Université de Montréal

Estimation de pose 2D par réseau convolutif

par

Samuel Huppé

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

avril, 2021

© Samuel Huppé, 2021

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Estimation de pose 2D par réseau convolutif

présenté par

Samuel Huppé

a été évalué par un jury composé des personnes suivantes:

Jean Meunier
Président-rapporteur

Sébastien Roy
Directeur

Bernhard Thomaszewski
Membre du jury

Mémoire accepté le _____

À Chloé

RÉSUMÉ

Magic: The Gathering est un jeu de cartes à collectionner stochastique à information imparfaite inventé par Richard Garfield en 1993. Le but de ce projet est de proposer un pipeline d'apprentissage machine permettant d'accomplir la détection et la localisation des cartes du jeu *Magic* au sein d'une image typique des tournois de ce jeu. Il s'agit d'un problème de pose d'objets 2D à quatre degrés de liberté soit, la position sur deux axes, la rotation et l'échelle, dans un contexte où les cartes peuvent être superposées. À travers ce projet, nous avons développé une approche par données synthétiques à deux réseaux capable, collectivement d'identifier, et de régresser ces paramètres avec une précision significative. Dans le cadre de ce projet, nous avons développé un algorithme d'apprentissage profond par données synthétiques capable de positionner une carte avec une précision d'un demi pixel et d'une rotation de moins d'un degré. Finalement, nous avons montré que notre jeu de données synthétique est suffisamment réaliste pour permettre à nos réseaux de généraliser aux cas d'images réelles.

Mots-Clés: Vision par ordinateur, Estimation de pose, Apprentissage machine, Apprentissage profond, Autoencodeurs, Réseaux Convolutifs, Jeux de données synthétiques

ABSTRACT

Magic: The Gathering is an imperfect information, stochastic, collectible card game invented by Richard Garfield in 1993. The goal of this project is to propose a machine learning pipeline capable of detecting and localising *Magic* cards within an image. This is a 2D pose problem with 4 degrees of freedom, namely translation in x and y, rotation, and scale, in a context where cards can be superimposed on one another. We tackle this problem by relying on deep learning using a combination of two separate neural networks. Our final pipeline has the ability to tackle real-world images and gives, with a very good degree of precision, the poses of cards within an image. Through the course of this project, we have developped a method of realistic synthetic data generation to train both our models to tackle real world images. The results show that our pose subnetwork is able to predict position within half a pixel, rotation within one degree and scale within 2 percent.

Keywords: Computer Vision, Pose Estimation, Machine Learning, Deep Learning, Autoencoders, Convolutional Neural Networks, Synthetic Datasets

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE D'ACRONYMES

RGB	Rouge, Vert, Bleu (Red, Green, Blue)
PnP	Perspective à n Points (Perspective-n-Points)
FCN	Réseau de Neuronnes Complètement Convolutif (Fully Convolutional Neural Network)
FC	Complètement Connecté (Fully Connected)
TRS	Translation, Rotation et Échelle (Translation, Rotation and Scale)
RGBA	Rouge, Vert, Bleu, Alpha (Red, Green, Blue, Alpha)
TP	Tenseur Partitions (Partitionning Tensor)
NMS	Suppression de Non-Maxima (Non-Maximum Suppression)
ReLU	Unité Linéaire Rectifiée (Rectified Linear Unit)
Adam	Adaptive Moment Estimation
Yolo	You Only Look Once
CNN	Réseau de Neuronnes à Convolutions (Convolutional Neural Network)

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de recherche, Sébastien Roy qui a sû, à tout instant, m'insufler la motivation nécessaire pour amener à son terme ce projet. En effet, dans un contexte de pandémie et de travail à distance, Sébastien a été un guide hors pair. Sa patience, son écoute, et son support dans un contexte difficile se sont avérés déterminants da ma réussite. Merci.

Je tiens également à remercier mes collègues pour leur présence et leur support inconditionnel. Leur camaraderie m'a manqué durant la pandémie. Je tiens à les saluer.

Je veux aussi remercier mon entourage proche sans lequel je n'aurais jamais réussi. Leur patience et support fûrent déterminants dans cet accomplissement.

Finalement, avec tout le respect et l'amour que je leur dois, je tiens à remercier grandement mes parents qui ont fait preuve d'une patience, d'un dévouement et d'une sagesse titanesques. Ce travail est certainement le fruit de leurs efforts et de leur amour envers moi. Je termine aujourd'hui ce travail grâce à leurs efforts et je ne saurai jamais comment suffisamment les remercier. C'est avec amour que je vous dis, merci.

INTRODUCTION

Dans le cadre de la vision par ordinateur, le problème de détection d'objets dans une image a maintes fois été étudié dans la littérature. De fait, la tâche de reconnaître un objet au sein d'une image donnée est considérée comme une des problématiques les plus simples à résoudre. De la reconnaissance des contours par transformé de Fourier, en passant par la reconnaissance faciale, jusqu'à l'étiquetage d'objets, l'idée de pouvoir identifier un objet saillant dans une scène offre un grand nombre de possibilités, toutes plus pertinentes aujourd'hui vu l'explosion importante du nombre de données disponibles. Cependant, au-delà d'être capable d'identifier un objet dans une scène, le problème d'identifier la *pose* du dit objet est une tâche toute aussi importante. En effet, dans de nombreuses situations, le contexte qui entoure un objet est souvent très révélateur sur le rôle de celui-ci. À titre d'exemple simpliste, connaître la pose d'une barrière d'un passage-à-niveau peut donner de l'information supplémentaire sur le contexte de l'objet au sein de l'image. Si la guérite est en position verticale, il est raisonnable de présumer que le passage est clair et sécuritaire. À l'opposé, si la pose de la barrière est horizontale, on peut inférer que le passage est obstrué et qu'il serait dangeureux de procéder sur le chemin. L'information qui entoure la pose d'un objet peut être considérée comme le *contexte géométrique*; elle consiste en une des motivations principales de ce travail.

Pour le présent ouvrage, la recherche de contexte géométrique est étudiée dans un cadre ludique. De fait, les jeux de société sont des exemples simplistes mais appropriés à l'étude du contexte géométrique. De fait, dans de nombreux jeux, la pose de certains objets permet d'inférer beaucoup d'informations sur l'état du jeu et la progression de celui-ci. De plus, les jeux de société donnent un environment de test assez isolé et facile

à contrôler, permettant une analyse de la pose généralement moins bruitée que dans des situations plus générales. Pour ces raisons, le présent travail tente d'étudier la question de la détection de pose d'objets dans le cadre de jeux de société, plus spécifiquement, du jeu de cartes **Magic: The Gathering**.

Magic: The Gathering (abrégé simplement *Magic*) est un jeu de cartes à collectionner inventé par Richard Garfield en 1993 et la propriété de *Wizards of the Coast LLC*. *Magic* se veut un jeu compétitif stochastique à information imparfaite, similaire au poker, où (dans sa version la plus simple) deux compétiteurs s'affrontent en jouant des cartes de leur main représentant, au sein de la fantaisie du jeu, des créatures, enchantements ou tout autre concept fantastique. Ce jeu est propice à l'étude du contexte géométrique car, dans *Magic*, la position et l'orientation d'un carte sur la table peut permettre d'inférer de l'information sur le rôle et l'état de la carte au fil d'un match. À titre d'exemple, une carte positionnée de telle sorte à ce qu'elle soit colinéaire avec la ligne reliant les deux joueurs[1] est dite *désengagée*. À l'opposé, si une carte est perpendiculaire à cette ligne, la carte est dite *engagée*. Dans le cadre du jeu, l'état d'une carte comme étant engagée ou non est important car il informe les actions que les joueurs peuvent prendre concernant la carte en question.

Ce travail s'inscrit donc dans ce contexte. La question de recherche se résume à vouloir faire de la reconnaissance de pose pour un objet plat sur une table avec un angle de vue au-dessus de celle-ci. Il s'agit formellement d'un problème d'estimation de pose avec quatre degrés de liberté: La position en x de la carte au sein de l'image, la position en y de la carte, la rotation de la carte par rapport à l'axe vertical de la caméra ainsi que l'échelle de la carte (qui est équivalent à la hauteur de la caméra par rapport à la table). Bien que la présente question puisse être résolue par les algorithmes classiques de vision, le présent travail veut aussi s'incrire dans le courant d'apprentissage profond devenu répandu depuis les dernières années. Ainsi, ce travail

[1] Présumant qu'ils soient face à face, ce qui est standard lors de matchs professionnels.

Figure 1: Exemple d'une carte *Magic* typique. La majorité des cartes ont la même structure composée d'un titre, une image ainsi que du texte réglementaire.

Figure 2: Exemple illustrant comment engager une carte. Il suffit de tourner la carte à 90 degrés par rapport à soi.

veut tenter d'utiliser une approche par apprentissage profond (et plus spécifiquement par l'utilisation d'autoencodeurs) pour déterminer s'il est possible de générer un modèle capable de, non seulement identifier une carte *Magic*, mais aussi d'en inférer sa pose en quatre dimensions (translation x et y, rotation et échelle).

ÉTAT DE L'ART

Approches au problème

Identifier la pose d'un objet dans une image RGB en utilisant que les données RGB n'est pas récent comme question. En effet, il s'agit d'une tâche étayée aussi bien dans le contexte classique que dans celui de l'apprentissage profond. Dans le contexte de l'apprentissage profond, la question de l'attitude est souvent étudiée comme un problème à 6 degrés de liberté dans un contexte 3D, soit trois degrés de liberté pour la position et trois degrés de liberté pour la rotation [1, 17, 13, 6]. Cependant, le niveau de précision donné par un détecteur à six degrés de liberté n'est pas justifié, ou réalisable. De fait, certaines approches réduisent la dimension à 2D, définissent la position sur moins de paramètres en présumant que l'objet se trouve au centre de l'image [15], ou préfèrent une approche par boîtes de frontières [18, 19, 9]. Aussi, il existe plusieurs approches qui tentent de recréer la prise de vue d'une image en régressant la pose d'un modèle 3D détaillé [15, 12, 2]. Dans cette section, nous allons étudier les différentes approches en plus d'étudier les enjeux propres à chacune d'elles.

Estimation de pose à six degrés de liberté

Comme mentionné ci-haut, de nombreuses approches étudient la question de l'estimation de pose comme un problème à six paramètres (translation en x, y et z avec rotation autour des axes x, y et z). De manière générale, les approches trouvent la rotation et la translation d'un objet dans une image comme des tâches séparées [17, 6]. De plus, de nombreux articles régressent souvent la position d'un objet comme un préalable à l'estimation de rotation. Certaines approches utilisent un réseau de proposition de région (ou "*Region Proposal Networks*") pour trouver des régions d'intérêt à partir desquelles trouver la position de l'objet [1, 6]. Afin de pouvoir étudier la question en

détail, nous allons regarder comment ces deux tâches sont abordées séparément.

Translation

Plusieurs articles débutent par une étape de détection de "*Regions of Interest*" ou régions d'intérêt qui définissent la position où réside l'objet au sein de l'image. Dans certaines approches, on définit la translation comme un vecteur $\mathbf{t} \in \mathbb{R}^3$ indépendant de la région d'intérêt définie pour l'objet [17, 13] alors que d'autres approches expriment plutôt la translation comme une seule composante de profondeur associée à chaque région d'intérêt trouvée [1]. De manière générale, la translation s'exprime à partir d'un système de coordonnées où la caméra se trouve à l'origine. De fait, en regression de translation, la position de la caméra est souvent le point de référence à partir duquel toutes les autres translations sont exprimées.

Rotation

Là où la translation est représentée de manière relativement uniforme à travers la littérature, la rotation, elle, est représentée par une multitude d'approches. Les approches peuvent se diviser en trois catégories: L'approche directe, l'approche par sous-espace des rotations, et l'approche par projection.

L'approche directe consiste à régresser ou classifier directement les angles de rotation autour des trois axes de l'espace. Cette approche, quoique simple, pose problème puisqu'elle est sujette au blocage de cadran ("gimbal lock"). De plus, régresser sur cet espace peut être ardu puisqu'il y a des discontinuités aux angles 0 et 2π. Wang *et al* et Su *et al* [15, 13] utilisent cette méthode. Wang *et al* définissent un système de coordonnées à 7 paramètres dont 3 déterminent l'orientation dans l'espace: l'azimuth, l'élévation, et la rotation dans le plan. Il s'agit d'une système angulaire similaire aux angles d'Euler [15]. Su *et al* utilisent un système à trois angles (azimuth, élévation et

rotation dans le plan) pour définir la rotation. Pour chacun des angles, la tâche est divisée en 360 classes pour une tâche de classification [12].

L'approche indirecte consiste à régresser la rotation dans un sous-espace correspondant aux rotations 3D. Ces sous-espaces définissent l'ensemble des rotations mais ont la propriété d'être continus sur $\mathbb{R}^n$ en plus d'éviter les problèmes de blocage de cadran. Levinson *et al* [5] utilisent la décomposition en valeurs singulières pour représenter la rotation. L'utilisation de la décomposition en valeurs singulières permet de faire une régression sur neuf degrés de liberté qui sont projetés sur l'espace $SO(3)$ des rotations en trois dimensions. Cette projection paramétrique de l'espace $SO(3)$ permet de faire une régression sur cet espace qui est normalement discontinue [5]. Do *et al* et Su *et al* [1, 13] utilisent plutôt l'algèbre de Lie pour régresser l'espace $SO(3)$ à partir d'un vecteur $\mathbf{x} \in \mathbb{R}^3$. Xiang *et al* [17] utilisent un quaternion pour représenter une rotation aléatoire dans l'espace 3D. Le modèle régresse un vecteur $\mathbf{x} \in \mathbb{R}^4$ et calcule la fonction de perte en fonction de la moyenne des distances entre les points de l'objet pivoté selon l'angle prédit et les points de l'objet pivoté selon l'angle donné comme "*groundtruth*". Puisque l'espace de quaternions définit une rotation valide pour l'entièreté de $\mathbb{R}^4$, la régression peut se faire sans contraintes. Xiang *et al* définissent une fonction de perte tenant en compte les symétries possibles d'un objet.

L'approche par projection évite de régresser sur l'espace des rotations 3D directement et préfère plutôt régresser la position 2D des points de contrôle de la boîte de contraintes de l'objet étudié. Une fois ces points régressés, un algorithme "*Perspective n Points*" (PnP) régresse l'orientation de la boîte de contraintes directement. Grabner, Roth et Lepetit [2] avec Li, Wang, et Ji [6] utilisent tous deux cette technique.

En outre, la régression de rotation a été étudiée par Richter-King et Frese qui ont développé une fonction de coût pour les réseaux convolutifs ayant la propriété d'être continue sur le domaine [10]. La fonction de coût a la particularité d'être facilement

dérivable sur son domaine et facilite donc la descente de gradient dans le contexte de l'apprentissage machine. En plus de la symétrie de base $0{,}2\pi$ de la rotation, un objet peut posséder ses propres symétries. Un objet est dit n-symétrique s'il possède n symétries autour d'un même axe. La fonction de coût prend en compte ces symétries de l'objet pour générer une fonction de perte agnostique à celle-ci. Une fois la rotation régressée, Richter-King et Frese utilisent un algorithme de PnP pour trouver laquelle des n symétries correspond au bon angle.

Estimation de pose 2D avec rotation

Une approche beaucoup plus similaire à ce projet est celle de l'estimation de la pose 2D avec rotation. Ce problème vise l'estimation de la position, de l'échelle, et de la rotation d'un objet dans le plan. Yang *et al* définissent la tâche comme une régression de boîte de contraintes avec rotation [18]. Pour un objet à détecter, on souhaite trouver la boîte de contraintes (x, y, w, h, θ) où x et y sont les coordonnées du centre de la boîte de contraintes, w et h, la largeur et hauteur de la boîte et finalement θ l'angle de la boite. De plus Zhu, Ma, et Du définissent un détecteur qui prend en compte, de manière implicite, la rotation. La sortie du réseau est composée de quatre sommets qui sont joints, dans l'ordre, pour former une boîte de contraintes avec un angle arbitraire [19].

Enjeux

De manière générale, on peut observer les enjeux suivants au sein de la littérature. Premièrement, on remarque que la régression de la translation est, de manière générale, beaucoup plus simple que la rotation. De fait, il s'agit simplement de définir la régression de la translation sur un vecteur réel $[0, 1]^2$ où $(0, 0)$ est le coin suppérieur gauche et $(1, 1)$ le coin inférieur droit pour définir une tâche de régression assez simple. À l'opposé, régresser la rotation pose quelques défis. De fait, il est nécessaire de trouver une représentation de la rotation sur un espace uniforme qui est facile à régresser.

Même si le problème de la rotation est moins difficile en 2D, il existe tout de même une discontinuité en 0 et 2π (en plus de symétries si l'objet est n-symétrique). Ainsi donc, toute forme de régression de la rotation devra prendre ces caractéristiques en compte. Deuxièmement, puisque la rotation est plus complexe à régresser, on remarque que la plupart des approches régressent la rotation et la translation séparément. Ainsi, cette approche séparée implique que la majorité des réseaux de régression de pose fonctionnent par branches. De fait, la topologie des réseaux présentés dans les articles ci-haut est, pour la plupart, une topologie par branches où une section du réseau trouve la translation (souvent par un mécanisme d'attention) et envoit son résultat à la branche de rotation qui utilise cette information pour régresser l'angle précis. Troisièmement, on remarque que les approches de la littérature font l'usage de réseaux profonds capables de régresser sur des "*attributs*" profonds qui prennent en compte le contexte de l'image en plus des détails. Finalement, on remarque comment, dans les cas appliqués sur des données synthétiques, une des problématiques les plus importantes est celle de lier les données synthétiques aux données réelles. Cela nous amène à réaliser l'importance de l'augmentation de données sur de telles applications, surtout dans les cas avec données synthétiques comme le nôtre.

APPRENTISSAGE PROFOND

La question de pouvoir reconnaître de l'information visuelle au sein d'une image est loin d'être nouvelle. De fait, il existe depuis longtemps des approches dites "classiques" qui sont capables d'analyser les formes et les fréquences d'une image pour en extraire des informations utiles à des tâches telles, la segmentation ou bien la détection d'objets. À titre d'exemple, des approches telles les filtres de Canny [16] peuvent, avec une certaine facilité, extraire les arêtes d'une image. Ces arêtes pourraient être utilisées afin de détecter la position d'objets. Ainsi donc, il est déjà possible de résoudre certaines des questions de recherche du présent travail avec des algorithmes dit "classiques". Cependant, l'intérêt du présent travail est d'étudier la question non pas sous l'angle classique, mais plutôt sous l'angle de l'apprentissage profond. Cet angle peut s'avérer avantageux dans les situations où les contours sont ambigüs tels des situations d'occultation ou de superposition.

L'apprentissage machine est une approche principalement statistique où, par le biais de données empiriques $\mathbf{u}_i = (\mathbf{x}_i, \mathbf{t}_i)$, un algorithme est capable d'apprendre une fonction f parmi la famille des fonctions F tel que celle-ci respecte la fonction de minimisation suivante:

$$\min_{f \in F} \sum_i L(f(\mathbf{x}_i), \mathbf{t}_i)$$

où L est une *fonction de coût* qui donne une mesure d'erreur entre la sortie désirée $\mathbf{t}_i$ et la sortie obtenue $f(\mathbf{x}_i)$. Ainsi l'apprentissage machine, est un domaine d'application de l'informatique où l'on tente de minimiser, par quelque méthode que ce soit, la fonction de minimisation définie ci-haut.

L'apprentissage profond est une discipline de l'apprentissage machine qui utilise une structure mathématique appelée un réseau de neurones pour définir non seulement la famille de fonctions F mais aussi comment f peut être apprise. Dans la prochaine

section, nous allons aborder l'apprentissage profond du point de vue du présent travail, en présentant les concepts importants du domaine.

Réseau de neurones

Un réseau de neurones est une structure mathématique permettant d'apprendre une fonction f dans le contexte de l'apprentissage machine. En tant que tel, un réseau de neurones est composé de couches qui sont, elles-même composées de neurones. Un neurone est simplement une fonction n où:

$$n(\mathbf{x}) = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

où, $\mathbf{x}$ est un vecteur de valeurs réelles envoyé en entrée au neurone, $\mathbf{w}$ représente les poids donnés aux valeurs d'entrée et b, le biais. Finalement σ est la fonction d'activation du neurone. Par lui-même, un neurone ne peut qu'exprimer une fonction linéaire, or, avec la fonction σ, le neurone peut alors exprimer une fonction non-linéaire. De manière générale, une couche d'un réseau de neurones peut être exprimée comme une matrice de poids $\mathbf{W}$ avec un vecteur de biais $\mathbf{b}$. Ainsi pour un vecteur d'entrée $\mathbf{x}$, la sortie $N(\mathbf{x})$ de la couche de neurones est définie comme:

$$N(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Ici $\mathbf{W}\mathbf{x}$ est le produit matriciel entre le vecteur d'entrée et la matrice de poids. Implicitement, N est une fonction $f_\theta : \{\mathbb{R}^n \to \mathbb{R}^m\}$ où $\theta = \{\mathbf{W}_{m \times n}, \mathbf{b}\}$ Ainsi, si une couche de neurones définit une famille de fonctions f, il s'en suit qu'un réseau de neurones est, en fait, la composition de plusieurs fonctions f. Un réseau de neurones à n couches peut donc être représenté par une fonction $R_\theta(x) = (f_{\mathbf{W}_1, \mathbf{b}_1} \circ \cdots \circ f_{\mathbf{W}_n, \mathbf{b}_n})(x)$ où $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \cdots, \mathbf{W}_n, \mathbf{b}_n\}$.

L'apprentissage profond cherche à trouver les paramètres θ^* de R tel que:

$$\theta^* = \min_\theta \frac{1}{n} \sum_{i=1}^{n} L(R_\theta(\mathbf{x}_i), \mathbf{t}_i)$$

Réseaux Convolutifs

À l'instar des réseaux de neurones qui définissent l'application d'une couche sur un vecteur d'entrée[2], un réseau convolutif définit l'application d'une couche sur un tenseur de dimensions $D \geq 1$. Nous allons ici décrire une couche convolutive pour une entrée image (2D) mais il faut noter que la convolution est une opération définie sur une entrée de dimensions arbitraire.

Soit une entrée 2D de taille $m \times m$ (on peut considérer le cas carré sans perte de généralité) et une convolution c_θ contrôlée par les paramètres $\theta = \{\mathbf{K}, p, \mathbf{s}\}$. Ici, $\mathbf{K}$ est un noyau (une matrice) de taille $k \times k$, p est un entier positif indiquant le remplissage ("*padding*"), et $\mathbf{s}$ est un vecteur de dimension 2 indiquant le pas ("*stride*"). Pour une image X:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,m} \end{bmatrix}$$

Un noyau K avec $k \leq m$ (ici k est supposé impair pour simplifier les calculs):

$$\mathbf{K} = \begin{bmatrix} k_{1,1} & \cdots & k_{1,k} \\ \vdots & \ddots & \vdots \\ k_{k,1} & \cdots & k_{k,k} \end{bmatrix}$$

Pour un remplissage de 0 (aucun remplissage) et un pas $\mathbf{s}$ de 1 ($\mathbf{s} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$), calculer la convolution $c_\theta(\mathbf{X})$ implique calculer la matrice $R_{d \times d}$ suivante ($d = \frac{m-k+2p}{\mathbf{s}_1} - 1$):

[2] En réalité, plusieurs entrées x peuvent être données à un réseau de neurones sous la forme d'une matrice. De manière générale, pour un tenseur de dimensions D, la première dimension est considérée celle des données. À titre d'exemple, appliquer un réseau sur une entrée matricielle de dimensions $m \times n$ est considéré comme appliquer le réseau sur m entrées de taille n simultanément. Ce concept tient sur des entrées de dimensions arbitraires $D > 1$.

$$r_{i,j} = \sum_{u=1}^{k} \sum_{v=1}^{k} k_{u,v} * x_{(u+i-1),(v+j-1)}$$

Signification dans un réseau convolutif

La convolution est une opération par laquelle une image est multipliée par un noyau. Chaque pixel du résultat est la somme pondérée des pixels dans l'image qui se trouve sous l'influence du noyau par les valeurs du noyau. Ainsi, pour un noyau **K**, le pixel du résultat est, en fait, une "*compression*" des données de l'image originale en une seule valeur. Le noyau va indiquer à quel genre de structure (dans l'image originale), le pixel de sortie est sensible. De plus, puisque le noyau est appliqué uniformément à travers l'image, la convolution va détecter une structure dans une image sans être influencée par la translation de celle-ci dans l'image.

Comme mentionné ci-haut, un des paramètres de la convolution est le "*stride*" ou "pas". Le pas implique qu'après avoir calculé le noyau **K** autour du pixel $x_{i,j}$ dans l'image, le prochain pixel calculé sera $x_{i+s_1,j+s_2}$ créant ainsi un "pas" dans l'image. Conséquemment, puisque le pas implique qu'il n'y aura plus de correspondance 1 à 1 entre les pixels de l'entrée, et ceux de la sortie, cela va générer une sortie considérablement plus petite que l'entrée. En effet, à titre d'exemple, avec un pas de 2, seulement un pixel sur deux dans l'image d'entrée sera considéré. Cela aura comme effet de générer une sortie deux fois plus petite que l'entrée. Cette particularité du pas fait en sorte que la sortie d'une convolution peut être une version spatialement condensée de l'entrée. Ainsi, avec un pas de plus en plus élevé, on peut contrôler la variation de l'échelle au niveau du réseau. En contrôlant ainsi la résolution, on peut faire en sorte que le réseau voit l'image à plusieurs résolutions permettant d'engendrer une invariance à l'échelle (voir la section *Invariance* ci-bas).

Ainsi, un réseau convolutif, au sense large, est défini comme tout réseau de neurones dont une partie des fonctions de couche f est définie comme une opération de convo-

lution. Dans le cas spécifique où toutes les couches d'un réseau sont convolutives, il s'agit d'un "*Fully Convolutional Network*" ou FCN.

Domaine d'application

Pour le présent travail, nous étudions la question de l'apprentissage profond appliqué au traitement d'images. Cela implique ainsi que pour une donnée $\mathbf{u}_i = (\mathbf{x}_i, \mathbf{t}_i)$, $\mathbf{x}_i$ est une image (soit un tenseur $H \times L \times 3$ pour une image RGB de hauteur H et largeur L). L'élément cible $\mathbf{t}_i$ est la donnée que nous voudrions extraire du traitement d'images (par exemple $\mathbf{t}_i$ pourrait être l'orientation d'un objet au sein de l'image $\mathbf{x}_i$). Ainsi, apprendre la fonction f implique trouver une fonction prenant une image en entrée, et retournant une information de traitement d'image en sortie. Il suffit ici de se procurer assez de paires $\mathbf{u}_i$ afin que f puisse être apprise avec succès.

Dans le domaine de l'apprentissage machine, il existe plusieurs grands problèmes, dont les deux plus importants sont la classification et la régression. La classification est un problème où l'élément $\mathbf{t}_i$ (et par extension $f(\mathbf{x}_i)$) peut être vu comme un nombre entier. Ce nombre entier $1 \geq n \geq N \in \mathbb{N}$ retourné par $f(\mathbf{x}_i)$ indique à quelle classe parmi N l'élément $\mathbf{x}_i$ appartient. Dans cette situation f est appelé un *classifieur*. Un exemple simple serait une fonction f capable de distinguer deux classes de photographies. Si la photograpie est une image d'une pizza, f retourne 1, sinon f retourne 2. Ici $\mathbf{x}_i$ est une photo quelconque, $\mathbf{t}_i$ est la "réponse" (si l'image $\mathbf{x}_i$ contient bel et bien de la pizza) et $f(\mathbf{x}_i)$ est la classification de l'image selon f.

Plutôt qu'une tâche de classification, il est aussi possible d'approcher un problème comme un tâche de régression. Dans le contexte de la régression, la fonction f retourne une (ou plusieurs) valeur(s) réelle(s) ($f(\mathbf{x}_i) \in \mathbb{R}$). Un exemple de regression serait une fonction f prenant une image en entrée $\mathbf{x}_i$ et retournant une valeur $f(\mathbf{x}_i)$ indiquant la position (en x) d'un objet par rapport au centre de celle-ci.

Dans le contexte de ce travail, le problème tel que défini à l'introduction est un problème de régression. De fait, l'image d'entrée est celle d'une carte *Magic*, et la sortie de la fonction f est un vecteur $\mathbf{o} = \begin{bmatrix} t_x & t_y & r_{sin} & r_{cos} & s \end{bmatrix} \in \mathbb{R}^4$. Les deux premières valeurs du vecteur sont les composantes x et y de la position de la carte dans l'image. L'orientation de la carte dans l'image est donnée par les composantes sinus et cosinus de l'angle soit r_{sin} et r_{cos}. Finalement, la valeur s est un facteur d'échelle (indiquant le rapport à l'échelle "standard" d'une carte *Magic*, arbitrairement choisie comme 204 × 146 pixels).

Extraction de Composantes

Dans le processus classique de vision, les images à être analysées ne peuvent que très rarement être utilisées telles quelles. De fait, si l'on souhaite, par exemple, détecter la position d'un objet utilisant un algorithme classique, il serait utile d'appliquer un filtre de Canny afin d'en extraire les frontières avant de tenter toute forme de positionnement. De la même manière, avec l'approche en apprentissage profond, il est aussi avantageux d'extraire des composantes utiles à la tâche. Cependant, contrairement aux méthodes classiques où les composantes doivent être extraites manuellement, dans le cas des méthodes d'apprentissage profond, l'extraction des composantes est apprise avec la fonction f. De fait, il est possible de décomposer $f(\mathbf{x}_i)$ en deux fonctions $f(\mathbf{x}_i) = g \circ h(\mathbf{x}_i) = g(h(\mathbf{x}_i))$ où h est la fonction d'extraction de composantes et g est une fonction qui associe les composantes de h à une sortie. Lorsque le réseau de neurones apprend la fonction f, il apprend les fonctions g et h implicitement.

Une caractéristique intéressante des réseaux de neurones est que les fonctions g et h se localisent à des endroits spécifiques du réseau. De fait, la fonction h (appelée encodeur) est située au début du réseau de neurones dû au fait qu'elle est appliquée sur les données d'entrée avant g. Ainsi, il est possible de physiquement extraire h d'un réseau une fois la fonction f apprise. Par exemple, supposons que nous avons

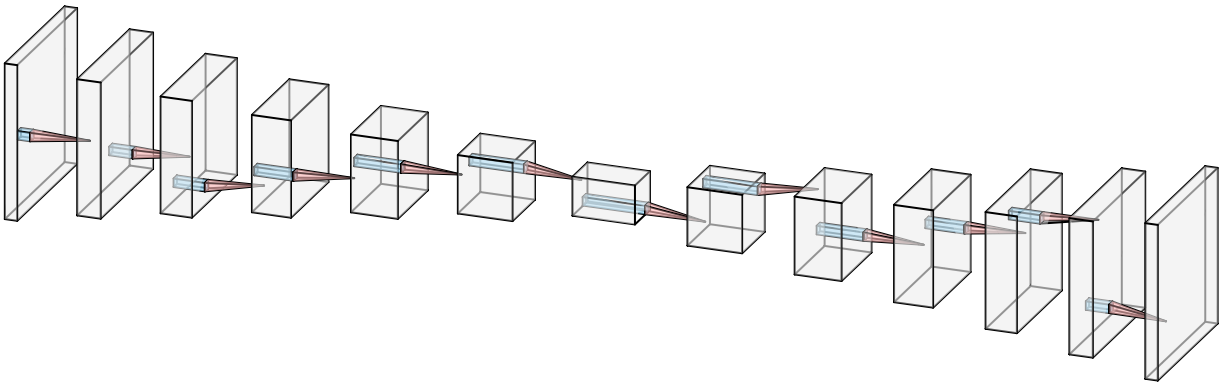

Figure 3: Exemple de la structure générale d'un autoencodeur. La dimension de l'entrée (gauche) est graduellement réduite à une dimension latente minimale (centre) et puis augmentée jusqu'à une taille de sortie (droite).

entraîné un réseau $f_1 = g_1 \circ h_1$ capable de reconnaître des types de voitures, il est possible d'extraire l'encodeur h_1 et de l'utiliser pour apprendre une deuxième fonction $f_2 = g_2 \circ h_1$. Le principe de cette démarche est que, si f_1 et f_2 sont utilisées pour des tâches connexes, alors, il serait logique de présumer que l'encodeur pour f_1 devrait être similaire à l'encodeur de f_2. Ainsi, si l'on connaît déjà h_1, il serait utile de simplement réutiliser h_1 pour f_2. Cela aurait comme effet de simplifier l'entraînement et permettrait de profiter de l'entraînement de f_1 pour l'entraînement de f_2.

Topologie de réseau

Quoique très variées en général, pour ce qui est des tâches d'analyse d'images, certaines topologies de réseau de neurones sont plus communes. De fait, il existe trois archétypes topologiques de réseaux reconnus pour leur succès en traitement d'images: L'autoencodeur, le réseau Res-Net, et le réseau U-Net.

Autoencodeur

Un autoencodeur est un archétype topologique en traitement d'images composé d'un encodeur h capable de compresser les informations d'une image $\mathbf{x}$ dans un espace tenseur beaucoup plus petit. L'idée sous-jacente à un autoencodeur, illustrée à la figure 3, est la suivante: Créer une structure neuronale qui prend l'entrée $\mathbf{x}$ et, par l'entremise de couches convolutives successivement plus petites, compresse l'information dans un espace de basse dimension. Une fois l'information compressée, le réseau décompresse l'image depuis l'espace latent jusqu'à son espace original.

La force d'un autoencodeur est apparente lorsque l'on étudie le processus d'entraînement. En effet, la cible du réseau est en fait l'image originale d'entrée ($\mathbf{u}_i = (\mathbf{x}_i, \mathbf{t}_i) = (\mathbf{x}_i, \mathbf{x}_i)$). Ce faisant, puisque $f(\mathbf{x}_i) = \mathbf{x}_i$, il s'en suit que $f = \mathbb{I}$. On peut alors exprimer f de la manière suivante: $f(\mathbf{x}_i) = \mathbb{I}(\mathbf{x}_i) = h^{-1}(h(\mathbf{x}_i))$. La logique est alors la suivante: Si l'on entraîne f de sorte à ce que la fonction soit l'identité, alors, l'autoencodeur sera composé de deux fonctions inverses l'une de l'autre, h et h^{-1}. Ainsi, en entraînant f vers l'identité, nous obtenons deux réseaux en un: un encodeur (h) et un décodeur (h^{-1}). L'encodeur h peut alors être extrait de f et utilisé dans d'autres problèmes et permet alors de compresser $\mathbf{x}_i$ dans un espace de faible dimension tout en conservant un maximum d'informations sur l'entrée.

De manière générale, l'encodeur et le décodeur sont composés de couches à convolution. Traditionnellement, une couche à convolution compresse l'image dans un espace plus petit si la couche est dans l'encodeur, ou bien, vers un espace plus grand si la couche est dans le décodeur. Ainsi, la fonction h est composée d'une séquence de convolutions compressant l'image d'entrée à répétitions jusque dans l'espace de représentation compressé. De la même manière, toutes les couches à convolution dans h^{-1} décompressent l'information dans l'espace de représentation jusque dans l'espace original de l'image.

Afin de ne pas simplement apprendre la solution triviale $h = h^{-1} = \mathbb{I}$), la plupart des

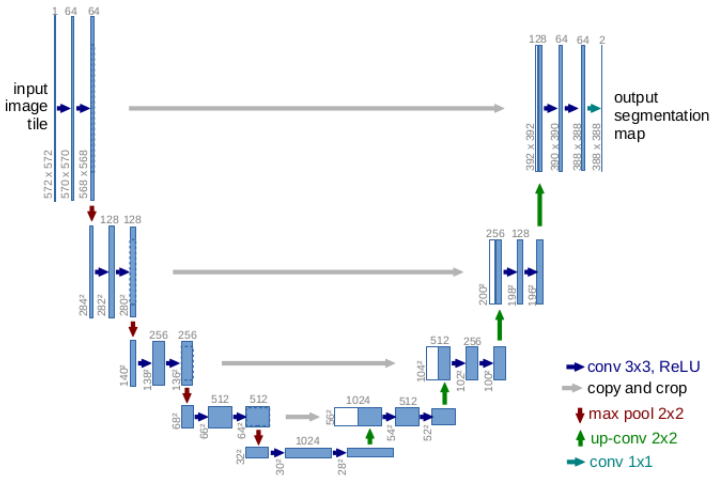

Figure 4: Réseau U-Net tel que défini par Ronneberger, Fisher et Brox. Récupéré du papier *U-Net: Convolutional Networks for Biomedical Image Segmentation*[11].

autoencodeurs font en sorte que h soit forcé de représenter son entrée dans un espace de plus faible dimension que celle-ci, rendant ainsi impossible le cas où $h = \mathbb{I}$ (un tel autoencodeur est dit "*undercomplete*" . Cependant, certaines publications affirment qu'avec suffisamment de régularisation, il n'est pas nécessaire d'avoir un autoencodeur dit "undercomplete" pour éviter le cas trivial [14]. De tels autoencodeurs sont appelés "*overcomplete*".

Réseau U-Net

Publié en 2015 par Ronneberger, Fisher, et Brox [11], le réseau U-Net, (illustré à la figure 4), est un réseau de convolution avec la particularité d'avoir des "*skip layers*" à chaque couche de convolution. Un *skip layer* est un lien entre une couche de convolution dans h et la couche de déconvolution équivalente dans h^{-1}. Ainsi, dans le décodeur, chaque déconvolution utilise, à la fois, la sortie de la couche précédente ainsi que la sortie de la couche de convolution correspondante dans la partie encodeur du réseau.

Cela a pour effet de permettre à la couche de déconvolution d'avoir accès à l'information bas-niveau venant de l'espace de représentation ainsi qu'à l'information de haut-niveau venant des convolutions de l'encodeur. Cela permet alors à l'encodeur d'accomplir la tâche en utilisant l'information de bas-niveau (détails) ainsi que l'information de haut-niveau (contexte).

Depuis la publication de U-Net, de nombreux réseaux ont démontré l'utilité des *skip layers* en les employant dans leurs réseaux (convolutifs ou non). Dans le cadre de ce travail, nous étudierons l'impact que les *skip layers* ont sur la performance du réseau.

Res-Net

Publié en 2015 par He *et al.*[3], Res-Net est un réseau profond à convolution avec représentation résiduelle dont la tâche est de classifier le contenu d'images (voir figure 5). Similairement à U-Net, Res-Net utilise des *skip layers* afin d'augmenter la capacité du réseau. Contrairement à U-Net qui tente d'insérer une structure de *skip layers* dans une architecture autoencodeur, Res-Net utilise plutôt les skip-layers dans un réseau profond. En tant que tel, Res-Net est un réseau composé d'un grand nombre de couches convolutives qui réduisent graduellement la taille des représentations jusqu'à une certaine taille. Cependant, contrairement à un autoencodeur qui tenterait de reconstruire les données vers une image de sortie, Res-Net utilise plutôt un couche FC (*Fully Connected*) pour générer un vecteur de classification (1000 classes) à partir de la représentation bas-niveau. Il ne s'agit donc pas d'un autoencodeur.

Représentation pour une tâche de vision

Le représentation des données dans un problème de vision a un impact important sur la structure du réseau. En effet, pour une même tâche, il existe une multitude de manières de représenter le résultat et la formulation du résultat va influencer la structure du réseau. Par exemple, prenons une tâche dont le but est de trouver l'angle θ d'un objet.

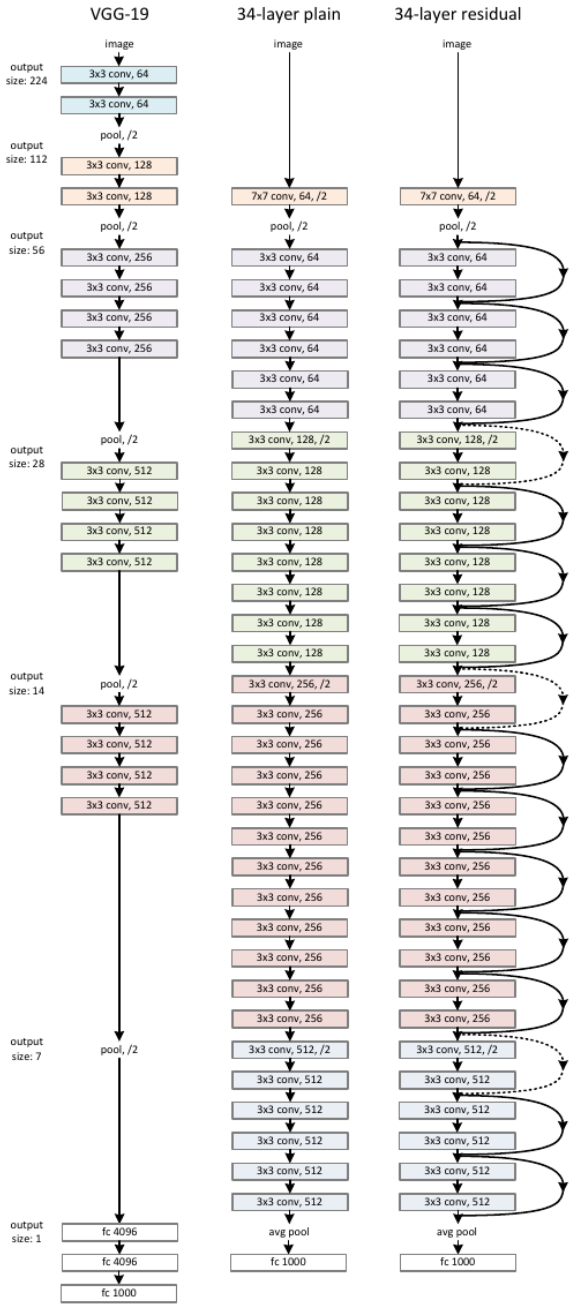

Figure 5: Réseau Res-Net tel que défini par *et al.* Récupéré du papier *Deep Residual Learning for Image Recognition* [3].

Il serait possible de définir la tâche comme une régression où la valeur est un réel indiquant l'angle de l'objet ou bien comme une classification, où la classe de l'objet indique à quel angle parmi 360 l'objet se trouve. Comme on le voit, même si la tâche reste la même, il existe plusieurs manières de la représenter. Dans le contexte de notre tâche de vision, on cherche à estimer trois propriétés: la position, la rotation et l'échelle. Nous allons voir les différentes options qui existent pour chacun de ces sous-problèmes.

Position

En ce qui concerne le positionnement d'objets, il y a deux manières de représenter la position d'un objet dans l'image. La première utilise deux valeurs réelles entre 0 et 1. Chaque valeur indique la distance de l'objet par rapport au coins supérieur gauche $(0, 0)$ dans l'image et au coin inférieur droit $(1, 1)$. Avec cette représentation, les valeurs de $(0.5, 0.5)$ indiquent que l'objet est dans le milieu de l'image. À l'opposé, le positionnement peut être vu comme une tâche de classification. En effet, soit une image $n \times n$, on peut considérer que chaque pixel est une classe et que le pixel au centre de l'objet correspond à la classe position de l'objet. On peut appeler une telle méthode un "*heatmap*".

Un avantage du heatmap est qu'il permet de représenter plusieurs positions dans une image. Par contre, le débalancement entre les échantillons (il y a beaucoup plus de 0 que de 1) peut poser problème en plus de ne pas garantir une grande précision.

Une dernière représentation est celle du masque. Avec celle-ci, un pixel est classifié de manière binaire comme appartenant, ou non, à l'objet. L'avantage est que, pour un objet symétrique, le centroïde des pixels classifiés comme étant "dans" l'objet va fournir un centroïde pour cet objet. Cette approche élimine le problème de débalancement et assure une plus grande précision de la localisation.

Rotation

En ce qui concerne la rotation, il existe aussi une manière régressive et classificatrice de représenter l'information. La manière régressive de trouver la rotation serait d'utiliser une valeur réelle entre 0 et 1. Une valeur de 0 représente une rotation de 0 radians, alors que 1 indique une rotation de 2π radians. Malheureusement, cette représentation pose un problème autour de 0 et 2π radians. Pour régler ce problème, il faudrait régresser deux valeurs entre -1 et 1 ou la première est $\sin\theta$ et la deuxième est $\cos\theta$. À l'opposé, il serait aussi possible de représenter la rotation comme une tâche de classification. De fait, il est possible de diviser le cercle angulaire en 360 classes uniques (une pour chaque degré entre 0 et 259, par exemple) et d'appliquer un classifieur sur une telle tâche.

Échelle

Finalement, l'échelle est un élément qui est plus limité dans sa représentation. Puisqu'il s'agit d'un facteur multiplicatif, le domaine consiste en une seule valeur réelle. Contrairement à la position et à la rotation, l'échelle n'est pas bornée. De fait, si la carte est suffisament grosse, n'importe quel réel positif peut être représenté par le facteur d'échelle. Conséquement, il serait impraticable d'en faire une tâche de classification. Ainsi, trouver la valeur d'échelle doit être une tâche de régression sur les réels positifs.

Augmentation de données

Un point important à apporter est celui de l'augmentation de données. Comme mentionné précédemment, pour un ensemble de données D quelconque composé de n éléments, un réseau de neurones va apprendre θ^* selon la fonction de minimisation suivante:

$$\theta^* = \min_\theta \hat{L}(D, f_\theta(D)) = \min_\theta \frac{1}{n} \sum_{x \in D} L(x, f_\theta(x))$$

Malheureusement, puisque θ^* est trouvé par optimisation, il est possible que f_θ apprenne des corrélations au sein de D qui ne sont pas représentatives de la réalité. Cela aurait comme effet de grandement réduire la précision d'un tel réseau dans des situations réelles. De fait, lors de la conception de l'ensemble de données, il faut prendre gare aux corrélations non-intentionelles qu'un réseau puisse apprendre. À titre d'exemple, si un ensemble D de données ne contient que des images composées d'un arrière-plan bleu, il se peut que le réseau déduise des données que le fond est toujours bleu. Cela aurait l'effet négatif de rendre la réponse du réseau imprévisible si jamais celui-ci était confronté à une situation où l'arrière-plan n'était pas bleu. Conséquemment, lors du design de l'ensemble de données, il faut s'assurer que le réseau apprenne les **invariances** appropriées.

Invariances

Une invariance au sein d'un réseau de neurones est une caractéristique où une variation dans l'entrée x d'un réseau, ne fera pas varier la sortie $f(x)$. À titre d'exemple si, lors du design de D, de nombreux arrières-plans variés avaient été inclus sans que la sortie de $f(x)$ ait été changée, le réseau aurait appris une fonction f de telle sorte à ce que faire varier l'arrière-plan de x ne change pas $f(x)$. Si un réseau apprend une telle fonction tel que le fond de x n'influence pas la sortie, on dit que le réseau est "invariant au fond". De la même manière, avec une conception appropriée de D il est possible de faire en sorte qu'un réseau soit invariant à une multitude de facteurs tels: la rotation, l'échelle, la translation, etc. Dans tout projet, il est crucial d'identifier les éléments de f qui doivent rester variables et ceux auxquels le réseau doit rester invariant. Nous verrons à la section *Jeu de données* les invariances priorisées dans ce projet.

JEU DE DONNÉES

Comme mentionné à la section *Apprentissage Profond*, les données utilisées à l'entraînement d'un réseau sont cruciales au bon fonctionnement de celui-ci. Malheureusement, il est souvent irréaliste de pouvoir construire un ensemble de données manuellement. En effet, récupérer les données photographiques nécessaires, en plus d'annoter ces photos de manière précise sont des tâches beaucoup trop grosses pour être complétées dans un temps raisonnable. Ainsi, il est possible d'utiliser des données synthétiques pour l'entraînement. Les données synthétiques présentent trois avantages importants. Premièrement, l'utilisation de données synthétiques permet de générer un grand nombre de données relativement rapidement. De fait, puisque l'ensemble de données est généré lors de l'entraînement, il est possible de générer un ensemble de données de taille arbitraire relativement rapidement. Faire l'équivalent avec des données réelles est tout simplement impossible. Deuxièmement, générer des données synthétiques permet de connaître précisément les caractéristiques de la donnée générée. Ainsi, en utilisant des données synthétiques, l'annotation manuelle n'est plus nécessaire. Troisièmement, l'utilisation de données synthétiques permet de modifier, sur mesure, les propriétés globales du jeu de données en cours de route si celà s'avère nécessaire. Malheureusement, malgré le fait que les données synthétiques présentent des avantages importants, elles présentent aussi un gros défi. En effet, pour que le réseau entraîné puisse généraliser aux cas réels, il faut que l'ensemble synthétique soit suffisament "réaliste" ou proche des données réelles. Ce prérequis est malheureusement nébuleux et la question de définir si un jeu de données est suffisament réaliste reste une question importante dans le domaine [13].

Notre jeu de données est synthétique, basé sur un ensemble de 24000 cartes *Magic* éditées depuis 1993. Pour chaque image, une carte est choisie du lot de 24000. Pour

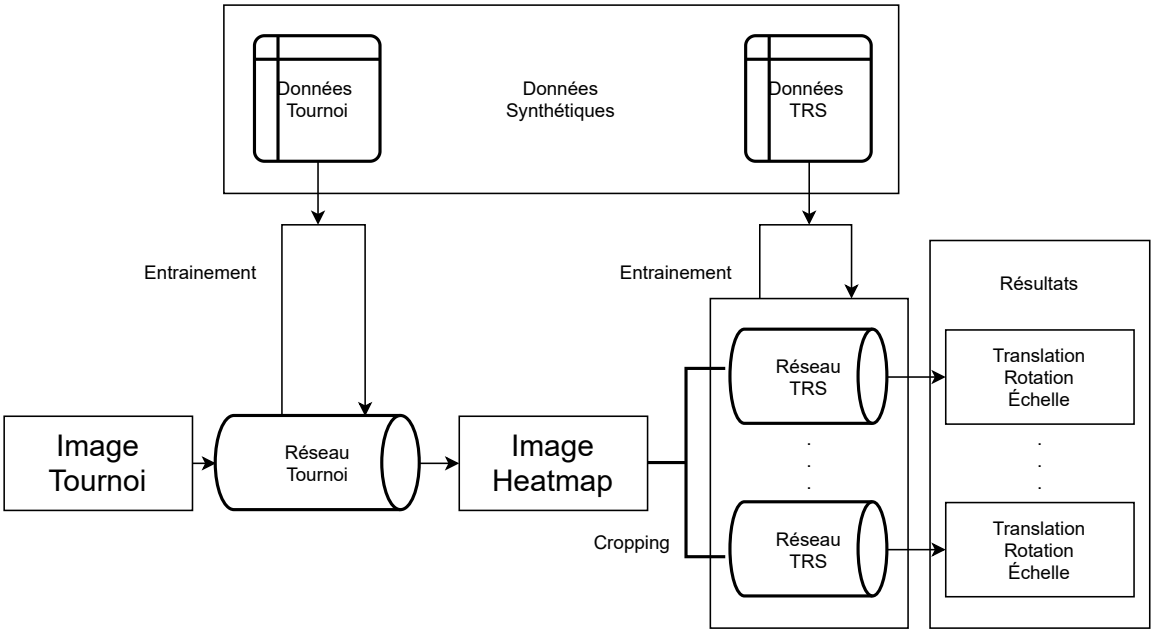

Figure 6: Flux de travail pour les réseaux Tournoi et TRS.

éviter qu'une carte soit sélectionnée plusieurs fois, les cartes sont sélectionnées séquentiellement du lot. Une fois la carte sélectionnée, une image synthétique est construite avec celle-ci. Nous allons, tout d'abord, voir comment l'image d'entrée est générée, puis, par la suite, nous analyserons comment la cible est créée. Pour ce travail, les cartes seront utilisées à l'intérieur de deux jeux de données différents. Le premier jeu de données, nommé le jeu Tournoi, est composé d'images synthétiques contenant plusieurs cartes et visant à simuler une image de tournoi *Magic* (voir section "Jeu de données"). Le deuxième ensemble de données, nommé jeu TRS, est un ensemble utilisé pour entraîner le réseau responsable d'identifier la position, la rotation, et l'échelle d'une carte (voir section "Jeu de donnée TRS"). La figure 6 illustre le flux de travail pour les deux réseaux. La présente section va analyser le rôle des données dans ce flux. Le chapitre *Approche* va décrire plus en détail le flux lui-même.

Figure 7: Composantes standards d'une carte Magic. (1) Nom de la carte, (2) Coût d'invocation, (3) Illustration, (4) Ligne de type, (5) Symbole d'extension, (6) Texte de capacités, (7) Texte d'ambiance, (8) Force / Endurance

La carte Magic

De manière générale, une carte *Magic* est composée de deux sections horizontales, comme illustré à la figure 7. La partie supérieure est composée de la barre de titre (composantes 1 et 2) et de l'image (composante 3). La partie inférieure est composée de la ligne de type (composantes 4 et 5), de l'encadré de texte (composantes 6 et 7) et, dans le cas de certaines cartes, une section "Force/Endurance" (composante 8). Comme on peut le voir, les cartes *Magic* sont, de manière générale, identifiables par leur boîte de texte. Ainsi, on voit comment il devrait être possible de différencier le "haut" et le "bas" d'une carte à partir de la boîte texte.

Cependant, malgré cette structure standardisée, les cartes individuelles peuvent dévier de manière importante de cette structure. En effet, Les cartes peuvent avoir une structure verticale plutôt qu'horizontale ou bien avoir une structure en quadrants (voir fig-

Figure 8: Variantes structurelles d'une carte *Magic*

ure 8). De plus, les choix stylistiques peuvent aussi faire varier de manière importante la structure de la carte, comme l'illustre la figure 9.

Il existe aussi des versions sans texte ("*textless*") de cartes. À titre d'exemple, la figure 10 montre deux cartes identiques (en terme d'utilité dans le jeu) mais la seconde carte n'a pas le texte de la première. Ces cartes sont particulières puisqu'elles possèdent du texte mais, pour des raisons stylistiques, le texte de la carte à été omis de celle-ci. Lors du jeu, la carte est considérée comme ayant le texte même si celui-ci a été omis.

Ainsi, on voit comment, quoique facile dans les cas "standards", identifier l'orientation d'une carte n'est pas un exercise trivial. Il faut être capable d'analyser non seulement les détails de chaque élément d'une carte mais il faut aussi le contexte qui entoure ces éléments pour identifier l'orientation avec précision.

Figure 9: Variantes stylistiques d'une même carte *Magic*

Figure 10: Variantes "textless" d'une même carte *Magic*

Figure 11: Image typique de tournoi *Magic*. Image récupérée du tournoi Pro Tour Dark Ascension ayant pris place en 2012 [7].

Jeu de données Tournoi

Dans le cadre de ce projet, l'objectif est de pouvoir appliquer l'algorithme du réseau sur une image de tournoi *Magic* et en retrouver les positions, orientations et échelles des cartes dans l'image. La figure 11 montre un exemple d'image de tournoi. De manière générale, la prise de vue est verticale, au-dessus de la table, regardant vers la table suivant un vecteur perpendiculaire à la normale de celle-ci. Ce point de vue est fixe et ne change pas entre les prises de vue. L'avantage de cet arrangement est que les cartes peuvent être considérées comme des objets 2D plats avec aucune déformation projective. Cela rend le problème plus facile car trouver l'échelle, la translation et la rotation de la carte est suffisant pour identifier parfaitement la pose de la carte par rapport à la camera.

Malheureusement, malgré ces éléments idéaux des tournois, ceux-ci présentent de nombreux obstacles dont il faut prendre compte pour pouvoir les utiliser dans un contexte d'apprentissage. Un premier défi est, tout d'abord, la difficulté d'obtenir des données bien formatées pour le problème que l'on souhaite résoudre. De fait, même s'il existe de nombreuses heures de vidéo de tournoi accessibles, ces vidéos ne sont pas propres au traitement. Premièrement, les vidéos de tournoi sont entrecoupées par d'autres éléments qui ne sont pas le "tournoi" en tant que tel (ex: publicités, interviews, etc.) Deuxièmement, même à l'intérieur d'images de tournoi, il existe des obstacles visuels tel que les bras ou les têtes de joueurs. Troisièmement, lors de tournois, il est commun pour des cartes d'être superposées. Ainsi, il faut prendre en compte le fait que l'algorithme doit être capable de détecter des cartes partiellement occluses. Cependant, chacun de ces problèmes peut être résolu avec un jeu de données synthétique.

Images Synthétiques

Pour résoudre les problèmes mentionnés ci-haut, il est possible de procéder à l'utilisation de données synthétiques pour permettre la génération d'exemples illimités. Générer les données a l'avantage de contrecarrer le problème des données insuffisantes en plus de permettre d'éliminer les obstacles visuels non-pertinents à l'exercice (bras, têtes, publicités). En ce qui concerne l'occultation, le jeu de données synthétiques peut représenter ces cas importants et forcer l'algorithme à tenter de localiser les cartes avec l'occultation. Pour ce faire cependant, un modèle statistique de ce qu'est une image de tournoi a été construit. Pour construire le modèle statistique, un ensemble relativement petit d'images de tournoi a été construit. À partir de ce petit ensemble, un écart-type est fait au niveau des pixels des images. Cela a comme résultat de créer une image monochrome où les pixels avec davantage de variation sont allumés et les pixels avec peu de variation sont éteints. Puisque la présence de cartes sur la table a comme effet de faire varier les pixels, les endroits avec davantage de cartes seront plus variables et

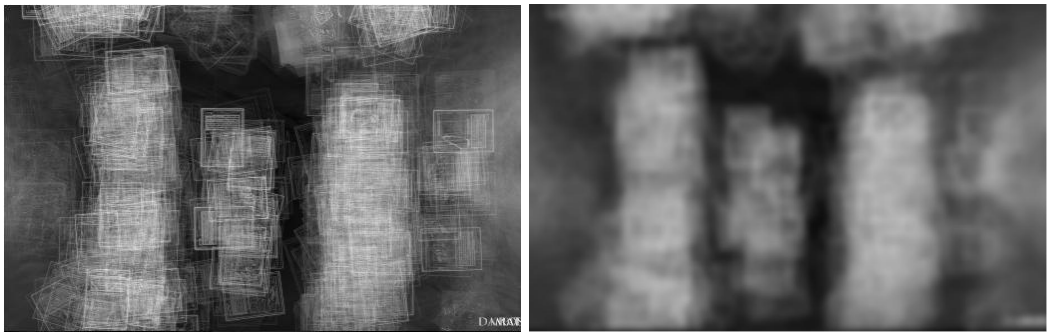

Figure 12: Image écart-type utilisée pour générer une image de densité de probabilité de tournoi synthétique. Gauche: écart-type simple, Droite: écart-type avec filtre gaussien.

donc plus allumés. À l'opposé, les endroits où les cartes ont peu tendance à se retrouver vont avoir un niveau bas car la couleur du pixel reste, de manière générale, constante. Ainsi, cette image écart-type (voir figure 12) permet d'identifier les endroits probables de la position d'une carte selon sa position en x et y. Puisque les joueurs peuvent avoir une main dominante différente et parce que le miroitement des cartes autour de l'axe horizontal ne devrait pas changer la validité d'une image de tournoi, on considère que la densité de probabilité est séparable. Une moyenne est ensuite faite sur l'axe vertical, créant ainsi une image $1D$. Celle-ci indiquant la probabilité d'une carte selon sa position horizontale indépendamment de sa position verticale. La figure 13 montre la fonction de probabilité d'une carte à la position horizontale donnée. Cette fonction est interpolée à partir des données de l'image 1D calculée à l'étape précédente.

Une fois la fonction de probabilité calculée pour la position horizontale, la position verticale est présumée uniforme. Ainsi, on obtient la fonction de probabilité 2D illustrée à la figure 14. Soit la variable aléatoire χ suivant la distribution définie précédemment, pour générer une image "tournoi" avec n cartes, un fond est généré et n positions sont sont tirées de la variable χ. Pour chacune des positions tirées, une carte est insérée dans

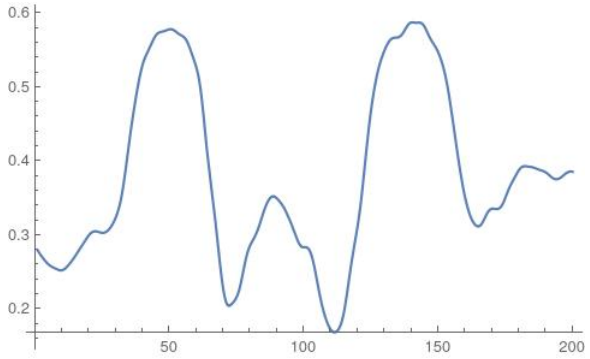

Figure 13: Fonction de probabilité de la position d'une carte sur l'axe horizontal de la table. La fonction est interpolée à partir des valeurs empiriques de l'image écart-type 1D.

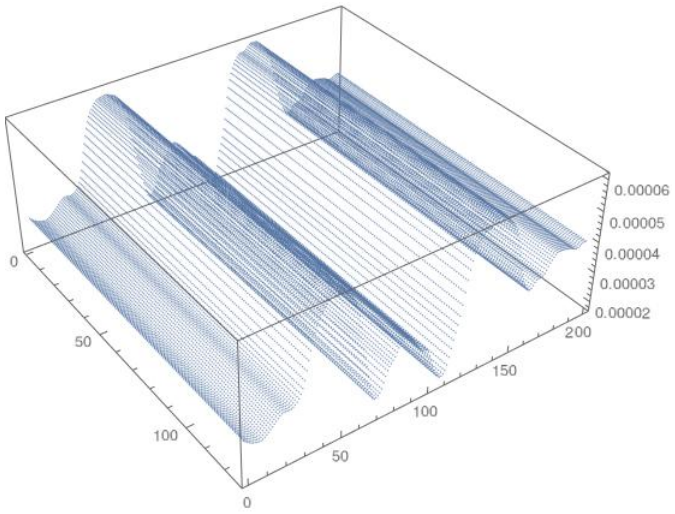

Figure 14: Fonction de probabilité de la position d'une carte sur la table.

Figure 15: Image de tournoi (gauche) générée avec cible (droite).

l'image à cette position. Si une carte existe déjà à cette position, la nouvelle carte est placée par-dessus la carte précédente afin de simuler les occultations observables durant les tournois. Après les insertions, l'image tournoi est terminée et peut être utilisée.

Cible Tournoi

Dans le contexte du travail actuel, il serait impraticable de localiser les positions et orientations de toutes les cartes au sein d'une image de tournoi. Ainsi, il faut procéder en deux étapes. Le première étape consiste à identifier la position des cartes dans l'image tournoi. La seconde est de prendre les positions retournées par la première étape et d'en retirer les éléments de translation, rotation et échelle de manière individuelle. Or, pour cette première étape, il faut développer un réseau capable de détecter la position de cartes dans une image. Cette première étape est celle à laquelle répond le réseau Tournoi. Celui-ci prend une image de tournoi et retourne une image monochrome où un pixel est alumé si celui-ci est proche du centre d'une carte (voir figure 15). C'est en visant le développement d'un tel réseau que la cible du réseau tournoi a été pensée.

Là où l'entrée est une image synthétique d'un tournoi, de l'autre côté la cible elle est une image monochrome où, plus un pixel est proche du centre d'une carte dans l'image d'entrée, plus celui-ci est allumé. Cela donne un effet de points chauds où les cartes

émettent une "chaleur" dans l'image cible qui se réduit plus on s'éloigne de leur centre. Ainsi, en apprenant cette représentation "chaleur" des cartes, le réseau tournoi va développer une habileté à définir le centre d'une carte ce qui lui permet d'accomplir son rôle. Utiliser cette méthode plutôt que la méthode naïve consistant à allumer que le pixel au centre des cartes vient du fait que la méthode naïve est sensible aux réponses triviales de la part du réseau durant l'entraînement[3], à cause du débalancement entre les étiquettes. Ainsi, il est nécessaire que la cible ait une composition suffisamment complexe pour que les réponses triviales n'engendrent pas de minima locaux. L'approche par points chauds permet au réseau "d'apprendre" le centre des cartes tout en évitant les problèmes des minima locaux. Nous allons maintenant étudier comment le jeu de données Tournoi est conçu.

[3] Si 10 cartes se trouvent dans une image to 10 Mégapixels, seulement 10 pixels vont être allumés dans l'image entière avec l'approche naïve. Dans ce cas réseau pourrait répondre l'image nulle et n'obtenir qu'une erreur de 0.001% sur la fonction de coût, rendant ainsi l'apprentissage difficile.

Génération de données

Image d'entrée

Figure 16: Image de fond utilisée pour générer les entrées du réseau Tournoi

Générer une image tournoi débute avec une image de fond RGB de taille 1028 par 648 représentative d'un tournoi standard (voir figure 16). Cette image ne contient que le fond et aucune carte. Par la suite, un entier $n \sim \mathcal{U}(1, 10)$ est généré aléatoirement. Par la suite, n positions (x, y) sont tirées de la distribution 2D énoncée à la section *Images Sythétiques* et, pour chacune de ces positions, une carte *Magic* est insérée dans l'image à cette position. La carte insérée a une rotation choisie aléatoirement entre 0 et 180 degrés si la carte est dans la moitié gauche de l'image. Dans le cas contraire, la rotation de la carte sera choisie entre 180 et 360 degrés. Toutes les cartes de l'image ont la même échelle puisque, dans un contexte de tournoi, toutes les cartes devraient, en temps normal, être à la même distance approximative de la caméra. Une fois, les n cartes insérées, l'image est retournée comme une image d'entrée d'un tournoi simulé.

Cible

La cible tournoi est une image monochrome (1 canal couleur) de taille 1028 par 648. La cible débute comme une image noire, où tous les pixels sont à zéro. Par la suite,

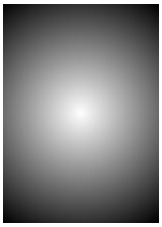

Figure 17: Points chauds insérés dans la cible du réseau Tournoi

pour chacune des positions tirées durant la génération de l'image d'entrée, un point chaud est généré au même endroit dans l'image cible. Un point chaud est, dans ce contexte, une image de taille identique à une carte *Magic* mais dont le contenu est une gaussienne centrée au milieu de la carte (voir figure 17). Cela a comme effet de générer une sorte d'indicateur de probabilité où la valeur du pixel indique la probabilité qu'a celui-ci d'être au centre d'une carte. Évidemment, la valeur au centre d'une carte est 1 et cette valeur réduit linéairement avec l'éloignement du centre d'une carte. La figure 17 illustre un exemple d'un point chaud. Dans le cas spécifique où deux points chauds se superposent, la valeur la plus élevée parmi les points chauds sera celle insérée dans l'image. La figure 15 montre un exemple de cible utilisée pour les tournois.

Absence de ground truth

Comme mentionné ci-haut, les jeux de donnée utilisés dans le cadre de ce projet sont presque exclusivement synthétiques. La raison de cette approche est simple: il n'existe pas de jeux de donnée "réels" sur les images de tournoi *Magic* telles celles illustrées précédemment. De fait, les images d'un tel jeu de donnée seraient probablement extraites d'images statiques ou de flux vidéos ne contenant aucune information sur le réel positionnement des cartes dans l'image. Ainsi, il serait impossible d'y appliquer une approche d'apprentissage supervisé telle que la nôtre sans un très grand effort d'annotation manuelle. Afin de contourner le problème de l'absence de "*ground truth*",

l'approche synthétique a été sélectionnée, car nous la considérons suffisamment réaliste pour substituer les données réelles.

Jeu de données TRS

Le sous-réseau en question (ici nommé le réseau "*Translation, Rotation and Scale*" ou TRS) est défini de la manière suivante: Soit une image de 300 par 300 pixels, pouvant contenir une carte *Magic* avec translation, rotation et échelle arbitraire pouvant être occluse par d'autres cartes, la tâche du réseau est de trouver les éléments TRS de cette carte. Le réseau TRS en tant que tel, n'est pas un réseau de régression (on ne tente pas ici de régresser les valeurs TRS depuis l'image) mais plutôt un réseau de classification. De fait, plutôt que de faire une régression, le réseau tente de classifier les pixels d'une image contenant une carte comme faisant partie, ou non, de la carte. Ce faisant, le réseau génère un masque sémantique isolant la carte du fond. Une fois ce masque trouvé, les valeurs TRS peuvent être facilement calculées. De fait, la translation dans l'image est le centroïde du masque et l'échelle est proportionelle à l'aire du masque. Cependant, en ce qui concerne la rotation, le masque ne peut, malheureusement, que trouver une classe d'équivalence pour la rotation. Puisque le masque est uniforme, il est impossible avec celui-ci d'inférer la rotation sans ambiguïté. Un masque d'une carte pivotée à θ degrés est, en effet, indistinguable du masque d'une carte pivotée à $\theta + 180$ degrés. Le réseau TRS ne peut pas simplement donner le masque comme réponse à la question TRS. Pour résoudre ce problème, une méthode à deux sorties a été adoptée. Le réseau TRS, en plus de donner le masque de la carte, donne aussi une classification entre la partie supérieure d'une carte et la partie inférieure (voir les composantes d'une carte à la section). Avec cette segmentation, il devient alors trivial de distinguer entre le cas θ et $\theta + 180$.

Puisque l'on ne dispose pas de données précises pour entraîner le TRS, il faut donc procéder, de nouveau, par l'entremise de données synthétiques. Chaque exemple

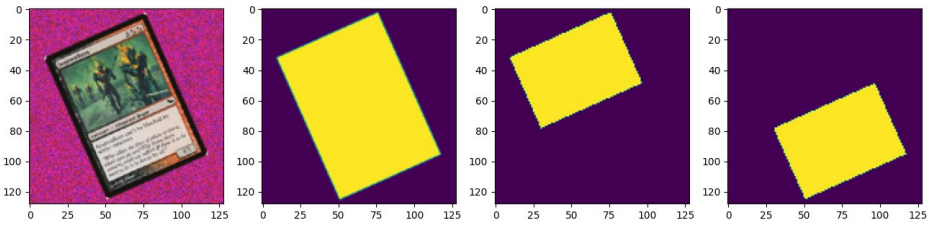

Figure 18: Exemple du jeu de données TRS

synthétique est composé d'une image contenant une ou plusieurs cartes qui peuvent s'occlure mutuellement. Pour chacun des exemples, les cartes se trouvent à l'intérieur d'une fenêtre de 300 par 300 pixels mais peuvent être partiellement occluses si leur position les amène suffisamment proche des bords de l'image. Pour chaque image, un fond est généré à partir de bruit gaussien (non centré en zéro) pour engendrer une invariance au fond. La sortie, quant à elle, est triple. Le premier élément est simplement une image monochrome du masque complet de la (ou les) carte(s). Le deuxième élément est un masque monochrome de la partie supérieure de la carte et la troisième en est de même pour la partie inférieure. La figure 18 illustre un exemple d'entrée de sortie du jeu de données.

Avec ce jeu de données, on peut ainsi créer un réseau TRS capable d'identifier la position, rotation et échelle d'une carte. Cependant, comme mentionné précédemment, il existe de nombreux obstacles à la reconnaissance d'une carte dans une image. De fait, il est, par exemple, possible pour une carte d'être obstruée par une autre ou bien pour une carte de voir sa couleur altérée par la lumière environnante. Pour pouvoir pallier à cette faiblesse, il est nécessaire d'augmenter le jeu de données TRS pour que celui-ci puisse engendrer une invariance à ces éléments.

Augmentation de Données

Lors du design du jeu de données, trois éléments devraient être présents pour répondre à l'énoncé du problème: une translation , une rotation et une échelle variables. Sans cette variance dans le jeu de données, il aurait été impossible pour un réseau d'apprendre la représentation de ces valeurs à la sortie. Or, au-delà de ces variances fondamentales, il a été nécessaire de rendre le réseau invariant à plusieurs éléments qui ne doivent pas influencer la sortie du réseau TRS. Ces éléments sont:

- La saturation des couleurs dans l'image;
- Le contraste de l'image (écart entre les extrêmes des pixels dans l'image);
- La variation du fond dans l'image;
- La possibilité qu'une carte en cache une autre partiellement (superposition).

Pour chacun de ces éléments, une certaine augmentation a été appliquée au jeu de données TRS. Nous allons maintenant étudier chacune de ces augmentations en détails.

Saturation et Contraste

Afin d'engendrer une invariance à l'environnement lumineux de la scène, il est nécessaire d'inclure une variance dans la saturation et le contraste d'une carte du TRS. Cela est accompli en appliquant un filtre de saturation aléatoire pouvant faire varier indépendamment la saturation et le contraste de 50% à 115% par rapport à l'image originale. Ce faisant, le réseau développe une invariance à la saturation et au contraste, ce qui augmente la capacité généralisatrice du réseau TRS. Les ajustements de saturation et contraste sont basés sur un "*alpha blend*" de l'image originale et d'une image moyenne dans le cas du contraste ou d'une image en tons de gris dans le cas de la saturation.

Figure 19: Exemple de saturations et contrastes variés. Contraste réduit (gauche), Original (centre), saturation réduite (droite).

Pour cette raison, ces filtres sont principalement utilisés pour réduire la saturation et le contraste plutôt que les augmenter (voir figure 19).

Invariance au Fond

Pour être généralisable à une variété de situations et tournois, le réseau TRS doit être invariant au fond car, il est impossible de prévoir sur quelle surface l'image est prise. Pour ce faire, lors de la génération d'images, un fond est généré aléatoirement. Pour chaque pixel du fond et chaque canal du pixel, sa valeur est tirée d'une variable aléatoire normale $\chi \sim \mathcal{N}(\mu, \sigma)$. Chaque valeur canal est tirée d'une distribution avec moyenne et écart-type tirés aléatoirement de distributions uniformes. Cela a comme effet de, non-seulement générer un bruit pour le fond, mais aussi de générer un bruit dont la moyenne est, elle aussi, non-uniforme. La raison de cette mesure est que si la moyenne de chacun des canaux couleur est la même, cela va générer un fond dont la couleur moyenne va tirer vers le blanc. Au fil de l'apprentissage, le réseau apprendra alors cette moyenne et cela va pénaliser la capacité de généralisation pour les fonds dont la moyenne n'est pas blanche. Ainsi, en créant un bruit variable pour le fond, on engendre une invariance au fond (voir figure 20).

Figure 20: Exemples de fonds variés

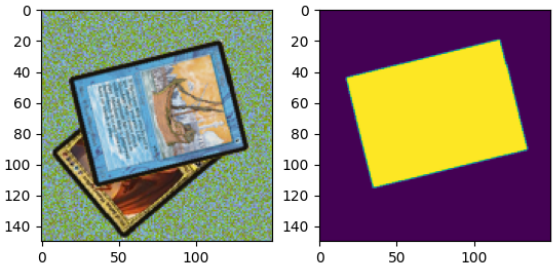

Figure 21: Exemple d'invariance à la superposition

Invariance à la superposition

Dans la situation où plusieurs cartes se superposent, il ne serait pas utile pour le masque ainsi que les partitions de prendre en comptes toutes les cartes. Ainsi, si plusieurs cartes se superposent, seulement la carte qui est complètement visible devrait être considérée pour le masque et les partitions. Afin d'introduire ce comportement au sein du réseau, le jeu de données TRS contient des images de cartes superposées. Avec ces images, la cible est simplement le masque et les partitions de la carte qui se trouvant par-dessus les autres. En incluant des cas où plusieurs cartes se superposent, le réseau apprend à ignorer les cartes qui se trouvent cachées par d'autres cartes. La figure 21 montre un exemple de cette invariance.

Génération de données

Image d'entrée

Le processus débute avec l'image de la carte sous forme de matrice RGB $C_{204\times146\times3}$. L'image est ensuite transformée en image RGBA $C_{204\times146\times4}$ où la nouvelle dimension est égale à une matrice $0_{204,146}$, soit $C_{i,j,4} = 0, \forall(i,j) < (204, 146)$. Une fois l'image convertie en RGBA, nous remplissons les deux premières dimensions jusqu'à (300×300) avec pixels noirs transparents (soit $C_{i,j} = [0\ 0\ 0\ 0]$). Par la suite, l'image est pivotée d'un angle $\theta \in [0, 2\pi]$. Une fois la rotation appliquée, une transformation afine $A_{t_x,t_y,s}$ est générée aléatoirement basée sur trois paramètres: $(t_x, t_y, s) \in ([-50, 50], [-50, 50], [0.8, 1.2])$. Les paramètres t_x et t_y représentent une translation x et y en pixels et le paramètre s est un facteur d'échelle affectant la taille de la carte au sein de l'image. $s = 0.8$ implique une échelle de 80% jusqu'à $s = 1.2$ pour une échelle de 120% par rapport à la taille originale de l'image. Une fois la transformation créée, elle est appliquée sur l'image résultant en une carte déplacée et mise à l'échelle voulue.

Comme dernière étape, le fond de l'image est remplacé par du bruit aléatoire. Pour déterminer la nature du bruit généré, trois paires de bornes $(d_R, f_R), (d_G, f_G), (d_B, f_B)$ sont créées aléatoirement, où $d_N \leq f_N$. Les bornes d_N et f_N indiquent, respectivement, le début et la fin du domaine d'une distribution uniforme. Chaque paire indique ainsi une distribution uniforme à partir de laquelle sera pigée la valeur d'un canal de couleur N pour un pixel donné dans le fond de l'image. Ainsi, trois distributions uniformes distinctes sont générées, soit: $Bruit_R = \mathbf{U}(d_R, f_R)$, $Bruit_G = \mathbf{U}(d_G, f_G)$ et $Bruit_B = \mathbf{U}(d_B, f_B)$. Ainsi, pour chaque pixel du fond, celui-ci se voit assigné une valeur à chaque canal couleur tirée de la distribution uniforme correspondante. Cela a pour effet de non seulement générer un bruit aléatoire, mais aussi un bruit aléatoire dont la moyenne est variable d'une image à l'autre. Ce processus complexe de génération de bruit pour le fond est mis en place pour éviter que le bruit soit entièrement uniforme. De fait, si le

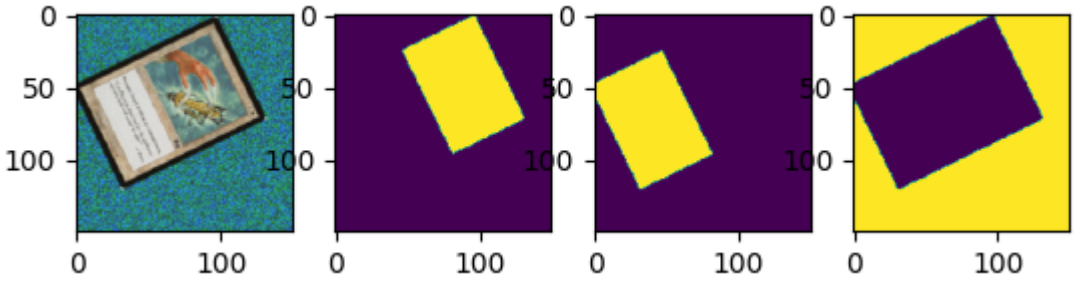

Figure 22: Exemple de sortie TP du réseau TRS. Dans l'ordre: l'image d'entrée, la masque supérieur, le masque inférieur, et le masque de fond.

bruit s'avère être uniforme, le fond sera invariablement du bruit blanc et cette constante créera un biais lors de l'apprentissage du réseau. Ce point sera analysé en profondeur à la section *Approche*. Une fois le fond bruité, l'image d'entrée est terminée et peut être envoyée au réseau.

Cible

La cible de notre jeu de données est double. Elle est composée d'une image monochrome $M_{300\times300}$ appelée le *Masque* et d'un tenseur $P_{300\times300\times3}$ appelé le *Tenseur Partitions* ou TP. Le masque M est simplement une image binaire où un pixel est allumé (ayant la valeur 1) si et seulement si son homologue dans l'image d'entrée se trouve dans la carte. Si, au contraire, celui-ci fait partie du fond, le pixel équivalent du masque est éteint (possède la valeur de 0).

Le TP est similaire au masque dans la mesure où il peut être considéré comme trois masques étant mutuellement exclusifs entre eux. Comme le montre la figure 22, pour le TP, la carte est séparée en deux sections: Une supérieure et une inférieure. Ainsi, les deux premiers canaux du TP sont en fait des masques pour leur moitié respective. Dans le TP, le premier canal correspond à la moitié supérieure et le deuxième à la moitié

inférieure. Le dernier canal indique que le pixel donné dans l'image ne se trouve dans aucune des deux moitiés de la carte (soit, celui-ci est à l'extérieur de la carte). Ainsi pour un pixel (x, y) donné, le TP indique sous la forme d'un "onehot", si le pixel fait partie d'une moitié de carte et, si tel est le cas, dans quelle moitié celui-ci se trouve. Nous verrons à la section *Approche* comment cette structure est utile avec un réseau de sortie "softmax" avec une fonction de perte de CrossEntropy catégorique, et comment il sera facile d'en extraire les paramètres de position, de rotation et d'échelle.

APPROCHE

De manière générale, ce projet cherche à implanter un pipeline qui récupère une image d'un tournoi *Magic*, et identifie, pour chaque carte dans l'image, la position, la rotation, et l'échelle de celle-ci. Pour ce faire, le flux de travail de ce pipeline est illustré à la figure 6 (voir section *Jeu de données*). Une image de tournoi est donnée au réseau "Tournoi" et celui-ci retourne une "carte de chaleur" indiquant la position approximative des cartes dans l'image. Pour chacune des cartes détectées, une section de l'image est coupée et donnée au réseau TRS. Le réseau TRS prend les extraits de l'image tournoi et, pour les cartes dans celles-ci, va retourner la position, rotation et échelle. Dans ce chapitre, nous allons analyser les deux réseaux qui composent ce flux de travail: Soit le réseau tournoi et le réseau TRS. Cependant, avant de procéder, il serait important de justifier une approche à deux réseaux.

Notre approche se veut en deux étapes: une de positionnement approximative suivie d'une tâche d'estimation de pose pour raffiner la position et identifier la rotation et l'échelle de la carte. La question se pose alors: pourquoi ne pas considérer une architecture monolithique où le réseau prend en entrée l'image tournoi et retourne directement la position, la rotation et l'échelle des cartes dans l'image. La raison principale est que les problèmes résolus par les réseaux sont très différents. Le premier problème cherche à détecter l'ensemble des cartes dans une image alors que le second cherche à identifier les éléments TRS d'**une seule** carte. Le premier réseau peut résoudre son problème en une étape alors que le second réseau doit être appliqué de multiples fois en fonction des cartes détectées. Il serait possible de combiner ces deux réseaux en un, cependant, cela obligerait à fixer le nombre de cartes pouvant être détectées simultanément (car la sortie d'un réseau doit être de taille fixe) mais aussi, cela augmenterait la taille du réseau de manière significative rendant l'apprentissage particulièrement long.

De plus, l'approche naïve, quoique prometteuse à la surface, amène des défis importants. Le plus importants de ceux-ci est le fait qu'avec cette approche, la totalité de l'image est prise en compte lors de l'inférence des éléments TRS. Cela a pour effet de rendre le modèle très susceptible aux variations dans l'image. Puisque tous les pixels de l'image sont pris en compte à l'entrée, il se pourrait très bien qu'une variation de quelques pixels dans une petite région de l'image fasse varier de manière importante les résultats à la sortie. Ainsi, avec cette méthode naïve, on entraîne la possibilité de créer des dépendences entre des pixels de l'image d'entrée qui ne sont normalement pas reliés.

Afin de contourner ce problème, une approche inspirée de Redmon et Farhadi [1] a été utilisée. Sous cette approche, un mécanisme d'attention est utilisé afin d'indiquer au réseau où "regarder". Ce mécanisme d'attention indique au réseau les régions où se trouvent les objets d'intérêt et permet à celui-ci de pouvoir identifier des objets avec une meilleure précision. Motivé par cette approche, notre modèle utilise le réseau tournoi comme manière de limiter la portée du réseau qui identifie les valeurs TRS. De fait, indépendamment du résultat du réseau tournoi, l'entrée du réseau tournoi est toujours locale ce qui élimine le principal problème de l'approche naïve.

Réseau Tournoi

Comme on peut le voir à la figure 6, le réseau Tournoi est un réseau de neurones qui prend une image de tournoi et qui retourne une image de points de chaleur (*heatmap*) indiquant où se trouvent les cartes dans l'image. L'image d'entrée est de taille 1028 par 648 et possède 3 canaux couleur (RGB). L'image de sortie est monochrome où un pixel est allumé s'il est proche du centre d'une carte. Elle possède la même taille que l'entrée à l'exception d'avoir un seul canal couleur.

Le réseau Tournoi possède une structure de réseau convolutif. L'image d'entrée est

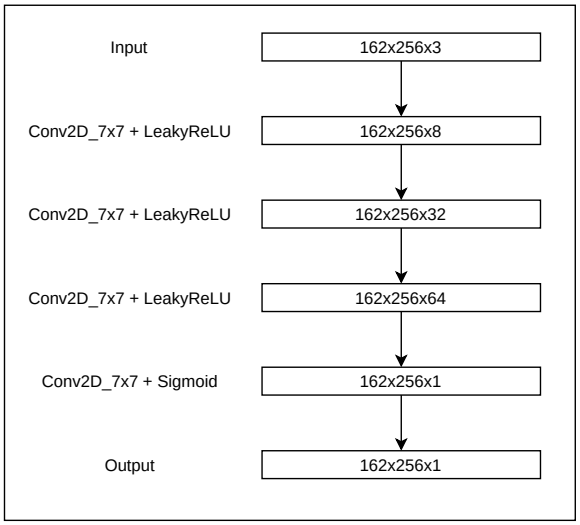

Figure 23: Structure du réseau Tournoi

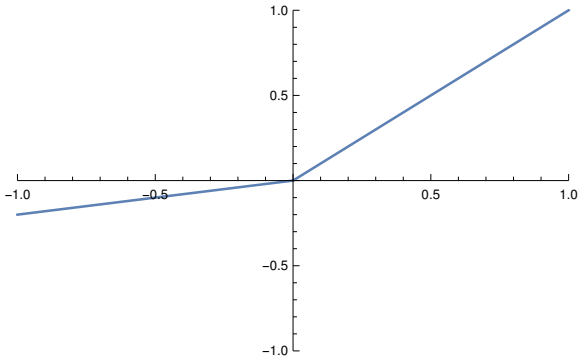

Figure 24: Fonction d'activation $LeakyReLU_{0.2}(x)$.

Algorithm 1: Fonction $nms(M, r, s)$

Data: Matrice $\mathbf{M}_{n\times m}$, rayon r, seuil s

Result: Matrice $\mathbf{N}_{n\times m}$ où $\mathbf{N}_{i,j} = 1 \iff$ une carte est centrée en (i, j)

$\mathbf{N} \leftarrow \mathbf{0}_{n\times m}$;

for (i, j) *from* $(1, 1)$ *to* (n, m) **do**

  if *Le rayon r centré en* (i, j) *ne dépasse pas les bords de l'image* **then**

    $\mathbf{N}_{i,j} \leftarrow detect\_max(\mathbf{M}_{[i-r,i+r],[j-r,j+r]}, s)$

  end

end

return N ;

Figure 25: Algorithme de suppression de non-maxima. Pour une image tournoi, on vérifie pour chaque pixel s'il est un maximum à l'intérieur de son voisinage de taille $2r + 1$. Pour être considéré comme un maximum, un pixel doit dépasser une valeur supérieure où égale à s.

passée à travers une série de blocs convolutifs pour retrouver une carte de chaleur indiquant la position des cartes dans l'image. Les blocs convolutifs sont composés d'une couche de convolution suivie d'une non-linéarité de type LeakyReLU[4] avec un α de 0.2 (la figure 24 illustre la fonction d'activation). Chaque convolution possède un noyau 7×7 avec un pas de 1 pour préserver la dimension de l'entrée. À chaque convolution, le nombre "d'*attributs*" est doublé. La seule exception est le dernier bloc convolutif qui, en plus de n'avoir qu'un seul canal en sortie, possède une non-linéarité sigmoïde[5] afin d'avoir une sortie dans le domaine $]0, 1[$. Le réseau est composé de 117313 paramètres. La structure générale du réseau est visible à la figure 23.

La sortie du réseau tournoi donne la probabilité d'un pixel d'être au centre d'une carte.

[4] $LeakyReLU_\alpha(x) = \mathbb{1}_{x>0} * x + \mathbb{1}_{x\leq 0} * (\alpha * x)$

[5] $\sigma(x) = \frac{1}{1+e^{-x}}$

Algorithm 2: Fonction $detect\_max(M,s)$

Data: Matrice $\mathbf{M}_{n\times n}$, seuil s

Result: 1 si le pixel au centre c est plus grand que s et s'il est le maximum de la matrice $\mathbf{M}$, 0 sinon.

$c \leftarrow \lfloor n/2 \rfloor$;

val $\leftarrow \mathbb{1}_{\geq 0}(\min(\mathbf{M}_{c,c} - \mathbf{M})) * \mathbb{1}_{\geq s}(\mathbf{M}_{c,c})$;

return val ;

Figure 26: Algorithme pour la fonction *detect_max*. Pour le voisinage donné, on vérifie si le pixel au centre est le maximum parmi tous les autres en plus de demander à ce que le pixel soit assez proche de 1. Si toutes les conditions sont remplies, l'algorithme retourne 1. Dans le cas contraire, il retourne 0.

Puisque nous voulons choisir qu'une seule position par carte, nous devons sélectionner le pixel avec la plus haute probabilité. Puisque plusieurs cartes peuvent être dans l'image, il n'est pas possible de simplement trouver le maximum de l'image de sortie. Pour trouver les multiples maxima, un algorithme de suppression de non-maxima (NMS) est appliqué à l'image pour retrouver la position approximative des cartes dans l'image. Les figures 25 et 26 décrivent cet algorithme. La figure 27 illustre un exemple de sortie du réseau Tournoi ainsi que le résultat de la suppression des non-maxima.

Réseau TRS

Le réseau TRS, comparativement au réseau Tournoi, est plus complexe. Plutôt que de retourner les positions de plusieurs cartes, le réseau présuppose qu'une seule carte est présente dans l'image d'entrée et essaie de retrouver sa translation, sa rotation, et son échelle. Comme l'illustre la figure 28, ce réseau peut être décomposé en quatre composantes: l'encodeur, le décodeur commun, le décodeur Masque, et le décodeur

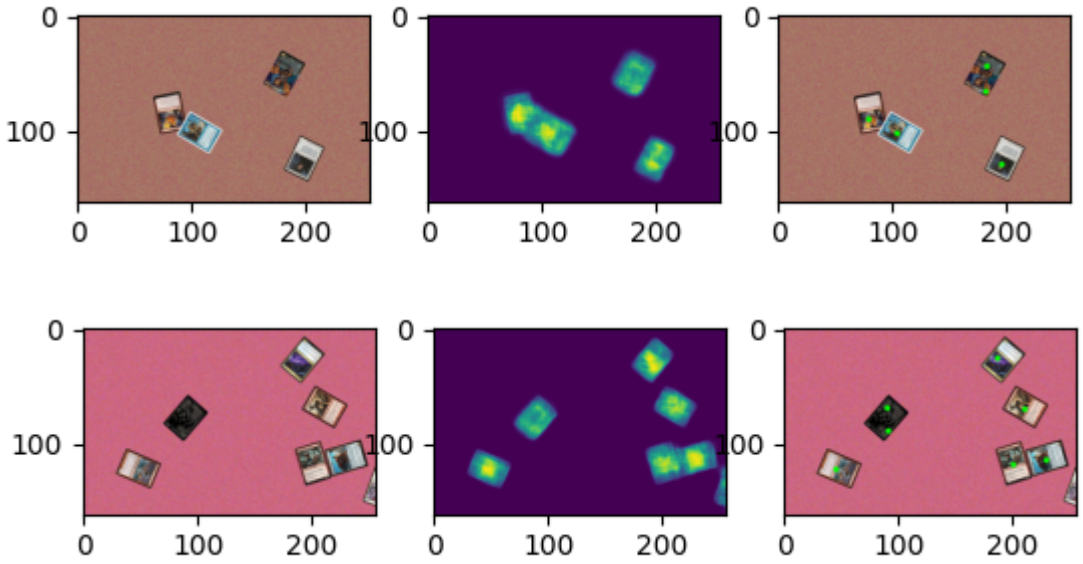

Figure 27: Exemple de sortie du réseau Tournoi. Image d'entrée à gauche et les centres trouvés par NMS à droite (points verts). L'image au milieu est la sortie du réseau Tournoi qui génère les points chauds à partir desquels l'algorithme NMS trouve les centres des cartes.

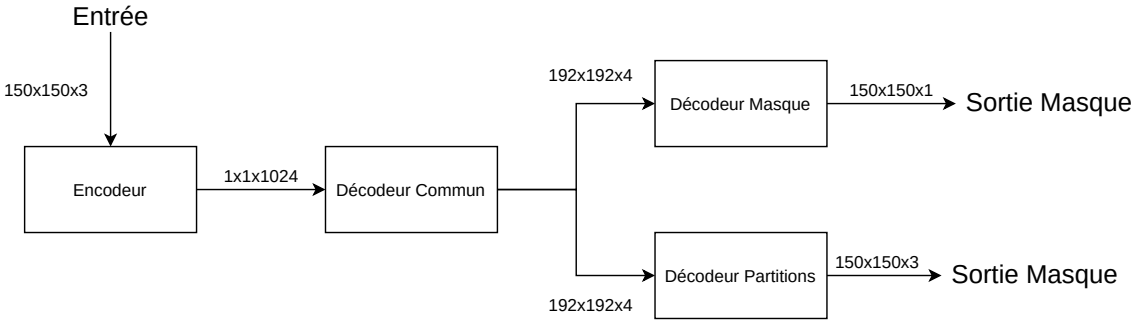

Figure 28: Structure globale du réseau TRS

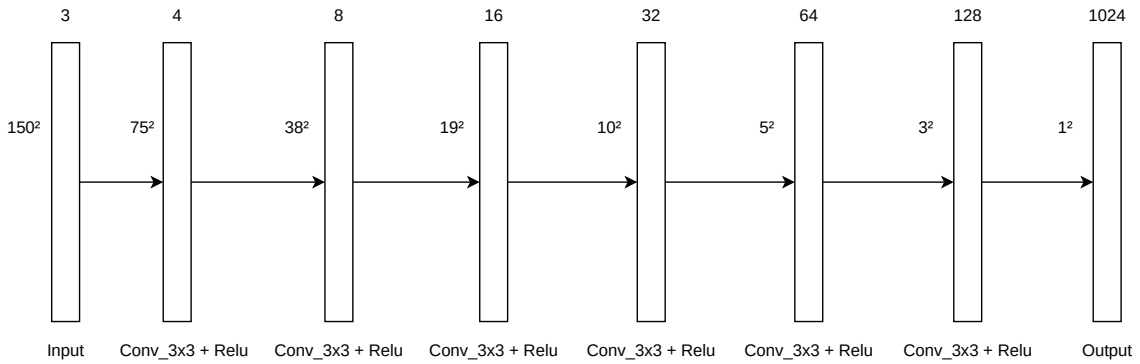

Figure 29: Structure de l'encodeur TRS

Partitions. Nous allons analyser chaque élément du réseau indépendamment avant d'étudier la topologie générale du réseau TRS et justifier cette topologie.

Encodeur

L'encodeur TRS (voir figure 29) a comme rôle, au sein du réseau TRS, d'encoder l'information d'une image de carte de telle sorte à ce que les trois autres composantes puissent utiliser cette information pour générer le masque ainsi que les partitions (haute et basse) de la carte dans l'image. Ainsi, le réseau prend une image de carte de taille 150×150 avec trois canaux couleur (RGB) (donc un tenseur $150 \times 150 \times 3$) et, à l'aide de blocs convolutifs, génère un encodage sur 1024 valeurs réelles ($1 \times 1 \times 1024$). Chaque bloc convolutif de l'encodeur est composé d'une convolution avec un noyau 3×3 et d'un pas de 2 afin de réduire progressivement la taille (à l'exception de la dernière convolution avant la sortie qui possède un pas de 1). Après chaque convolution, une activation de type ReLU est ajoutée.

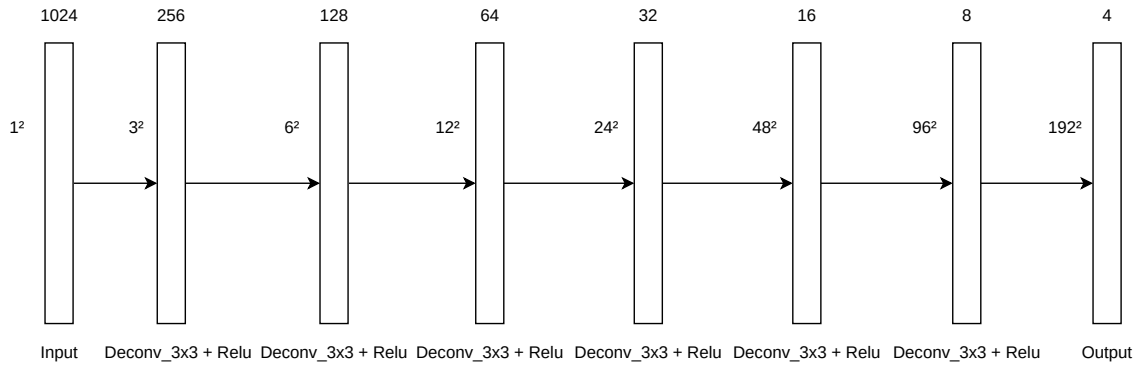

Figure 30: Structure du décodeur commun TRS

Décodeur Commun

Le décodeur commun TRS a la tâche de prendre l'encodage de l'encodeur TRS et de le décompresser vers une représentation que les deux dernières composantes du réseau vont utiliser pour accomplir leur tâche. La figure 30 illustre la structure de cette composante. Le décodeur est composé d'une série de blocs déconvolutifs qui augmentent la taille de l'image depuis l'espace d'encodage jusqu'à un tenseur $194 \times 194 \times 4$. Le choix de 4 cannaux à la sortie est empirique et sélectionné pour donner suffisamment d'informations aux deux réseaux qui lui succèdent. À chaque bloc déconvolutif, le nombres d'attributs est réduit de moitié, partant de 256 jusqu'à 4. Chaque bloc déconvolutif est composé d'une déconvolution suivie d'une activation ReLU. Les déconvolutions du décodeur ont toutes un noyau 3×3 et un pas de 2 (à l'exception de la première convolution qui a un pas de 1).

Décodeur Masque

Le décodeur Masque (figure 31) est un réseau simple dont l'objectif est d'extraire de la représentation du décodeur commun un masque qui identifie l'intérieur la carte. Ce décodeur est simplement une déconvolution avec un noyau 3×3, un pas de 1, et pos-

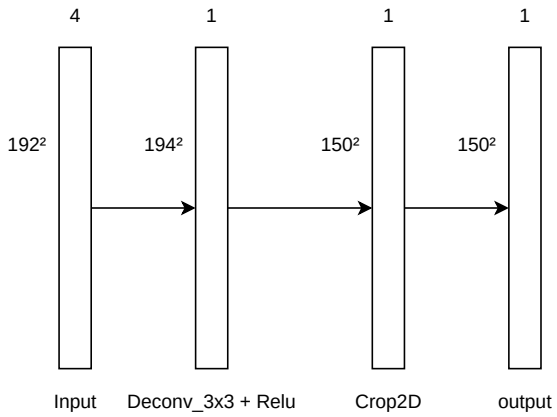

Figure 31: Structure du décodeur masque TRS

sédant un seul attribut, le masque lui-même. Suivant la déconvolution, une couche de découpage ("*cropping*") ajuste la taille de l'image à 150×150 (qui est la taille originale de l'entrée). Finalement, une couche de sortie avec activation sigmoïde ramène toutes les valeurs de sortie entre 0 et 1. La figure 32 montre un exemple de sortie sur une carte typique, avec et sans occultation partielle.

Une fois le masque obtenu, il suffit simplement de calculer le centroïde de celui-ci pour trouver la position de la carte dans l'image. De la même manière, le ratio de l'aire du masque sur l'aire d'une carte standard nous donne l'échelle. On note que si plusieurs cartes sont visibles, mais partiellements cachées par une superposition, c'est la carte du dessus qui est considérée.

Décodeur Partitions

Contrairement au décodeur Masque qui identifie la position et l'échelle d'une carte, le décodeur Partitions cherche plutôt à estimer l'orientation de la carte. Le décodeur Partitions utilise l'information du décodeur commun pour générer une carte de probabilité où chaque pixel de l'image est classifié selon trois catégories: soit, le pixel est

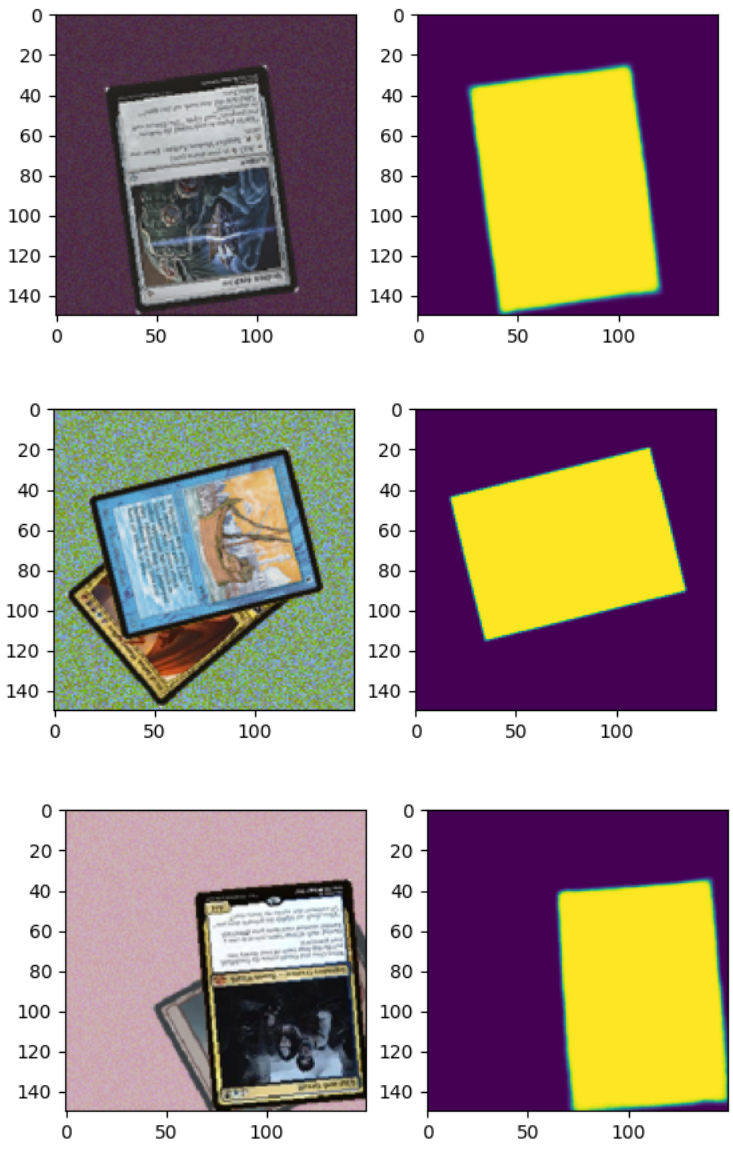

Figure 32: Exemple de sortie du décodeur masque. Lorsqu'il y a plusieurs cartes, seulement la carte du dessus est pertinente au masque. Comme on peut le voir à l'image du bas, un décalage de la carte par rapport au centre n'affecte pas la détection du masque. On remarque aussi une invariance à la superposition, propriété essentielle pour estimer la position et l'échelle avec précision.

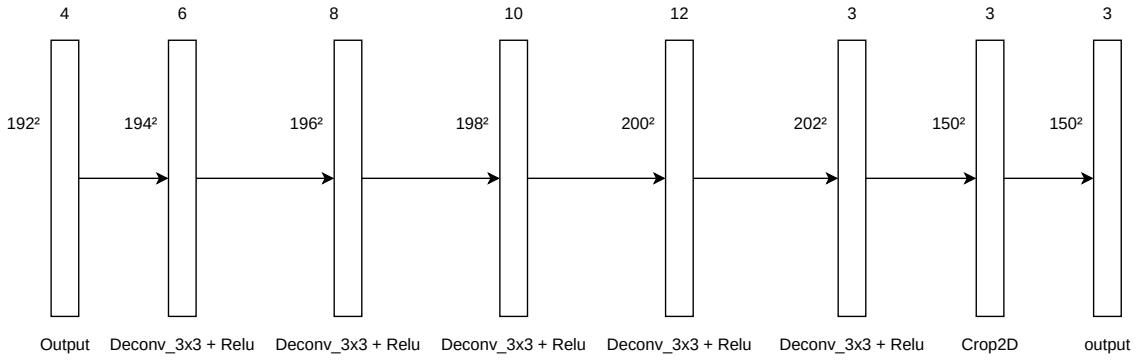

Figure 33: Structure du décodeur Partitions du réseau TRS

dans la partie supérieure de la carte, soit il est dans la partie inférieure, soit il est à l'extérieur de la carte. Le décodeur (voir figure 33) est composé de blocs déconvolutifs où chaque bloc est composé d'une déconvolution suivie d'une activation ReLU. Chaque déconvolution possède un noyau 3×3 et un pas de 1 tout en augmentant le nombre de d'attributs progressivement de 4 à 12. La dernière déconvolution, elle, ramène le nombre d'attributs à 3 pour la sortie. Après les blocs convolutifs, une couche de "*cropping*" ajuste l'image à 150×150 pour la sortie. Finalement, une couche d'activation softmax est appliquée à la sortie pour normaliser les probabilités des trois classes pour chaque pixel. Des exemples de sorties sont illustrés à la figure 34.

Topologie générale

Le réseau TRS complet prend une image représentée sous forme d'un tenseur de taille $150 \times 150 \times 3$ et applique l'encodeur pour obtenir un encodage de taille $1 \times 1 \times 1024$. L'encodage est ensuite décodé vers une taille de $194 \times 194 \times 4$ par le décodeur commun. Finalement, les deux branches utilisent ce décodage pour en générer deux sorties: la sortie Masque de taille $150 \times 150 \times 1$ et la sortie de Partitions de taille $150 \times 150 \times 3$. La raison d'une topologie par branche est simple. En forçant la branche masque et la branche segmentation à partager les poids de l'encodeur et du décodeur commun, on

permet à la branche Partition d'utiliser les données de la branche Masque pour obtenir la bonne segmentation. Cela vient du fait que le segment de décodeur Masque est extrêmement petit. Ainsi, l'information venant du décodeur commun doit être assez similaire à la sortie Masque pour que la branche Masque ait une petite valeur de perte. Cela implique que la branche Partitions, qui est basée sur le décodeur commun, doit apprendre à utiliser les informations du décodeur commun (qui va avoir tendance à être proche du masque) pour trouver les bonnes segmentations. Ainsi, le système par branches permet à l'information relative au masque d'influencer la sortie de Partitions.

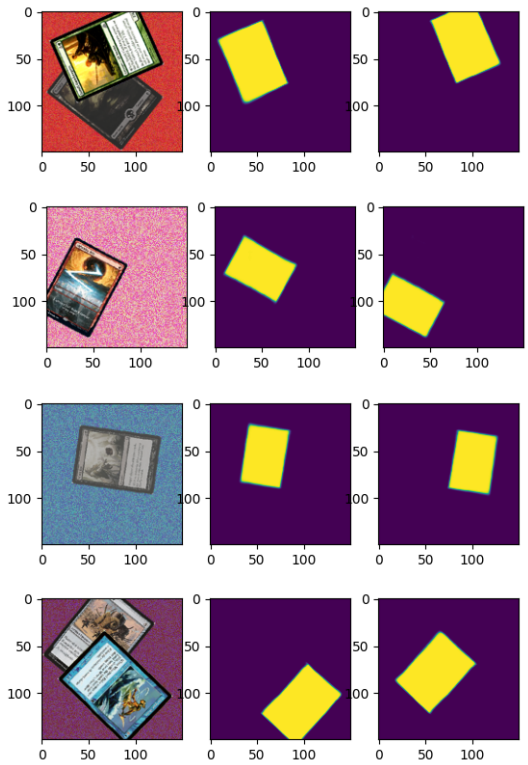

Figure 34: Exemple de sortie du décodeur Partitions du réseau TRS. Image d'entrée (gauche), masque de la partie supérieure (centre), masque de la partie inférieure (droite). On remarque une invariance à la superposition de carte, une propriété essentielle pour estimer l'orientation avec précision.

.

RÉSULTATS

Le problème auquel s'intéresse ce travail est d'identifier la pose de cartes *Magic* dans une image de tournoi *Magic*. Comme mentionné au chapitre *Approche*, la stratégie est d'utiliser les deux réseaux (Tournoi et TRS) en tandem pour définir un système capable de prendre une image de tournoi et d'en extraire les éléments TRS pour chaque carte (avec exceptions). Afin de pouvoir bien analyser les résultats, nous étudierons la performance qualitative et quantitative de ces deux réseaux avant de fournir les résultats finaux.

Il est à noter que tous les résultats démontrés ci-dessous ont été générés sur une entrée d'images fixes. Il serait possible d'appliquer un tel système sur une séquence vidéo (en appliquant l'algorithme sur les images de la séquence) mais cela reste en dehors des objectifs de ce travail. De plus, puisque le système est un réseau non-récurrent, sa performance sur des séquences reste limitée. Pour ces raisons, les résultats de cette section sont basés sur des images fixes *tirées de véritables séquences vidéo de tournoi*.

Résultats Intermédiaires

Avant de discuter des résultats de l'algorithme complet sur des situations réalistes, il serait utile d'étudier la performance des sous-réseaux de manière individuelle.

Réseau Tournoi

Comme décrit au chapitre *Approche*, le réseau Tournoi a comme tâche de prendre une image d'une séquence de tournoi *Magic* et d'en identifier les centres des cartes. Simplement dit, le réseau Tournoi est la première étape de l'algorithme et son but est d'indiquer au réseau TRS où "regarder".

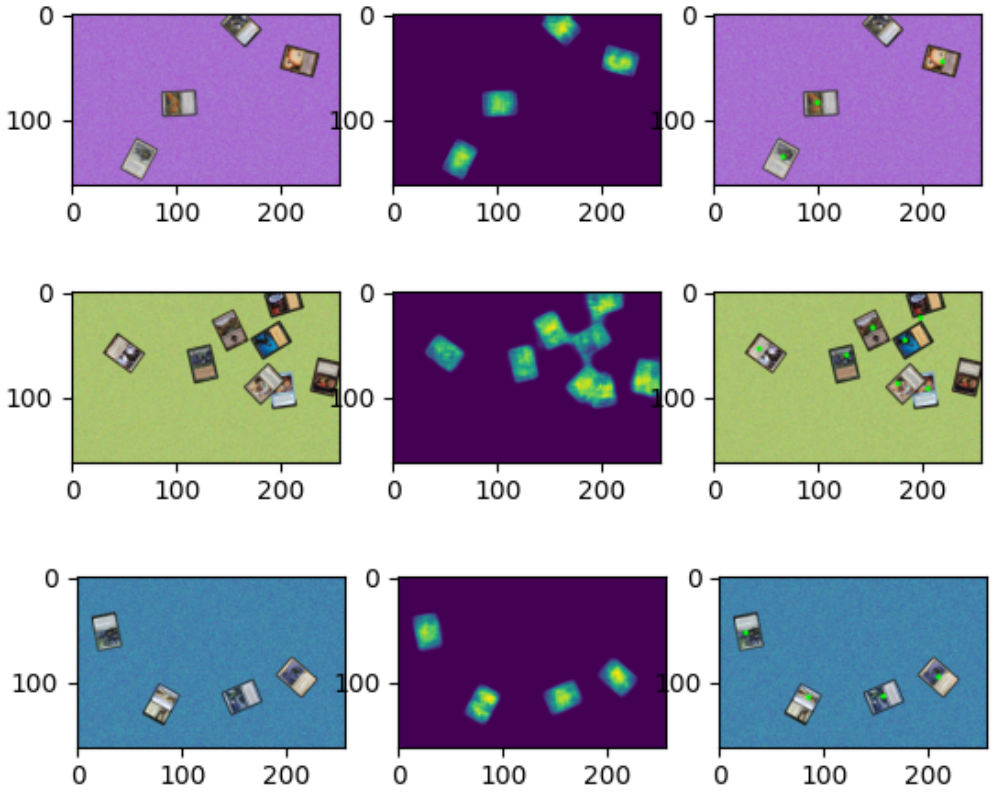

Figure 35: Résultats du réseau Tournoi. À gauche, image synthétique en entrée. Au milieu, estimation des "distances au centre de la carte". À droite, résultat de la suppression des non-maximums.

Figure 36: Résultats du réseau Tournoi sur des images réelles de tournoi. En vert: maximums détectés après supression des non-maximum.

Entraînement

L'entraînement s'est fait à partir du jeu de données décrit au chapitre *Jeu de données*. Une image de taille 256×162 pixels RGB est générée avec un nombre de cartes pigées de manière uniforme de l'intervalle $[0, 10]$. Puisque les données du jeu Tournoi sont générées sur demande, chaque point de donnée n'est utilisé qu'une seule fois, évitant les problèmes de surapprentissage. Pour l'entraînement, 4500 points de données synthétiques ont été générés à chaque époque avec une taille de lot (*batch size*") de 16 et un taux d'apprentissage est $\lambda = 10^{-4}$ avec l'algorithme d'optimisation *Adam*[4] et une condition d'arrêt prématuré sur 50 époques[6]. Sur une carte graphique Nvidia Geforce GTX 1060, l'entraînement a duré 221 époques, avec une moyenne de 211 secondes par époque, pour une durée totale approximative de 13 heures. Au total, le réseau aura appris sur un ensemble de 994500 points de donnés uniques sans aucune répétition.

Les figures 35 (synthétique) et 36 (réel), illustent des exemples de performance du réseau Tournoi. Pour la figure 35, la colonne de gauche est l'image d'entrée, et celle de droite est l'image d'entrée avec les centres trouvés superposés sur celle-ci (points verts). Finalement, la colonne du milieu montre la sortie brute du réseau qui est utilisée pour calculer les centres. La figure 36 illustre simplement des images réelles de tournoi avec les centres trouvés superposés sur l'image.

Comme on peut le voir, le réseau est capable d'atteindre son objectif assez facilement sur les images synthétiques et réussit à assigner un centre à presque chaque carte dans l'image. De fait, on peut voir que le réseau Tournoi génère bel-et-bien des "points chauds" pour chaque carte dans l'image permettant à l'algorithme NMS de trouver des maxima dans l'image.

Les résultats sur des données réelles sont plus mitigés mais restent concluants. Le

[6] L'entraînement s'arrête automatiquement s'il n'y a aucun amélioration de la fonction de coût sur l'ensemble de validation pendant 50 époques.

Figure 37: Impact du seuil sur les résultats tournoi. Seuil de 0.5 (gauche) et seuil de 0.8 (droite).

réseau est capable d'identifier la majorité des cartes dans les images et donne un centre raisonnable à chaque carte trouvée. Il est à noter que l'absence de certains centres peut être due au seuil utilisé dans l'algorithme NMS. De fait, comme le montre la figure 37, le seuil peut engendrer de nombreux faux positifs. Puisque le réseau TRS ne peut pas bien représenter l'absence de carte, le contrôle des faux positifs à l'étape tournoi est important.

Limitations

Malgré une bonne performance sur l'ensemble de test et dans une bonne partie des cas réels, le réseau possède une limitation importante. Comme le montrent les figures, 35, 36, et 38, le réseau est de manière générale, efficace pour détecter la présence d'une carte mais reste peu précis quant à la position du centre des cartes. De fait, comme le montre la figure 39, l'erreur de distance moyenne entre le centre estimé et le centre réel pour le réseau Tournoi est de 30 pixels (sur 450 images 1028x648), ce qui est assez élevé. En fait, seulement environ 6% des résultats du réseau Tournoi ont une erreur inférieure à 10 pixels. Ainsi, il n'est pas raisonnable de s'attendre à des positionnements précis de la part du réseau. Cela n'est pas nécessairement un

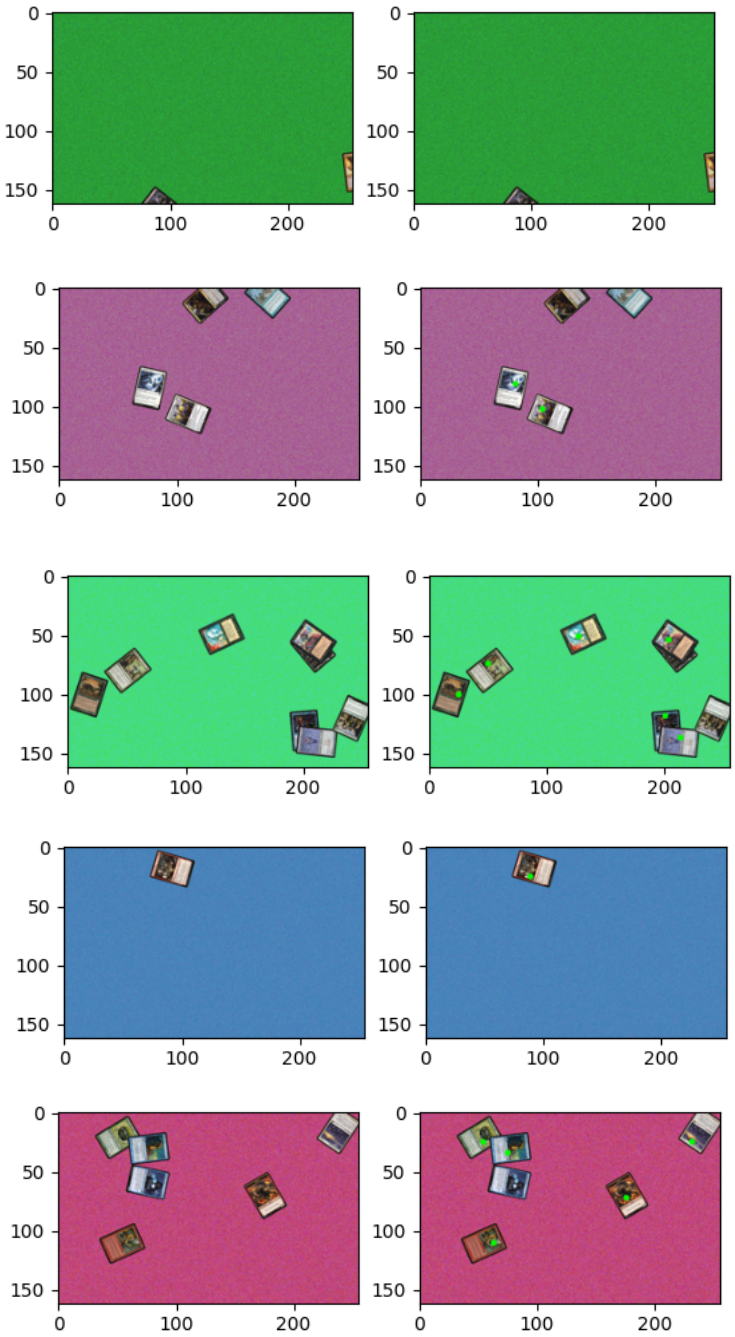

Figure 38: Exemples de faiblesses du réseau Tournoi. Comme on peut le voir, le positionnement est approximatif. De plus, le réseau est incapable d'identifier les cartes coupées par les bords de l'image.

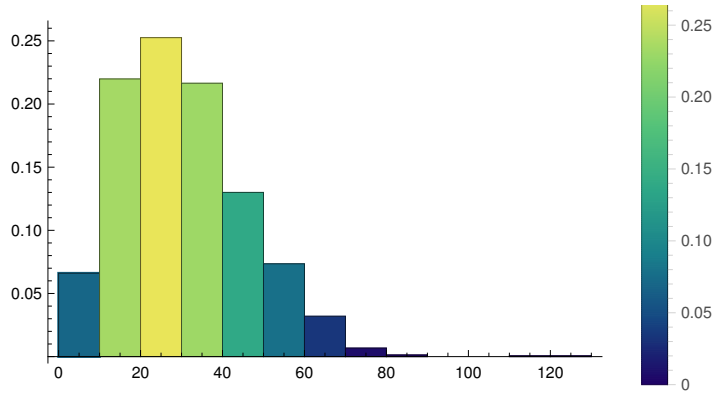

Figure 39: Erreur de localisation du réseau Tournoi. Histogramme de la distance entre le centre estimé et le centre réel (images 1028×648). Axe des X: Erreur de positionnement en pixels.

problème, car le réseau Tournoi vise avant tout la détection, pas la localisation. Ainsi, l'erreur de positionnement engendrée par le réseau Tournoi est ensuite corrigée par le réseau TRS. Comme nous allons le voir, la précision du réseau TRS est telle que l'erreur du réseau Tournoi, pour la majorité des cas, est complètement éliminée des résultats finaux.

Réseau TRS

Comme mentionné ci-haut, le réseau TRS est la deuxième composante du pipeline complet. Une fois les centres des cartes trouvés par le réseau Tournoi, le réseau TRS, lui, doit trouver la position précise de la carte (composante "T" ou translation), l'angle de la carte (la composante "R" ou rotation), et finalement, l'échelle de la carte (la composante "S" ou "*scale*").

Entraînement

L'entraînement s'est fait à partir du jeu de données TRS décrit à la section *Jeu de données*. L'entraînement a été fait sur 1000 époques de 16000 points de données, ceux-ci regénérés sur demande à chaque époque comme pour le réseau Tournoi. Puisque chaque point de donnée est généré aléatoirement sur demande, le réseau aura vu, durant son entraînement, près de 16 millions de points de données uniques. Le "*batch size*" était 16 et le taux d'apprentissage $\lambda = 10^{-4}$ avec l'algorithme d'optimisation *Adam*[4] et une condition d'arrêt prématuré sur 100 époques. Les fonctions de coût utilisées sont une entropie croisée[7] catégorique pour la branche segmentation et d'une entropie croisée binaire[8] pour la branche masque. La fonction de perte ("*loss*") générale est simplement la somme des deux pertes. Sur une carte graphique Nvidia Geforce GTX 1060, l'entraînement a duré 1000 époques, avec une moyenne de 155 secondes par époque, pour une durée totale de approximative de 45 heures.

De manière concrète, une fois les maxima trouvés par le réseau Tournoi, une sous-image carrée (la taille de ce carré est un hyper-paramètre de l'algorithme complet) autour de chaque maxima est récupérée et envoyée au réseau TRS. Celui-ci indique alors la translation de la carte dans la sous-image, sa rotation et son échelle. Les composantes de la translation sont additionnées à la position de la fenêtre dans l'image originale totale pour retrouver la position absolue de la carte. Pour cette section, nous allons étudier les performances de l'algorithme TRS sur sa fenêtre. En d'autres termes, pour cette section, les performances du réseau TRS ont être analysées indépendamment de celle du réseau précédent.

Comme le montre la figure 40, le réseau TRS est particulièrement précis, même dans les

[7] L'entropie croisée entre deux points y et $\hat{y}$ est définie comme: $L(y, \hat{y}) = \sum_c \hat{y}_c \log(y_c)$

[8] La distinction entre l'entropie croisée catégorique ou binaire est simplement que l'entropie croisée binaire est un cas spécial de l'entropie catégorique où le nombre de classes c est égal à 2.

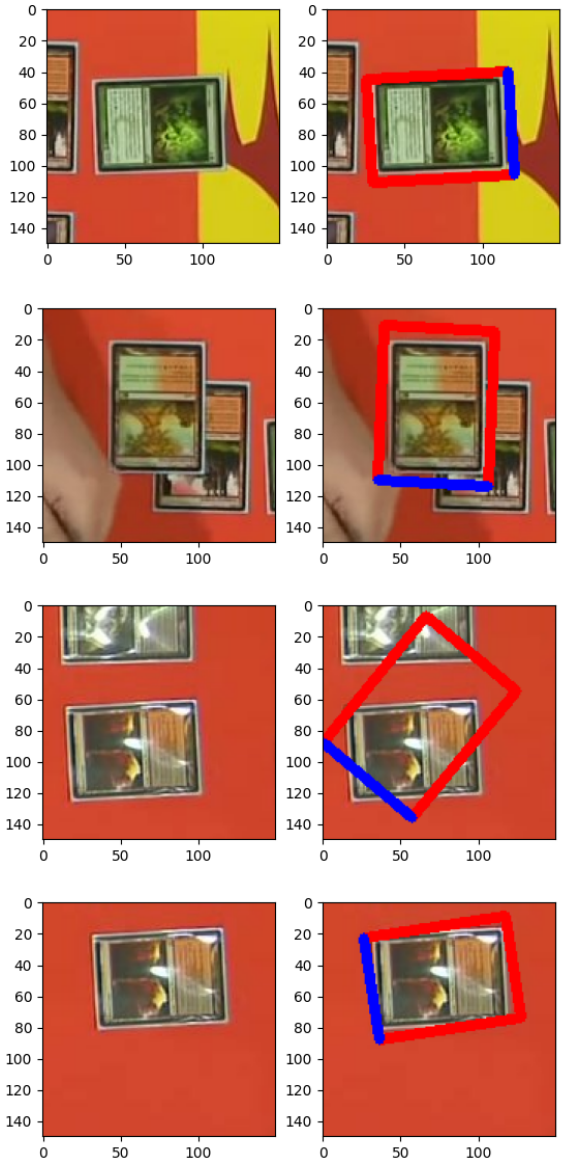

Figure 40: Résultats du réseau TRS sur des images réelles de tournoi. Deux exemples du haut: localisation précise, avec et sans présence de superposition avec une autre carte. Troisième exemple: Erreur engendrée par la présence d'une autre carte dans l'image sans que celle-ci se superpose avec la carte visée. Quatrième exemple: Erreur engendrée par le reflet de la pochette de protection autour de la carte.

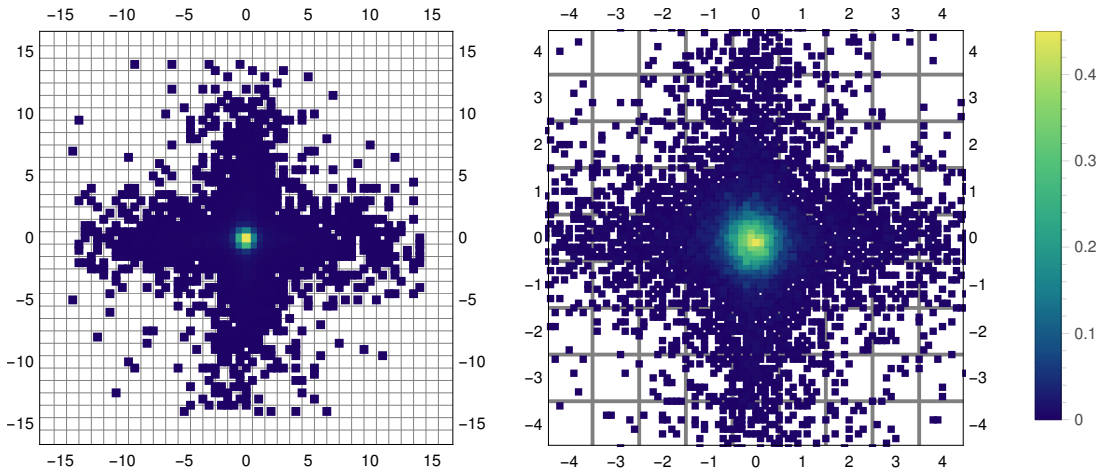

Figure 41: Histogramme de l'erreur de positionnement engendrée par le réseau TRS (sous-image 150 × 150). Les axes X et Y sont en pixels.

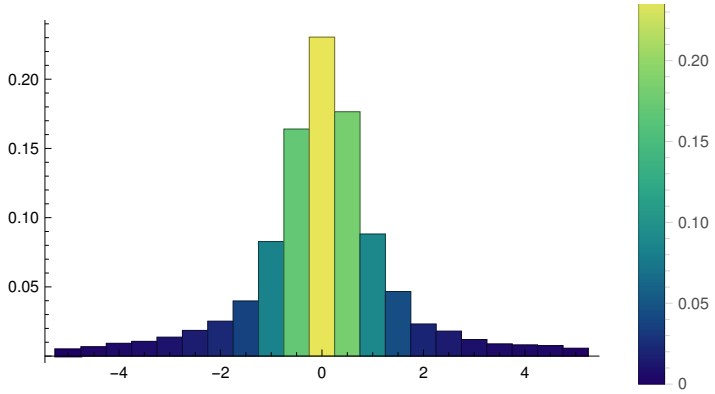

Figure 42: Histogramme de l'erreur moyenne de rotation engendrée par le réseau TRS. Axe des X: erreur de rotation en degrés par rapport à la bonne rotation.

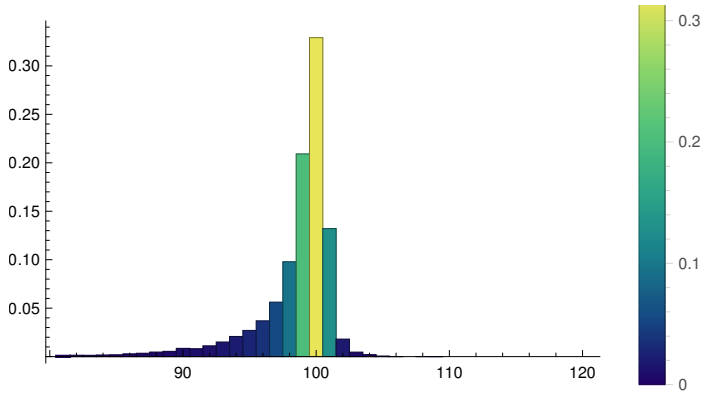

Figure 43: Histogramme de l'erreur d'échelle engendrée par le réseau TRS sur une carte pleine échelle (échelle = 1.0). Axe des X: rapport (en pourcentage) entre l'échelle estimée et l'échelle réelle. Une valeur de 100 indique une carte avec une échelle identique à l'échelle réelle.

cas réels. De fait, le réseau TRS réussit à précisément orienter une boîte de contrainte autour des cartes dans les images données (l'arête bleue indique le haut de la carte). Sur un ensemble de près de 17600 images, le réseau TRS atteint une grande précision tant sur la position que l'orientation ou l'échelle. La figure 41 illustre l'erreur de positionnement pour le réseau. Sur cet ensemble de données, près de 64% des points ont été positionnés avec ±1 pixel de précision sur une image de taille 150 × 150. Cette statistique augmente à 84% à l'intérieur d'une fenêtre de ±3 pixels. En ce qui concerne l'orientation de la carte (voir figure 42), le réseau identifie correctement l'angle de 63% des points de données à l'intérieur de ±1 degré, et de 84% à l'intérieur de ±3 degrés. La figure 43 montre que le réseau peut aussi identifier l'échelle de 74% des points de données à l'intérieur de ±2% de l'échelle réelle. On y voit que la majorité des points se voient attribués la bonne échelle mais que le réseau a tendance à sous-estimer l'échelle.

Interaction avec le réseau Tournoi

Comme mentionné ci-haut, le réseau Tournoi possède une limitation de ne pas fournir une grande précision sur le positionnement des cartes car il vise la détection et non la localisation . À lui seul, ce réseau est peu utile mais lorsqu'il est combiné avec le réseau TRS, l'erreur disparaît presque entièrement. De fait, le réseau TRS calcule la distance de la carte par rapport au centre de sa sous-image. Or le centre de la sous-image est nécessairement le pixel centre trouvé par le réseau Tournoi. Ainsi, en mesurant la translation de la carte avec le réseau TRS, nous calculons essentiellement la "correction" nécessaire à la prédiction du réseau Tournoi. De fait, en additionnant la position Tournoi avec la translation TRS, on obtient un positionnement précis malgré l'imprécision du réseau Tournoi.

Résultats Finaux

Comme mentionné à la section *Approche*, le pipeline complet utilise les deux réseaux en tandem pour générer des résultats. La figure 44 montre des exemples du pipeline complet sur des images de tournoi réelles. Comme on peut le voir, La majorité des cartes sont identifiées et leur pose estimée avec une bonne précision. On peut aussi voir que les seuls échecs du pipeline découlent de faux positifs et faux négatifs du réseau Tournoi. De fait, puisque l'algorithme TRS n'est appliqué que si et seulement si un centre est détecté, si le réseau Tournoi ne réussit pas à identifier un centre, alors le réseau TRS ne pourra jamais en déterminer la pose. De plus, comme on peut le voir, dans les cas de faux positifs du réseau Tournoi, le réseau TRS se trouve contraint de trouver un masque à une image vide. Dû à l'utilisation des masques dans le calcul des composantes TRS, même une image vide va se voir attribuer des composantes TRS. Or puisqu'il n'y a pas de carte, ces composantes sont forcément erronéés.

Figure 44: Résultats du pipeline Tournoi + TRS sur des images réelles de tournoi.

Résultats Comparatifs

À titre de comparaison, nous avons exécuté l'algorithme de détection d'image YoloV3 [8] sur les images de tournoi. Comme le montre la figure 45, le réseau est capable d'identifier les cartes dans l'image mais, contrairement à notre pipeline, le réseau est incapable de fournir l'angle de la carte. De fait, peu importe l'angle de la carte, les boîtes de contraintes de YoloV3 vont toujours être alignées avec les axes de l'image car celui-ci n'estime pas l'orientation. L'algorithme YoloV3 cherche simplement à délimiter les limites de chaque objet plutôt que d'inférer sa pose. Par exemple, la figure 46 montre que les boîtes de contraintes restent invariables à la rotation alors que notre algorithme peut s'adapter à la rotation de l'image. Finalement, on remarque que Yolo a tendance à identifier le contenu des cartes (personnages, animaux, etc..) plutôt que les cartes elles-même.

Méthodes Classiques

Le but du présent mémoire n'est pas d'explorer les méthodes classiques mais plutôt l'apprentissage profond. En pratique, les méthodes classiques présentent certaines faiblesses comme la sensibilité aux paramètres ou le manque de robustesse. La figure 47 illustre un algorithme simple de composantes connexes sur une image binarisée. Les résultats pour plusieurs seuils montrent que la segmentation va, soit, ne pas détecter de cartes, soit, sur-segmenter ou sous-segmenter les cartes dans l'image. Il est évident qu'avec d'importants efforts il serait possible de réduire cette sensibilité, or, les résultats seront tout de même fortement dépendants de l'image analysée. En outre, dans le contexte du problème visé, on observe que la transformée de Hough s'applique mal car les bordures de carte forment des lignes trop courtes. Ainsi, les méthodes classiques, contrairement à l'approche par apprentissage profond, s'appliquent mal aux cas plus généraux.

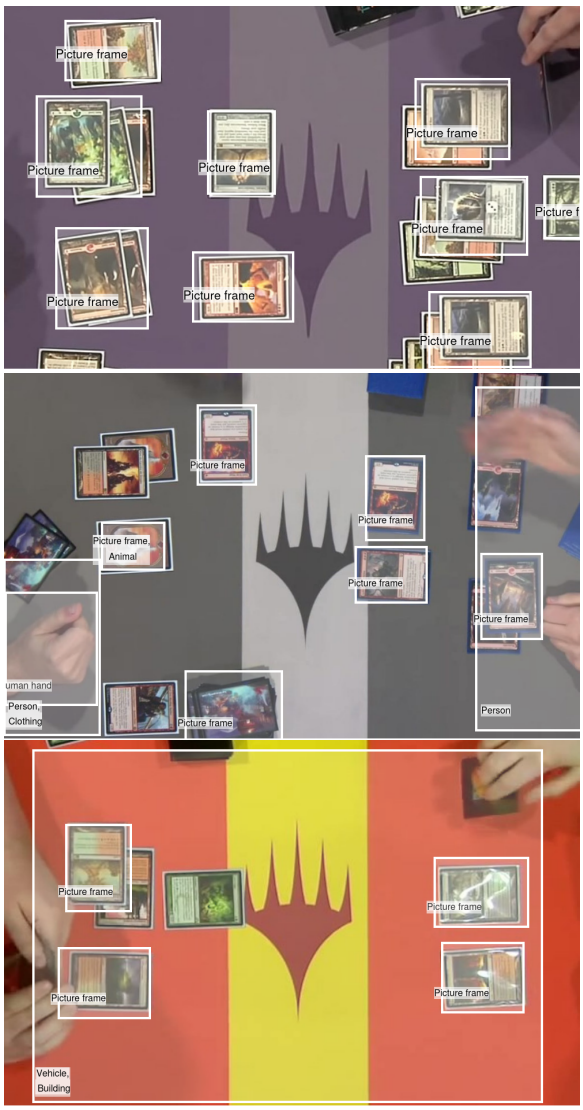

Figure 45: Résultat d'une application du réseau YoloV3 sur des images réelles de tournoi. [8]

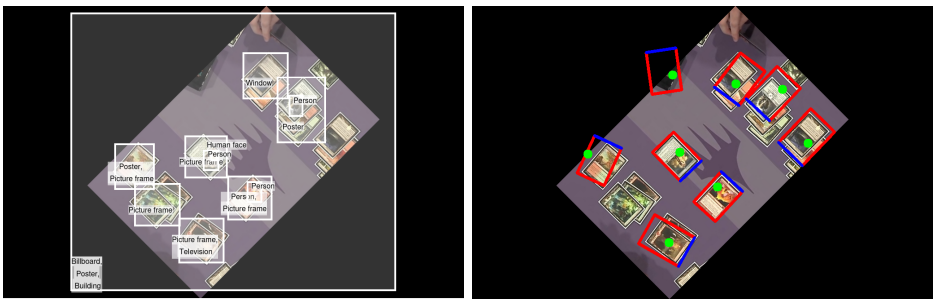

Figure 46: Impact de la rotation sur YoloV3 (gauche) et notre algorithme (droite).

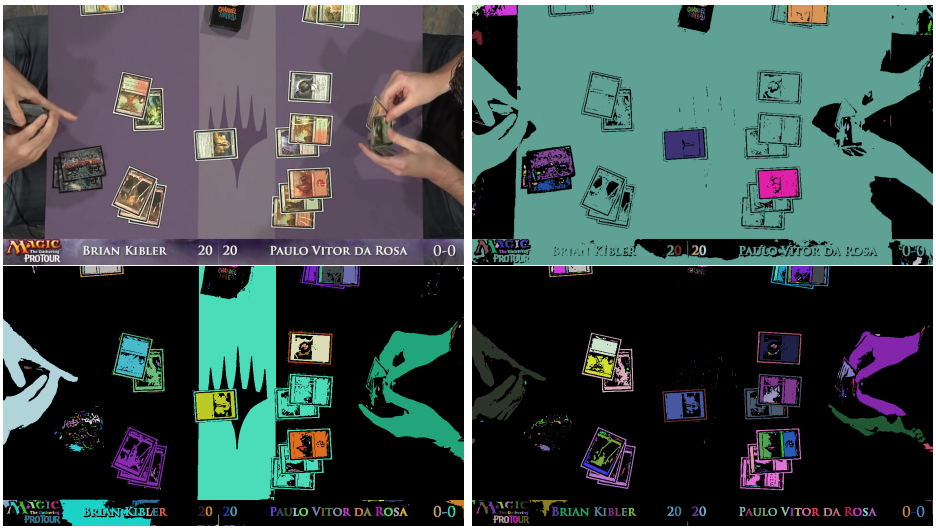

Figure 47: Exemples d'applications d'algorithmes de segmentation classiques (composantes connexes après binarisation) sur des images de tournoi. Haut gauche: Image réelle. Haut droit, bas gauche, bas droit: seuils de 0.3, 0.4 et 0.5 respectivement. Une couleur distincte est attribuée à chaque composante connexe.

CONCLUSION

Le but de ce projet était de proposer un algorithme capable de détecter et d'estimer la position et l'orientation de cartes *Magic* dans une image de tournoi donnée. Pour accomplir cet objectif, nous avons, non seulement, développé un pipeline composé de deux réseaux de neurones, mais, nous avons aussi développé une méthode de génération de données synthétiques capable de bien représenter les données réelles dans les cas non-obstrués. Le pipeline ayant été divisé en deux, il nous a été possible de séparer la tâche en deux sous-tâches: d'une part, la détection et la localisation approximative, et d'autre part le positionnement précis.

Le réseau Tournoi, quoiqu'imparfait, atteint son objectif de manière satisfaisante. Le réseau sait détecter et positionner les cartes *Magic* au sein d'une image. Le positionnement est imprécis mais adéquat pour la tâche puisque l'emphase est sur la détection.

Le réseau TRS est la réussite principale de ce projet. Le réseau à convolution est capable d'identifier une carte *Magic* au sein d'une image et d'en donner un masque et une partition si précise qu'il est possible d'utiliser ces informations pour calculer le positionnement de la carte avec une précision de moins d'un pixel, la rotation jusqu'à un degré et l'échelle jusqu'à 2%. Le réseau est suffisamment puissant pour être capable d'absorber, voir même compenser pour l'imprécision du réseau Tournoi. Malgré l'utilisation de données synthétiques, les résultats démontrent une capacité à généraliser sur les cas réels tout en évitant les dangers du surapprentissage.

De manière générale, le domaine de l'apprentissage machine est en explosion de popularité et il est commun pour les approches par réseaux de neurones d'être prisées simplement dus à leur popularité. Ainsi, il est important de se questionner sur la pertinence de l'approche par apprentissage profond dans le cadre de ce projet. En ce qui

concerne ce travail, il est clair que l'apprentissage profond apporte un avantage marqué qui n'aurait pas été possible avec les algorithmes classiques, tout spécialement par la robustesse à la superposition et à la précision des estimations. De plus, une approche par masque ou classification de pixels nous a permis d'atteindre une performance très précise pour la tâche en question.

Malheureusement, malgré les succès de ce projet, celui-ce montre tout de même quelques lacunes. Le réseau Tournoi, quoique suffisant pour notre objectif, laisse place à de nombreux faux-positifs et faux-négatifs. De plus, notre réseau ne détecte que les cartes non-obstruées. Celui-ci est robuste à la superposition mais si une des cartes se retrouve cachée par une autre carte, ou par une main par exemple, alors l'algorithme ne peut performer. L'utilisation de données réelles pourrait aider le réseau Tournoi à généraliser aux cas les plus complexes. De plus, le système est sensible aux ratios d'aspect des images d'entrée. Développer un système pleinement convolutif à taille variable pourrait davantage aider les réseaux à généraliser.

En outre, ce projet laisse place à d'autres améliorations telles l'ajout d'une branche d'identification capable d'identifier la carte une fois celle-ci détectée et positionnée. Aussi, il serait possible de définir un autre réseau capable d'augmenter les capacités du réseau TRS afin de lui permettre d'identifier les cartes obstruées. De plus, la question d'utiliser un réseau récurrent pour profiter de la cohérence temporelle d'un vidéo reste ouverte. De fait, un réseau capable de conserver une mémoire de la séquence du positionnement des cartes pourrait aider celui-ci à mieux positionner les cartes dans un contexte de mouvements.

En concluant, nous considérons avoir atteint l'objectif du projet décrit dans le présent document. Nous avons prouvé la capacité de notre modèle à généraliser les cas réalistes à partir de données synthétiques tout en développant un pipeline complexe capable de rapidement répondre à la tâche.

RÉFÉRENCES

[1] Thanh-Toan Do, Ming Cai, Trung Pham, et Ian Reid. Deep-6dpose: Recovering 6d object pose from a single rgb image. *arXiv preprint arXiv:1802.10367*, 2018.

[2] Alexander Grabner, Peter M Roth, et Vincent Lepetit. 3d pose estimation and 3d model retrieval for objects in the wild. Dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3022–3031, 2018.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et Jian Sun. Deep residual learning for image recognition. Dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Diederik P Kingma et Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Jake Levinson, Carlos Esteves, Kefan Chen, Noah Snavely, Angjoo Kanazawa, Afshin Rostamizadeh, et Ameesh Makadia. An analysis of svd for deep rotation estimation. *arXiv preprint arXiv:2006.14616*, 2020.

[6] Zhigang Li, Gu Wang, et Xiangyang Ji. Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation. Dans *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7678–7687, 2019.

[7] Wizards of the Coast. Pro tour dark ascension: Top 8 finals.

[8] Joseph Redmon et Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, et Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[10] Jesse Richter-Klug et Udo Frese. Handling object symmetries in cnn-based pose estimation. *arXiv preprint arXiv:2011.13209*, 2020.

[11] Olaf Ronneberger, Philipp Fischer, et Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. Dans *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[12] Hao Su, Charles R Qi, Yangyan Li, et Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. Dans *Proceedings of the IEEE International Conference on Computer Vision*, pages 2686–2694, 2015.

[13] Yongzhi Su, Jason Rambach, Alain Pagani, et Didier Stricker. Synpo-net—accurate and fast cnn-based 6dof object pose estimation using synthetic training. *Sensors*, 21(1):300, 2021.

[14] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, et Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[15] Yaming Wang, Xiao Tan, Yi Yang, Xiao Liu, Errui Ding, Feng Zhou, et Larry S Davis. 3d pose estimation for fine-grained object categories. Dans *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[16] Wikipedia contributors. Canny edge detector — wikipedia, 2021. [En ligne; accédé le 1er Mai 2021].

[17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, et Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[18] Xue Yang, Jirui Yang, Junchi Yan, Yue Zhang, Tengfei Zhang, Zhi Guo, Xian Sun, et Kun Fu. Scrdet: Towards more robust detection for small, cluttered and rotated objects. Dans *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8232–8241, 2019.

[19] Yixing Zhu, Chixiang Ma, et Jun Du. Rotated cascade r-cnn: A shape robust detector with coordinate regression. *Pattern recognition*, 96:106964, 2019.