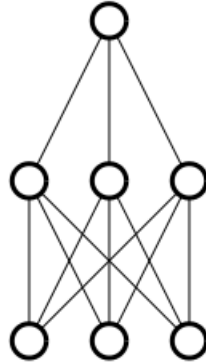


**Figure 2.1** – An example of a binary classification dataset that the SVM can learn to perfectly separate: the crosses represent the examples of the first class and the circles represent examples of the second class. The blue lines and the green all represent valid hyperplanes that perfectly separate the data but the green one is the one the SVM will learn.

the data in a new space with a non-linear transformation and then learning a linear decision function in this space which will correspond to a non-linear decision function in the original inputs space. Kernel SVMs can do this without ever explicitly projecting the data, which can be a very costly operation if the projection is complex or if the new space has very high dimension, but it's only possible for certain types of projections. In this thesis, the so-called Radial Basis Function (RBF) Kernel SVM is abundantly used.

### 2.2.2 Multilayer Perceptron

Multilayer Perceptrons (MLPs) are neural networks where the neurons are organized in layers (see figure 2.2) where the activation of any neuron in a layer is equal to a linear function of the activations of the neurons at the previous layer to which an “activation function” is applied. This linear function is simply a weighted sum to which a bias term is added. Thus, the activation of a neuron is described by equation 2.2 where  $W$  is the vector containing the weights from the previous layer to the neuron,  $b$  is the bias term of that neuron,  $x$  is the vector of the activations of the previous layer,  $n$  is the number of neurons in the previous layer and  $\sigma$  is the neuron's activation function.



**Figure 2.2** – A multilayer perceptron with 3 inputs, 1 hidden layer with 3 neurons and an output layer with 1 neuron

$$\text{activation} = \sigma \left( \sum_{i=1}^n W_i x_i + b \right) = \sigma(Wx + b) \quad (2.2)$$

Common activations functions for MLPs include the logistic function (see equation 2.3) and the hyperbolic tangent (see equation 2.4). Neurons are often designated by their activation function, thus a neuron with a logistic activation function is called a logistic unit. A neuron can also have no activation function, in that case it is a linear unit.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.3)$$

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.4)$$

The last layer of an MLP is the output layer and it has as many neurons as the number of values the MLP must learn to predict from the inputs. The MLP is trained using a loss function that measures a dissimilarity between the labels  $y$  and the outputs of the network  $\hat{y}$ . This dissimilarity is measured differently in classification and regression tasks so, while MLPs can be used for both, the choice of loss function will depend on the nature of the task. Also, since some loss functions require the labels and the network outputs to be in certain formats, the choice of the loss function influences what activation function is used in the output layer of the network. Popular loss functions include the Mean Squared Error for regression

---

and the Binary Cross-Entropy for binary classification.

---

## 2.3 Unsupervised Learning

Unsupervised learning is a class of problems where no labels are provided. Every example  $D_i$  is exclusively composed of inputs  $x_i$ . In this class of problems, the goal is that the model learns something about the structure of the data. Many problems fall in the category of unsupervised learning, the following list describes some of the most common ones :

- Density estimation : the model attempts to learn the probability density function (PDF) of a distribution  $F$  such that, given a new data point  $x^*$ , it can express how likely  $x^*$  is under  $F$ .
- Synthesis : the model attempts to learn to generate previously unseen examples from the distribution  $F$ . The problem of speech synthesis falls under that category; given a set of words, the model must generate a valid sound wave. For the speech to sound genuine and natural, words must not be pronounced exactly the same way every time. Thus the model must have the capacity to generate very varied but still valid pronunciations of each word.
- Clustering : the model attempts to discover natural groups in the data such that the elements of the same group are related to each other in some way. K-means Clustering and Spectral Clustering (?) are widely used clustering methods.

The following subsections describe models for unsupervised learning that are both common and relevant to this thesis.

### 2.3.1 Autoencoders

Autoencoders are a widely used class of unsupervised neural network models. They have one hidden layer and one output layer and are trained to reconstruct their own inputs. As such, autoencoders are trained to minimize the reconstruction error (often the Mean Squared Error when the inputs are continuous and the Binary Cross-entropy when the inputs are binary or continuous between 0 and 1). The hidden layer of the autoencoder is often referred to as the 'encoder' and the

---

output layer as the 'decoder' as a way of expressing that the hidden layer converts the inputs to the representation learned by the autoencoder and the output layer converts it back to its original format.

The main objective of the autoencoder is to learn a new and useful representation for the data. For this reason, standard autoencoders are often trained with under-complete representations (a hidden layer smaller than the input layer) to avoid the case where the autoencoder simply learns nothing about the data itself and simply learns an identity mapping between the input and output layers. To allow the use of over-complete representations (hidden layers larger than the input layers), new varieties of autoencoders have been introduced.

These varieties of autoencoders include the Denoising Autoencoder (DAE) introduced in ? and the Contractive Autoencoder (CAE) introduced in ?. In the DAE setup, the inputs are corrupted with stochastic noise and the model is trained to reconstruct the clean version of its inputs. This prevents the model from simply learning the identity mapping from its inputs to its outputs and forces it to learn the structure of the data. In the CAE setup, a new term is added to the reconstruction loss that the autoencoder attempts to minimize. This term, expressed in equation 2.5, is equal to the Frobenius norm of the Jacobian of the hidden layer  $h$  with regard to the input  $x$ . This penalty forces the encoder to be as insensitive as possible to variations in the inputs and thus, only remain sensitive to variations that are important to reconstruct them correctly.

It has recently been shown that it is possible to sample from an autoencoder by starting from a randomly initialized sample and alternating between a reconstruction step through a trained autoencoder and a step where the sample is corrupted with stochastic noise (?).

$$\sum_{i,j} \left( \frac{\partial h_j(x)}{\partial x_i} \right)^2 \tag{2.5}$$

### 2.3.2 Restricted Boltzmann Machine

A Restricted Boltzmann Machine, or RBM, is an undirected probabilistic graphical model that learns a probability distribution over a set of binary input variables. It models the input variables  $x$  (also designated "visible units") as being explained by a set of unknown factors  $h$  (also designated "hidden units").

---

The RBM is an *energy-based model* which means that it assigns a scalar value, the “energy”, to each possible configuration of values for the units in  $x$  and  $h$ . Under energy-based models, the unnormalized probability of a configuration is proportional to the exponential of its negative energy. Thus, the probability of a configuration of  $x$  and  $h$  is obtained as in equation 2.6 where  $E(x,h)$  is the energy assigned by the model to the values of  $x$  and  $h$  and  $Z$  is the partition function : the sum of  $e^{-E(x,h)}$  for all possible configurations of  $x$  and  $h$ . The probability of a configuration of  $x$  can be obtained by marginalizing out  $h$  as in equation 2.7.

This means that training a Restricted Boltzmann Machine is done by tuning the parameters that define its energy function so as to minimize the energy of desired configurations of  $x$  and  $h$  and maximize it for other configurations. Many algorithms exist to do this, with the most common ones being Contrastive Divergence (?) and Persistent Contrastive Divergence (?), but they are not explained here as they fall outside of the scope of this thesis.

$$P(x, h) = \frac{e^{-E(x,h)}}{Z} \quad (2.6)$$

$$P(x) = \frac{\sum_h e^{-E(x,h)}}{Z} \quad (2.7)$$

Many variants of the Restricted Boltzmann Machine have been proposed over the years. The ones that are relevant to this thesis are the Gaussian Restricted Boltzmann Machine (GRBM) (?), which has continuous visible units instead or binary, and the Spike-and-Slab Restricted Boltzmann Machine (ssRBM) (?). The ssRBM is a variant of the RBM where the binary hidden units are replaced with two sets of interacting hidden units : a set of binary hidden units (the “spikes”) and a set of continuous units (the “slabs”).

---

## 2.4 Deep learning

Deep learning refers to the idea of training model with many levels (“layers”) of representation. This is desirable because deep models have the potential to be exponentially more efficient than shallow models in terms of number of parameters

---

(??). Until recently the only way to efficiently train deep neural models was by using convolutional networks.

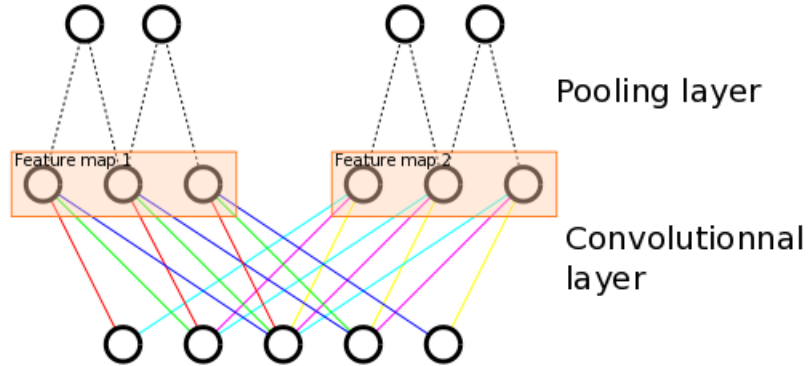
Convolutional networks (?) are neural networks like multilayer perceptrons but they differ from standard MLPs in three ways; local connectivity, weight sharing and pooling. These three elements are integrated in convolutional networks by the use of two new types of layers: convolutional layers and pooling layers.

In a convolutional layer, each neuron is only connected to a small spatially-contiguous subset of the layer’s input. These neurons are grouped in “feature maps”. A feature map is defined as a set of neurons which share weights and bias but differ in that they apply those weights to different subsets of the layer’s input. A convolutional layer can have as many feature maps as desired. A convolutional layer allows a network to detect multiple features at various locations in the inputs by using only a small set of parameters which makes the network more efficient and easy to train.

A pooling layer is usually used right after each convolutional layer. In a pooling layer, subsets of the layer’s inputs are aggregated together by using a pooling function. Popular pooling functions include the mean, maximum and sum functions. The pooling layer has the effect of reducing the size of the representation and making the network invariant to small spatial changes in the input. Figure 2.3 illustrates the architecture obtained by employing a convolutional layer followed by a pooling layer.

More recently, deep non-convolutional networks have been trained by stacking shallow unsupervised models such as Restricted Boltzmann Machines (?) and Autoencoders (??). To perform this, a first shallow model is trained on the inputs themselves. Then, a second shallow model is trained on the representation learned by the first shallow model. The process can continue like this for as long as desired and the parameters of the shallow models are used to initialize a deep neural network. This process is called *greedy layer-wise unsupervised pretraining* and is often followed by a phase of *fine-tuning* in which the layers of the deep network are trained together.

The process of jointly training a deep neural network without any sort of unsupervised pretraining is now possible thanks to a variety of techniques and models that have appeared and/or have become widely used in the last few years. The following subsections introduce some of those which had the largest impact: Rectified



**Figure 2.3** – A convolutional layer followed by a pooling layer. In the convolutional layer, connections of identical colors share weights.

Linear Units, Maxout Units, momentum and dropout.

### 2.4.1 Rectified Linear Units and Maxout

Rectified Linear Units (ReLU for short) and Maxout Units are new types of neurons that have become widely used as hidden units in neural networks. They have been shown to work well in a variety of tasks and to help tremendously in training larger and deeper neural networks.

A ReLU is very similar to a sigmoid unit and differs only in the activation function. Equation 2.8 describes the Rectified Linear activation function of the ReLU unit.

$$\text{RectifiedLinear}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.8)$$

A Maxout unit has  $N$  sets of weights and  $N$  biases which it uses those to compute  $N$  linear activations. The activation of the Maxout unit is the maximum value of the linear activations. Explained differently, a Maxout unit can be defined as a max-pooling over  $N$  linear units. When  $N = 1$ , a Maxout unit is simply a linear unit. As  $N$  grows, the Maxout unit is able to model more and more complex

---

functions.

### 2.4.2 Momentum

Momentum is a technique that can be used to improve the speed and performance of stochastic gradient descent. In standard stochastic gradient descent, whenever the model is trained on a minibatch of data, the updates to the model's parameters  $\Theta$  is equal to the learning rate times the gradient of the loss with regard to  $\Theta$ . With momentum, the updates are equal to a moving average of the gradient of the loss over the current minibatch *and* the previous minibatches.

In momentum-enhanced stochastic gradient descent, the updates take the form described in equation 2.9 where  $\alpha$  is a parameter, between 0 and 1, controlling the strength of the momentum.

$$\begin{aligned}\Delta_t &= \gamma_t \cdot \nabla_{\Theta_t} L(D, \Theta_t) + \alpha \cdot \Delta_{t-1} \\ \Theta_{t+1} &= \Theta_t - \Delta_t\end{aligned}\tag{2.9}$$

Momentum allows to speed up training by acquiring velocity in the directions where the gradients on consecutive minibatches agree.

### 2.4.3 Dropout

Dropout is a regularization technique for neural networks and it has been used to achieve state of the art performance on a number of tasks. Dropout works in the following way: every time the neural network is presented with a training example, each hidden neuron in the network is dropped with probability  $p$ . The neurons that are dropped do not participate in the neural network for this example and they also do not receive any gradient for this example. When the network is used outside of training, every neuron is kept but the outgoing weights of the hidden layers have their weights scaled by  $p$  so that the expected inputs of the neurons of the next layer will remain the same.

The process of randomly dropping units limits *co-adaptation* (units that learn to rely on the presence of one another) between neurons. It has also been demonstrated that dropout is roughly equivalent to averaging together all the different neural networks that can be obtained by independently keeping or dropping each hidden neuron in the network on which dropout was used during training (?). In



---

a neural network with 100 hidden neurons, there are  $2^{100}$  possible models resulting from keeping or discarding each neuron. This makes dropout a very efficient form of model averaging.

It has been empirically shown that the optimal value for  $p$  tends to be close to 0.5, making it a good default value (?).

# 3

## Disentangling factors of variation

The notion of “disentangling factors of variation” refers to the capacity of a model to learn a representation that teases apart the various factors of variation that explain the input data. ? argues that this trait makes for a better representation because a representation that disentangles the factors of variations allows a model to reason about a subset of those factors while being invariant to the others. Also, in opposition to pooling which discards information to achieve invariance, disentangling the factors of variation does not imply information loss as it is, in essence, a form of information reorganization to make it easier to work with. Most research efforts in this regard have focused on trying to achieve this property in an unsupervised or semi-supervised setting.

---

### 3.1 Explicit Orthogonality Enforcing

? introduces a semi-supervised approach for the task of emotion recognition in images of faces. This approach, called Contractive Discriminative Analysis (CDA), is based on the contractive autoencoder. In essence, the CDA is a CAE where the hidden units have been partitioned into two groups,  $h^{(o)}$  and  $h^{(d)}$ . Both  $h^{(o)}$  and  $h^{(d)}$  are used to compute the reconstruction of the input but  $h^{(d)}$  is also used as inputs to a logistic regression trained to predict the emotion so it also receives a supervised signal related to that task. Finally, a penalty (see equation 3.1)

$$\gamma \sum_{i,j} \left( \frac{\partial h_i^{(d)}(x)}{\partial x} \cdot \frac{\partial h_j^{(o)}(x)}{\partial x} \right)^2 \quad (3.1)$$

is added to the cost that the model tries to minimize to attempt to make every neuron in  $h^{(o)}$  and  $h^{(d)}$  learn a feature that is orthogonal to the features learned by the neurons in the other group. This penalty achieves this by penalizing - for

---

every pair of neurons where one is in  $h^{(o)}$  and the other is in  $h^{(p)}$  - the square of the dot product of their partial derivatives with regards to the input  $x$ . The results presented by ? demonstrate that the neurons in  $h^{(d)}$  learn features that are highly discriminant for emotions while the neurons in  $h^{(o)}$  rather tend to learn features related to the facial structure of the subjects.

---

## 3.2 Bilinear models

Many researchers have investigated the use of bilinear models for the task of disentangling the factors of variations in the data. In general terms, bilinear models are models in which the outputs are generated by two sets of latent factors that interact multiplicatively to produce it. If any of these two groups of factors is held constant, the outputs of the models are linear in the factors of the other group. The multiplicative interaction between the factors allows the efficient modeling of rich interactions between the factors which makes the bilinear models well suited to the task of teasing apart the factors of variation.

? describes two simple shallow bilinear models and explores their use in separating style and content in a variety of tasks; spoken word classification in a multi-speaker setting, extrapolation of fonts to unseen letters and generalization of faces to new lighting conditions. These models describe the inputs as a weighted sum of the interaction between elements  $a_i$  of the vector  $A$  (representing style-related latent factors of variation) and elements  $b_j$  of the vector  $B$  (representing content-related latent factors of variation).

In the first model, the symmetrical model, the weight for the interaction between  $a_i$  and  $b_j$  is dependent only on  $i$  and  $j$ . In the second model, the asymmetrical model, this interaction weight is also dependent on the value of  $a_i$ , allowing style-related factors to modify how the interactions are combined to generate the inputs. Both of these models, especially the asymmetrical model, are shown to offer good performance in generalizing style to new content.

? introduces a more complex bilinear model, the *disentangling Boltzman Machine* (disBM) similar to the Restricted Boltzmann Machine. Unlike the standard RBM, the bilinear RBM has its hidden units partitioned into two sets who interact multiplicatively between them. The authors observe that the training of such a

---

model often gives mediocre results but that it is possible to improve the results significantly by using labels or by learning by correspondence.

Using the labels is relatively straightforward. In the task of emotion recognition, for example, the authors add connections from one set of hidden units to labels related to emotion and from the other set of hidden units to labels related to the identity of the subjects. Training the model in this setting enforces that the first group of hidden units will learn features related to emotions and the second one to identity. The idea of learning by correspondence is to take advantage of known relationships between training examples. For example, if two data points are known to match according to some factor of variation, the hidden units modeling this factor of variation can be constrained to take the same value for these two data points. This constraint can also be relaxed and the model instead pressured to make those hidden units take similar values between matching data points and dissimilar values between non-matching data points. This second approach is designated “manifold-based training”.

Finally, the paper demonstrates the efficiency of the resulting disBM in representing rich interaction by showing that a small disBM can have much better performance than even an RBM with four times more hidden units.

? uses an approach very similar to the disRBM and introduces a new bilinear model, the Higher Order Spike-and-Slab Restricted Boltzmann Machine (hoss-RBM) based on the ssBRM. In the Higher Order Spike-and-Slab Boltzmann Machine, 2 sets of binary hidden units ( $g$  and  $h$ ) interact multiplicatively and their interactions are modulated by a set of continuous hidden variables ( $s$ ) and gated by a third set of binary hidden units ( $f$ ). Thus the Higher Order Spike-and-Slab Boltzmann Machine models multiplicative interactions between four sets of hidden variables.

To make the interactions between the hidden variables more sparse, and thus, more manageable for the model to learn with,  $g$  and  $h$  are restricted to interact according to a block-like structure : the first  $n$  units of  $g$  interact only with the first  $m$  units of  $h$ , the next  $n$  units of  $g$  interact only with the next  $m$  units of  $h$ , and so on. Thus a block corresponds to a subset of  $g$  interacting with a subset of  $h$ . Each interaction between a unit from  $g$  and a unit from  $h$  is scaled by a variable from  $s$  and each block is itself gated by a unit in  $f$ .

The authors argue that this interactivity structure can be interpreted as a form

---

of multi-way pooling. Within a block where  $n$  units of  $g$  interact with  $m$  units of  $h$ , the interaction between the  $g$ s and the  $h$ s can be seen as forming a matrix of size  $n \times m$  where each  $g_i$  pools over row  $i$  of the interaction matrix and each  $h_j$  pools over column  $j$ . Other than the elements mentioned so far, a key difference between the `hossRBM` and the `disRBM` is that the former is trained in a fully unsupervised setting, receiving no information from labels during training.

# 4 Learning with Noisy Labels

The concept of noisy labels refers to the idea of training on data for which some of the labels have been corrupted. This topic has received increasing attention in the last few years, presumably due to the difficulty in obtaining large datasets of clean data. Therefore, the use of large datasets, necessary to train large and complex neural models, often requires the capacity to deal with the noise present therein. Most of the research on this topic assumes a binary classification task and label noise that follows one of the three following noise models :

- Classification Noise Process (CNP) : a model for label noise in a binary classification task introduced by ?. This model assumes an oracle able to flawlessly sample elements from the right distribution  $D$  but that the oracle will report the wrong class with probability  $n < 0.5$  and the right class with probability  $1 - n$ .
- Class Conditional Noise (CCN) : a model similar to CNP but the probability that an example's label will be misreported depends on the example's class. CNP is a subset of the Class Conditional Noise model.
- Adversarial Noise : a model in which the oracle knows the right label for every sample but is allowed to willingly misreport some of them. Given a limit on how many labels can be misreported, the oracle tries to maximize the detrimental effect on the learner's generalization performance. ? and ? give examples of adversarial label noise strategies against a Support Vector Machine learner.

---

## 4.1 Surrogate Loss Functions

A fair amount of research has been done in coming up with *surrogate losses*, alternate loss functions that aim to minimize the effect of the label noise on training.

---

This is because many of the loss functions often used to train classifiers have very poor tolerance to noise in the labels, as demonstrated by ? in a theoretical study of the most commonly used loss functions.

In the context of class conditional noise, ? proposes a surrogate loss for the Hinge loss (the loss used to train SVMs). Given the parameters of the conditional noise model, this surrogate loss applied to a set of noisy examples is an unbiased estimator of the Hinge loss on the set of corresponding clean examples. This approach has a major drawback, however. Whereas training an SVM using the Hinge loss is a convex optimization problem, training an SVM using this surrogate loss is not. This is an important distinction because a convex optimization problem has only one minimum, the global minimum. Finding the global minimum is thus as easy as finding a local minimum. A non-convex optimization problem offers no such guarantee and it is therefore possible for the optimization process to end up in a local minimum without ever discovering the global minimum.

? builds on the previous approach and describes a method for modifying any loss applied to noisy examples so that it will become an unbiased estimator of the loss on clean examples. It shows that, if the original loss is convex and its second derivative satisfies a simple symmetry property, the surrogate loss thus obtained will also be convex. It also describes a second method for obtaining a proxy loss, based on the observation that the optimal decision functions on the clean training set and the noisy training set differ only in the threshold value they use to assign a label to an example. This second method is merely a loss function where the loss of each example is scaled by a label-dependent weight so as to modify the threshold used in the decision function. Both of these methods require that the parameters of the class conditional noise model be known. If they are unknown, they can be approximated using cross-validation but both methods benefit from being provided the right values.

---

## 4.2 Optimization Methods

Some of the proposed approaches directly alter the way the models update their weights in an effort to minimize the unwanted effect of the corrupted labels on the model parameters. In that direction, one of the earliest papers is ? who describes

---

a noise tolerant version of the Perceptron (?) algorithm by adding a correction factor to each weight update.

In the same spirit ? introduces Passive-Aggressive, a new online algorithm for binary classification which very aggressively updates the model parameters every time it doesn't correctly classify a training example by a wide enough margin. When this occurs, the model parameters are updated by however much is needed to make the model correctly classify the training example. This paper also demonstrates that it is possible to improve the algorithm's resistance to label noise by introducing a new hyper-parameter to control how aggressively the parameters are updated.

? builds on the Passive-Aggressive framework but uses a different approach which they call confidence weighting. Their updated algorithm keeps track of a measure of confidence in each of the model's parameters. This confidence is then used to constrain the updates to the model's parameters so that the more confident parameters are updated less aggressively than the less confident ones.

The AROW online learning algorithm introduced in ? reuses the concept of confidence weighting but changes how it is applied during training, using it as a regularization on the model instead of a hard constraint of the weights updates.

---

## 4.3 Label Confidence Modelling

Finally, there are other approaches which attempt to assign a measure of confidence to labels: low confidence if it seems likely that the label for a given example has been corrupted and high confidence if it seems likely.

For instance, ? employs the Expectation Maximization (EM) (?) algorithm to attempt to simultaneously learn the parameters of a classifier and the probability that the label has been flipped for each training examples. This approach has been shown to work well with high noise rates on small datasets but the algorithm's high computational cost restricts its ability to scale to large datasets.



# 5

## Multi-task Learning

Multi-task learning (?) refers to the idea of using data for one or more tasks as a way to improve the performance of a learner on a different task. There are multiple ways of implementing multi-task learning. It is often implemented as a simple model that is simultaneously trained on every task and learns a representation that is shared between all of them but it is also sometimes implemented as a collection of models, each trained on its own task, with some mechanism for sharing information between them.

is a classical example of the first scenario : a single deep neural network is trained to simultaneously perform a variety of natural language processing tasks; part-of-speech tagging, chunking and named entity recognition (all of these tasks consisting in attributing tags indicating the semantic or syntactic roles of words or groups of words in a sentence).

In some cases of multi-task learning, we care about the joint performance of the model on these tasks. In others, we only care about performance on one of these tasks (designated as “*main task*”, “*target task*” or “*task of interest*”) and thus attempt to leverage data from the other tasks (“*side tasks*” or “*extra tasks*”) to improve the performance of the model on the task of interest. Multi-task learning is sometimes referred to as ‘learning to learn’(?).

The core intuition behind multi-task learning is that similar tasks tend to require the learner to learn similar features and concepts. The resulting effect is that, very often, features that are useful for one task will be useful for another similar one. In the context of computer vision, for example, the detection of people’s eyes and mouths in images are two very closely related tasks. At a very low level of abstraction, both tasks require the learner to be able to detect edges and corners at various scales and orientations. At a higher level of abstraction, these tasks interact even more; knowledge of the position and orientation of the eyes in a picture is often enough to formulate a reasonable guess as to the position and orientation of the mouth. Similarly, the position of the mouth gives a strong prior on the location of

---

the eyes. It follows that a model learning to identify both the eyes and the mouths in pictures can rely on this relationship between the two tasks to solve both of them more easily. Multi-task learning can also be seen as a form of regularization; by forcing a learner to be competent at many tasks, it is harder for it to overfit any single one of them.

The notion of 'similar tasks' is subtle. While some tasks are obviously similar and some obviously dissimilar, there is a sizeable grey area between these two extremes. ? tackles the problem of rigorously defining the concept of "related tasks". It argues that it is far from trivial to possess full knowledge of whether two tasks are close enough that multi-task learning will provide any gain but proposes a heuristic definition for "task relatedness" : "Two tasks are related for [multi-task learning with gradient back-propagation] if there is correlation (positive or negative) between the training signal for one task (or more correctly the back-propagated errors for that task's training signal) and what is learned in the hidden layer for the other task when they are trained together"(?). The authors also demonstrate that the correlation between tasks is not necessarily an adequate proxy for task relatedness by showing how two completely uncorrelated tasks can greatly benefit each other in the context of multi-task learning.

---

## 5.1 Learning Task Relationships

Some researchers have proposed approaches where models learn the relationship between the various tasks so as to better leverage the similarities, or dissimilarities, between them. Many of the approaches that fall in this category rely on a 'linear task space' hypothesis (?). This hypothesis assumes that tasks reside in a linear latent subspace of unknown dimension. The basis of this subspace is a set of unknown tasks and, as a result, any task can be expressed as a linear combination of those basis tasks. It also hypothesizes that the representation of most tasks in that subspace is sparse; that is, only a small number of the basis tasks need to be combined to express almost any new task. The linear task space hypothesis does not appear to have ever been formally validated or invalidated but it has nonetheless given rise to methods that perform very well.

---

Based on the 'linear task space' hypothesis, ? introduces the Latent Independent Component Analysis (LICA) algorithm, inspired by Independent Component Analysis (ICA) (?). The ICA is an algorithm to learn, given some data, the independent factors that have been linearly combined to generate this data. The LICA is a hierarchical version of ICA with two levels of latent factors : LICA jointly learns the factors of each task that give rise to the data for that task and the global latent factors that give rise the task-specific factors themselves. The paper compares factors learned by LICA and ICA by using them as new representations for the data and feeding them in a logistic regression model and shows that LICA outperforms ICA, especially in the case where there is very little data for each task. ? and ? build on this work by introducing probabilistic frameworks to jointly learn the task space and the factors associated with each task.

? employs a different approach to relate the tasks to each other. It performs a form of clustering of the tasks themselves. Then, it trains a model where some of the weights are task-independent and some are task-dependent. The task-independent weights are shared across all tasks. The task-dependent weights change between tasks and their values for a given task are a function of the task's cluster assignment. This way, tasks that are very similar tend to have similar task-dependent weights while dissimilar tasks have very different task-dependent weights.

---

## 5.2 Learning Global Features

Some of the approaches that have been explored explicitly attempt, given some tasks, to discover features that are useful across all the tasks. ? proposes an algorithm to learn shared features with a model that has no hidden layer. It essentially works by alternating between two steps; a supervised step in which the model learns on every task independently and an unsupervised step in which the model learns a low rank representation of its weights matrix. This unsupervised step is shown to exert a pressure on the model to make its features similar across tasks. The model's propensity to share features across tasks can be controlled by making the model more or less aggressive in learning a lower rank representation during the unsupervised step.

---

Alternating Structure Optimization (ASO), introduced by ? also operates by alternating an independent learning step on every task and a grouping step which encourages the features to be similar across tasks. In the unsupervised step, ASO processes the weights that have been learned so far with Singular Value Decomposition (SVD) and keeps only a fixed number of the leading left-singular vectors of the decomposition. In the supervised step, these leading singular vectors are used to learn new independent weights for every task. In ASO, the unsupervised step can be seen as keeping from the weights only the most predictive directions and thus encouraging the model to learn features that are useful across tasks.

? build upon ASO with the improved Alternating Structure Optimization *i*ASO. In ASO, the problem of learning features common to a set of tasks is formulated as a non-convex optimization problem and, as such, there is no guarantee that the global minimum will be reached by the optimization process. In *i*ASO the problem is reformulated as a convex one and a new algorithm *c*ASO is proposed to efficiently solve it.

Unlike approaches like ASO and *i*ASO which attempt to *learn* features that are useful across tasks, other approaches attempt to *select*, from the inputs, those that possess this usefulness. ? introduces a method which involves sharing *regularization*, not parameters, between models learning on distinct tasks. Specifically, it penalizes, for every input, the Euclidean norm of the vector containing the weights that each of the models associate with that input. This pressures the models into learning similar sparsity patterns for each of the models. If this regularization is strong, the model learns to rely only on a few of the inputs which are relevant for most of the tasks.

# 6

## Prologue to First Article

---

### 6.1 Article Details

**MNIST+: a simple benchmark for factor disentangling** Pierre Luc Carrier, Guillaume Desjardins, Aaron Courville, and Yoshua Bengio

The main contribution of this article is a new suite of datasets to allow easier benchmarking of methods for factor disentangling and multi-task learning. The article also benchmarks popular machine learning models for each of the introduced datasets.

This article has not yet been published.

---

### 6.2 Individual Contributions

Guillaume Desjardins came up with the original idea for this dataset and he wrote the first version of the code to generate it. I then slightly modified this code to generate the final version of the dataset.

I generated the final version of the dataset and performed the benchmark experiments on it. This included writing the necessary code to perform the experiments, doing the hyper-parameter optimization as well as running all the experiments. Aaron Courville and Yoshua Bengio suggested the commonly used machine learning models on which to benchmark the datasets.

Finally, I wrote the article itself and generated the figure included it it.

# 7

# MNIST+: a simple benchmark for factor disentangling

---

## 7.1 Introduction

In this technical report, we introduce a new benchmark dataset for handwritten digits classification. This dataset is intended to allow to easily benchmark models and techniques that aim to learn representations that disentangle the factors of variation of the data. This dataset is based on the MNIST dataset, the well-known digit recognition benchmark dataset, to which various factors of variation have been added to make it suitable for this task.

---

## 7.2 Motivation

Deep learning, which has recently brought many successes to the field of machine learning, refers to the idea of training a “deep” model. Such a model is composed of multiple levels of computation, or layers, able to capture increasingly abstract properties of the inputs. As previously argued by some researchers (?), the performance of such deep models is highly dependent on the quality of the representations learned by the model’s various layers.

In the classification setting, for instance, it is easy to see that if the last layer takes as input a representation that captures nothing of the data distribution, it is bound to fail. On the other hand, if that representation contains the information required by the output layer, performances are expected to be higher.

This leads one to ask what are the properties of a “good” representation and ? explores that very question. The authors argue that being *distributed* (the ratio of the number of possible inputs configurations captured by the representation over the size of the representation should be high), *invariant* (the representation should

---

be invariant to local changes in the input) and *disentangling the factors of variation* of the data are amongst the desirable qualities for a representation.

The MNIST+ dataset, which we introduce in this technical report, aims to allow easier experimentation and evaluation of last this property. To that end, MNIST+ is really a dataset suite. It includes four variants of the same dataset, each building on the previous one by adding a new factor of variation to it. This allows to see, in a controlled setting, how models behave when dealing with data that exhibits progressively more and more degrees of freedom.

---

## 7.3 Previous Work

The MNIST+ dataset is based on the MNIST handwritten digits dataset (?) a benchmark for the computer vision task of digit recognition. It is a simple but very widely used dataset comprising 70000 images of size 28x28 pixels.

The MNIST Variations dataset (?) was created a few years ago for a similar purpose as the MNIST+ dataset. It is based on the MNIST dataset to which new factors of variation (rotations and/or replacing the background of the MNIST digits by random noise or patches from natural images) are added.

The MNIST+ dataset suite differs from the MNIST Variations dataset in the nature and the number of factors that are simultaneously added to the data. It contains four distinct variants, each adding a new factor of variation over the previous one. It also differs from MNIST Variations in that it provides labels for the variations introduced in the data, allowing multiple ways to experiment with the factors of variation in more ways than simple vanilla digit classification. This includes using these new labels in the context of multi-task learning or even feeding them as additional inputs to the model.

---

## 7.4 Content and characteristics

The MNIST+ dataset suite contains four distinct datasets, or variants of the same data. Each one of those variations adds a new factor of variation to the data,

---

making it more complicated, and therefore harder, than the other versions before it. This allows us to observe the behaviour and performance of machine learning techniques and see how they scale as the data they are used on starts to vary in new ways. It allows to better see the difference between the models that scale well to highly multi-modal data and appropriately disentangle the factors of variation present in it and the models that don't.

Each dataset in MNIST+ was created from the MNIST dataset using the following process applied to every example in MNIST:

1. If the dataset includes the factor of variation “rotation” perform a random spatial rotation of the MNIST image with rotation angle  $\theta$  sampled uniformly from  $[0, 2\pi[$ . Bicubic interpolation was used to obtain the new pixel values resulting from the rotation.
2. Rescale the MNIST digit from its original size of 28x28 pixels to 48x48 pixels using bicubic interpolation.
3. If the dataset includes the factor of variation “texture”, sample a 48x48 patch from a random location in a randomly selected texture of the Brodatz dataset (?) and add the MNIST digit on top of it; for every pixel position, we use the pixel from the MNIST digit if its intensity is higher than 0.1 and the pixel from the texture otherwise.
4. Apply an embossing filter on the MNIST digit. The embossing filter takes an image and an angle describing the position of a light source as inputs and produces an output image of the same size as the input image where the pixel intensities are decided according to the changes in pixel intensities in the input. Regions of the inputs where pixel increase in intensity as they get further away from the light source will produce regions of light pixels in the output. The sharper the increase in pixel intensity, the lighter the output region. On the other hand, regions of the inputs where pixel decrease in intensity as they get further away from the light source will produce regions of dark pixels in the output. In this case, the sharper the decrease, the darker the pixels. Finally, regions of the inputs where the intensities of the pixels vary little will produce grey regions in the output. If the dataset includes the factor of variation “azimuth”, the angle of the light source is sampled independently for each image from  $[0, 2\pi[$ . Otherwise, a light source angle



---

of  $\frac{3\pi}{4}$  is used for the whole dataset.

This process is used to create the four variants of the MNIST+ dataset :

- MNIST+ Basic (MNIST+B) : Obtained by resizing the digits in the MNIST dataset and applying the embossing operation. It mainly serves to establish the difficulty of the MNIST dataset transformed with scaling and embossing before adding new factors of variation to it.
- MNIST+ Texture (MNIST+T) : Uses the “texture” variation on top of the scaling and embossing transformations.
- MNIST+ Texture and Rotation (MNIST+TR) : Employs both the “texture” and the “rotation” variations on top of the scaling and embossing transformations.
- MNIST+ Texture, Rotation and Azimuth (MNIST+TRA) : Employs every additional factor of variation (“texture”, “rotation” and “azimuth”) on top of the scaling and embossing transformations.

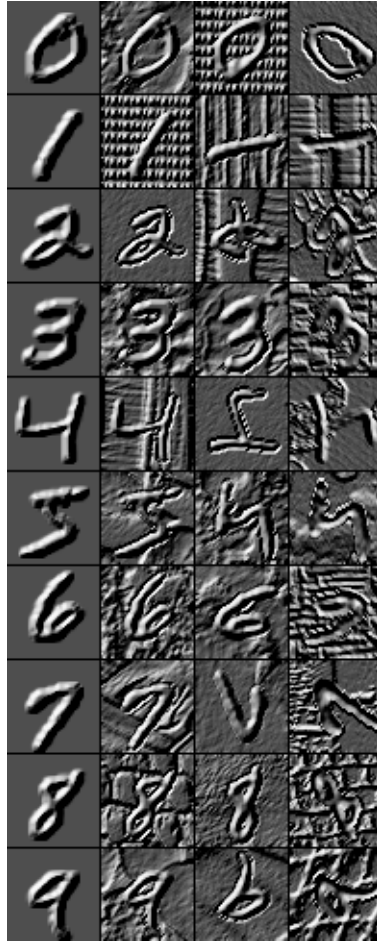
Every dataset in MNIST+ has a training set comprising 50000 examples (created from the first 50000 examples in the MNIST training set), a validation set of 10000 examples (created from the remainder of MNIST’s training set) and a test set of 10000 examples (created from the digits of the MNIST test set). The figure 7.1 shows digits samples from every variant in MNIST+.

---

## 7.5 Benchmarks

In order to provide an estimation of the difficulty level of this dataset, as well as provide a reasonable starting point for future comparison between different models and techniques, we have selected a few models commonly used in computer vision and trained them on every variation of the MNIST+ dataset. The table 9.2 shows the best classification error that was obtained for each of the model classes by using the hyper-parameter optimization process described below. The same process was used for each of the MNIST+ variations :

- Linear SVM : We trained linear SVMs using 30 distinct values for the regularization hyper-parameter C, between 1e-4 and 1e2 and roughly equally



**Figure 7.1** – Samples from the variants of the MNIST+ dataset suite. From left to right, the columns show samples from the “Basic”, “Texture”, “Texture and Rotation” and finally “Texture, Rotation and Azimuth” variants. The samples for each variant have been generated from the same digits in the original MNIST dataset.

spaced in log-scale. The SVMs were trained in a one-versus-all fashion meaning that, for each class, an SVM was trained to differentiate between the examples of that class and the examples of the remaining classes.

- RBF Kernel SVM : The hyper-parameter optimization was done in much the same way as for the linear SVM. The only difference is that using the RBF kernel introduces a new parameter  $\gamma$ . For this new hyper-parameter, we selected 15 values between  $1e-3$  and 1 and roughly equally spaced in log-scale. We trained SVMs in a one-versus-all fashion for each pair of values for  $C$  and  $\gamma$ , resulting in a total of 450 hyper-parameter combinations being used.

- 
- Maxout Networks (?) : For each type of Maxout network (convolutional or fully connected), we performed a very coarse grid search to identify a reasonable region in the hyper-parameter space and then launched 200 experiments in that hyper-parameter region using random search. For the fully connected networks, we experimented with anywhere between 0 and two hidden layers before the output layer. For the convolutional networks, we tried between one and three convolutional layers followed by up to two fully-connected hidden layers before the output layer. For both fully connected and convolutional networks, we used momentum (?) with momentum coefficients between 0.3 and 0.95 for the momentum coefficient and we also used dropout (?) with  $p = 0.5$ .

<b>Mnist+ variant</b>	<b>Mnist+B</b>	<b>Mnist+T</b>	<b>Mnist+TR</b>	<b>Mnist+TRA</b>
Linear SVM	6.95%	23.03%	63.08%	89.15%
RBF Kernel SVM	2.39%	16.99%	52.02%	70.96%
Fully connected Maxout	1.37%	8.46%	41.06%	56.44%
Convolutional Maxout	0.45%	0.84%	3.19%	5.19%

**Table 7.1** – Benchmarking results on the variants of the MNIST+ dataset. Performance is expressed in proportion of misclassified digits in the test set.

The benchmark results shown in table 9.2 illustrate several interesting properties of this dataset and of the models used. First and most obvious is the gradual increase in difficulty as new factors of variation are added to the data. Every model considered has its accuracy steadily decreasing as the tasks become more difficult. The results on the Texture, Rotation and Azimuth variation are also very interesting. Most models are wrong over 50% of the time with the linear SVM even performing only marginally better random guess. The convolutional network, however, does reasonably well on this MNIST+ variant which demonstrates its ability to generalize very well in the presence of data with a high number of factors of variation and only very few training examples for each combination of those factors.

---

## **7.6 Dataset availability**

The MNIST+ dataset can be obtained by contacting the authors of this article. Should you use this dataset for research purposes, please cite this technical report.

# 8

## Prologue to Second Article

---

### 8.1 Article Details

**Facial Expression Recognition 2014 Dataset** Pierre Luc Carrier, Aaron Courville, and Yoshua Bengio

The main contribution of this article is a new dataset for the task of facial expression recognition. This new dataset is larger than previous ones for this task both in the number of images and in the number of distinct facial expressions. This article also redefines the facial expression recognition task as a *tagging task*, or *detection task*, meaning that facial expressions are not generally mutually exclusive. Finally, the article also benchmarks popular machine learning models on the newly introduced dataset.

This article has not yet been published.

---

### 8.2 Individual Contributions

Aaron Courville came up with the idea of automatically collecting images from the Web to assemble a dataset for the task of facial expression recognition. With some help from Aaron Courville, I came up with the specific process by which to collect the images which I then implemented and executed.

I implemented the tool to allow the labelling of the collected images. Aaron Courville provided useful suggestions which I implemented. These include the idea of having the users label the images that were the results of the same image search query together, in the same order as they were returned by the image search engine. Since the first images returned are more likely to show the desired facial expression, this allowed the users to stop labelling the images associated with a given query once the image search engine started returning a low proportion of valid results.

---

I came up with the idea of using face detection tools like OpenCV and Picasa to obtain initial bounding boxes around the faces in the collected images and thus reduce the number of times the users had to draw these bounding boxes themselves. I also had the idea of using the Keystroke-Level Model to make the labelling interface more efficient, greatly reducing the labelling time.

With the implemented tool I labelled the majority of the images collected but also received the help of many colleagues of the LISA lab who labelled the rest of the images. Yoshua Bengio is the one who enlisted the help of these colleagues, organized the labelling activities and managed the human labellers.

Then, I performed the benchmark experiments on the dataset. Like for the MNIST+ article, this included writing the necessary code to perform the experiments, doing the hyper-parameter optimization as well as running all the experiments themselves. Aaron Courville suggested the models on which to benchmark the datasets. Finally, I wrote the article and generated the figures present in it.

# 9

# Facial Expression Recognition 2014 Dataset

---

## 9.1 Introduction

This technical report introduces a new dataset, the Facial Expression Recognition 2014 (FER-2014) dataset. It contains a large number of images of faces, collected from the internet using Google’s Image Search service, and is intended to serve as a new benchmark dataset for the task of facial expression recognition from images.

---

## 9.2 Previous Work and Motivation

The current trend of scaling up machine learning efforts to ever-bigger models fuels the need for bigger quantities of quality data to train those models. This is especially true when models are trained to perform very subtle or complex tasks. Facial expression recognition is one of such tasks.

Multiple datasets already exist for this task with the Toronto Face Database (TFD) (?), a concatenation of many other existing datasets, being one of the most popular. It contains a few thousands images, each displaying one of seven basic emotions (anger, disgust, fear, happiness, sadness, surprise or neutrality). Many of these images were captured in conditions of controlled lighting and background and/or feature subjects acting the desired emotions in an exaggerated fashion instead of genuinely expressing them. Moreover, the images were preprocessed to eliminate variance in subject pose. This makes for a dataset that is easier to learn from, but it also makes it different from the kind of data on which humans perform facial expression recognition every day.

The FER-2014 dataset seeks to improve upon the Toronto Face Database in both the quantitative and the qualitative aspects. To achieve this goal, we as-

---

sembled this data with images collected from the Web. Since this process of data collection is cheaper than hiring actors, it allowed us to collect a larger number of images at small financial cost. Moreover, since the images come from very varied sources, they have the potential for much more variability in terms of background, lighting conditions, subject identity, the way the subjects express emotions, etc. Finally, while some of the images available on the internet show subjects who display acted and/or exaggerated facial expressions, the majority of them show subjects that display expressions that seem both spontaneous and genuine.

---

### 9.3 Data collection process

The first step of the data collection process was to establish a list of 184 emotion-related keywords. Most of them were either the name of an emotion or an adjective related to one. By combining those keywords with words related to people's identity (ethnicity, age and sex), we assembled a total of 788 query strings.

For each of the query strings assembled, a query was made on the Google Image Search service, using the option to return only images containing a face, and up to the first 1000 results were kept (many queries did not return as many results). For every image obtained this way, we also collected the URL of the image, the small caption displayed by Google alongside the image as well as other information. Following this, we used OpenCV (?), an open-source computer vision library, and the Picasa software (?) to perform face detection on every image collected and obtain the coordinates of the biggest face present in it. OpenCV performs its facial detection feature by employing a cascade decision tree based on Haar features (??).

Once the data were collected according to the process defined above, it was manually filtered by human labellers. For every query, the labellers split the images into two groups : those in which the face found by OpenCV displayed a facial expression compatible with the emotion-related keyword used in that query and those where it wasn't. For images in which OpenCV fails to detect a face or produces an invalid bounding box, labellers were tasked with drawing a new bounding box around a face present in the image, if possible, and then classify the image in the same way than if that face had been found by OpenCV. If no face was found in the image, then it was simply discarded. To maximize the use of the labellers'



---

time, they processed the images in the same order as they were returned by Google and, whenever they processed 10 images without seeing one with the desired facial expression, they stopped processing that query and moved on to the next one.

At that point of the process, for each of the 184 emotion-related keywords that we used, we had positive examples (images that are known to contain the corresponding emotion) and/or negative examples (images that are known to contain a different emotion). We grouped together the keywords that were synonyms or had nearly identical meaning and then dropped those for which we had no positive examples. This left us with 18 distinct facial expressions for which we had both positive and negative examples.

Of those 18 expressions, some were mutually exclusive (like sadness and happiness, for instance) but some were not (like surprise and happiness). For every pair of expressions we found to be mutually exclusive, we made the positive examples of each expression also be negative examples of the other expression. However, since the 18 expressions are not all mutually exclusive, the task of facial expression recognition on the FER-2014 dataset is not one of classification, but tagging.

The data collection and filtering process described above is not particularly complex, but it was designed to maximize labelling speed to allow the collection of a large dataset at minimal cost. Using OpenCV and Picasa as a first attempt at face detection reduced the number of bounding boxes that labellers had to draw by approximately 75%. Similarly, in small-scale tests that we performed, filtering a group of images based on a single emotion proved much faster than independently labelling each of them with its most likely emotion, even when considering that the former method ends up discarding some of the images whereas the latter keeps every one of them. Additionally, we used the Keystroke Level Model (?), often used in UI Design, to produce a very efficient GUI for labellers to use. Combining these three elements, individual labellers were able to achieve filtering rates of up to 3000 images per hour.

---

## 9.4 Relationship to the Facial Expression Recognition 2013 Dataset

During the data collection process for the FER-2014 dataset, a subset of it was preprocessed and released as the Facial Expression Recognition 2013 (FER-2013) dataset in the context of the ICML2013’s Challenges in Representation Learning Workshop for the facial expression recognition task recognition challenge (?). In the FER-2013 dataset, unlike the FER-2014 dataset, the emotions are considered mutually exclusive. This makes it a classification task as opposed to a tagging task.

The FER2013 dataset was also used by a team at University of Montreal’s machine learning laboratory, the LISA, to win the Emotion Recognition In The Wild Challenge and Workshop (EmotiW 2013) competition (?). In that competition, the task was to automatically recognize emotions played out by actors in short video sequences taken from popular movies (?). The FER2013 dataset proved instrumental to that victory by drastically increasing the amount of training data available, thereby allowing to train bigger, more complex and better performing neural networks.

To assemble the FER2013, each of the 184 emotion-related keywords used to collect images was mapped to one of the following 7 emotions : anger, disgust, fear, happiness, sadness, surprise or neutral. This mapping was then used to assign a class to every image that had been collected at that point and that had been labelled as a positive example for any of the 184 keywords used. The negative examples for each keyword were ignored because it was impossible to automatically infer the correct classes for them.

As a result of this process, the FER-2013 contained 35887 images with 4953 for the class “Anger”, 547 for “Disgust”, 5121 for “Fear”, 8989 for “Happiness”, 6077 for “Sadness”, 4002 for “Surprise”, and 6198 for “Neutral”.

---

## 9.5 Content and characteristics

The FER-2014 dataset contains a total of 73125 images, with 58334 of them composing the training set, 7297 making up the validation set and the remaining

---

Facial expression	Nb. positive examples	Nb. negative examples
Anger	4332	32376
Boredom	479	27039
Concern	704	27064
Crying	951	28842
Disappointment	271	25781
Discouragement	934	33069
Disgust	604	36517
Displeasure	899	25552
Elation	2050	23264
Fear	1770	33891
Happiness	10869	23451
Nervousness	1618	33361
Neutral	4700	32861
Sadness	1764	35390
Screaming	1797	23668
Shame	128	33324
Surprise	4170	23558
Tiredness	263	12726

**Table 9.1** – Contents of the FER-2014 dataset

7494 images composing the test set.

However, not all images in the dataset have labels for every facial expression. For this reason, the table 9.1 describes, for each of the 18 facial expression in this dataset, the number of images that have been labelled as positive and negative examples of this expression. The figure 9.1 shows representative examples of the faces present in the FER-2014 dataset.

---

## 9.6 Benchmarks

In order to provide an estimation of the difficulty level of this dataset, as well as provide a reasonable starting point for future comparisons between different models and techniques, we have selected a few of the commonly used models in computer vision and trained them on the FER-2014 dataset.

To run these benchmarks, we preprocessed the dataset by expanding the smallest dimension of every bounding box to obtain a squared bounding box for each



**Figure 9.1** – Samples from the FER-2014 dataset

face, extracting the faces using those new bounding boxes and then resizing the faces to 48 by 48 pixels. This expansion of the bounding boxes ensures that the extracted faces could be resized to 48 by 48 pixels without altering their aspect ratios. We then performed feature standardization on the dataset (normalize each feature so that it has 0-mean and unit standard deviation). This is not guaranteed to be the best preprocessing scheme for this data. It is, however, reasonable enough to see the differences in performance between different models and get a sense for the complexity of the task of facial expression recognition on this dataset.

The table 9.2 shows the best performance that was obtained for each of the model classes by using the hyper-parameter optimization process described below. The performance metric used in these benchmarks is the Mean Average Precision as used in the Visual Object Classes Challenge 2010 (?). In this variation of the Mean Average Precision metric, before computing the area under the precision-recall curve, every precision value is set to be equal to the maximum of the precisions associated with equal or higher recalls.

- Linear SVM : For each one of the 18 binary detection tasks, 30 linear SVMs were trained using 30 distinct values, between  $1e-4$  and  $1e2$  and roughly equally spaced in log-scale, for the hyper-parameter  $C$ . For each of the 18 detection tasks, the SVM with the best average precision was kept, resulting in 18 SVMs whose performance were averaged to obtain the mean average precision across tasks.

---

Model	Test performance
Linear SVM	0.2390
RBF Kernel SVM	0.4259
Fully connected Maxout Network	0.3866
Convolutional Maxout Network	0.4961

**Table 9.2** – Benchmarking results on the FER-2014 dataset. The performance is expressed in Mean Average Precision. The best possible value is 1.0 and the worst possible value is 0.0

- RBF Kernel SVM : The hyper-parameter optimization was done in much the same way as for the linear SVM. The only difference is that using the RBF kernel introduces a new parameter  $\gamma$ . For this new hyper-parameter, we selected 15 values between 1e-3 and 1 and roughly equally spaced in log-scale. We trained SVMs for each pair of values for  $C$  and  $\gamma$ , resulting in 450 SVMs being trained for each binary detection task. Again, the performance reported in table 9.2 is the mean of the best performance obtained for each detection task.
- Maxout Networks (?) : For each type of Maxout network (convolutional or fully connected), we performed a very coarse grid search to identify a reasonable region in the hyper-parameter space and then launched 200 experiments in that hyper-parameter region using random search. For the fully connected networks, we experimented with anywhere between 0 and two hidden layers before the output layer. For the convolutional networks, we tried between one and three convolutional layers followed by up to two fully connected hidden layers before the output layer. For both fully connected and convolutional networks, we used momentum (?) with momentum coefficients between 0.3 and 0.95 for the momentum coefficient and we also used dropout (?) with  $p = 0.5$ .

---

## 9.7 Discussion

The process used to collect the data is not without faults. Even with the precautions taken to ensure a proper level of quality in the data, some noise is

---

still present in the emotion labels. Discussed below are the main sources of noise identified in the data collection and labelling process :

- The process of mining images from the Web with image search queries has the unfortunate effect that some of the properties of the distribution of images found online can be reflected in the FER-2014 dataset. For instance, individuals often prefer to put online pictures of themselves looking happy rather than looking ashamed. Because of this, queries relying on keyword related to happiness tended to return many more images than those using keywords related to shame.
- The mining process also has the effect of making some properties of the language in which the queries were made (english) reflect in the dataset. For example, the more commonly used in conversation a keyword is, the more results the queries using it tend to return. Also, the greater the number of keywords related to a given emotion, the greater the number of queries that can be done for that emotion. The combination of these two effects explains why the dataset contains much more data for emotions like 'happy' and 'surprise' than for 'shame' or 'disgust'.
- The task of facial expression recognition is not an easy one. While exaggerated or very strong expressions are often simple to recognize, milder and genuine facial expressions are much more difficult. The recognition of such expressions often relies on subtle cues and different people tend to express emotions differently. This makes it difficult, even for humans, to assign the right label to an image. Because of this, it is very possible that the labels assigned to some images are wrong.

---

## 9.8 Obtaining the data

The Facial Expression Recognition 2014 dataset is available on request from the authors of this technical report. This includes the original images and the coordinates of the faces bounding boxes, as well as the preprocessed 48x48 version that was used for the benchmarks. The previously mentioned additional informations that were collected at the same time as the images are also included in the dataset;

---

URLs and image caption amongst others. Should you use this dataset for research purposes, please cite this technical report.

---

## 9.9 Additional Information

The table [9.3](#) details the 184 emotion-related keywords that were used, and combined with identity related keywords, to generate the query strings which were then used to perform image searches.

---

afraid	disappointed	grief	peevd
aghast	disappointment	grumpy	perplexed
agitated	discouraged	guilt	perturbed
agony	disgust	happiness	petrified
alarmed	disgusted	happy	pleased
amazed	dismal	hateful	proud
amused	dismayed	heartbroken	rage
anger	displeased	heartsick	raging
angered	dissatisfied	heavyhearted	regret
angry	distraught	hopeful	regretful
annoyance	distressed	horrified	regretting
annoyed	distrustful	humiliated	relieved
anxious	disturbed	humiliation	remorse
appalled	doubt	hurting	resentful
ashamed	dreading	in awe	sad
astonished	ecstatic	infuriated	sadness
astounded	elated	intimidated	satisfaction
awe	elation	irate	satisfied
bashful	embarrassment	irked	scared
bitter	ennuied	irritated	screaming
blissful	enraged	jealous	shame
bored	enthusiasm	jealousy	shamed
chagrined	envious	jolly	sheepish
cheerful	envy	joy	shock
cheerless	euphoric	joyful	shocked
chirpy	exasperated	joyous	shy
concern	exasperation	jubilant	sorrowful
concerned	excited	laughing	stress
confident	exhaustion	low-spirited	stupefied
confused	exhilarated	mad	sulky
confusion	exhilaration	melancholy	surprise
contempt	exultant	merry	surprised
contrite	fear	mirthful	suspicion
coward	fearful	misery	tense
crestfallen	fed-up	morose	ticked off
crying	ferocious	mortified	tired
defiant	flustered	mournful	troubled
dejected	fretful	mourning	unhappy
delight	frightened	nervous	unimpressed
delighted	frustrated	not happy	upset
depressed	frustration	not impressed	vexed
depression	fuming	offended	weary
desiring	furious	outraged	weeping
despaired	glad	overjoyed	worried
despairing	gleeful	panic	wrathful
despondent	gloomy	panic-stricken	yelling

**Table 9.3** – Emotion-related keywords used to assemble image search queries



---

## 10.1 Article Details

**Discriminative Disentangling via Multi-way Pooling** Pierre Luc Carrier, Ian Goodfellow, Aaron Courville, and Yoshua Bengio

This article contributes two new methods, for the multi-task setting, to improve the performance of a neural network on a target task by leveraging data with labels for other tasks. The first method is a generalization of Maxout Networks designated Multi-way Pooling Maxout and the second is a discriminative application of a previously introduced Contractive Discriminant Analysis (CDA) penalty.

This article has not yet been published.

---

## 10.2 Individual Contributions

Yoshua Bengio deserves the credit for motivating and pushing the objective of learning models that disentangle the factors of variation in the data. It is this objective that eventually led to the techniques introduced in this paper.

Aaron Courville and Ian Goodfellow came up with the original idea for the Multi-way Pooling Maxout, while the supervised application of the CDA penalty is an idea both Aaron Courville and I had. I performed experiments to explore these ideas, the results of which have allowed Aaron Courville and I to refine these ideas until they reached the form in which they are introduced and used in this article.

I defined the process to obtain noisy labels for the task of identity recognition from the side informations provided in the FER-2014 dataset and also performed all the experiments detailed in this article. Doing these experiments involved implementing the different approaches introduced in this paper, performing the hyperparameter optimization as well as executing the experiments themselves.

---

Aaron and I collaborated to write the article, resulting in most sections in the article being the product of joint work with the exception of the discussion section which I wrote by myself based on my personal analysis of the experimental results presented in the paper.

# Discriminative Disentangling via Multi-way Pooling

---

## 11.1 Introduction

There has been a tremendous amount of recent interest in the application of deep learning methods to discriminative tasks, such as classification. This interest has been fuelled by recent high profile successes in a number of key areas of application such as speech recognition (?) and object recognition (?). These successes seem to hinge on the use of very large datasets as well as the application of high capacity models that can efficiently and effectively take advantage of large amounts of labelled data.

While this is undoubtedly true, this perspective ignores that the successes of machine learning also often share the characteristic that they rely on architectures and techniques that promote *invariance*. This includes large convolutional networks in which the compounded effect of many pooling layers leads to a high level of spatial invariance. This also includes Maxout which performs a form of cross-channel pooling which results in a different kind of invariance. Contractive Autoencoders, which are explicitly trained to be as invariant as possible to their inputs without compromising their capacity to reconstruct their inputs, also fall in this category.

Invariance allows a model to easily generalize its behavior to new, previously unseen, configurations of the inputs, likely improving its generalization performance. Of course, not any invariance will do. A model that is invariant to the factors of variation in the data that are needed to perform a certain task will surely do poorly because it has learned to discard the very information that it needs to perform this task well. On the other hand, a model that is sensitive only to the factors of variation needed for its task and invariant to every other direction of variance in the data is likely to perform well in a variety of conditions. This makes invariance, under the assumption that it is to factors unrelated to the task, a very desirable

---

property. Elaborating further on the concept of invariance leads to the idea of *disentangling*.

**Disentangling** Disentangling is the notion of a learned representation teasing apart the factors of variation in the data so that every factor has its own subset of the representation that is sensitive to it while the rest of the representation is invariant to this factor.

In a sense, disentangling is a *reorganization* of the information in the data that makes it easier to consider some variations in the data while being invariant to others. Because of this, disentangling is arguably preferable to simple invariance: it provides the same benefits of easily allowing models to only be sensitive to some factors of variation but it does so with no, or minimal, loss of information in the process.

**Our approach** The problem of disentangling factors of variation has, so far, mostly been explored in fully unsupervised or a semi-supervised settings (????). This paper aims to tackle this problem in a purely discriminative training paradigm and explores the question of whether or not *labels* can be sufficient to disentangle the factors of variation that give rise to the data.

Our approach can be seen as a multi-task approach. It relies on the notion that solving a task requires sensitivity to a subset of the factors of variation in the data. An unrelated task on the same data will rely on different factors of variation. If the model can successfully be trained to solve each task while learning for each task a representation that is invariant to the representation for the other task, it will have successfully disentangled the factors of variations that the first task relies on from those that the second task relies on.

In this paper, we attempt to employ this idea to improve a model’s performance on a task of interest, or target task, by leveraging data from other tasks, side tasks, that are unrelated to the target task and on which the model’s performance is not important to us.

In particular, we are interested in the setting where the labels for the side tasks are noisy. This is because collecting clean data for every side task can often prove costly in both money and man-hours. On the other hand, large amounts of noisy data are often easy to obtain at little or not cost and, therefore, the ability to

---

efficiently leverage them to improve a model’s performance is highly desirable.

In this paper, we introduce two approaches in an attempt to achieve this goal of disentangling the factors of variation, both of which rely on a generalization of Maxout to the multi-task setting.

The first approach modifies Maxout’s pooling structure itself, resulting in a multi-way pooling generalization of Maxout. Maxout employs a form of cross-channel pooling to achieve invariance to some directions of variance in the data and we generalize that pooling structure to a higher dimension in which a Maxout layer can produce multiple representations, each of them sensitive to distinct directions of variance.

Our second approach does not alter the model but rather the optimization process itself. It relies on the recently introduced Contrastive Discriminant Analysis (CDA) penalty (?) to force the model to learn representations for each task that are orthogonal to each other.

**Our contributions** In this paper we make three contributions to the literature. First, we formulate the discriminative disentangling task and relate it to similar multi-task learning formulations in the literature. We also formulate a multi-way pooling generalization of Maxout and a discriminative application of the previously proposed CDA penalty that both support the exploitation of the multi-task learning paradigm to build features that are explicitly invariant to irrelevant factors of variation present in the data. We compare these approaches on the tasks of handwritten digit recognition and facial expression recognition.

---

## 11.2 Previous work

As discussed in the previous section, the literature contains a few examples of *generative disentangling*: where the goal is to explicitly tease apart the factors that give rise to the variations in the data by – in some sense – modeling the *generative entangling process* that gave rise to the data.

Note that while we refer to these approaches collectively as generative disentangling, it is not meant to imply that there was no use made of the discriminative labels in learning the disentangled representation (as was done in both ? and ?).

---

While this line of work is undoubtedly important and has led to very interesting results, it is our view that there are likely simpler approaches that could disentangle the factors of variations without having to rely on modelling their entangling process.

One disadvantage in attempting to explicitly model the generative entangling process is that it will suffer the same disadvantage as every generative approach, that the true underlying generative process is far more complex than what current models can represent and that the power of the disentangling is limited accordingly.

Another, perhaps more pressing and relevant criticism is that the kinds of factors we often consider – and put forth as examples of the kinds of factors we wish to disentangle (facial identity cues, such as age and gender, and emotional expression remain the most popular choice) – are typically fairly abstract in comparison to the kinds of low level features we typically see in the lower layers of all deep learning models.

Building a generative model that would faithfully reflect the generative interaction of factors such as age, gender and emotional expression seems like an extremely challenging path towards the exploitation of our prior knowledge that we would like classifiers trained to classify one such factor should be invariant to the remaining factors.

In some sense, we view the factors that are often used as being very high level and not directly or at least readily available in low-level processing of the data (such as in ?, where generative disentangling was attempted in the first hidden layer).

Our approach is quite different. Here we do not attempt to model the generative entangling process. Our perspective is to attempt to take advantage of the recent progress in the discriminative setting while still recognizing the need to disentangle the factors underlying the variations in the data.

The work that most closely resembles ours in addressing the problem of discriminative disentangling is that of ?. They also cast the problem of learning to be invariant to certain entangled factors of the data as a multi-task learning problem.

While the problem statements are similar, the approaches we’ve taken are quite different. Their approach is able to leverage both side tasks that are positively correlated with the target task and side tasks that are negatively correlated with it. On the other hand, our approaches rely on the side tasks to be, starting at a

---

certain level of abstraction, unrelated to the target task. Another key difference is the fact that their approach is employed in the context of a linear model whereas our approaches are easily integrated into non-linear models.

---

## 11.3 Maxout Networks

Our approach to discriminative disentangling is based on a state-of-the-art neural network architecture, the Maxout architecture (?).

A Maxout network is a feed-forward neural network architecture that predicts an output  $y$  given input vector  $x$ . These architectures contain a series of hidden layers  $\mathbf{h} = \{h^{(1)}, \dots, h^{(L)}\}$  that can be interpreted as transformed, intermediate representations of the data. The network parameters  $\theta$  parametrizes a family of conditional distributions  $p(y | x; \theta)$  over the output variable.

Where Maxout is distinguished from other feed-forward neural network architectures is in its unit activation function. Each unit in the network uses a piecewise linear, convex activation function. Specifically, given a unit input  $x \in \mathbb{R}^d$ , (where  $x$  could be the input  $v$  or the hidden unit activations of the previous layer) a Maxout hidden unit implements the function

$$h_i(x) = \max_{j \in [1, k]} z_{ij}$$

where  $z_{ij} = x^T W_{\dots ij} + b_{ij}$ , and  $W \in \mathbb{R}^{d \times m \times k}$  and  $b \in \mathbb{R}^{m \times k}$  are learned parameters.

A single Maxout unit can be interpreted as learning a piecewise linear approximation to an arbitrary convex activation function. The more linear pieces  $z_{ij}$  the Maxout unit  $h_i$  max-pools over, the better the approximation. This means that Maxout networks learn not just the relationship between hidden units, but also the activation function of each hidden unit. This is trivially generalized to the convolutional setting where a Maxout feature map can be constructed by taking the maximum across  $k$  affine feature maps (making every value in the Maxout feature map the result of both spatial and cross-channel pooling).

Maxout units are similar to another popular activation function: the rectified linear unit (?). The difference between the two of them is that, whereas a Maxout

---

unit’s activation is the maximum taken over two or more linear functions of its inputs, a rectified linear unit’s activation is the maximum over a single linear function of its inputs and the value 0. This makes it much more likely for a rectified linear unit to have an activation of 0 than it is for a Maxout unit, which often makes for a sparser representation.

In the next two sections, we introduce the architectural innovations we make to Maxout in order to extend it to the multi-task setting with irrelevant tasks.

### 11.3.1 Multi-task Maxout

Our first contribution is really a straightforward extension of the Maxout network to the multi-task setting. Our extension follows previous neural network approaches to multi-task learning (such as ?). In this work we employ multi-task Maxout for classification and tagging tasks, but nothing prevents the use for regression tasks.

In the multi-task Maxout setting, the neurons of the first few layers of the Maxout model are common to all tasks. Then, starting at a given layer  $m \in \{1, L\}$ , the neurons of the model are partitioned into separate and non-interacting columns, or paths, through the model. Each path eventually produces the output(s) for one of the tasks. That is, every neuron starting at hidden layer  $m$  is associated with a single task. The neurons of layer  $m$  take as inputs the activations of neurons at layer  $m - 1$  as usual, but the neurons at layer  $m + 1$  or above can only take as inputs the neurons at the layer below that are associated with the same task.

With this model change, feed-forward inference and learning proceed as expected. The loss function used to train the model’s parameters is a simple weighted sum of the losses for each of the tasks that the model is trained on. Every task  $i$  other than the target task has its loss weighted by hyper-parameter  $\lambda_i$  (see equation 11.1).

$$\mathcal{L}_{MT} = \mathcal{L}_{TargetTask} + \lambda_1 \mathcal{L}_{SideTask1} + \lambda_2 \mathcal{L}_{SideTask2} + \dots \quad (11.1)$$

When a label is missing for a particular task-example pair, no error is back-propagated through the network for this pair.



---

## 11.4 Multi-way pooling

One important conclusion that one might draw from the relative success of Maxout compared to rectified linear methods is that the cross-channel pooling that Maxout employs (where we interpret channels here to be the pre-max linear filters of the Maxout unit) is an important part of its success. Empirically, Maxout seems to shine relative to rectified linear models in settings where rectified linear models had a tendency to overfit the training data to a point where validation and test performance started to degrade. Thus we might interpret the cross-channel pooling strategy as a method of building in additional invariance in the feature representation that allows the model to be more robust to overfitting the training data.

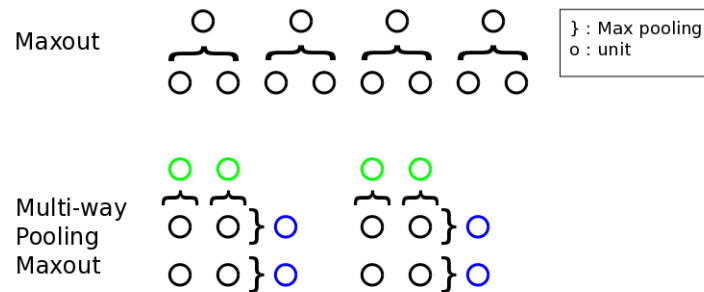
Our multi-way pooling strategy extends this kind of cross-channel pooling mechanism in the service of disentangling factors of variation in a multi-task setting. We describe the model details in the simplified setting where we have one target task and one irrelevant task. The framework readily extends to a higher number of tasks.

To understand multi-way pooling Maxout, it helps to visualize a Maxout layer as a simple linear layer followed by a pooling layer. In a standard Maxout layer, the pooling is done by organizing the linear pieces in non-overlapping groups of size  $n$  and the result of max-pooling over each of those groups is the output representation of the Maxout layer and gets used as input by the next layer.

In a multi-way pooling Maxout, the pooling is done by organizing the linear pieces in non-overlapping groups of dimension  $d \geq 2$  and pooling over each group  $d$  times, each time pooling over all dimensions except  $i$  for  $i \in \{1, d\}$ .

In the multi-task setting, we employ multi-way pooling Maxout at layer  $m - 1$  (where layer  $m$  is the first layer where neurons are partitioned according to task). We use blocks with the same number of dimensions as the number of tasks the model is trained on and – instead of taking all the values that result from the max-pooling operations and use them all as inputs to the neurons of the next layer – the values that were obtained by pooling over all the block dimensions except the first one are used as inputs for the neurons of the next layer assigned to task 1, those that were obtained by pooling over all the block dimensions except the second one are used as inputs for the neurons of the next layer that are assigned to task 2, etc.

For example, in a multi-task setting with two tasks, the linear pieces are organized in rectangular blocks of size  $n \times m$  and max-pooling is done twice over each block, first pooling over columns for every line in the block (resulting in  $n$  values) and then pooling over lines for every column in the block (resulting in  $m$  values). The neurons of the next layer assigned to task 1 only receive as inputs the values obtained by max-pooling over the columns of each block while those assigned to task 2 receive only those obtained by pooling over the rows of each block. See figure 11.1 for a visualization of this.



**Figure 11.1** – Visualization of the pooling mechanism of Maxout and multi-way pooling Maxout. In Maxout, linear filters are grouped in one dimensional entities and aggregated via max-pooling resulting in a single representation for the next layer. In multi-way pooling Maxout, the linear filters are grouped in entities of higher dimensions and multiple orthogonal pooling operations are performed. In this case, the groups have two dimensions and so the pooling operations result in two representations that can be used for distinct tasks. The green units form the first representation and the blue units form the second.

Multi-way pooling creates multiple representations, each obtained by keeping and pooling over different directions of variance. In a multi-task setting, it allows to learn a representation for each task that is invariant to directions of variance that have been learned by the model as being useful, or discriminative, for the other tasks. This amounts to disentangling factors of variation in the data, based on what task they are useful for.

This multi-way pooling structure was inspired by a similar pooling structure used in ?. As previously stated in section 11.2, while that work was focused primarily on the use of probabilistic generative models, we are interested in how we can leverage explicitly irrelevant auxiliary tasks to help disentangle the factors of variation in a purely discriminative way.

---

## 11.5 Discriminative CDA

Our proposed multi-way pooling strategy is not the only conceivable strategy to discriminative disentangling in our context. In the context of the existing literature, it is perhaps not even the most obvious approach. Indeed, we can apply an adaptation of the semi-supervised Contractive Discriminative Analysis approach (?) to our purely discriminative framework. The resulting approach can reasonably be considered a non-linear generalization of the multi-task learning approach to discriminative disentangling proposed in ?.

### 11.5.1 Contractive Discriminative Analysis

CDA is a semi-supervised version of the Contractive Auto-Encoder (CAE). While the standard CAE encodes the data into a single feature vector  $h(x)$  – corresponding to a layer of a neural network architecture, CDA learns a mapping from an input into two<sup>1</sup> distinct feature vectors: one that encodes factors of its input that are associated with a target discriminative task,  $h^{(d)}(y) = s(Wy+c)$ , and one (or more) that encode all other factors,  $h^{(o)}(y) = s(Vy+b)$ . Both feature blocks are trained to cooperate to reconstruct their common input  $y$  with a reconstruction loss function, e.g.,

$$L_{\text{RECON}}(y, \hat{y}) = \|y - \hat{y}\|^2 \quad (11.2)$$

where  $\hat{y}$  is the CDA reconstruction, given by a linear combination of learned features:

$$\hat{y} = g([h^{(d)}(y), h^{(o)}(y)]) = s_2(W^T h^{(d)}(y) + V^T h^{(o)}(y) + \rho). \quad (11.3)$$

where  $\rho_i$  is an offset to capture the average value of  $y_i$ . In the CDA formulation, an additional discriminative component is added to the loss function to ensure that the discriminative features,  $h^d(y)$  are also predictive of the expression label  $z(y)$  when that information is available.

CDA enforces disentangling between the discriminative features and the other features (those meant to account for the remaining variance in the input) through

---

1. As discussed in ?, the framework readily generalized to more than two feature blocks

---

a CAE-inspired contractive penalty  $\mathcal{J}_{\text{CDA}}(y)$ :

$$\mathcal{J}_{\text{CDA}}(y) = \left| \frac{\partial h^{(d)}(y)}{\partial y} \right|_F^2 + \left| \frac{\partial h^{(o)}(y)}{\partial y} \right|_F^2 + \gamma \sum_{i,j} \left( \frac{\partial h_i^{(d)}(y)}{\partial y} \cdot \frac{\partial h_j^{(o)}(y)}{\partial y} \right)^2. \quad (11.4)$$

The first two terms penalize sensitivity in  $h^{(d)}(y)$  and  $h^{(o)}(y)$  respectively to local variations in  $y$  (as in the standard CAE). The third term encourages  $h^{(d)}(y)$  and  $h^{(o)}(y)$  to represent different directions of variation in the input  $y$ , by penalizing non-orthogonality between the sensitivity vector  $\frac{\partial h_i^{(d)}(y)}{\partial y}$  of each discriminant feature ( $h_i^{(d)}$  for all  $i$ ) and the sensitivity vector  $\frac{\partial h_j^{(o)}(y)}{\partial y}$  associated with each non-discriminant feature  $h_j^{(o)}$  (for all  $j$ ).

### 11.5.2 Learning orthogonal representations for orthogonal tasks

Adapting CDA to the discriminative setting is conceptually straightforward. We wish to incorporate our prior knowledge of the irrelevance of the set of side tasks to the target task. This can be done by reusing the third term of the  $\mathcal{J}_{\text{CDA}}$  and applying it on the neurons at layer  $m$  – where neurons are first divided into independent task-oriented paths through the model – between every group of neurons assigned to different tasks. Thus, the neurons assigned to different tasks are encouraged to be orthogonal to each other.

This explicitly enforces the prior that, from a certain level of abstraction, information regarding one task is irrelevant to other tasks and only represents undesired possible variations in the data.

---

## 11.6 Experiments

We explore and report the performances of the previously introduced approaches on two very distinct datasets: the MNIST+ dataset and the FER-2014 dataset.

---

### 11.6.1 Experiments on MNIST+

The MNIST+ dataset (??) is a recently introduced handwritten digit recognition dataset suite. It is based on the MNIST (??) dataset to which new factors of variation have been added to make it more challenging. MNIST+ contains four variants of the same data, each adding new factors of variations to the previous ones. The following experiments have been performed on the MNIST+ Texture variant in which every example have been generated by embossing a MNIST digits over a randomly selected patch from a random texture from the Brodatz dataset of textures (??).

In the following experiments on the MNIST+ Texture dataset, for every example, the target task is to identify the digit present in the example and the side task, when applicable, is to predict the label of the texture that was used to generate the example. Thus, both target and side tasks are classification tasks with, respectively, 10 and 112 classes.

Since in these experiment we care only about performance on the main task, the hyper-parameters  $\lambda$ , which weight the loss of every side task in the model's global loss function, are optimized as to maximize performance on the target task.

Table 11.1 gives the classification error reported in the MNIST+ technical report for a Maxout convolutional network using dropout (??) and momentum (??) trained only on the target task of digit classification. It also provides the performance for multi-task Maxout, multi-way pooling Maxout in a multi-task setting and multi-task Maxout with the CDA penalty.

Multiple models were trained for each approach; we report the test performance of the single model with the best performance on the validation set. For a fair comparison, we trained the models with dropout and momentum and using the same hyper-parameter optimization process as was used in ?? to train the Maxout convolutional network, with the exception that we also optimized the new hyper-parameters introduced by our approaches. The performance of the models is measured in term of classification error.

From these results, we can observe that both of our approaches offer a significant improvement in model performance over both the baseline Maxout convolutional network and the multi-task version of that baseline. In fact, our approaches more than double the improvement of the multi-task model over the baseline model with Multi-way Pooling Maxout providing nearly four times the reduction in classifica-

---

Model	Test performance
Maxout ConvNet	0.84 %
Multi-task Maxout ConvNet	0.80 %
Multi-way Pooling Maxout Convnet (multi-task)	0.66 %
Maxout ConvNet with CDA penalty (multi-task)	0.74 %

**Table 11.1** – Best performance obtained, for every considered approach, on the MNIST+ Texture dataset. Performance is given in misclassification errors.

tion errors in this particular setting.

### 11.6.2 Experiment on FER-2014

The FER-2014 dataset (?) is a dataset for the task of facial expression recognition in images. It features 18 non-exclusive facial expressions and, as such, models the facial expression recognition task as a detection, or tagging, task. Because of this, performance on this dataset is measured with the Mean Average Precision ordering metric as employed in the Pascal 2010 Visual Object Classes Challenge (?).

The images in the FER-2014 dataset were collected from results of image search queries in Google’s image search service and the dataset contains not only the images, but also additional data related to the image collection process: the original name of the image files, the captions displayed alongside each image in Google’s result page, etc.

The experiments performed on the FER-2014 dataset are very similar to those performed on the MNIST+ dataset. The difference lies only the nature of the target task and of the side task. On the FER-2014 dataset, the main task is facial expression recognition. For these experiments, we establish the side task as being the prediction of the sex and age category of the subjects in the image.

We obtained targets for these identity-related attributes by establishing a list of words associated with each possible value of these attributes. We then automatically processed every image, searching their Google captions and their original URLs for these words, and automatically inferred a value for these identity-related attributes.

The attribute “sex”, for instance, can be either male or female. For “male”, we used the keywords : “man”, “boy”, “male”, “guy” as well as their plural forms.

---

Model	Test performance
Maxout ConvNet	0.4961
Multi-task Maxout ConvNet	0.5106
Multi-way Pooling Maxout Convnet (multi-task)	0.5236
Maxout ConvNet with CDA penalty (multi-task)	0.5074

**Table 11.2** – Best performance obtained, for every considered approach, on the FER-2014 dataset. The performances are expressed in Mean Average Precision which lie in the interval  $[0, 1]$  with 0 being the worst possible performance and 1 being the best possible one.

For “female”, we used “woman”, “girl”, “female”, “lady” and their plural forms. For a given image, if the caption and/or the URL contains keywords from the male category, it is classified as showing a male subject. If they contain keywords from the female category, the subject is classified as female. If they contain keywords from both or from neither, the image is not given a value for the attribute sex. The same was done for the age attribute with the three different possible values being “baby”, “young” and “adult”.

This is a very noisy process to estimate the age and sex of subjects in pictures and we chose it because it reflects the level of noise that can be expected in data collected the internet without any manual cleaning process to remove noise in the labels. This gives an interesting possibility to witness how the proposed approaches fare in a real life setting.

Table 11.2 provides the results obtained on the FER-2014 dataset. Once again the baseline number is provided by the FER-2014 dataset’s technical report and was obtained by training a Maxout convNet with momentum and dropout on the single task of facial expression recognition. The other results were obtained by applying multi-task Maxout and our two proposed approaches in the same fashion as on the MNIST+ dataset. Once again, for a fair comparison, we trained the models with dropout and momentum and using the same hyper-parameter optimization process as the benchmark models to which we compare, with the exception that we also optimized the new hyper-parameters introduced by our approaches. The performance is given in terms of Mean Average Precision.

These results seem to indicate that Multi-way Pooling Maxout remains useful even when the labels for the side task exhibit a significant level of noise. At this level of noise, however, the CDA penalty no longer has any positive effect. In fact, in these experiments, the use of the CDA penalty harms performance, although

---

not by a large amount.

### 11.6.3 Experiment with noise control

While the use of the CDA penalty brought disappointing results on the FER-2014 dataset, our Multi-way pooling approach gave very good results both on MNIST+ and on the FER-2014 dataset, offering an improvement in performance over both the multi-task model and the single-task model. Since our results suggest Multi-way Pooling Maxout to be the most promising multi-task approach explored in this paper, we perform further experiments on it in the presence of controlled label noise.

In this section, we attempt to empirically explore Multi-way Pooling Maxout’s tolerance to noise in the labels of the side task. To achieve this, we revisit the MNIST+ dataset. We reuse the hyper-parameters that provided the best performance on this dataset and use them to train new models, but this time we introduce varying levels of noise in the labels for the texture classification side task. This allows to get a sense of how useful this approach is in various noise settings.

We consider noise levels of 0.0, 0.1, ..., 0.5 and 0.6 where the noise level is the proportion of the texture labels that have been randomly corrupted. For every noise level considered, we train three models reusing the previous hyper-parameters, but different random seeds. Figure 11.2 illustrates these results.

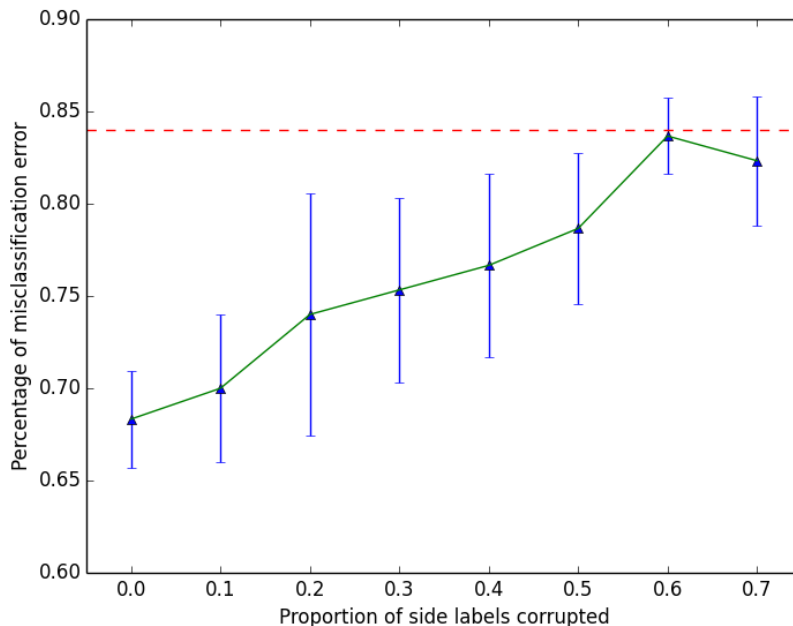
---

## 11.7 Discussion

The experiments above suggest that the Multi-way Pooling Maxout is an efficient way to leverage side information. On the MNIST+ Texture dataset, it offers a larger increase in performance than the standard multi-task approach. In the situations where these side informations have noise, like on the FER-2014 dataset, the improvement is more modest, but it still improves over the multi-task approach as much as the multi-task approach improves over the convolutional network trained on a single task.

On the other hand, the CDA penalty does not perform quite as well. The experiments on MNIST+ show that it offers a significant improvement over the





**Figure 11.2** – Performance of Multi-way Pooling Maxout (in green) on the MNIST+ Texture dataset with varying levels of noisy on the labels for the side task. Performance is expressed in term of misclassification error. For each level of noise, both the average and standard deviation are plotted. For comparison, the best performance obtained without using the side labels is shown in red.

standard multi-task approach, though not as high as Multi-way Pooling Maxout. However, when the labels for the side task start exhibiting noise, the performance falls drastically and, in our experiments on the FER-2014 dataset (see table 11.2), using the CDA penalty with the multi-task approach actually performed worse than multi-task Maxout, although only by a small amount.

The two following subsections explore the factors that could explain the performance of the supervised CDA penalty being lower than expected and avenues to explore to increase the tolerance of both methods the noise in the labels of the side tasks.

### 11.7.1 Supervised CDA

This section analyses the factors that could have contributed to the poor performance of our supervised application of the CDA penalty. It focuses mainly on the differences between our experimental setup and the experiments done in ? where,

---

it is important to remember, the CDA penalty was instrumental in obtaining state of the art performance on the task of emotion recognition on the Toronto Face Database dataset.

### **Logistic units and Maxout units**

One of the key differences between the experimental settings in ? and ours is that the former applies the CDA penalty between groups of *logistic* hidden units whereas we employ Maxout hidden units. This has the potential to make a huge difference because these two types of units have very different saturation behaviors.

This has to do with how the third term of the original CDA penalty (see equation 11.4), the term which promotes orthogonality, is defined.

Analysis of this third term shows that having the hidden units in different groups learn orthogonal features is not the only way to minimize this penalty. Specifically, it can be reduced by minimizing the partial derivatives of individual neurons with regard to their inputs, thereby making them less sensitive to variations in their inputs and making the scalar product of their partial derivatives and those of any other hidden unit closer to 0. The following is a very extreme example, but if all the units in one group of hidden units were perfectly invariant to their inputs, the value of the third term of the CDA penalty would effectively be equal to 0.

To reiterate, the third term of the CDA penalty can be minimized by making the groups of hidden units learn orthogonal features *or* by making some of the units as invariant as possible to their inputs. This is important because logistic and Maxout units differ very much in the ways they can be made more invariant to their inputs.

Recall that the output  $y$  of a logistic unit is obtained by applying the function described in equation 11.5 with  $W$  being the input weight vector of the logistic unit and  $b$  its bias. Taking the derivative of that function with regard to the inputs  $x$  yields equation 11.6. From this, we can see that sensitivity of a sigmoid unit with regard to its inputs can be done in three ways : making the output of the unit close to 0, close to 1 or making the weights  $W$  small. The first two ways can be achieved by making the weights and or biases large which will push the neuron towards more extreme activation values like 0 and 1. Thus, when the CDA penalty is applied to groups of logistic units, the model has three mechanisms it can combine to minimize of the penalty applied to it; it can make the neurons in different groups learn more

---

orthogonal features, it can increase drastically the weights and or biases of some individual neurons and it can make the weights of other individual neurons close to 0.

$$y = \sigma(Wx + b) \text{ with } \sigma(x) = 1/(1 + e^{-x}) \quad (11.5)$$

$$\frac{\partial y}{\partial x} = (y)(1 - y)W \quad (11.6)$$

On the other hand, because the output of a Maxout unit is simply the maximum over multiple linear activations, the derivative of a Maxout unit with regard to its inputs is simply the weights vector associated with the linear activation that dominated the maximum operation. This means that Maxout units cannot be made more invariant to their inputs by increasing the norms of their weights, only by reducing them. Thus, when the CDA penalty is applied to groups of Maxout units, the network has only two mechanisms it can combine: making the neurons learn orthogonal features and/or making the weights of some neurons closer to 0.

It is possible that this third mechanism that a Maxout model doesn't have access to, increasing the weights, is critical to the success of the CDA penalty. For instance, it allows a network to keep features even if they are not orthogonal. In the case of logistic units, if one unit learns a feature that is so important to the model that the gradient descent keeps pushing the norm of its weights upward, then this unit eventually saturates and the model doesn't have to pay a cost to keep this feature even if it is not orthogonal to the features learned in the other group of hidden units. The Maxout model doesn't have this option; it can either make the features orthogonal or reduce its confidence – reduce the weights – in some of those features.

### **Relationship between the tasks**

Another potentially key difference is that our supervised application of the CDA penalty depends strongly on the orthogonality between the tasks, a difficulty that the original Contractive Discriminant Analysis model did not have to face.

In our setting, each group of hidden units that the CDA penalty is applied on receives a supervised signal from some task so the model has to conciliate two goals, the first one being to make the hidden units in each group learn valuable

---

features for their assigned task and the second one being making those features orthogonal between groups. If the tasks do not rely on orthogonal features, the model’s attempt to make the features of different tasks orthogonal may lead to learning low quality features for all the tasks.

In the original CDA model, only one group of hidden units receives a supervised signal from a task and both groups receive a signal from the reconstruction task. In this context, the model has to conciliate two different goals : make the hidden units learn valuable feature for reconstruction and organize them such that the features useful for the supervised task are in the first group of hidden units and the features that are irrelevant to those are in the second group of hidden units. In this setting, the CDA penalty can almost be seen more as a feature selection mechanism than a constraint on the features that can be learned; it attempts to ensure that only the features relevant to the supervised task are learned in the first group of hidden units and the others are learned in the second group.

## 11.7.2 Tolerance to noise

While we care significantly for the setting where the labels of the side tasks are noisy, our approaches have not been explicitly designed to maximize their robustness in the presence of noise.

The literature on learning with noisy labels offers directions to explore to remedy this problem and improve our approaches’ tolerance to noise in the labels. The most obvious direction would be to employ the Hinge Loss for tagging and binary classification side tasks as it has been shown to be more tolerant to noise than the binary cross-entropy loss (?). It is also possible to instead use the procedures put forward by ? and ? which allow to adapt loss functions to minimise the effect of noise on a classifier.

Finally, we could consider an approach inspired by “confidence weighting” (??) to modify how the weights are updated depending on the confidence that the model has in them. This approach is interesting in the multi-task setting because, if we have knowledge of which tasks have noisy labels and which tasks have clean labels, we can leverage this information as a prior on the confidence that the models should have on each weight; the model can be confident about the weights used exclusively for tasks with clean labels, but it should be less confident about the weights used

---

for tasks with labels known to be noisy.

---

## 11.8 Conclusion

In this paper, we proposed two new approaches to improving a model’s performance on a task of interest by leveraging labels associated with a different and unrelated task. The first approach is a supervised application of a penalty to promote orthogonality between sets of neurons. That penalty was previously proposed in the context of the semi-supervised Contrastive Discriminant Analysis model. In this paper, we propose using this penalty in the multi-task setting to enforce that the hidden representations associated with different tasks be orthogonal to each other. The second approach is a generalization of Maxout in which we extend Maxout’s pooling structure to a higher dimension in which we can perform multiple orthogonal pooling operations by pooling over different dimensions.

In our experiments, Multi-way Pooling Maxout offered a sizable improvement in performance both on the MNIST+ dataset where the labels for the side task were noise-free and on the FER-2014 dataset where those same labels were noisy. It is also easy to implement and scales well to large neural models.

On the other hand, the supervised use of the CDA penalty provided a lesser, but still significant, improvement in the case of clean side labels but actually harmed performance in our experiments with noisy side labels.

We have explored possible explanations behind the performance of the supervised application of the CDA penalty as well as avenues to consider to improve the noise tolerance of both our proposed approaches. Both of these should prove interesting future research directions and help further our capacity to leverage noisy side information in a supervised setting.



# Bibliography