République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Seddik Benyahia Jijel
Faculté des Sciences Exactes et Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Intelligence Artificielle

Thème

Un Modèle CNN guidé SIFT pour la Reconnaissance des Classes d'Objet

Réalisé par

MAOUDJ Salah Eddine
REMMOUCHE Brahim

Encadré par

Dr. TAFFAR Mokhtar

Promotion 2021

Remerciements

Tout d'abord nous remercions ***ALLAH****, le tout puissant qui a illuminé notre chemin et qui s'est armé de courage et de patience pour accomplir ce travail.*

Nous remercions notre encadreur ***Dr. Taffar Mokhtar****, qui a été très disponible tout au long de la réalisation.*

Nous remercions également toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail, sans oublier les membres du jury pour avoir accepté de la juger.

Résumé

Dans ce projet, nous avons proposé un modèle de réseau de neurones à convolution (CNN) guidé par les points d'intérêt SIFT (scale-invariant feature transform) pour la reconnaissance de classes d'objets. Notre modèle admet deux entrées, la première entrée reçoit l'image originale et la seconde est alimentée par une partie de l'image originale qui contient la plus grande densité de points SIFT appartenant à l'objet. L'idée d'utiliser deux entrées dont l'une est une partie de l'autre est de permettre au modèle d'apprendre des informations globales à partir de l'image entière et des informations locales à partir de l'image partielle, cela aide à reconnaître des objets très similaires dans leur forme globale ainsi que les objets occultés. Dans ce dernier cas, les caractéristiques SIFT servent à guider le modèle à reconnaître les parties partielles non occultées de l'objet. Un classifieur SVM a été utile pour identifier et extraire la région de l'image qui contient la plus grande densité de points SIFT appartenant à l'objet. Les caractéristiques SIFT de l'objet sont sélectionnées selon leurs capacités discriminatives de l'apparence de l'objet à classifier. Les expérimentations ont montré les résultats remarquables de notre modèle par rapport à un simple CNN, avec une nette amélioration de la performance de reconnaissance qui atteint une précision de 92% sur le jeu de données kaggle cats-vs-dogs , En revanche le simple CNN n'a obtenu que 88%. Notre modèle a également prouvé son efficacité à reconnaître les objets occultés avec un taux de 87% contre un faible taux de reconnaissance du modèle CNN simple.

Mots clés : *Caractéristiques SIFT, Classifieurs, CNN, Détecteurs locaux, Détection d'objets occultés, Reconnaissance des classes d'objets.*

Abstract

In this project, we have proposed a convolutional neural network (CNN) model guided by SIFT (scale-invariant feature transform) interest points for object class recognition. Our model admits two inputs, the first input receives the original image and the second input is fed by a part of the original image that contains the highest density of SIFT points belonging to the object. The idea of using two inputs where one is a part of the other is to allow the model to learn global information from the whole image and local information from the partial image, this helps to recognize objects that are very similar in their global shape as well as occluded objects. In the latter case, the SIFT features are used to guide the model to recognize the unoccluded partial parts of the object. An SVM classifier was useful to identify and extract the region of the image that contains the highest density of SIFT points belonging to the object. The SIFT features of the object are selected according to their discriminative abilities of the appearance of the object to be classified. Experiments have shown the remarkable results of our model compared to a simple CNN, with a clear improvement of the recognition performance which reaches an accuracy of 92% on the kaggle cats-vs-dogs dataset, while the simple CNN obtained only 88%. Our model also proved to be effective in recognizing occluded objects with a rate of 87% against a low recognition rate of the simple CNN model.

Keywords : *Classifiers, CNN, Local detectors, Object classes recognition, Occulted objects detection, SIFT characteristics.*

Table des matières

Table des Matières i

Table des figures iv

Liste des acronymes vii

Introduction Générale 1

1 Détecteurs Locaux et Classifieurs 4
- 1.1 Introduction . . . 4
- 1.2 Objectif de l'Apprentissage . . . 4
- 1.3 Approches de l'Apprentissage . . . 5
  - 1.3.1 Apprentissage supervisé . . . 5
  - 1.3.2 Apprentissage non supervisé . . . 5
- 1.4 Données d'apprentissage . . . 5
  - 1.4.1 Données d'entraînement . . . 5
  - 1.4.2 Données de test . . . 5
  - 1.4.3 Données de validation . . . 6
- 1.5 Régression et Classification . . . 6
- 1.6 Détecteurs de Caractéristiques Locales . . . 7
  - 1.6.1 Détecteur de Harris . . . 7
  - 1.6.2 Détecteur SIFT . . . 8
  - 1.6.3 Détecteur de SURF . . . 10
  - 1.6.4 Détecteur de HOG . . . 11
- 1.7 Classifieurs . . . 12
  - 1.7.1 K-PPV . . . 12
  - 1.7.2 K-Means . . . 12
  - 1.7.3 SVM . . . 13
  - 1.7.4 Réseaux de Neurones . . . 14
- 1.8 Conclusion . . . 19

2 Apprentissage Profond 20
- 2.1 Introduction . . . 20
- 2.2 Position de l'Apprentissage Profond . . . 20
- 2.3 Notion de Profondeur . . . 22
- 2.4 Architectures d'Apprentissage profond . . . 22
  - 2.4.1 Réseau neuronal convolutif . . . 22
  - 2.4.2 Réseaux neuronal récurrent . . . 23

2.5 Fonctionnement de l'apprentissage profond . . . 23
2.5.1 Fonction de Loss . . . 24
2.5.2 Optimiseurs . . . 25
2.6 Sur-apprentissage . . . 27
2.7 Méthodes pour éviter le Surapprentissage . . . 27
2.7.1 Augmentation des données . . . 27
2.7.2 Arrêt précoce (Early stopping) . . . 28
2.7.3 Méthodes de régularisation . . . 29
2.8 Conclusion . . . 29

3 Réseau de Neurones Convolutifs 30
3.1 Introduction . . . 30
3.2 Principes des réseaux neuronaux convolutifs . . . 30
3.2.1 Couches de convolution . . . 31
3.2.2 Couches de pooling . . . 32
3.2.3 Couches entièrement connectées . . . 32
3.3 Architectures des réseaux neuronaux convolutifs . . . 32
3.3.1 LeNet-5 . . . 32
3.3.2 AlexNet . . . 33
3.3.3 VGGNet . . . 33
3.3.4 ResNet . . . 34
3.3.5 EfficientNet . . . 34
3.3.6 Pseudo labels (Noisy Student) . . . 35
3.3.7 Meta Pseudo Labels . . . 36
3.4 Jeux de données . . . 36
3.4.1 ImageNet . . . 36
3.4.2 Cifar-10 . . . 37
3.4.3 Cifar-100 . . . 37
3.4.4 JFT-300M . . . 38
3.4.5 Kaggle Cats-vs-Dogs . . . 38
3.5 Visualiser ce qu'un CNN Apprend . . . 38
3.5.1 Visualisation des activations intermédiaires . . . 39
3.5.2 Visualisation des Filtres . . . 39
3.5.3 Visualisation de heatmap pour la classe d'activation . . . 39
3.6 Conclusion . . . 40

4 Modèle CNN guidé SIFT pour la reconnaissance des classes d'objet 41
4.1 Méthode proposée . . . 41
4.2 Description de la fonction du Modèle proposé . . . 43
4.2.1 Prétraitement des données . . . 43
4.2.2 Modèle SVM pour la sélection des points objets . . . 43
4.2.3 Modèle CNN proposé . . . 44
4.3 Modèle proposé vs CNN simple . . . 47
4.4 le jeu de données . . . 48
4.5 Reconnaissance des Objets Occultés . . . 48
4.6 Conclusion . . . 49

5 Expérimentation et Résultats **50**
5.1 Représentation des outils 50
5.1.1 Outils logiciels 50
5.1.2 Matériel informatique 52
5.2 Interfaces Graphiques 52
5.3 Implementation 59
5.3.1 Code source de pré-traitement des images 59
5.3.2 Code source de modèle CNN simple 62
5.3.3 Code source de modèle CNN proposé 63
5.4 Expérimentation et résultats 64
5.4.1 Évaluation de performance de modèle SVM 64
5.4.2 Comparaison du Modèle proposé avec un simple CNN 67
5.5 Interpretation 70
5.6 Conclusion 70

Conclusion Générale **71**

Bibliographie **viii**

Annexe **xiii**

Table des figures

1.1 Illustration de classification binaire et de classification multi-classe . . 6
1.2 Régression linéaire simple 7
1.3 Convoluer l'image à l'échelle multi-espace avec la différence de gaussienne [1] 8
1.4 Détecter les points extrêmes en comparant chaque point à ses 26 voisins [1] 9
1.5 Descripteur $4 * 4$ calculé à partir d'un ensemble d'échantillons $16 * 16$ [1] 10
1.6 Exemple d'algorithme de k-ppv dans le cas où $k = 3$ et $k = 5$ 12
1.7 Exemple d'algorithme de k-means avec $k = 3$ (3 clusters) 13
1.8 Classification binaire par SVM 13
1.9 Projection de données dans un espace plus grand 13
1.10 Classification multi-classe par SVM 14
1.11 Exemple typique d'un perceptron 16
1.12 Un perceptron multicouche avec deux couches cachées 16

2.1 La relation entre l'intelligence artificielle, l'apprentissage automatique et l'apprentissage profond [2] 20
2.2 Comparaison entre l'ancien paradigme de l'IA et le paradigme de l'apprentissage automatique 21
2.3 Comparaison des performances des algorithmes d'apprentissage automatique et d'apprentissage profond par rapport à la quantité de données 21
2.4 Comparaison des processus d'extraction de caractéristiques dans les algorithmes d'apprentissage automatique et d'apprentissage profond . 22
2.5 Réseau de neurones profonds pour la classification des chiffres avec 4 couches cachées 22
2.6 Exemple typique d'un réseau de neurones récurrent 23
2.7 Un réseau de neurones paramétré par ses poids [2] 24
2.8 Une fonction de loss pour mesurer la qualité de la sortie du réseau [2] 24
2.9 Utilisez le score de loss comme signal de retour pour ajuster les poids par l'optimiseur [2] 25
2.10 Génération d'images de chats par augmentation aléatoire des données 28
2.11 Comment fonctionne l'arrêt précoce 28
2.12 Comment fonctionne le dropout 29

3.1 Architecture typique d'un réseau de neurones convolutifs 31
3.2 Illustration de l'opération de convolution 31
3.3 Illustration des opérations average pooling et max pooling 32

3.4 Architecture détaillée de LeNet-5 [3] . . . 32
3.5 Architecture détaillée de AlexNet . . . 33
3.6 Architecture détaillée de VGGNet [4] . . . 33
3.7 Blocs Résiduels [5] . . . 34
3.8 Architecture de ResNet [5] . . . 34
3.9 (a) Exemple d'un réseau, (b) Mise à l'échelle de la largeur, (b) Mise à l'échelle de la profondeur, (d) Mise à l'échelle de la résolution, (e) Mise à l'échelle composée [6] . . . 35
3.10 Illustration de l'apprentissage du modèle noisy student [7] . . . 36
3.11 La différence entre les pseudo-labels et les méta-pseudo-labels [8] . . . 36
3.12 Quelques images appartiennent au jeu de données ImageNet . . . 37
3.13 Quelques images appartiennent au jeu de données Cifar-10 . . . 37
3.14 Quelques images appartiennent au jeu de données Kaggle Cats-vs-Dogs 38
3.15 Les activations de certaines couches pour une image de chat . . . 39
3.16 Visualisation du quatrième bloc dans la première convolution pour le modèle VGGNet pré-trainé du jeu de données ImageNet . . . 39
3.17 Visualisation de heatmap . . . 40

4.1 Les étapes de la méthode proposée . . . 42
4.2 Description du fonctionnement de la méthode proposée . . . 42
4.3 Prétraitement des données . . . 43
4.4 Détection des points d'intérêt SIFT . . . 43
4.5 Classifier les points SIFT appartient ou non à l'objet . . . 44
4.6 Sélection des points SIFT appartient à l'objet . . . 44
4.7 Sélection de la région image qui a la plus forte densité de points SIFT appartenant à l'objet . . . 44
4.8 L'architecture du modèle proposé . . . 46
4.9 L'architecture du simple CNN . . . 47
4.10 Quelques images du jeu de données kaggle cats-vs-dogs . . . 48
4.11 Quelques objets partiellement occultés par l'arrière-plan de l'image . 49

5.1 Fenêtre principale . . . 53
5.2 Fenêtre de description SIFT . . . 53
5.3 Fenêtre de classification des points SIFT . . . 54
5.4 Fenêtre d'apprentissage de modèle CNN simple . . . 55
5.5 Fenêtre de prédiction et d'évaluation du modèle SVM . . . 56
5.6 Fenêtre de classification . . . 56
5.7 Fenêtre d'évaluation de la prédiction des images par les deux modèles CNN . . . 57
5.8 Fenêtre d'évaluation des performances des deux modèles CNN . . . 58
5.9 Fenêtre de traitement d'image . . . 59
5.10 Le code source de la méthode *cut_image* . . . 59
5.11 Le code source de la méthode *image_by_point* . . . 60
5.12 Le code source de la méthode *calc_center* . . . 60
5.13 Le code source de la méthode *select_points* . . . 61
5.14 Code source du modèle CNN simple . . . 62
5.15 Code source du modèle CNN proposé : Réseau 1 . . . 63
5.16 Code source du modèle CNN proposé : Réseau 2 . . . 64

5.17 Code source du modèle CNN proposé : Concaténation des réseaux . . 64
5.18 Quelques images avec une bonne classification 66
5.19 Quelques images avec une mauvaise classification 67
5.20 L'évolution d'accuracy et de loss dans l'entraînement et la validation des deux modèles pour les 30 epoches 67
5.21 Les valeurs de loss et d'accuracy pour l'entraînement et la validation du modèle proposé dans les dernières 5 epoches 68
5.22 Les valeurs de loss et d'accuracy pour l'entraînement et la validation du modèle simple dans les dernières 5 epoches 68
5.23 Les valeurs de loss et accuracy pour le TestSet du modèle simple . . . 68
5.24 Les valeurs de loss et accuracy pour le TestSet du modèle proposé . . 68
5.25 Quelques images où notre modèle reconnaît correctement les objets et où le simple CNN échou 69
5.26 Les valeurs de loss et accuracy pour le TestSet du modèle simple dans le cas des objets occultés 69
5.27 Les valeurs de loss et accuracy pour le TestSet du modèle proposé dans le cas des objets occultés 69
5.28 Quelques images d'objets occultés où notre modèle les reconnaît correctement et le simple CNN échou 70

Listes des acronymes

AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
ANN	Artificial Neural Network
CAM	Class Activation Map
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Units
HOG	Histogram of Oriented Gradients
IA	Intelligence Artificial
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LSTM	Long-Short Term Memory
K-PPV	k Plus Proches Voisins
MLP	MultiLayer Perceptron
MNIST	Modified National Institute of Standards and Technology
ReLU	Rectified Linear Unit
ResNet	Residual Network
RMSProp	Root Mean Square Propagation
RN	Réseaux de Neurones
RVB	Rouge Vert Bleu
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
SVM	Support Vector Machine
TPU	Tensors Processing Unit
VGGNet	Visual Geometry Group Network

Introduction Générale

La vision par ordinateur est un domaine scientifique interdisciplinaire qui traite de manière dont les ordinateurs peuvent acquérir une compréhension de haut niveau à partir d'images. L'un de ses sous-domaines est la reconnaissance d'objets qui fait référence à la capacité d'identifier des objets en vue sur la base d'une entrée visuelle. Une signature importante des méthodes de la reconnaissance visuelle d'objets est leur invariance aux apparences des objets. Cette propriété nécessaire aux algorithmes de reconnaissance des objets témoigne de la capacité d'identifier des objets à travers les changements dans des environnements complexes, tels que les changements d'illumination, de pose de l'objet ou d'angle de vue, d'occultation, etc., dans lesquels les objets visuels sont détectés et reconnus.

Plusieurs méthodes de reconnaissance des objets ont été proposées dans ce domaine au cours des deux dernières décennies. Ces méthodes peuvent être classées en deux grandes catégories d'approches, l'approche classique basée sur les caractéristiques visuelles locales et la nouvelle approche basée sur l'apprentissage profond. Les méthodes basées sur les caractéristiques locales comme SIFT et SURF extraient d'abord les points d'intérêt d'un ensemble d'images de référence pour construire le modèle de l'objet ou de la classe d'objet à traiter en utilisant un classifieur. Ces points sont représentés par un descripteur invariant à l'échelle, l'orientation, les changements d'illumination, et partiellement invariant à la distorsion affine. Ensuite, un objet est reconnu dans une nouvelle image en comparant individuellement chaque caractéristique de la nouvelle image aux caractéristiques du modèle construit . En faisant une mise en correspondance entre les caractéristiques de l'image requête et celles du modèle, le but est de trouver un ensemble de caractéristiques image qui match ou apparient en se basant sur une mesure de distance entre les descripteurs de caractéristiques. Cette approche peut être efficace dans le cas de l'identification d'objets dans des images complexes ou encombrées ou encore dans le cas où les objets sont partiellement occultés, mais elle a beaucoup d'inconvénients, l'un d'eux est l'extraction manuelle des caractéristiques ou features. En outre, les approches basées des descripteurs locaux présentent de mauvaises performances dans les cas de reconnaissance multi-objets. Bien que ces approches ont dominé le domaine de la reconnaissance d'objets, l'apparition des réseaux neurones convolutifs, vers les années 2012, a complètement bouleversé la performance des algorithmes dédiés à la vision par ordinateur en particulier et l'intelligence artificielle en générale.

Bien que le premier réseau de neurone convolutif réussi soit apparu dans les années 1990 par Yann Lecun, ce réseau n'a pas pris beaucoup de notoriété à cette époque en

raison de l'absence de puissance de calcul dans les ordinateurs. La situation reste inchangée jusqu'à l'apparition des unités de traitement graphique (GPU) pour le calcul parallèle. Cela a donné une chance à ces modèles d'aller plus loin dans leur architecture. En 2012, Alex Krizhevsky a proposé AlexNet, le premier réseau de neurones convolutifs gagné le défi de reconnaissance visuelle à grande échelle ImageNet (ILSVRC). Depuis, ces modèles de réseaux profonds convolutionnels dominent la plupart des approches et méthodes de la vision par ordinateur.

Problématique

En raison du grand succès des réseaux de neurones convolutifs dans la reconnaissance d'objets et de leur avantage d'extraire automatiquement les caractéristiques à travers des données images brutes sous forme de représentation interne au réseau, l'utilisation des méthodes classiques basées sur les détecteurs locaux est devenue presque inefficace dans les domaines de la détection, reconnaissance, suivi, etc. Mais comme toute approche, les algorithmes dédiés à l'apprentissage profond ont leurs propres inconvénients. Parmi les inconvénients on peut citer celui qui apparaît lors de la détection ou la reconnaissance des objets occultés où les objets dans les images sont partiellement cachés ou occultés par d'autres objets qui font partie du fond d'écran ou encore des objets qui ne font pas partie de l'étude, ce qui relève souvent des cas très probable dans lesquels les objets apparaissent dans la réalité.

Un autre inconvénient est lorsque les classes d'objets sont trop similaires dans leur structure globale comme la classification des feuilles d'arbres et la classification des animaux où les objets ont approximativement la même longueur, largeur et forme. Ces cas peuvent être traités et résolus en proposant des architectures de réseaux plus profondes et plus convolutives avec un réglage très minutieux des hyper-paramètres des réseaux. Tout cela nous conduit à un autre problème plus technique lié à la quantité énorme de paramètres que peut contenir un réseau, ce qui restreint un modèle de réseau profond à être utilisé dans certains dispositifs spécialisés à la fois lors de l'entraînement, les tests et encore lors de son intégration réelle.

Motivation

Partant du constat, que d'une part, un modèle de reconnaissance classique utilisant à la base un classifieur, tel de type SVM ou K-ppv, et un détecteur de caractéristiques visuelles locales, admet des performances mitigées dépendantes à la fois des trois composantes essentielles : de la nature d'apprentissage qu'il a subit sur un certain Dataset, du type de classifieur, ainsi que du type de descripteur local pour modéliser la classe d'objets à traiter, mais qui offre une certaine efficacité envers les objets partiellement occultés puisque les points d'intérêts peuvent être saisis et extraites sur les régions apparentes de l'objet ; et que d'autres part, un modèle de reconnaissance d'objets utilisant un réseau profond convolutionnel admet des performances indéniables qui dépassent tout prévision mais dont les capacités de classification chute drastiquement dès qu'il s'agit d'objets occultés à détecter ou à reconnaître dans des images ou vidéo mais également la résolution de ce problème nécessite un modèle plus profond et plus

convolutif ou même plus récurrent, ce qui complique inévitablement sa mise en oeuvre avec un matériel banalisé.

Idée

Pour éviter ce genre de problèmes cités précédemment, et principalement pour résoudre le cas reconnaissance des objets occultés, l'idée est concevoir un système utilisant un réseau de neurones convolutif à deux entrées, l'une pour l'image originale et l'autre pour la partie de l'image qui a la plus grande densité de points d'intérêt SIFT appartenant à l'objet. Pour décider si un point appartient ou non à l'objet, nous entraînons un classificateur SVM. Le but de l'utilisation d'un modèle de réseau de neurones convolutif à deux entrées est que, lorsque des objets de classes différentes sont similaires ou encore ont une ressemblance visuellement, ce modèle peut apprendre des informations globales à partir de l'image entière et des informations locales à partir de l'image partielle. Ces informations locales concernant l'objet occulté sont généralement situées dans les parties non occultées de l'objet et leurs détection et extraction ne nécessite pas l'apparition de l'objet en entier dans l'image. Ces informations locales sont localisées là où se trouve la plus grande densité de points d'intérêt dans l'image de l'objet. Ainsi, nous estimons que ce modèle peut fonctionner efficacement dans le cas d'objet occulté parce que les points d'intérêt servent à guider le modèle de réseau profond à reconnaître les parties de l'objet sans l'obligation que l'objet apparaisse en sa totalité dans l'image.

Cette mémoire est composée de cinq chapitres organisés comme suit :

- Le premier chapitre présente les bases de l'apprentissage automatique, les détecteurs locaux et certains classificateurs couramment utilisés dans les processus d'apprentissage et de classification.
- Le deuxième chapitre traite de l'apprentissage profond en général. Nous discutons de la différence entre l'apprentissage profond et l'apprentissage automatique, la notion de profondeur, des éléments importants pour en construire un modèle profond, de la notion d'overfitting et des moyens de l'éviter.
- Le troisième chapitre présente l'architecture d'apprentissage profond la plus couramment utilisée en vision par ordinateur, à savoir les réseaux de neurones convolutifs. Nous discutons de son fonctionnement, de certaines de ses architectures communes et de ses jeux de données.
- Le quatrième chapitre expose la méthode que nous avons proposée, dont nous avons discuté en détail les étapes, le processus d'apprentissage et le jeu de données utilisé.
- Le cinquième chapitre décrit les outils matériels et logiciels utilisés dans notre travail, les expérimentations réalisées et les résultats obtenus.

Enfin, nous terminons ce mémoire par une conclusion et des perspectives concernant notre projet.

CHAPITRE 1
Détecteurs Locaux et Classifieurs

1.1 Introduction

L'apprentissage automatique est un domaine d'étude qui s'intéresse aux algorithmes qui apprennent à partir d'exemples. Un algorithme d'apprentissage supervisé ou non supervisé apprend à attribuer une étiquette de classe à l'exemple appris à travers sa description visuelle dans l'image. Chaque exemple est représenté par un descripteur composé de ses caractéristiques élémentaires. La classification est une tâche qui nécessite l'utilisation du modèle des classes d'exemples construit par apprentissage pour reconnaître les différents objets qui se présentent. Ce processus de reconnaissance se base sur la notion de distance entre les descripteurs représentant les objets pour affecter ces derniers aux différentes classes. Les objets image sont décrits par des caractéristiques locales détectées et extraites des images/vidéo par des détecteurs locaux. Les classifieurs interviennent dans une étape en amont pour exploiter les descripteurs d'objets pour l'apprentissage puis dans une autre étape en aval pour classifier et reconnaître des objets nouveaux.

Dans ce chapitre, nous discutons de l'apprentissage artificiel, de son objectif, de ses approches de description des objets, de ses algorithmes ainsi que de la classification comme composant majeur d'un système de détection et de reconnaissance des objets. Enfin, nous verrons quelques descripteurs permettant de décrire les contenus visuels des objets dans les images.

1.2 Objectif de l'Apprentissage

Le grand intérêt de l'apprentissage automatique est d'apprendre aux ordinateurs à découvrir comment ils peuvent effectuer des tâches sans être explicitement programmés pour le faire. Il s'agit pour les ordinateurs d'apprendre à partir de données fournies afin qu'ils effectuent certaines tâches [9].

Les principales tâches d'apprentissage automatique peuvent être divisées en trois catégories : la classification, comme décider si un Émail est un spam ou non, la régression, comme estimer le prix d'une maison sur le marché, et le regroupement, comme identifier des segments de clientèle et des données démographiques pour aider à élaborer des campagnes publicitaires ciblées.

1.3 Approches de l'Apprentissage

Les approches d'apprentissage automatique sont divisées en deux grandes catégories.

1.3.1 Apprentissage supervisé

Les algorithmes d'apprentissage supervisé sont un type d'algorithmes qui fonctionnent en construisant un modèle mathématique d'un ensemble de données qui contient à la fois les entrées et les sorties souhaitées. Ce modèle sera utilisé plus tard pour prédire la sortie associée à de nouvelles entrées que le modèle n'a jamais vues auparavant [10].

1.3.2 Apprentissage non supervisé

Les algorithmes d'apprentissage non supervisé prennent un ensemble de données qui ne contient que des entrées, et trouvent une régularité ou une similarité dans ces données. Cela consiste principalement à faire des regroupements ou du clustering sur les données dont le résultat est un ou plusieurs groupes de données d'apprentissage réparties selon leurs ressemblances. Par conséquent, ils apprennent à partir de données de test qui n'ont pas été étiquetées, classées ou catégorisées, contrairement aux algorithmes d'apprentissage non supervisé qui répondent à la rétroaction. Les algorithmes d'apprentissage non supervisé identifient les points communs dans les données et réagissent en fonction de la présence ou de l'absence de ces points communs dans chaque nouvelle donnée.

Remarque : En plus de ces deux types d'apprentissage, nous pouvons rencontrer d'autres variants, à savoir l'apprentissage semi-supervisé pour lequel certains des exemples d'apprentissage sont dépourvus d'étiquettes d'apprentissage ; et l'apprentissage par renforcement lequel est guidé par l'environnement sous forme de récompenses et de pénalités données en fonction de l'erreur commise par l'algorithme.

1.4 Données d'apprentissage

Les données d'apprentissage sont généralement divisées en 3 catégories

1.4.1 Données d'entraînement

Appelé également population d'apprentissage, un ensemble d'exemples utilisés pour générer le modèle d'apprentissage. Par exemple, si la mission est la reconnaissance vocale, les données peuvent être des fichiers sonores de personnes qui parlent. Si la mission est la reconnaissance d'objets, il peut s'agir de photos.

1.4.2 Données de test

Composé des candidats que le modèle d'apprentissage n'a jamais vu auparavant. Toujours à partir du même type de données d'entraînement, les données test per-

mettent de mesurer la capacité du modèle à généraliser en utilisant une métrique spécifique (précision, matrice de confusion, etc.).

1.4.3 Données de validation

Ce type de données est une sous-population de l'ensemble de données d'apprentissage mais souvent utilisées pour confirmer la performance du modèle construit avant son intégration dans son environnement réel.

1.5 Régression et Classification

Les algorithmes de régression et de classification sont tous deux utilisés pour la prédiction en apprentissage et fonctionnent avec des ensembles de données étiquetées. La différence entre les deux est la manière dont ils sont utilisés pour différents problèmes d'apprentissage automatique.

La classification est le processus qui consiste à trouver ou à découvrir un modèle ou une fonction qui aide à séparer les données en plusieurs classes. Ainsi, les données sont classées sous différentes étiquettes en fonction de certains paramètres donnés en entrée, puis les étiquettes sont prédites pour les données.

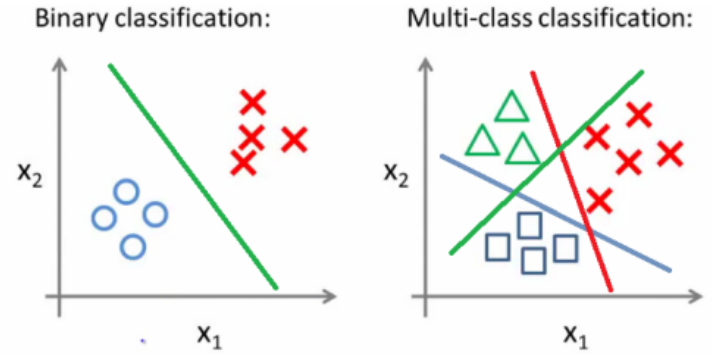


FIGURE 1.1 – Illustration de classification binaire et de classification multi-classe

La régression est le processus qui consiste à trouver un modèle ou une fonction permettant de distinguer les données en valeurs réelles continues au lieu d'utiliser des classes ou des valeurs discrètes. Elle peut également identifier le mouvement de distribution en fonction des données historiques. Comme un modèle de régression prédit une quantité, l'habileté du modèle doit être rapportée comme une erreur dans ces prédictions.

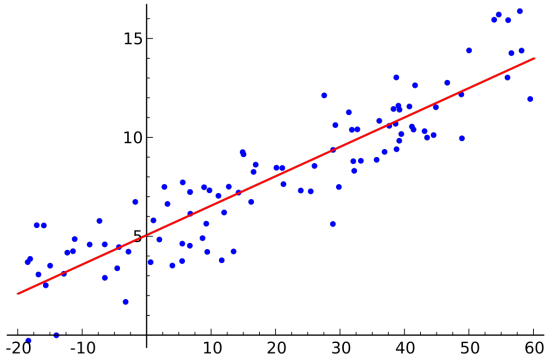


FIGURE 1.2 – Régression linéaire simple

1.6 Détecteurs de Caractéristiques Locales

L'extraction de caractéristiques locales visuelles consiste en des transformations mathématiques calculées sur les pixels d'une image. Les caractéristiques locales permettent de mieux rendre compte de certaines propriétés visuelles des objets dans l'image. Elles sont utilisées pour un traitement ultérieur dans des applications telles que la détection d'objets ou la recherche d'images basée sur le contenu, etc. Les modèles de classification basés sur l'apprentissage des descripteurs locaux dominaient les domaines de l'analyse d'image et de la vision par ordinateur. Ils étaient la base des applications destinées à la détection/reconnaissance des différentes classes d'objets avant l'émergence des réseaux de neurones convolutionnels et profonds. Les descripteurs décrivent des caractéristiques élémentaires telles que les bords et segments, les points d'intérêt et les régions d'intérêt ou blobs, etc.

La suite de ce paragraphe présente les détecteurs communément utilisés dans le domaine de la vision par ordinateur.

1.6.1 Détecteur de Harris

1.6.1.1 Définition

Dans [11], Chris Harris et Mike Stephens ont proposé le détecteur Harris, ce détecteur de coins qui est couramment utilisé pour extraire les coins et déduire les caractéristiques des objets dans une image.

1.6.1.2 Etapes de l'algorithme

En général, l'algorithme du détecteur de coins de Harris se base sur cinq étapes

- Conversion de la couleur en niveaux de gris.
- Calcul de la dérivée spatiale dans les deux directions X et Y.
- Calcul de la matrice M

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{1.1}$$

- Calcul la matrice de score de réponse R était

$$R = det(M) - k.tr(M)^2 \tag{1.2}$$

Où
k : est une constante déterminée empiriquement entre 0,04 et 0,06
tr : c'est la trace de matrice
det : c'est le déterminant de matrice

- Suppression non-maximale par le choix d'un seuil de points acceptés pour tous les éléments de la matrice de score R.

1.6.2 Détecteur SIFT

1.6.2.1 Définition

D. Lowe [1] a proposé Sift, un algorithme de détection de caractéristiques locales dans les images. Les caractéristiques extraites sont formulées sous forme de descripteur SIFT invariant à l'échelle, à l'orientation, aux changements d'illumination, et partiellement invariant à la distorsion affine. L'algorithme SIFT a diverses applications en détection, reconnaissance et la mise en correspondance des objets. Ces caractéristiques sont locales et basées sur l'apparence de l'objet à des points d'intérêt particuliers, et sont invariantes à l'échelle et à la rotation de l'image.

1.6.2.2 Etapes de l'algorithme

En général, le détecteur SIFT présente essentiellement quatre étapes

- **Détection des extrema dans l'espace des échelles ou multi-échelle :** Pour détecter les points d'intérêt, le détecteur SIFT utilise d'abord le Laplacien du Gaussien pour convoluer l'image avec différentes valeurs de σ. le Laplacien du Gaussien agit comme un détecteur de taches (régions ou blobs) qui détecte les taches de différentes tailles en raison du changement de σ, de sorte que les maxima locaux fondés à travers l'échelle et l'espace qui donne une liste de valeurs (x, y, σ), ce qui signifie qu'il y a un point clé potentiel à (x, y) à l'échelle σ.

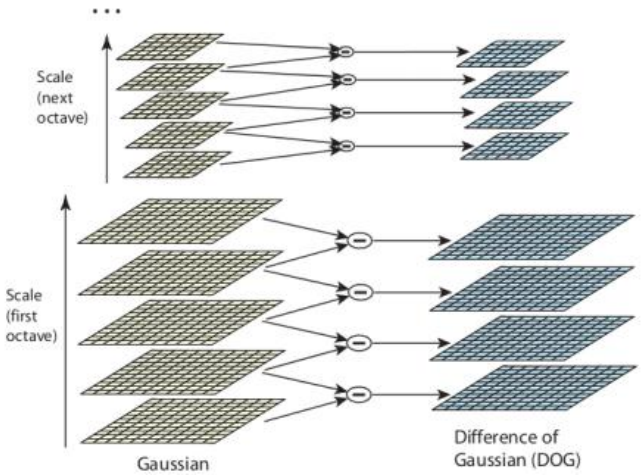


FIGURE 1.3 – Convoluer l'image à l'échelle multi-espace avec la différence de gaussienne [1]

Une fois ce Laplacien de Gaussien trouvé, les images sont recherchées pour les extrema locaux sur l'échelle et l'espace. Chaque pixel d'une image est comparé à ses 8 voisins ainsi qu'aux 9 pixels de l'échelle suivante et aux 9 pixels des échelles précédentes. S'il s'agit d'un extremum local, il s'agit d'un point clé potentiel.

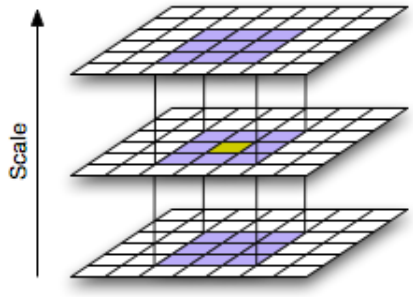


FIGURE 1.4 – Détecter les points extrêmes en comparant chaque point à ses 26 voisins [1]

- **Localisation des points clé :** La détection des extrema dans l'espace échelle produit un trop grand nombre de points clés candidats, dont certains sont instables. L'étape suivante de l'algorithme consiste à effectuer un ajustement détaillé aux données relatives afin de déterminer avec précision la localisation, l'échelle et le rapport des courbures principales. Ces informations permettent de rejeter les points clés qui présentent un faible contraste (et sont donc sensibles au bruit) ou qui sont mal localisés le long d'un bord.
- **Assignement de l'orientation :** Une orientation est ensuite attribuée à chaque point clé afin d'obtenir une invariance par rapport à la rotation de l'image. Un voisinage est pris autour de l'emplacement du point clé en fonction de l'échelle, et la magnitude et la direction du gradient sont calculées dans cette région. Un histogramme d'orientation avec 36 cases couvrant 360 degrés est créé. Le pic le plus élevé de l'histogramme est pris et tout pic au-dessus de 80% de celui-ci est également considéré pour calculer l'orientation.
- **Calcul du descripteur du point-clé :** pour extraire le descripteur de point clé ; tout d'abord, un ensemble d'histogrammes d'orientation est créé sur des voisinages de $4 * 4$ pixels avec 8 bins chacun. Ces histogrammes sont calculés à partir des valeurs de magnitude et d'orientation des échantillons dans une région $16 * 16$ autour du point clé, de sorte que chaque histogramme contient des échantillons d'une sous-région $4 * 4$ de la région de voisinage d'origine. Puisqu'il y a $4 * 4 = 16$ histogrammes, chacun avec 8 cases, le vecteur a 128 éléments. Ce vecteur est ensuite normalisé à une longueur unitaire afin d'améliorer l'invariance aux changements affines et d'illumination. Pour réduire les effets de l'illumination non linéaire, un seuil de 0.2 est appliqué et le vecteur est à nouveau normalisé.

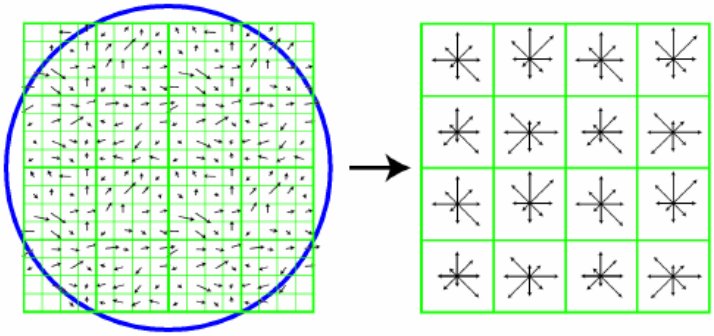


FIGURE 1.5 – Descripteur $4 * 4$ calculé à partir d'un ensemble d'échantillons $16 * 16$ [1]

1.6.3 Détecteur de SURF

1.6.3.1 Définition

Dans [12], Herbert Bay et *al.* ont proposé SURF, un nouveau détecteur de caractéristiques locales brevetées. Il s'inspire en partie du descripteur SIFT (scale-invariant feature transform), l'algorithme est invariant à l'échelle mais pas à la rotation de l'image.

1.6.3.2 Etapes de l'algorithme

L'algorithme SURF est basé sur les mêmes principes et étapes que SIFT, mais les détails de chaque étape sont différents

- **Détection des points d'intérêt :** SURF utilise des filtres en forme de carré comme une approximation du lissage gaussien. Le filtrage de l'image avec un carré est beaucoup plus rapide si l'on utilise l'image intégrale

$$S(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j) \tag{1.3}$$

  Ensuite, SURF utilise un détecteur de blob basé sur la matrice Hessienne pour trouver les points d'intérêt.
- **Localisation des points clés :** contrairement à SIFT, les espaces échelle dans SURF sont mis en œuvre en appliquant des filtres en boîte de différentes tailles. En conséquence, l'espace d'échelle est analysé en augmentant la taille du filtre plutôt qu'en réduisant itérativement la taille de l'image. Il en résulte des filtres de taille $9 * 9, 15 * 15, 21 * 21, 27 * 27$... La suppression non maximale dans un voisinage $3 * 3 * 3$ est appliquée pour localiser les points d'intérêt dans l'image et sur les échelles.
- **Affectation de l'orientation :** surfez d'abord pour calculer les réponses en ondelettes de Haar dans les directions x et y, et ce dans un voisinage circulaire de rayon 6s autour du point clé, puis calculez la somme des réponses en ondelettes verticales et horizontales dans une zone de balayage, puis changez l'orientation du balayage (ajoutez $\frac{\pi}{3}$), et recalculer, jusqu'à trouver l'orientation avec la plus grande valeur de somme.

- **Formation du descripteur :** SURF utilise les réponses Wavelet dans la direction horizontale et verticale (là encore, l'utilisation d'images intégrales facilite les choses). Un voisinage de taille $20s * 20s$ est pris autour du point clé où s est la taille. Il est divisé en $4 * 4$ sous-régions. Pour chaque sous-région, les réponses en ondelettes horizontale et verticale sont prises et un vecteur est formé comme ceci, $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$... On obtient ainsi un vecteur descripteur pour toutes les sous-régions $4 * 4$ de longueur 64.

1.6.4 Détecteur de HOG

1.6.4.1 Définition

Dans [13], Navneet Dalal et Bill Triggs ont proposé HOG, une technique de descripteur de caractéristiques qui compte les occurrences de l'orientation des gradients dans des parties localisées d'une image. Cette méthode est similaire à celle des histogrammes d'orientation des bords, des descripteurs de SIFT et des contextes de forme, mais elle diffère en ce qu'elle est calculée sur une grille dense de cellules uniformément espacées et qu'elle utilise une normalisation du contraste local par chevauchement pour une meilleure précision.

1.6.4.2 Etapes de l'algorithme

Le détecteur HOG comporte les étapes suivantes

- Trouver le gradient dans les directions x et y à l'aide du filtre de sobel, puis calculer l'amplitude et la direction des gradients avec des deux formules suivantes

$$g = \sqrt{g^2 + g_y^2} \tag{1.4}$$

$$\theta = arctan\frac{g_y}{g_x} \tag{1.5}$$

- **Calculer l'histogramme des gradients dans les cellules** $8 * 8$ **:** Dans cette étape, l'image $64 * 128$ est divisée en $8 * 8$ cellules et un histogramme de gradients est calculé pour chaque cellule et formé un histogramme à 9 cases (correspondant aux degrés 0, 20, 40 .. 160) pour chaque cellule, la valeur de la magnitude est relativement divisée entre les cases de l'histogramme (cela signifie que la valeur 35 sera divisée en 75% pour la case 40 et 25% pour la case 20).
- **Normalisation des blocs** $16 * 16$ **:** avec l'utilisation d'une fenêtre glissante avec Un bloc $16*16$ a 4 histogrammes qui peuvent être concaténés pour former un vecteur de $36 * 1$ éléments et le normaliser, La fenêtre est alors déplacée de 8 pixels et un Un vecteur normalisé $36 * 1$ est calculé sur cette fenêtre et le processus est répété.
- **Calculer le vecteur caractéristique HOGs :** il ne reste plus qu'à concaténer les blocs normalisés pour former le descripteur, qui sont $7*15 = 105$ blocs, chaque bloc a un vecteur de dimension $36 * 1$, ce qui signifie que la dimension du descripteur final est $36 * 105 = 3780$.

1.7 Classifieurs

En général, le choix d'algorithme de classification dépend de données et du problème à résoudre. Voici les classificateurs les plus courants de l'apprentissage des classes d'objets.

1.7.1 K-PPV

Dans [14], Evelyn Fix et Joseph Hodges ont proposé k plus proches voisins (k-ppv), un algorithme de classification supervisée non paramétrique, Il est utilisé à la fois pour la classification et la régression. Dans les deux cas, l'entrée est constituée des k exemples d'apprentissage les plus proches dans l'ensemble de données, où k est un nombre entier positif, généralement petit.

Dans la classification k-ppv, la sortie est une appartenance à une classe. Un objet est classé par une pluralité de votes de ses voisins. Dans le cas de la régression k-ppv, la sortie est la valeur de la propriété de l'objet. Cette valeur est la moyenne des valeurs des k-voisins les plus proches.

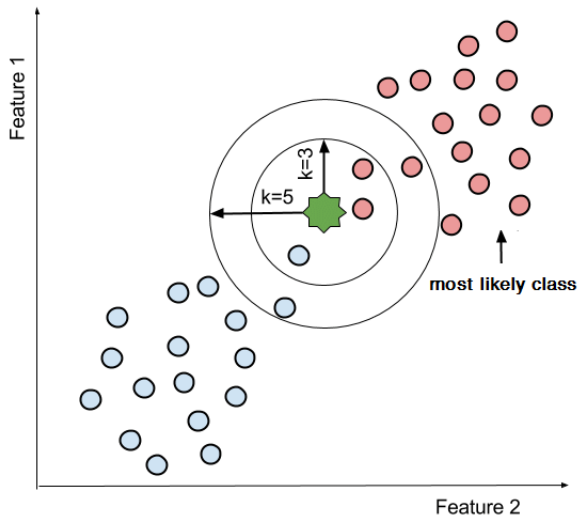


FIGURE 1.6 – Exemple d'algorithme de k-ppv dans le cas où $k = 3$ et $k = 5$

1.7.2 K-Means

Dans [15], James MacQueen a proposé K-Means, un algorithme itératif d'apprentissage automatique non supervisé qui tente de partitionner l'ensemble de données en K groupes disjoints appelés clusters. Il tente de rendre les points de données intra-cluster aussi similaires que possible tout en gardant les clusters aussi différents (éloignés) que possible. La figure suivante représente le fonctionnement de la classification K-means.

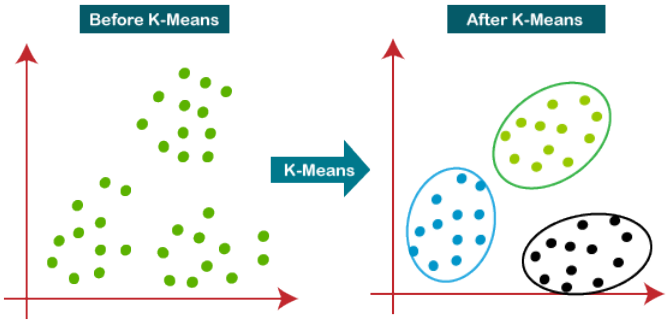


FIGURE 1.7 – Exemple d'algorithme de k-means avec $k = 3$ (3 clusters)

1.7.3 SVM

Dans [16], Vladimir Vapnik et *al.* ont proposé les SVM, sont des modèles d'apprentissage supervisé avec des algorithmes d'apprentissage associés qui analysent les données pour les problèmes de classification et de régression. Ils sont basés sur l'idée de trouver l'hyperplan parfait qui divise le mieux un ensemble de données en deux classes et maximise la distance inter-classe.

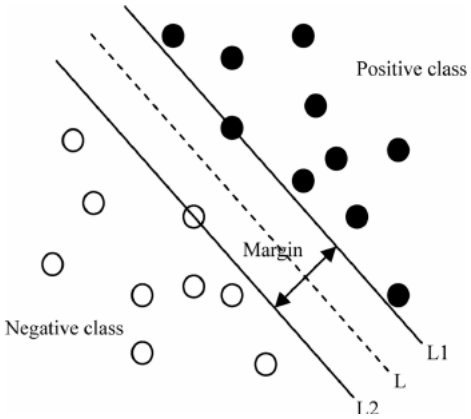


FIGURE 1.8 – Classification binaire par SVM

Dans le cas des classes non linéairement séparables, le SVM projette les données dans une plus grande dimension en utilisant l'astuce du noyau(kernel trick) qui applique certaines transformations aux données et crée de nouvelles données séparables.

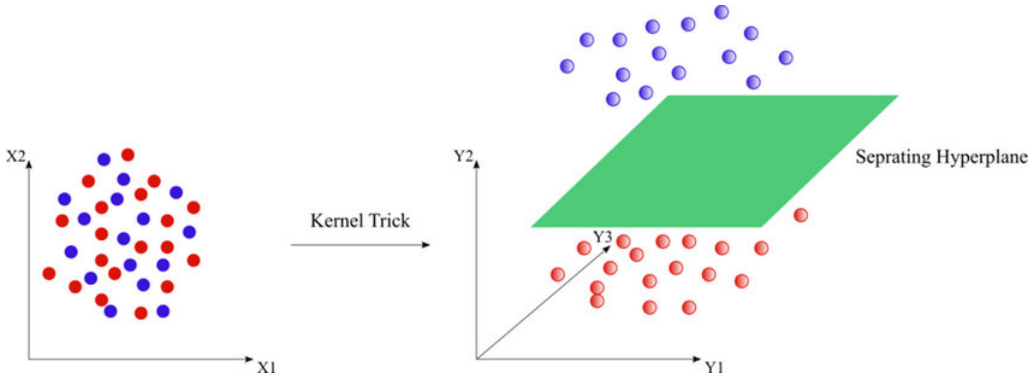


FIGURE 1.9 – Projection de données dans un espace plus grand

Pour la classification multi-classe, le même principe est utilisé après avoir décomposé le problème de classification multi-classe en plusieurs problèmes de classification binaire. L'idée est de faire correspondre les points de données à un espace de haute dimension pour obtenir une séparation linéaire mutuelle entre chaque deux classes, nous avons donc besoin d'un hyperplan pour séparer chaque deux classes, ce qui signifie qu'avec n classes nous aurons besoin de $\frac{n(n-1)}{2}$ hyperplans.

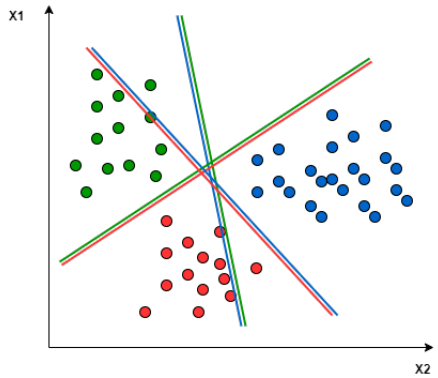


FIGURE 1.10 – Classification multi-classe par SVM

1.7.4 Réseaux de Neurones

Les réseaux neuronaux (RN) sont un modèle d'apprentissage artificiel qui utilise un réseau de fonctions pour comprendre et traduire une entrée de données d'une certaine forme en une sortie souhaitée, généralement sous une autre forme. Le concept de réseau neuronal artificiel s'inspire de la biologie humaine et de la façon dont les neurones du cerveau humain fonctionnent ensemble pour comprendre les entrées des sens humains.

1.7.4.1 Composants des RNs

Neurones et Couches Les RNs sont composés de neurones artificiels où chaque neurone a des entrées et produit une sortie unique qui peut être envoyée à plusieurs autres neurones. Les entrées peuvent être les valeurs des caractéristiques d'un échantillon de données externes, telles que des images ou des documents, ou les sorties d'autres neurones. Les couches sont un ensemble de neurones dont chaque neurone n'est pas connecté à un autre neurone de la même couche, mais est connecté à certains ou à tous les neurones de la couche précédente et/ou suivante.

Connexions et poids Le réseau est constitué de connexions, chaque connexion fournissant la sortie d'un neurone comme entrée à un autre neurone. Chaque connexion se voit attribuer un poids qui représente son importance relative.

Fonction de propagation La fonction de propagation calcule l'entrée d'un neurone à partir des sorties de ses neurones prédécesseurs et de leurs connexions sous la forme d'une somme pondérée. Un terme de biais peut être ajouté au résultat de

la propagation, puis le résultat de cette somme est soumis à une transformation linéaire ou non linéaire appelée fonction d'activation pour déterminer la valeur finale du neurone, la somme des activations communes étant

- Fonction sigmoïde

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.6}$$

- Fonction tangente hyperbolique

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{1.7}$$

- ReLU

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{1.8}$$

- La fonction Leaky ReLU

$$f(x) = \begin{cases} \alpha x & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \tag{1.9}$$

Où α est généralement égal à 0.01

1.7.4.2 Perceptron

Bien que Warren McCulloch et Walter Pitts aient été les premiers à ouvrir le sujet en créant un modèle computationnel pour les réseaux neuronaux en 1943 dont le travail a été publié dans [17], l'algorithme du perceptron a été inventé et utilisé en 1958 au Cornell Aeronautical Laboratory par Frank Rosenblatt [18].

Le perceptron est un classifieur binaire qui peut décider si une entrée, représentée par un vecteur de nombres, appartient ou non à une classe spécifique. C'est un type de classificateur linéaire, ce qui signifie qu'il ne peut résoudre que les cas linéairement séparables.

La perception fonctionne par une fonction de seuil qui fait correspondre son entrée (un vecteur de valeur réelle) à une valeur de sortie (une seule valeur binaire) :

$$f(x) = \begin{cases} 1 & \text{si } w*\text{x}+b > 0 \\ 0 & \text{sinon} \end{cases} \tag{1.10}$$

Où w est un vecteur de poids à valeur réelle, $w * x$ est le produit scalaire $\sum_{i=1}^{m} x_i * w_i$ où m est le nombre d'entrées du perceptron, et b est le biais. Le biais éloigne la limite de décision de l'origine et ne dépend d'aucune valeur d'entrée, La figure suivante représente un perceptron typique.

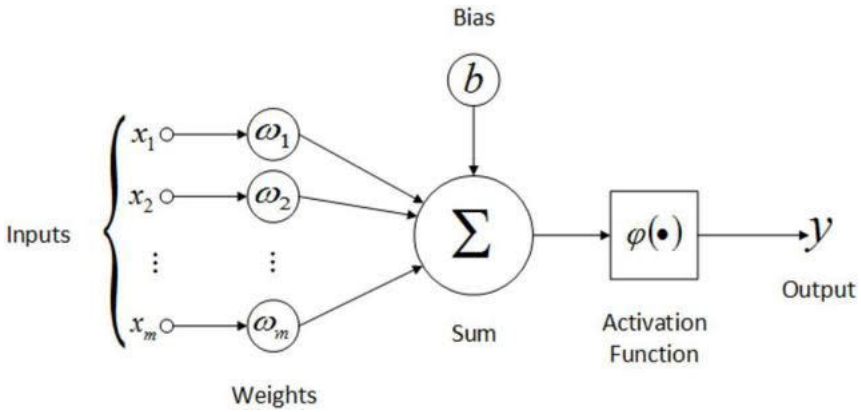


FIGURE 1.11 – Exemple typique d'un perceptron

1.7.4.3 Perceptron multicouche

Dans [19], Ivakhnenko et Lapa ont proposé le perceptron multicouche (MLP), lequel est une classe de réseaux neuronaux artificiels à anticipation. Il est constitué d'au moins trois couches des neurones : une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque nœud est un neurone qui utilise une fonction d'activation non linéaire. Le MLP utilise une technique d'apprentissage supervisée appelée rétropropagation pour l'apprentissage [20] (voir section suivante). Contrairement au perceptron, le perceptron multicouche peut distinguer des données qui ne sont pas linéairement séparables.

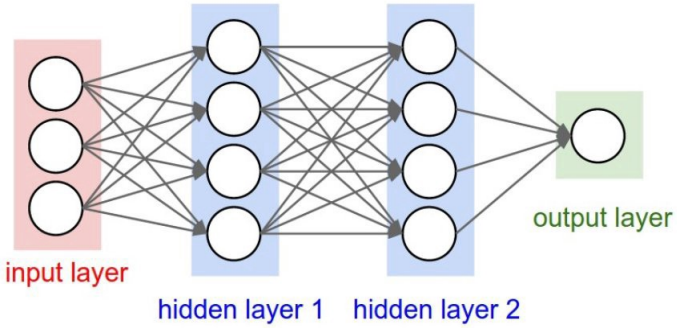


FIGURE 1.12 – Un perceptron multicouche avec deux couches cachées

1.7.4.4 Algorithme de Rétropropagation

Rumelhart, Hinton et Williams dans leurs travaux ([21], [22]) ont énoncé puis élaboré et popularisé l'algorithme de rétropropagation d'erreur, qui est utilisé pour l'entraînement des réseaux neuronaux feedforward.

L'algorithme de rétropropagation consiste ainsi à déterminer l'erreur entre les sorties de réseau et les sorties attendues. Puis, à rétropropager cette erreur à travers les couches du réseau dans le sens inverse (de la sortie vers l'entrée).

L'objectif de l'algorithme de la rétropropagation est d'adapter les poids afin de minimiser la fonction de loss (la soustraction entre la sortie attendue et la sortie de neurone) donnée par

$$E = \frac{1}{2}\sum_{j\in G} e_j^2 \tag{1.11}$$

Avec

$$e_j = d_j - o_j \tag{1.12}$$

Où
E : la somme de L'erreur quadratique moyenne des moindres carrés.
e_j : L'erreur du neurone j.
d_j : la sortie attendue du neurone j.
o_j : la sortie du neurone j.

La sortie o_j est définie par

$$o_j = f(s_j) = f(\sum_{i=1}^{N} w_{ij}x_i) \tag{1.13}$$

Où
f : La fonction d'activation du neurone j.
w_{ij} : Le poids de la connexion entre le neurone i de la couche précédente et le neurone j.
x_i : C'est l'entrée i.

Pour corriger l'erreur du réseau, il s'agit de modifier le poids W_{ij} dans le sens inverse au gradient de l'erreur notée $\nabla E = \frac{\delta E}{\delta W_{ij}}$

Nous pouvons appliquer le théorème de dérivation des fonctions composées (parfois appelé règle de dérivation en chaîne ou règle de la chaîne) pour calculer $\frac{\delta E}{\delta W_{ij}}$

$$\nabla E = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial o_j}\frac{\partial o_j}{\partial W_{ij}} \tag{1.14}$$

Nous exprimons la variation de poids ΔW_{ij} sous la forme suivante

$$\Delta W_{ij} = -\alpha \nabla E = -\alpha \frac{\partial E}{\partial W_{ij}} \tag{1.15}$$

Où
α : Représente le taux ou le pas d'apprentissage plus petit.
∇E : représente un facteur de sensibilité

Correction de l'erreur au niveau de la couche de sortie Les étapes de correction de poids pour la couche de sortie sont les mêmes pour le perceptron monocouche. La dérivée partielle de la fonction E se fait cette fois par la règle de chaînage.
Soit

$$\nabla E = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial o_j}\frac{\partial o_j}{\partial sj}\frac{\partial s_j}{\partial W_{ij}} \tag{1.16}$$

Par l'évaluation de chacun des termes du gradient (l'équation 1.16), nous obtenons donc

$$\frac{\partial E}{\partial W_{ij}} = -e_j o_j [1 - o_j] o_i \tag{1.17}$$

Et la règle dite de <delta> pour la couche de sortie s'exprime donc par

$$\Delta W_{ij} = -\alpha \frac{\partial E}{\partial W_{ij}} = -\alpha \delta_j o_j \tag{1.18}$$

Avec

$$\delta_j = c_j o_j [1 - o_j] \tag{1.19}$$

Où
δ_j : S'appelle le gradient local.

Correction de l'erreur au niveau de la couche cachée Considérons désormais le cas des neurones sur la dernière couche cachée (le cas des autres couches cachées est semblable).

L'expression de la dérivée partielle de l'erreur totale E par rapport à W_{ij} reste la même, mais nous ne dérivons pas par rapport d l'erreur car celle-ci est inconnue

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial s_j} \frac{\partial s_j}{\partial W_{ij}} \tag{1.20}$$

Nous remarquons que le 2^{eme} et le 3^{eme} terme de (l'équation 1.20) sont ceux (l'équation 1.16) de la couche de sortie, ils restent alors inchangés. et pour le 1^{er} terme la formule suivante est utilisée

$$\frac{\partial E}{\partial o_j} = \frac{\partial [\frac{1}{2} \sum_{k \in C} e_k^2]}{\partial o_j} \tag{1.21}$$

Où
C : Représente la couche de sortie.

Après l'évaluation de (l'équation 1.21) nous obtenons

$$\frac{\partial E}{\partial W_{ij}} = -o_j [1 - o_j] [\sum_{k \in C} \delta_k W_{jk}] \tag{1.22}$$

Et

$$\Delta W_{ij} = -\alpha \frac{\partial E}{\partial W_{ij}} = -\alpha \delta_j o_i \tag{1.23}$$

Avec

$$\delta_j = -o_j [1 - o_j] [\sum_{k \in C} \delta_k W_{jk}] \tag{1.24}$$

Où
δj : le gradient local de la couche cachée j.

Les deux équations (équation 1.23) et (équation 1.24) sont valables pour toutes les couches cachées.

Remarque : la fonction de loss utilisée dans cette section est appelée l'erreur quadratique moyenne mais il existe d'autres fonctions de loss. Également, l'algorithme de rétropropagation dans l'apprentissage profond est aujourd'hui appelé l'optimiseur et à différents types chacun avec sa philosophie, ces deux concepts seront discutés en détail dans le prochain chapitre.

1.8 Conclusion

Dans ce chapitre, nous avons discuté de certaines notions importantes de l'apprentissage automatique comme son objectivité, la façon dont il divise ses données, ses approches. Nous avons également vu la différence entre la classification et la régression et certains classifieurs communs dans ce domaine comme les svm, les réseaux de neurones, etc. nous avons mentionné certains détecteurs visuels très utilisés dans l'extraction des caractéristiques visuelles des images. Dans le chapitre suivant, nous aborderons une branche d' apprentissage automatique basée sur les réseaux de neurones artificiels qui a prouvé sa puissance descriptive et son efficacité dans le domaine de l'intelligence artificielle en général et de la vision par ordinateur en particulier, qui est l' apprentissage profond.

CHAPITRE 2

Apprentissage Profond

2.1 Introduction

Avec l'amélioration rapide de la capacité de calcul au cours des dernières années, et l'essor des GPU et TPU, l'utilisation de l'apprentissage profond a été appliquée à des domaines tels que la vision par ordinateur, la reconnaissance vocale, le traitement du langage naturel, la traduction automatique, la bioinformatique, la conception de médicaments, l'analyse d'images médicales, etc.

L'apprentissage profond est basé sur des réseaux de neurones artificiels avec apprentissage par représentation, qui s'inspirent du traitement de l'information et des nœuds de communication distribués dans les systèmes biologiques (voir chapitre 1).

Dans ce chapitre, nous expliquerons la relation entre l'intelligence artificielle, l'apprentissage automatique et l'apprentissage profond, la notion de profondeur dans l'apprentissage profond, certaines fonctionnalités de base de l'apprentissage profond, la notion de sur-apprentissage et certaines méthodes courantes pour l'éviter.

2.2 Position de l'Apprentissage Profond

Avant d'aller plus loin dans ce chapitre, nous devons clarifier la relation entre l'intelligence artificielle, l'apprentissage automatique et l'apprentissage profond.

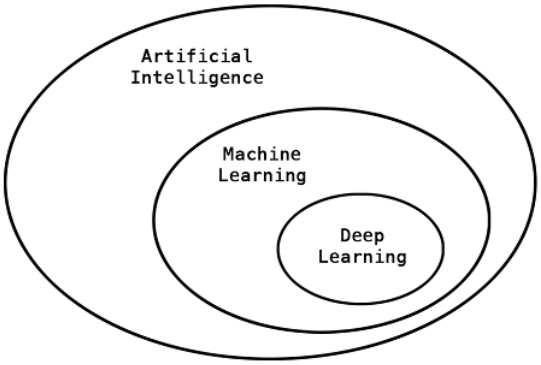


FIGURE 2.1 – La relation entre l'intelligence artificielle, l'apprentissage automatique et l'apprentissage profond [2]

L'apprentissage automatique est la science de la création d'algorithmes qui s'améliorent automatiquement grâce à l'expérience [23], ces algorithmes construisent un modèle en apprenant à partir d'un ensemble de données appelées données de formation et prédisent des données du même type sans avoir été vues auparavant.

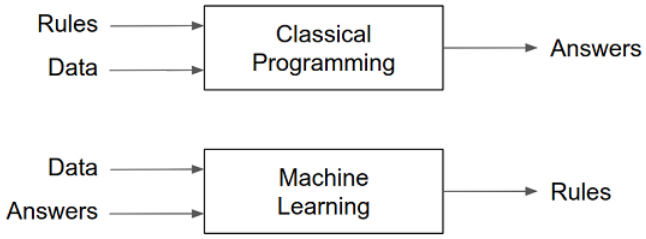


FIGURE 2.2 – Comparaison entre l'ancien paradigme de l'IA et le paradigme de l'apprentissage automatique

Apprentissage profond (Deep Learning en anglais) est un ensemble d'algorithmes d'apprentissage artificiel qui tentent d'apprendre à plusieurs niveaux correspondant à différents niveaux d'abstraction. Il utilise à cette fin le modèle des réseaux neuronaux artificiels. Les niveaux de ces modèles statistiques appris correspondent à des niveaux distincts de concepts, où les concepts de niveau supérieur sont définis à partir de ceux de niveau inférieur, et les mêmes concepts de bas niveau peuvent aider à définir de nombreux concepts de niveau supérieur [24].

L'avantage de l'apprentissage profond sur l'apprentissage automatique est la quantité de données : Les algorithmes d'apprentissage automatique fonctionnent souvent bien même si l'ensemble de données est petit, mais l'apprentissage profond est avide de données : plus vous avez de données, plus il est susceptible d'être performant.

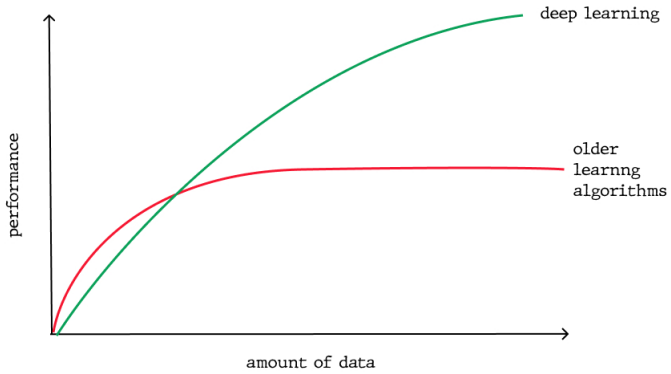


FIGURE 2.3 – Comparaison des performances des algorithmes d'apprentissage automatique et d'apprentissage profond par rapport à la quantité de données

Un autre grand avantage de l'apprentissage profond sur l'apprentissage automatique est que l'apprentissage automatique extrait les caractéristiques des données manuellement en utilisant des descripteurs locaux comme SIFT [1], SURF [12], HOG [13], contrairement à l'apprentissage profond qui les extrait automatiquement en utilisant un réseau de neurones artificiels.

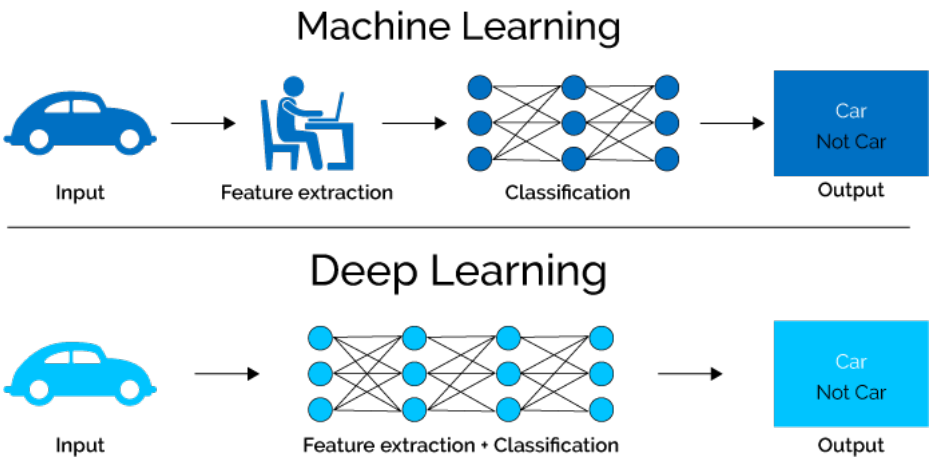


FIGURE 2.4 – Comparaison des processus d'extraction de caractéristiques dans les algorithmes d'apprentissage automatique et d'apprentissage profond

2.3 Notion de Profondeur

L'apprentissage profond est une nouvelle approche de l'apprentissage de représentations à partir de données qui met l'accent sur l'apprentissage de couches successives de représentations de plus en plus significatives. Le terme "profond" ne fait pas référence à une quelconque compréhension plus profonde obtenue par cette approche ; il désigne plutôt cette idée de couches successives de représentations. Le nombre de couches qui contribuent à un modèle de données est appelé la profondeur du modèle.

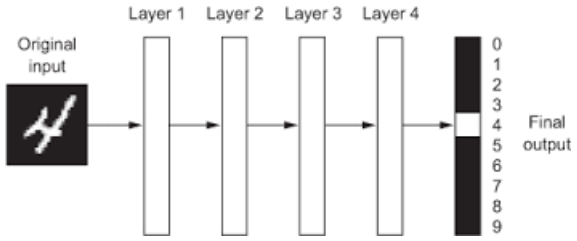


FIGURE 2.5 – Réseau de neurones profonds pour la classification des chiffres avec 4 couches cachées

2.4 Architectures d'Apprentissage profond

Les architectures des réseaux de neurones en apprentissage profond sont divisées par leur mode d'apprentissage en deux types : les architectures d'apprentissage supervisé et non supervisé, mais dans cette mémoire, nous ne sommes intéressés que du premier type.

2.4.1 Réseau neuronal convolutif

un réseau de neurones convolutif est un réseau multicouche qui a été biologiquement inspiré par le cortex visuel animal [25], ce type d'architecture est particulièrement utile dans les applications de vision par ordinateur,son mode de fonctionnement

peut être séparé en deux phases : l'extraction de caractéristiques à l'aide d'opérations de convolution et de pooling ,et la classification à l'aide d'un perceptron multicouche moderne,cette architecture est discutée en détail dans le chapitre suivant.

2.4.2 Réseaux neuronal récurrent

Un réseaux neuronal récurrent est une architecture d'apprentissage profond qui diffère des autres par ses connexions entre les couches, ils peuvent avoir des connexions qui rétroagissent sur les couches précédentes ou sur la même couche, cette rétroaction permet de conserver la mémoire des entrées passées et de modéliser les problèmes dans le temps, cette architecture est très utilisée dans des applications telles que la traduction automatique, la prédiction de séries temporelles, la reconnaissance vocale, l'apprentissage de la grammaire, etc. Elle existe dans de nombreuses variantes, telles que l'unité récurrente complète, l'unité bidirectionnelle, la mémoire à long et à court terme (LSTM), l'unité récurrente à grille (GRU), etc.

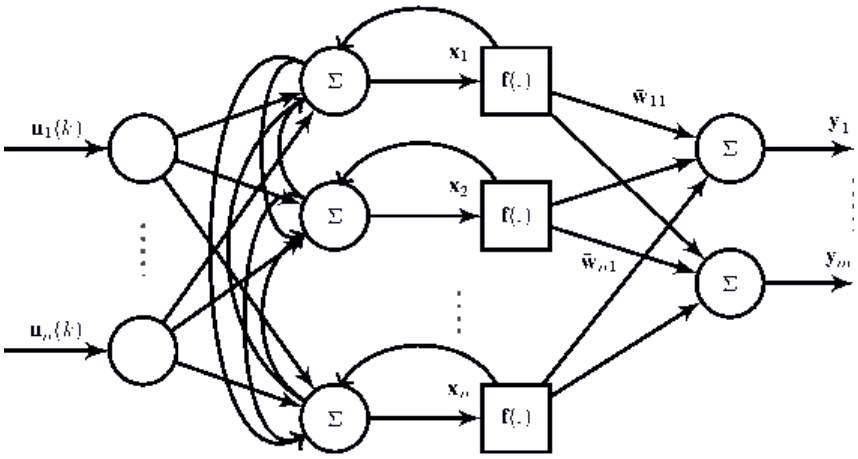


FIGURE 2.6 – Exemple typique d'un réseau de neurones récurrent

Remarque : Il existe un type d'architecture d'apprentissage profond supervisé appelé réseau neuronal récurrent convolutif (CRNN) qui présente à la fois les caractéristiques du convolutif et du récurrent. Ce type de réseau est généralement utilisé pour reconnaître les symboles séquentiels dans les images.

2.5 Fonctionnement de l'apprentissage profond

Les réseaux de neurones profonds font un mappage entrée-cible via une séquence profonde de transformations de données simples (couches) et que ces transformations de données sont apprises par exposition à des exemples, et la transformation mise en œuvre par une couche est paramétrée par ses poids, de sorte que nous pouvons dire que l'objectif de l'apprentissage profond est de trouver la valeur parfaite de ces poids de manière à ce que le réseau associe correctement les entrées des exemples à leurs cibles associées

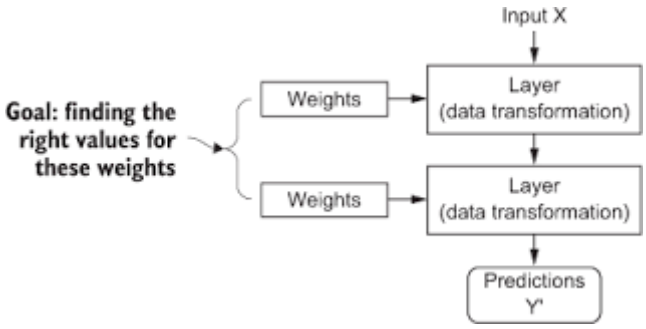


FIGURE 2.7 – Un réseau de neurones paramétré par ses poids [2]

Pour contrôler quelque chose, il faut d'abord être capable de l'observer. Pour contrôler la sortie d'un réseau neuronal, il faut pouvoir mesurer la distance entre cette sortie et ce que l'on attendait, puis ajuster un peu la valeur des poids, dans une direction qui réduira le score de loss pour l'exemple actuel. C'est le travail de la fonction de loss et de l'optimiseur respectivement.

2.5.1 Fonction de Loss

La fonction de Loss, également appelée fonction objective, prend les prédictions du réseau et la cible réelle (ce que vous vouliez que le réseau produise) et calcule un score de distance.

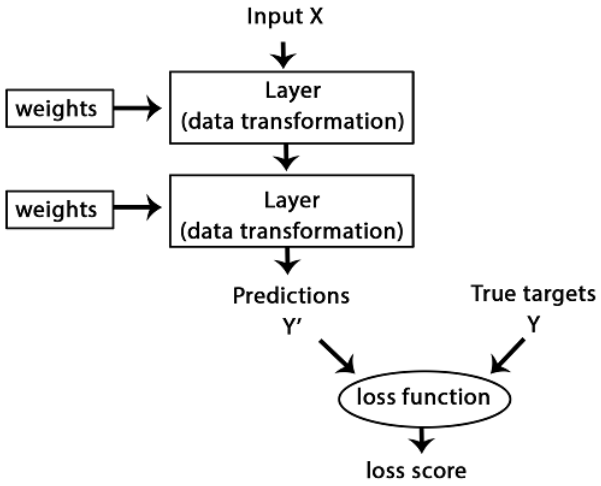


FIGURE 2.8 – Une fonction de loss pour mesurer la qualité de la sortie du réseau [2]

2.5.1.1 Erreur quadratique moyenne

Comme son nom l'indique, l'erreur quadratique moyenne (Quadratic Loss, en anglais) est mesurée comme la moyenne de la différence au carré entre les prédictions et les observations réelles.

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}{n} \tag{2.1}$$

2.5.1.2 Erreur absolue moyenne

L'erreur absolue moyenne, quant à elle, est mesurée comme la moyenne de la somme des différences absolues entre les prédictions et les observations réelles.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \widehat{y}_i|}{n} \tag{2.2}$$

2.5.1.3 Erreur moyenne de biais

Il est beaucoup moins courant dans le domaine de l'apprentissage automatique que son homologue. C'est la même chose que MSE, à la seule différence que nous ne prenons pas de valeurs absolues.

$$MBE = \frac{\sum_{i=1}^{n} (y_i - \widehat{y}_i)}{n} \tag{2.3}$$

2.5.1.4 Erreur d'entropie croisée (Negative Log Likelihood)

Il s'agit du paramètre le plus courant pour les problèmes de classification. La loss d'entropie croisée augmente lorsque la probabilité prédite diverge de l'étiquette réelle.

$$CrossEntropyLoss = -(y_i log(\widehat{y}_i) + (1 - y_i) log(1 - \widehat{y}_i)) \tag{2.4}$$

2.5.2 Optimiseurs

l'optimiseur implémente ce qu'on appelle l' algorithme de Rétropropagation de l'erreur dans le premier chapitre, qui est l'algorithme central de l'apprentissage profond pour ajuster les poids.

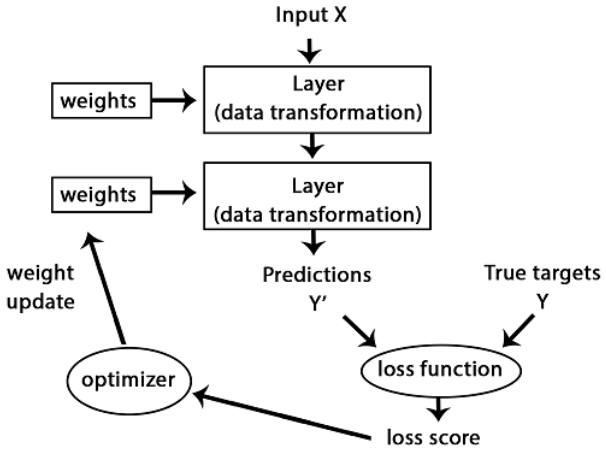


FIGURE 2.9 – Utilisez le score de loss comme signal de retour pour ajuster les poids par l'optimiseur [2]

2.5.2.1 Descente de gradient

La descente de gradient est l'algorithme de rétropropagation le plus basique, c'est un algorithme d'optimisation du premier ordre qui dépend de la dérivée du premier

ordre d'une fonction de loss. Il calcule de quelle manière les poids doivent être modifiés pour que la fonction puisse atteindre un minimum, sa formule est

$$\theta = \theta - \alpha \nabla J(\theta) \tag{2.5}$$

Où α est le taux d'apprentissage.

2.5.2.2 Momentum

Développé par Rumelhart et *al.* en 1986 [21],momentum est comme une balle qui descend une pente. La balle prend de l'élan en descendant la colline, ce qui contribue à accélérer la descente en gradient (GD) lorsque les surfaces présentent une courbe plus raide dans une direction que dans une autre [26].

$$V(t) = \gamma V(t-1) + \alpha \nabla J(\theta) \tag{2.6}$$

$$\theta = \theta - V(t) \tag{2.7}$$

Où γ est un terme de quantité de mouvement(momentum term) généralement égal à 0.9

2.5.2.3 AdaGrad

Adagrad ou Algorithme de gradient adaptatif est une méthode de taux d'apprentissage adaptatif publiée en 2011 par Duchi et al. [27]. Adagrad adopte le taux d'apprentissage aux paramètres et effectue des mises à jour plus importantes pour les paramètres peu fréquents et des mises à jour plus petites pour les paramètres fréquents [27].

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{g_{t,i} + \varepsilon}} . g_{t,i} \tag{2.8}$$

Où $g_{t,i}$ dérivée de la fonction de loss à un instant t donné.

2.5.2.4 RMSProp

RMSprop est une méthode de taux d'apprentissage adaptatif proposée par Geoff Hinton [54], elle utilise la magnitude des descentes de gradient récentes pour normaliser le gradient, dans RMSProp le taux d'apprentissage est ajusté automatiquement et il choisit un taux d'apprentissage différent pour chaque paramètre [28].

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{(1-\gamma) g_{t-1}^2 + \gamma g_t + \varepsilon}} . g_t \tag{2.9}$$

Où g_t est la moyenne du gradient carré.

2.5.2.5 Adam

Développé par Kingma et *al.* en 2014 [29], Adam (Adaptive Moment Estimation) peut être considéré comme une combinaison d'AdaGrad, qui fonctionne bien sur les gradients sparse et de RMSprop qui fonctionne bien dans des paramètres en ligne et non stationnaires [29]. L'algorithme Adam met d'abord à jour les moyennes mobiles exponentielles du gradient mt et du gradient au carré vt qui sont les estimations du premier et du deuxième moment.

Les hyper-paramètres $\beta 1, \beta 2 \in [0, 1)$ contrôlent les taux de décroissance exponentielle de ces moyennes mobiles comme indiqué ci-dessous.

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.10}$$

$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2.11}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \varepsilon} \widehat{m}_t \tag{2.12}$$

2.6 Sur-apprentissage

Le Surapprentissage se réfère à un modèle qui modélise trop bien les données de l'entraînement, il se produit lorsqu'un modèle apprend les détails et le bruit des données de l'entraînement au point d'avoir un impact négatif sur les performances du modèle sur les nouvelles données. Cela signifie que le bruit ou les fluctuations aléatoires des données de l'entraînement sont captés et appris en tant que concepts par le modèle. Le problème est que ces concepts ne s'appliquent pas aux nouvelles données et ont un impact négatif sur la capacité du modèle à généraliser.

Heureusement, il existe de nombreuses techniques pour lutter contre le surapprentissage, que nous verrons dans la section suivante.

2.7 Méthodes pour éviter le Surapprentissage

2.7.1 Augmentation des données

L'augmentation des données consiste à générer davantage de données d'apprentissage à partir d'échantillons d'apprentissage existants, en augmentant les échantillons par un certain nombre de transformations aléatoires qui produisent des images d'apparence crédible [30].

FIGURE 2.10 – Génération d'images de chats par augmentation aléatoire des données

2.7.2 Arrêt précoce (Early stopping)

Une autre technique facile et efficace pour lutter contre le surapprentissage est l'arrêt précoce, lorsque nous formons un modèle, nous pouvons mesurer la performance de chaque itération du modèle, jusqu'à un certain nombre d'itérations, les nouvelles itérations améliorent le modèle. Après ce point, cependant, la capacité du modèle à généraliser peut s'affaiblir car il commence à s'adapter excessivement aux données de formation, l'idée d'arrêt précoce fait référence à l'arrêt du processus de formation avant que l'apprenant ne passe ce point [31] et est expliqué de manière crédible dans la figure suivante.

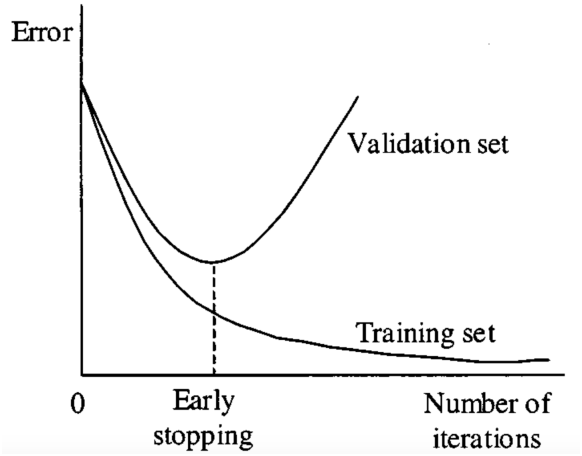


FIGURE 2.11 – Comment fonctionne l'arrêt précoce

2.7.3 Méthodes de régularisation

2.7.3.1 Dropout

Dans [32], Geoff Hinton et *al.* ont proposé Dropout, une des techniques de régularisation les plus efficaces et les plus couramment utilisées pour les réseaux de neurones. Elle applique à une couche, consiste à abandonner de manière aléatoire (en mettant à zéro) un certain nombre de caractéristiques de sortie de la couche pendant l'apprentissage.

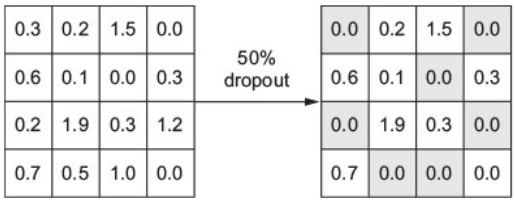


FIGURE 2.12 – Comment fonctionne le dropout

2.7.3.2 Régularisation de poids

Une autre façon courante d'atténuer le surapprentissage consiste à imposer des contraintes sur la complexité d'un réseau en forçant ses poids à ne prendre que de petites valeurs, ce qui rend la distribution des valeurs de poids plus régulière. C'est ce que l'on appelle la régularisation des poids, qui s'effectue en ajoutant à la fonction de loss du réseau un coût associé à l'utilisation de poids importants [34]. Ce coût est de deux types

- **Régularisation L1** - Le coût ajouté est proportionnel à la valeur absolue des coefficients de pondération.
- **Régularisation L2** - Le coût ajouté est proportionnel au carré de la valeur des coefficients de pondération, L2 régularisation est également appelée 'weight decay' dans les livres et articles.

2.8 Conclusion

Dans ce chapitre, nous avons tout d'abord vu ce qu'est l'apprentissage profond et comment il diffère des algorithmes traditionnels d'apprentissage artificiel. Nous avons clairement expliqué la notion de profondeur dans l'apprentissage profond, son fonctionnement à l'aide de la fonction de loss et de l'optimiseur. Nous avons également expliqué l'un des plus grands problèmes des modèles d'apprentissage profond qui est le surapprentissage ainsi que les méthodes efficaces pour en lutter.

Dans le chapitre suivant, nous verrons une des architectures de l'apprentissage profond la plus couramment utilisée en vision par ordinateur, à savoir le réseau de neurones convolutionnels.

CHAPITRE 3

Réseau de Neurones Convolutifs

3.1 Introduction

Le réseau de neurones convolutifs (CNN) est une classe de réseaux de neurones profonds et le plus communément appliqué pour analyser l'imagerie visuelle [35]. Ces modèles ont été proposés en 1962 par Hubel et Wiesel [25] pour imiter le comportement d'un cortex visuel, mais leur premier succès fut dans le travail proposé dans [36] par Lecun et *al.* dénommé LeNet-5 pour la classification de chiffres manuscrits (voir section 3.3.1). Ils ont des applications dans la reconnaissance d'images et de vidéos, les systèmes de recommandation, la classification d'images, la segmentation d'images, l'analyse d'images médicales, le traitement du langage naturel, les interfaces cerveau-ordinateur, etc.

Dans ce chapitre, nous examinerons les principaux composants d'un réseau de neurones convolutifs : les couches de convolution et de pooling, quelques architectures célèbres, les jeux de données habituellement utilisées pour tester les performances de ces CNN, et quelques méthodes de visualisation où le modèle apprend le plus.

3.2 Principes des réseaux neuronaux convolutifs

L'architecture d'un réseau neuronal convolutif est composée de trois éléments principaux : les couches de convolution, les couches de pooling et les couches entièrement connectées.

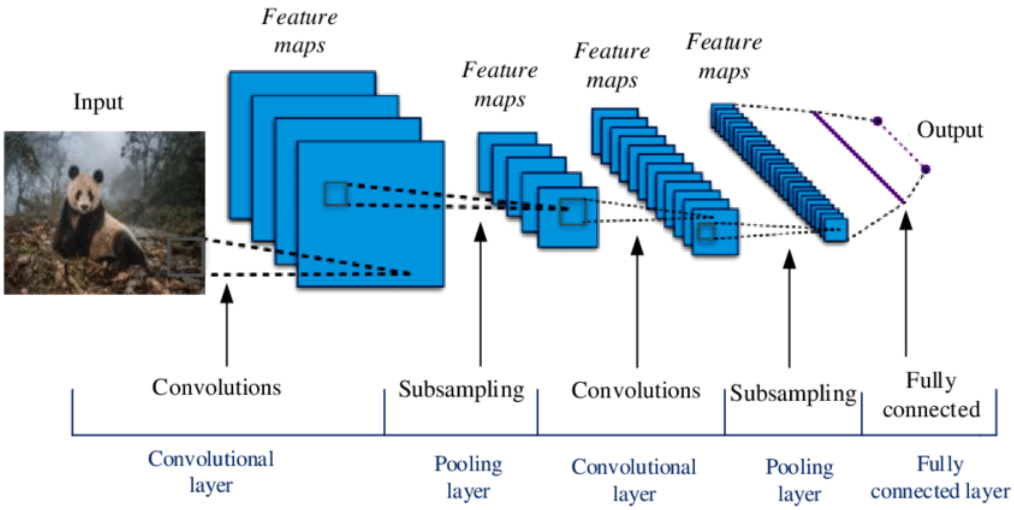


FIGURE 3.1 – Architecture typique d'un réseau de neurones convolutifs

Les couches de convolution et de pooling sont utilisées pour extraire les caractéristiques et produire ce que l'on appelle la feature map, puis elles sont aplaties pour devenir un tenseur 1D et vont vers les couches entièrement connectées qui sont un perceptron multicouche traditionnel pour classifier l'image.

3.2.1 Couches de convolution

L'opération de convolution prend un tenseur (2D ou 3D) en entrée et lui applique un certain nombre de noyaux. nous faisons l'opération de convolution exactement comme nous le faisons en traitement d'image dans le cas d'un tenseur 2D, dans le cas d'un tenseur 3D, il utilise un noyau 3D pour la convolution, puis utilisez la somme élémentaire entre les tenseurs résultants pour produire un tenseur 2D, le résultat de cette opération est N tenseurs 2D filtrés où N est le nombre de noyaux appliqués.

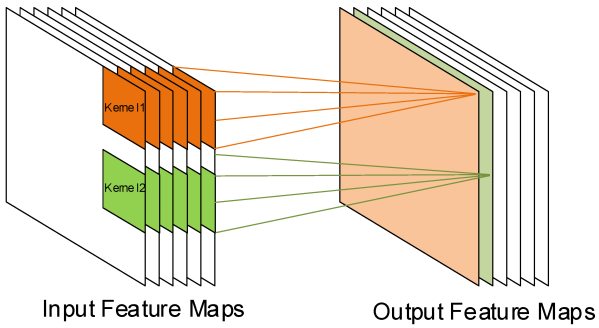


FIGURE 3.2 – Illustration de l'opération de convolution

Remarque : En apprentissage profond, chaque tableau numpy multidimensionnel est un tenseur.

3.2.2 Couches de pooling

L'opération de pooling permet de réduire la dimension des tenseurs, cette opération nécessite deux paramètres : la taille de la fenêtre carrée glissante, et le stride, le plus connu est le pooling max et le pooling moyen qui prennent une part $N * N$ de l'image généralement 2∗2, et sélectionnez la valeur la plus grande et la valeur moyenne respectivement à l'intérieur de la fenêtre, en commençant par le coin supérieur droit et en suivant le reste de l'image en fonction de le stride choisie.

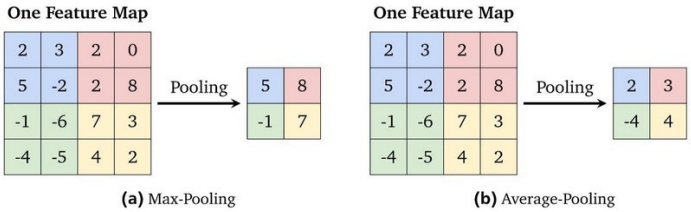


FIGURE 3.3 – Illustration des opérations average pooling et max pooling

3.2.3 Couches entièrement connectées

Après l'extraction des caractéristiques en utilisant des couches de convolution et de pooling , nous devons transférer notre feature map 3D empilées à 1D (Flatten), puis la donner à une série de couches entièrement connectées pour apprendre à classer les images. Les neurones et les poids dans ces couches sont ceux que vous connaissez des perceptrons multicouches traditionnels.

3.3 Architectures des réseaux neuronaux convolutifs

3.3.1 LeNet-5

Dans [3], Yann LeCun et *al.* ont proposé LeNet-5, le premier modèle réussi. Ils ont utilisé cette architecture pour reconnaître les chiffres manuscrits dans le jeu de données MNIST [40]. L'architecture de ce modèle est simple et efficace pour classer les chiffres.

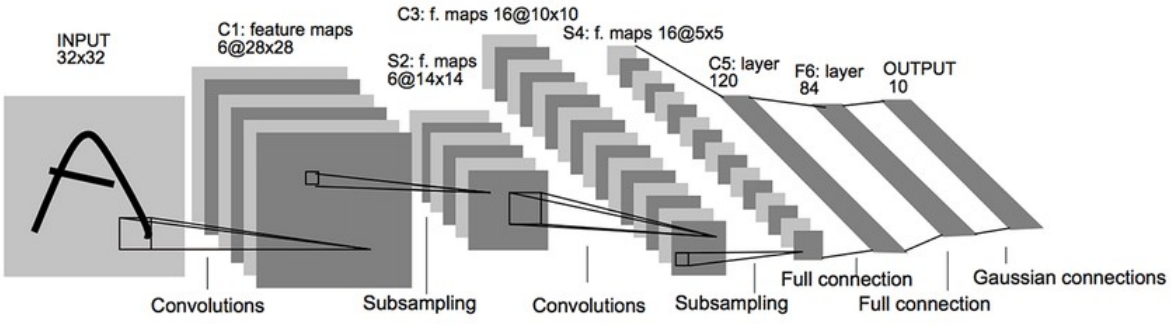


FIGURE 3.4 – Architecture détaillée de LeNet-5 [3]

3.3.2 AlexNet

Dans [41], Alex Krizhevsky et *al.* ont proposé AlexNet, la première architecture CNN a gagné la compétition ImageNet ILSVRC-2012, ils ont atteint le top-1 et le top-5 des taux d'erreur de 37.5% et 17.0%, le réseau a 60 millions de paramètres. Ils ont utilisé la fonction d'activation ReLU au lieu de tanh pour ajouter la non-linéarité, dropout au lieu de la régularisation pour traiter le sur-apprentissage et overlap pooling pour réduire la taille du réseau.

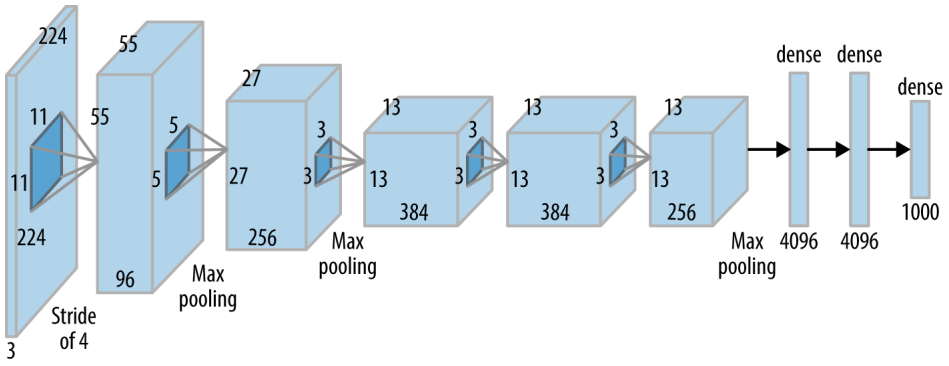


FIGURE 3.5 – Architecture détaillée de AlexNet

3.3.3 VGGNet

Dans [4], Karen Simonyan et *al.* ont proposé VGG16, un modèle de réseau de neurones convolutif a gagné la compétition ILSVRC-2014 avec une erreur top-1 de 24,8% et une erreur top-5 de 7.5%, contrairement au modèle AlexNet qui utilise des filtres de grande taille (11 et 5 dans la première et la deuxième couche convolutive respectivement), VGG utilise des champs réceptifs très petits ($3 * 3$ avec un stride de 1), l'image d'entrée est un RVB de $224 * 224$ pixels, le seul prétraitement qu'ils font est de soustraire la valeur RVB moyenne.

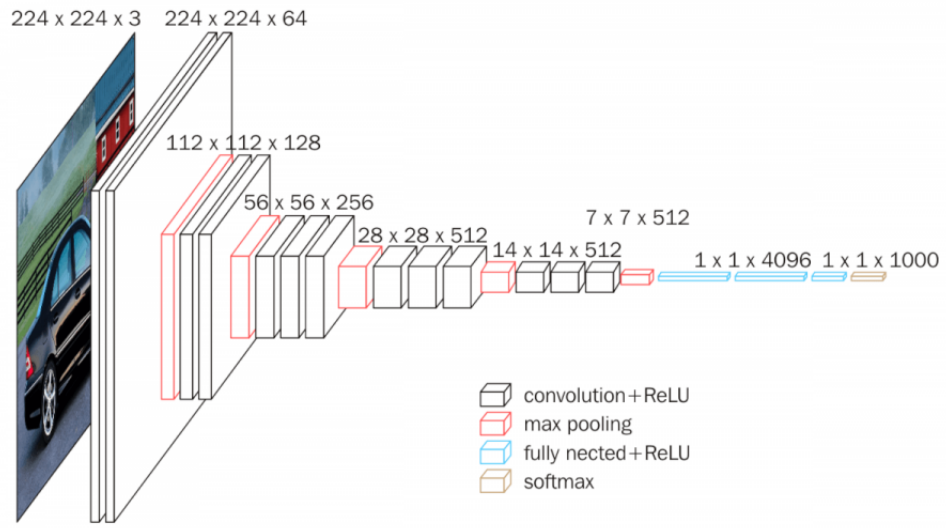


FIGURE 3.6 – Architecture détaillée de VGGNet [4]

3.3.4 ResNet

Dans [5], Kaiming He et *al.* ont proposé ResNet, un type spécifique de réseau de neurones qui utilisait les blocs résiduels pour faciliter l'apprentissage des réseaux qui sont sensiblement plus profonds, et le succès d'éviter le problème de gradient disparaissant/explosant [42] que les modèles d'apprentissage profond souffrent quand ils sont plus profonds. Leur idée leur a permis de remporter la première place à la tâche de classification ILSVRC-2015.

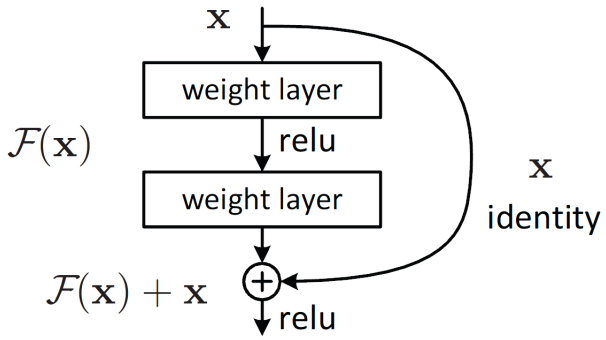


FIGURE 3.7 – Blocs Résiduels [5]

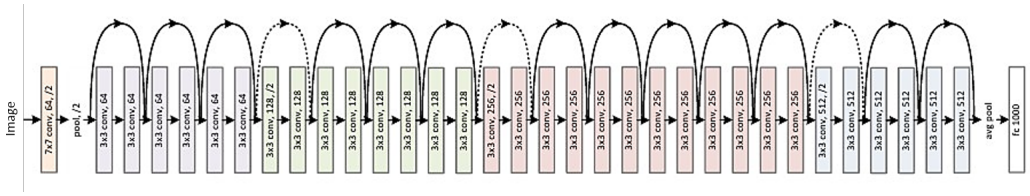


FIGURE 3.8 – Architecture de ResNet [5]

3.3.5 EfficientNet

Dans [6], Mingxing Tan et *al.* ont proposé EfficientNet, une architecture de réseau neuronal convolutif et une méthode de mise à l'échelle. L'idée de cette méthode est de mettre à l'échelle uniformément toutes les dimensions de profondeur-largeur-résolution en utilisant un coefficient composé. Par exemple, si nous voulons utiliser des fois plus de ressources informatiques, alors nous pouvons simplement augmenter la profondeur du réseau par a^n, la largeur par b^n, et la taille de l'image par c^n, où a, b, c sont des coefficients constants déterminés par une recherche de grille sur le petit modèle original, ils utilisent la recherche d'architecture neuronale pour concevoir un nouveau réseau de base et le mettre à l'échelle pour obtenir une famille de modèles, appelés EfficientNets (EfficientNet-B0, EfficientNet-B1, ... EfficientNet-B7), ils ont utilisé la profondeur stochastique [43] et la compression et l'excitation [44], avec l'architecture EfficientNet-B7 ils obtiennent une accuracy de pointe de 84,3% sur ImageNet.

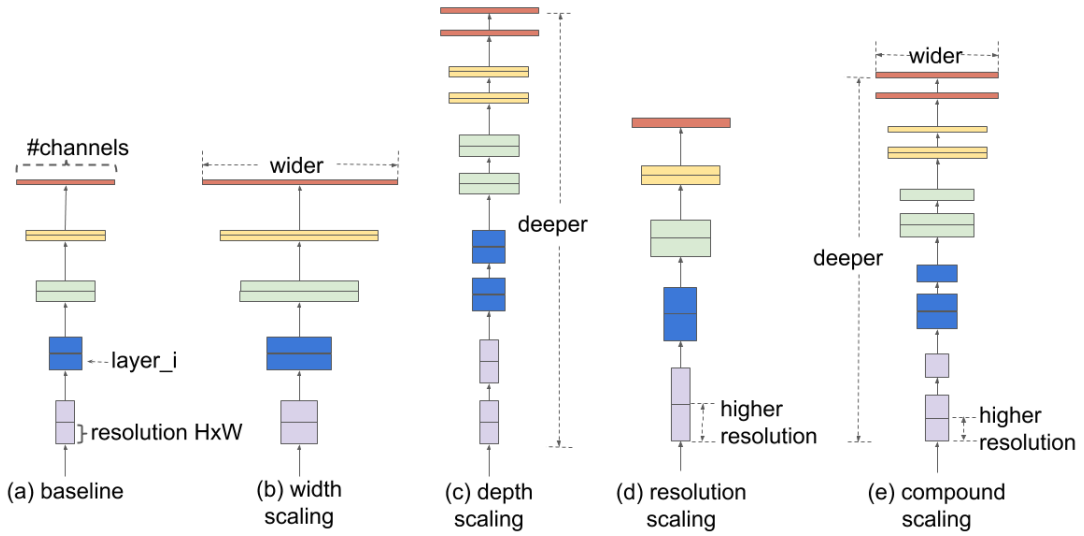


FIGURE 3.9 – (a) Exemple d'un réseau, (b) Mise à l'échelle de la largeur, (b) Mise à l'échelle de la profondeur, (d) Mise à l'échelle de la résolution, (e) Mise à l'échelle composée [6]

3.3.6 Pseudo labels (Noisy Student)

Dans [7], Qizhe Xie et *al.* ont proposé Pseudo labels, une approche d'apprentissage semi-supervisée proposée, elle étend l'idée d'auto-formation et de distillation avec l'utilisation de modèles d'étudiants égaux ou plus grands et de bruit ajouté à l'étudiant pendant l'apprentissage. Sur ImageNet, ils forment d'abord un modèle EfficientNet sur des images étiquetées et l'utilisent comme enseignant pour générer des pseudo-étiquettes pour 300 millions d'images non étiquetées du jeu de données JFT-300M [45]. Ils forment ensuite un Efficient-Net plus grand comme modèle étudiant sur la combinaison d'images étiquetées et pseudo-étiquetées. Ils réitèrent ce processus en mettant l'étudiant comme enseignant. Pendant l'apprentissage de l'étudiant, ils injectent du bruit comme dropout [32], profondeur stochastique [43] et augmentation des données via RandAugment [46] à l'étudiant de sorte que l'étudiant généralise mieux que l'enseignant, avec cette nouvelle approche ils atteignent 88,4% d'accuracy top-1 sur ImageNet, ce qui est 2,0% de mieux que le modèle de pointe à ce moment-là.

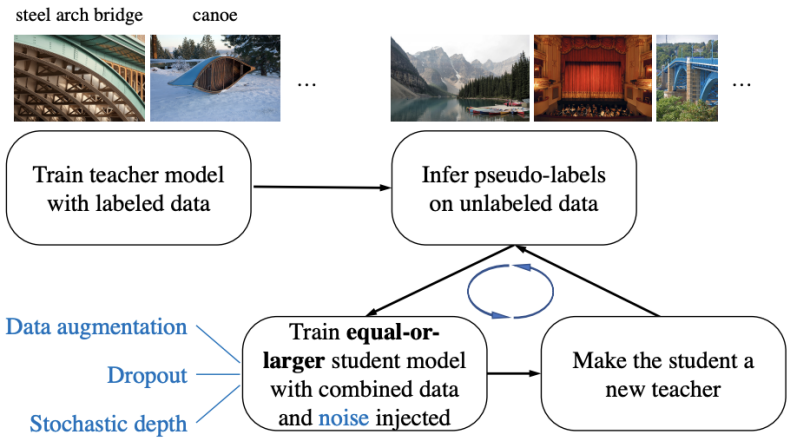


FIGURE 3.10 – Illustration de l'apprentissage du modèle noisy student [7]

3.3.7 Meta Pseudo Labels

Dans [8], Hieu Pham et *al.* ont proposé Meta Pseudo Labels, une méthode d'apprentissage semi-supervisée proposée qui détient actuellement la meilleure accuracy de l'état de l'art de 90,2% sur ImageNet, soit 1,6% de mieux que l'état de l'art précédent. Comme Pseudo Labels, Meta Pseudo Labels a un réseau d'enseignants qui génère des pseudo-étiquettes sur des données non étiquetées pour enseigner à un réseau d'étudiants. Cependant, contrairement aux Pseudo Labels où l'enseignant est fixé, l'enseignant dans Meta Pseudo Labels est constamment adapté par le retour des performances de l'étudiant sur l'ensemble de données étiquetées. En conséquence, l'enseignant génère de meilleures pseudo-étiquettes pour enseigner à l'étudiant.

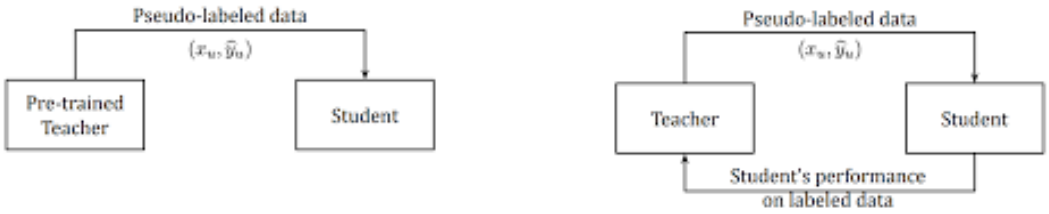


FIGURE 3.11 – La différence entre les pseudo-labels et les méta-pseudo-labels [8]

3.4 Jeux de données

3.4.1 ImageNet

ImageNet est un grand jeu de données de photographies annotées destiné à la recherche en vision par ordinateur [47], Le but du développement du jeu de données était de fournir une ressource pour promouvoir la recherche et le développement de méthodes améliorées pour la vision par ordinateur, il a plus de 14 millions d'images dans le jeu de données, un peu plus de 21 mille groupes ou classes, depuis 2010, le projet ImageNet a lancé le Défi de reconnaissance visuelle à grande échelle l'ImageNet

ou ILSVRC pour faire court, dont les tâches utilisent des sous-ensembles du jeu de données ImageNet [48], ce sous-ensemble est devenu en bref le jeu de données le plus utilisé en vision par ordinateur, il a 1 million d'images pour l'ensemble d' apprentissage, 50000 pour l'ensemble de validation et 150000 pour l'ensemble de test.

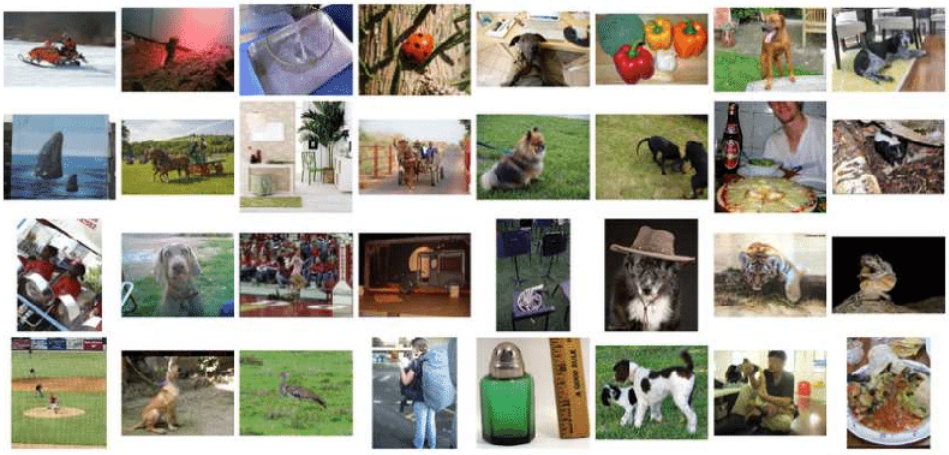


FIGURE 3.12 – Quelques images appartiennent au jeu de données ImageNet

3.4.2 Cifar-10

Le jeu de données CIFAR-10 est une collection d'images couramment utilisées pour l'entraînement d'algorithmes d'apprentissage automatique et de vision par ordinateur [49]. Elle se compose de 60 000 images couleur $32 * 32$ réparties en 10 classes, avec 6000 images par classe. Il y a 50000 images d'entraînement et 10000 images de test, les 10 classes différentes représentent des avions, des voitures, des oiseaux, des chats, des cerfs, des chiens, des grenouilles, des chevaux, des bateaux et des camions.

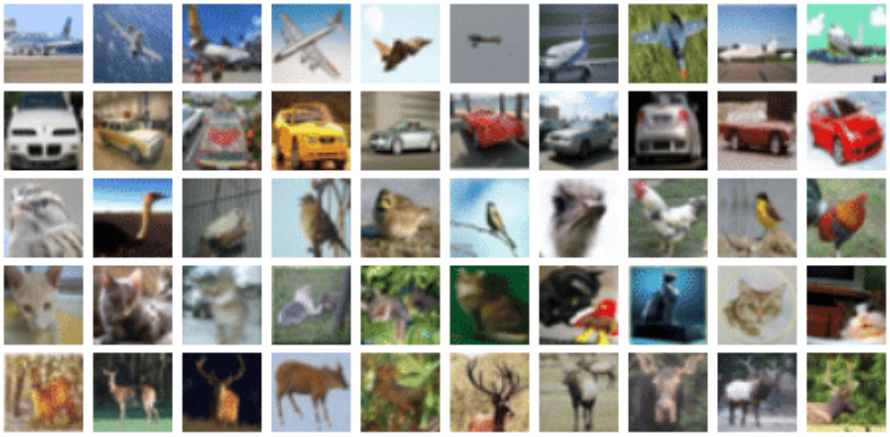


FIGURE 3.13 – Quelques images appartiennent au jeu de données Cifar-10

3.4.3 Cifar-100

Ce jeu de données est identique au CIFAR-10, sauf qu'il comporte 100 classes contenant chacune 600 images. Il y a 500 images d'entraînement et 100 images de

test par classe. Les 100 classes du CIFAR-100 sont regroupées en 20 superclasses. Chaque image est accompagnée d'une étiquette 'fine' (la classe à laquelle elle appartient) et d'une étiquette 'grossière' (la superclasse à laquelle elle appartient), par exemple 'fleurs' est une superclasse pour 'orchidées', 'coquelicots', 'roses', 'tournesols' et 'tulipes'.

3.4.4 JFT-300M

JFT-300M est un jeu de données interne à Google utilisé pour l'entraînement des modèles de classification d'images [45]. Les images sont étiquetées à l'aide d'un algorithme qui utilise un mélange complexe de signaux web bruts, de connexions entre les pages web et de commentaires des utilisateurs. Il en résulte plus d'un milliard d'étiquettes pour les 300 millions d'images (une seule image peut avoir plusieurs étiquettes). Ce jeu de données est couramment utilisé dans les modèles d'pprentissage auto-supervisé comme les pseudo-labels [7] et les méta-pseudo-labels [8].

3.4.5 Kaggle Cats-vs-Dogs

cats-vs-dogs est un jeu de données largement utilisé en vision par ordinateur comprenant essentiellement les deux classes d'objets étudiées : les chiens et les chats. Cet ensemble de données a été utilisé pour un concours d'apprentissage automatique organisé par Kaggle en 2013. Il se compose de 25000 photos taguées : 12500 chiens et le même nombre de chats. Des prédictions ont ensuite été demandées sur un ensemble de données de test de 12500 photographies non étiquetées.

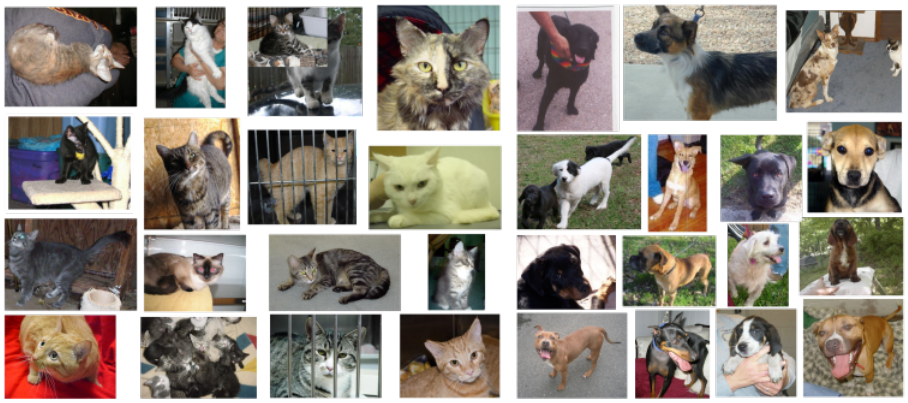


FIGURE 3.14 – Quelques images appartiennent au jeu de données Kaggle Cats-vs-Dogs

3.5 Visualiser ce qu'un CNN Apprend

On dit souvent que les modèles d'apprentissage profond sont des boîtes noires : Bien que cela soit partiellement vrai pour certains types de modèles d'apprentissage profond, ce n'est pas le cas pour les CNNs. Les représentations apprises par les CNNs se prêtent très bien à la visualisation. Un large éventail de techniques a été développé pour visualiser et interpréter ces représentations. Les trois techniques les plus accessibles et les plus utiles sont les suivantes.

3.5.1 Visualisation des activations intermédiaires

La visualisation des activations intermédiaires consiste simplement à afficher les cartes de caractéristiques qui sont produites par les différentes couches de convolution et de pooling d'un réseau.

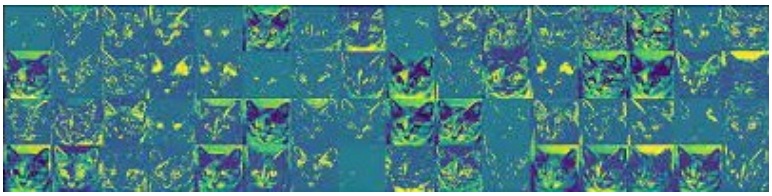


FIGURE 3.15 – Les activations de certaines couches pour une image de chat

3.5.2 Visualisation des Filtres

Une autre façon simple d'inspecter les filtres appris par les CNNs est d'afficher le motif visuel auquel chaque filtre est censé répondre. Ceci peut être fait avec l'ascension de gradient dans l'espace d'entrée : appliquer la descente de gradient à la valeur de l'image d'entrée d'un convnet de manière à maximiser la réponse d'un filtre spécifique [50], en partant d'une image d'entrée vierge. L'image d'entrée résultante sera celle à laquelle le filtre choisi répondra le mieux. Le processus est simple : nous allons construire une fonction de loss qui maximise la valeur d'un filtre donné dans une couche de convolution donnée, puis nous utiliserons la descente de gradient stochastique pour ajuster les valeurs de l'image d'entrée de manière à maximiser cette valeur d'activation.

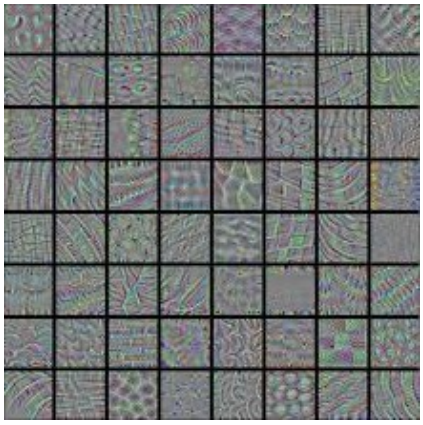
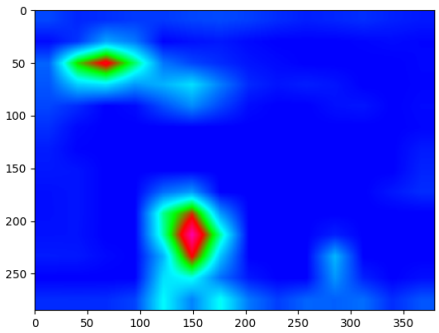


FIGURE 3.16 – Visualisation du quatrième bloc dans la première convolution pour le modèle VGGNet pré-trainé du jeu de données ImageNet

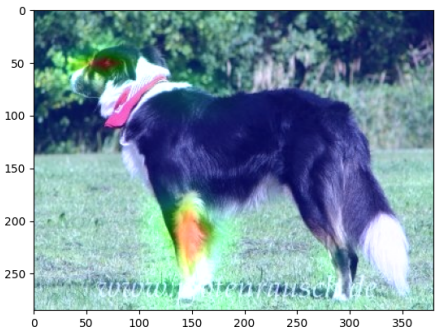
3.5.3 Visualisation de heatmap pour la classe d'activation

Une autre technique utile pour comprendre quelles parties d'une image donnée ont conduit un CNN à sa décision finale de classification est la visualisation de la carte d'activation de classe (CAM), qui est très utile pour déboguer le processus de décision d'un CNN, en particulier dans le cas d'une erreur de classification [51], la façon dont elle fonctionne est très simple, elle consiste à prendre la feature map de

sortie d'une couche de convolution, étant donné une image d'entrée, et de pondérer chaque canal dans cette feature map par le gradient de la classe par rapport au canal.



(a) Visualisation de heatmap pour la classe d'activation chien.



(b) Superposition de la heatmap de la classe d'activation sur l'image originale.

FIGURE 3.17 – Visualisation de heatmap

3.6 Conclusion

Dans ce chapitre, nous avons mentionné comment le CNN fonctionne généralement à partir de l'opération de convolution, l'opération de pooling et l'empilement des couches. Nous avons également vu l'histoire du réseau de neurones convolutifs à partir du petit modèle LeNet jusqu'au modèle géant de méta pseudo labels avec ses 480 millions de paramètres. Nous avons également eu la chance de regarder les jeux de données les plus communs comme imageNet et Cifar, et enfin nous avons vu quelques méthodes intéressantes pour visualiser ce que le CNN apprend. Dans le prochain chapitre, nous développons l'idée derrière la conception et de l'implémentation de notre modèle CNN. Ce dernier sera guidé par un détecteur SIFT pour la reconnaissance de deux classes d'objets (chats et chiens) dans les cas simples (de la base d'images kaggle cats-vs-dogs) et en particulier pour résoudre le problème de l'occultation des objets dans les images (e.g., dans les cas où les objets chats et chiens sont occultés partiellement).

CHAPITRE 4

Modèle CNN guidé SIFT pour la reconnaissance des classes d'objet

Dans ce chapitre, nous allons présenter la méthode proposée pour reconnaître des objets en particulier les objets partiellement occultés sur le jeu de données kaggle Cats-vs-Dogs. Le modèle proposé est construit autour d'un réseau CNN guidé par un détecteur SIFT. Notre approche utilise deux modèles de classification, un classifieur SVM et un modèle CNN a deux entrées. Ainsi, le classifieur SVM distingue entre les points d'intérêt détectés par l'algorithme SIFT ceux appartenant à l'objet ou non. Ensuite, il sélectionne les 10 premiers points appartenant à l'objet ordonnés par leur magnitude, calcule la moyenne de ces points et recadre une région de l'image centrée sur ce point. Le CNN proposé à deux entrées : une entrée pour l'image originale et l'autre pour l'image recadrée. Les deux images sont exposées à des convolutions séparées et concaténées dans les couches entièrement connectées, et complétées comme un seul modèle.

L'idée d'utiliser deux entrées pour le modèle CNN, où la deuxième entrée est une partie de la première, est que le modèle peut apprendre des informations globales à partir de l'image complète et des informations locales à partir de l'image rognée, ces informations locales sont généralement situées là où la densité des points d'intérêt est la plus grande.

4.1 Méthode proposée

Les deux figures suivantes représentent la structure générale de model CNN proposée

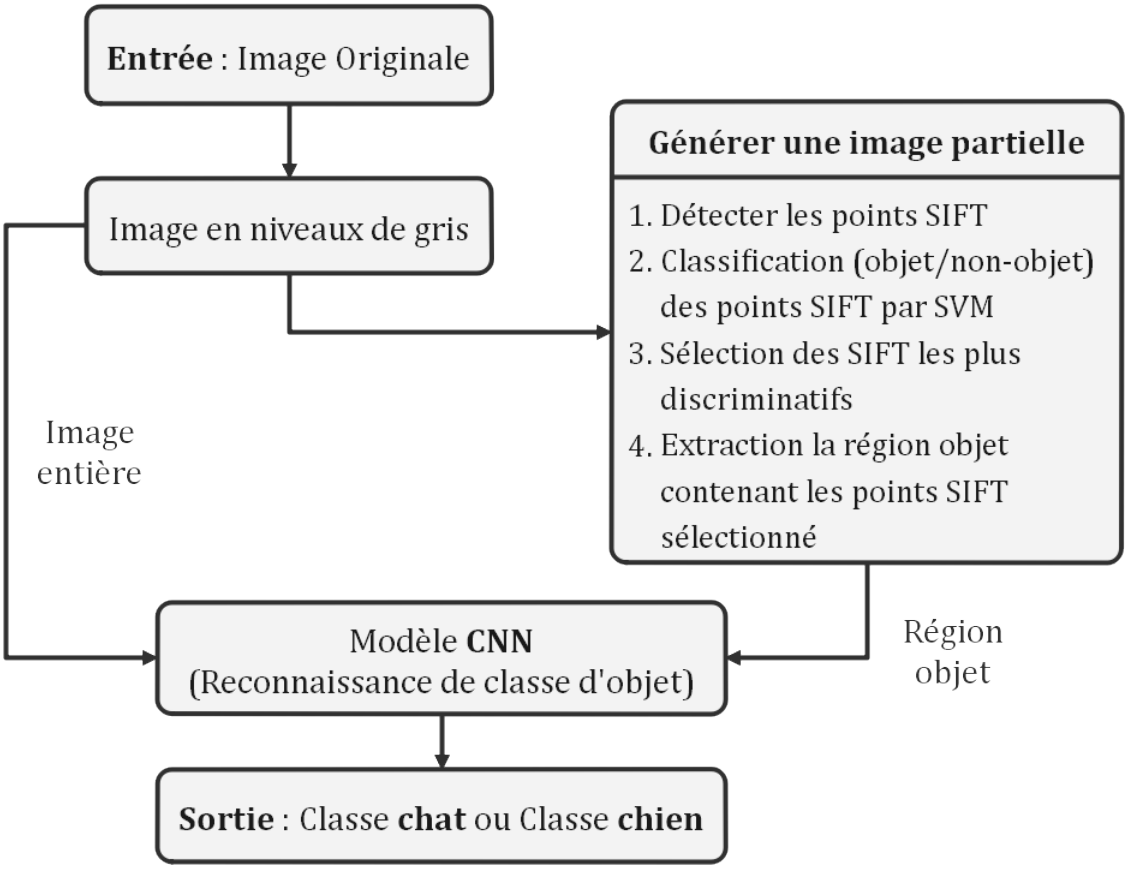


FIGURE 4.1 – Les étapes de la méthode proposée

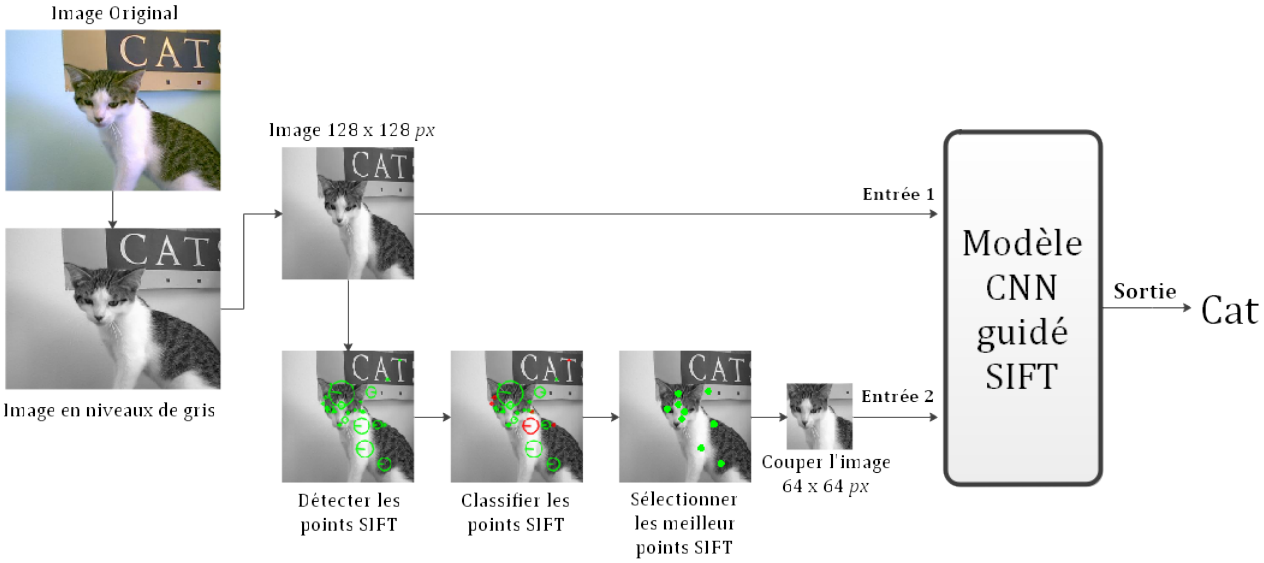


FIGURE 4.2 – Description du fonctionnement de la méthode proposée

4.2 Description de la fonction du Modèle proposé

4.2.1 Prétraitement des données

Comme un prétraitement des données image, nous transformons les images en niveau du gris et nous les redimensionnons en des images de taille fixe $128 * 128$.

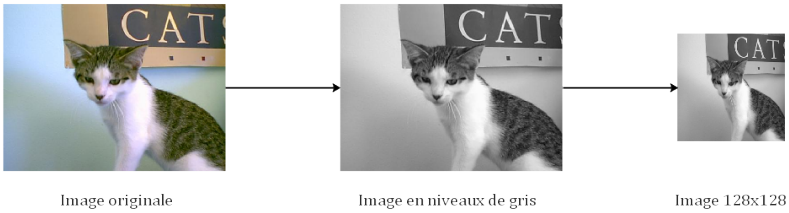


FIGURE 4.3 – Prétraitement des données

4.2.2 Modèle SVM pour la sélection des points objets

Le rôle du modèle SVM consiste à détecter les points d'intérêt et à décider s'ils appartiennent ou non à l'objet. L'étape consistant à reconnaître l'objet dans l'image n'a pas été élaborée par un algorithme de décision ou un classifieur classique car cette étape sera traitée par un réseau profond.

4.2.2.1 Détection des points d'intérêt

Pour détecter les points d'intérêt nous avons utilisé l'algorithme SIFT (Scale-invariant feature transform) [1]. selon l'étude dans le travail [52], SIFT peut être plus performant dans notre cas que les autres détecteurs, car l'algorithme génère des points d'intérêt invariants à l'échelle uniforme, l'orientation, les changements d'illumination, et partiellement invariant à la distorsion affine. Ces points sont représentés par un descripteur de 128 éléments contenant des informations élémentaires sur chacun d'eux.

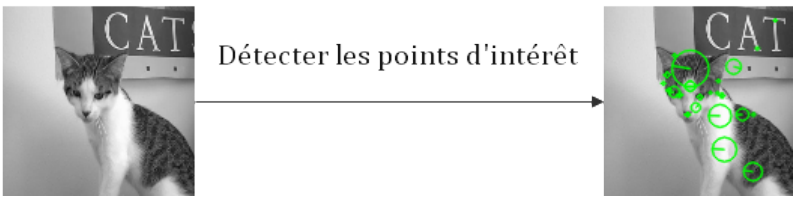


FIGURE 4.4 – Détection des points d'intérêt SIFT

4.2.2.2 Apprentissage

Dans cette étape, nous avons utilisé un classifieur SVM binaire avec le noyau polynomial, la raison pour laquelle nous avons choisi pour cette tâche est qu'il s'agit d'un algorithme d'apprentissage automatique simple et efficace qui peut effectuer une classification binaire ou multiple, et qui n'a pas besoin de beaucoup de données pour le faire.

Nous avons fait le processus d'apprentissage manuellement, d'abord nous avons sélectionné environ 100 images pour l'ensemble d'entraînement, puis pour chaque

image nous avons généré ses points d'intérêt, et nous avons passé par chacun d'eux et décidé si c'est un point objet ou non, ou pas important pour l'apprentissage. Comme nous l'avons mentionné dans la section précédente, chaque point est présenté avec un descripteur de 128 éléments, le reste est de donner les points descripteurs avec leurs étiquettes au SVM pour la classification.

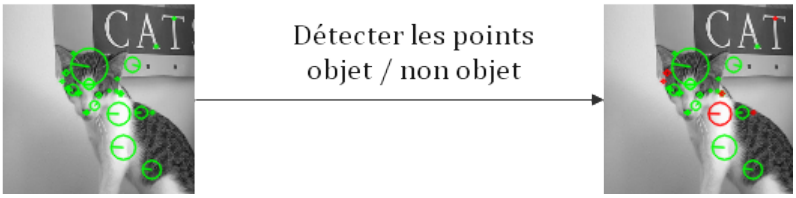


FIGURE 4.5 – Classifier les points SIFT appartient ou non à l'objet

4.2.3 Modèle CNN proposé

Le second modèle est la base de la méthode proposée où il utilise le premier modèle qui extrait la partie de l'image contenant vraisemblablement l'objet détecté. Cette nouvelle image obtenue a partir de la région d'intérêt localisé à travers les caractéristiques SIFT décrivant l'objet. Cette image ainsi que l'image originale alimentent notre modèle CNN à deux entrées.

4.2.3.1 Générer des images partielles

La génération d'images partielles est faite automatiquement en utilisant le premier modèle, dans cette étape nous avons utilisé plusieurs paramètres de point d'intérêt, après avoir généré le point d'intérêt en utilisant l'algorithme SIFT, tout d'abord nous avons utilisé le descripteur pour classer si un point appartient à l'objet ou non en utilisant le premier modèle, puis la magnitude pour choisir les 10 meilleurs points, et enfin leur coordonnées pour trouver le point moyen et recadrer la sous-région $64 * 64$ de l'image centrée sur ce point.

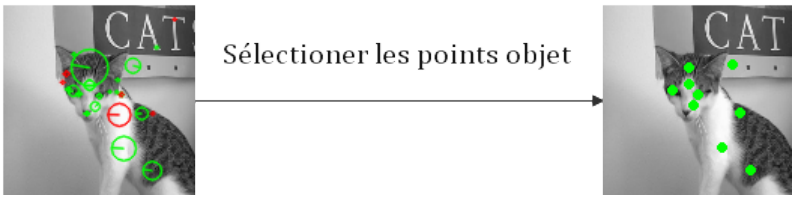


FIGURE 4.6 – Sélection des points SIFT appartient à l'objet

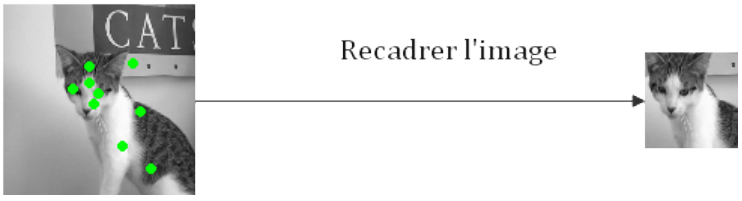


FIGURE 4.7 – Sélection de la région image qui a la plus forte densité de points SIFT appartenant à l'objet

Algorithme 1: Algorithme de sélection ou d'extraction de la région objet

Pour *toute* ***image*** $\in$ *Jeu de Données* **faire**

1. Générer les points SIFT
2. Classifier par le SVM des points SIFT en points objet et non-objet
3. Sélectionnez les 10 *points* d'objet qui ont la plus grande magnitude
4. Calculer le centre $C[x, y]$ de ces 10 *points*
5. Extraire la partie **image** de centre $C[x, y]$ et de taille $64 * 64$ px
6. Stocker la nouvelle image

fin

Résultat: Les nouvelles images $64 * 64$ px

4.2.3.2 Apprentissage

Après avoir généré des images partielles, chaque image et sa partie dans l'ensemble d'entraînement est normalisée entre 0 et 1 et donnée à un CNN fonctionnel avec deux entrées, où elles sont soumises à des opérations de convolution séparables, et finalement elles sont concaténées dans les couches entièrement connectées pour donner une seule sortie.

4.2.3.3 Architecture du Modèle Proposé

L'architecture du modèle est composée de sept couches de convolution, quatre pour l'image originale et trois pour l'image recadrée, et de deux couches entièrement connectées. Nous avons utilisé la normalisation par lot [53] pour accélérer le processus d'apprentissage, le dropout [32] pour éviter le sur-apprentissage. Pour compiler et ajuster le modèle nous avons utilisé un l'optimiseur d'Adam avec un taux d'apprentissage de 0.01, la loss d'entropie croisée binaire, une taille de lot de 24 et un nombre d'époques égale à 30.

- La première entrée est une image en niveaux de gris de $128 * 128$,nous commençons l'opération de convolution avec 32 filtres suivis par un max-pooling de taille $2 * 2$,une fonction d'activation Relu, une normalisation par lot et un dropout de 0.2, les deuxième et troisième couches de convolution sont les mêmes que la première sauf pour le nombre de filtres où nous avons utilisé 64 et 128 filtres respectivement, la quatrième couche est la même que la troisième sauf que nous n'avons utilisé ni la normalisation par lot ni le dropout, la feature-map produite est aplatie et concaténée avec celle produite par la deuxième entrée.
- La deuxième entrée est une image recadrée en niveaux de gris $64 * 64$, dans cette partie du réseau, nous avons utilisé trois couches de convolution en commençant par 32 filtres et en doublant le nombre de filtres après chaque couche, chaque opération de convolution est suivie d'un max-pooling de taille $2 * 2$ et d'une fonction d'activation Relu,la première et la deuxième couche de convolution est suivi aussi avec un dropout de 0.1 et une normalisation par lot, et enfin la feature-map produite par ces trois couches de convolution est aplatie et concaténée avec celle produite par la première entrée.

- Après avoir concaténé les feature-maps, nous avons effectué une normalisation par lot et un dropout de 0.5, puis ajouté la première couche entièrement connectée avec ses 1024 neurones suivis d'une fonction d'activation Relu, d'une normalisation par lot et d'un dropout de 0,5, et enfin ajouté la deuxième et dernière couche entièrement connectée où elle est composée d'un seul neurone qui est la sortie du modèle qui utilise la fonction d'activation sigmoïde.

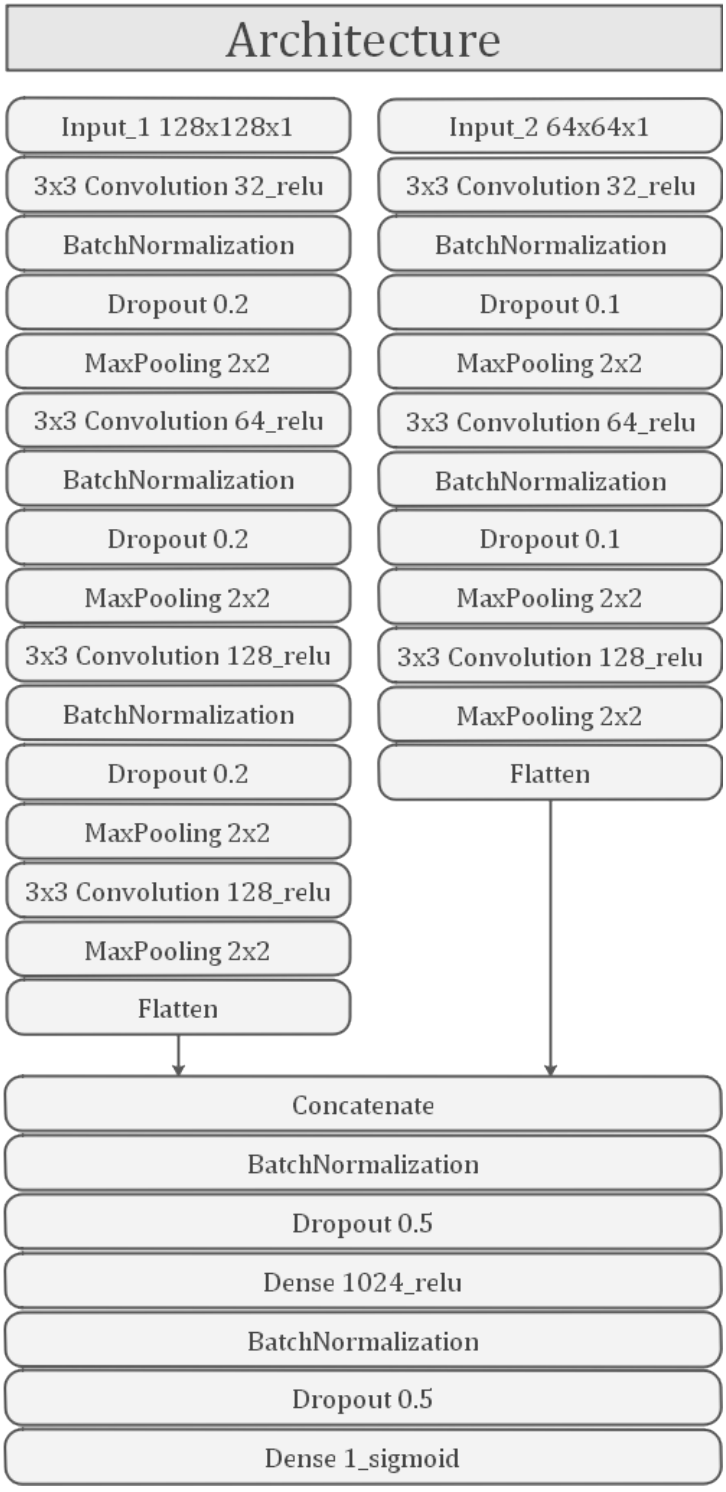


FIGURE 4.8 – L'architecture du modèle proposé

4.3 Modèle proposé vs CNN simple

Pour valider les performances de notre méthode proposée, nous avons développé un CNN simple afin de comparer en termes d'accuracy dans différents cas d'apparition d'objets.

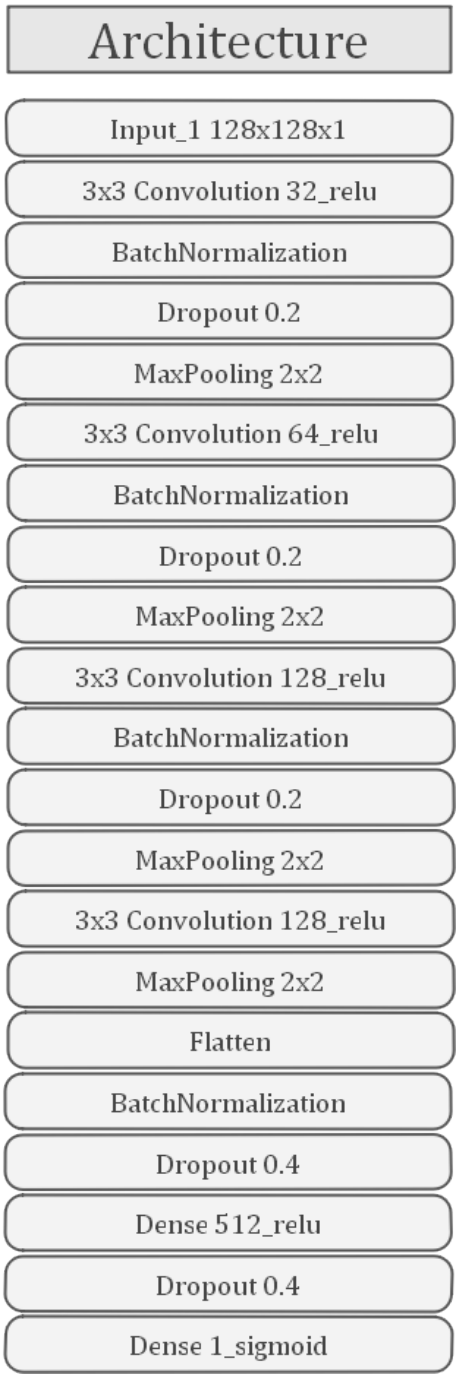


FIGURE 4.9 – L'architecture du simple CNN

L'entrée du modèle est une image en niveaux de gris $128 * 128$, nous commençons l'opération de convolution avec 32 filtres suivis d'un pool max de taille $2 * 2$, fonction d'activation Relu, normalisation batch et dropout de 0.2, les deuxième et troisième

couches de convolution sont les mêmes que la première sauf pour le nombre de filtres où nous avons utilisé respectivement 64 et 128 filtres, la quatrième couche est la même que la troisième sauf que nous n'avons utilisé ni la normalisation batch ni le dropout, la feature map produite est aplatie et suivie d'une normalisation batch et un décrochage de 0.4, puis a ajouté la première couche entièrement connectée avec ses 512 neurones suivi d'une fonction d'activation Relu, une normalisation par lots et un décrochage de 0.4, et enfin ajouté la deuxième et dernière couche entièrement connectée où elle est composée d'un seul neurone c'est la sortie du modèle qui utilise la fonction d'activation sigmoïde.

Les paramètres de compilation et d'ajustement du modèle sont les mêmes que pour la méthode proposée.

4.4 le jeu de données

Pour tester notre modèle, nous avons utilisé le le jeu de données kaggle dogs-vs-cats mentionné dans le chapitre 2.parce que l'ensemble de tests n'est pas disponible pour le téléchargement, nous avons divisé les 25000 photos étiquetées en 16000 pour l'ensemble d'entraînement, 4000 pour l'ensemble de validation et 5000 pour l'ensemble de test. Cet ensemble de données est disponible en ligne et peut être téléchargé à partir d'ici kaggle dogs-vs-cats.

FIGURE 4.10 – Quelques images du jeu de données kaggle cats-vs-dogs

4.5 Reconnaissance des Objets Occultés

C'est la partie la plus importante pour valider l'efficacité du modèle proposé par rapport au modèle CNN simple. Ainsi l'approche que nous suggérons permet un model CNN de traiter les cas d'occultation d'objet dans les images. Pour ce faire, nous avons choisi au hasard 100 images de l'ensemble de données testé, puis nous avons partiellement couvert l'objet manuellement avec une partie de l'arrière-plan de l'image, la figure suivante représente quelques images occultées.

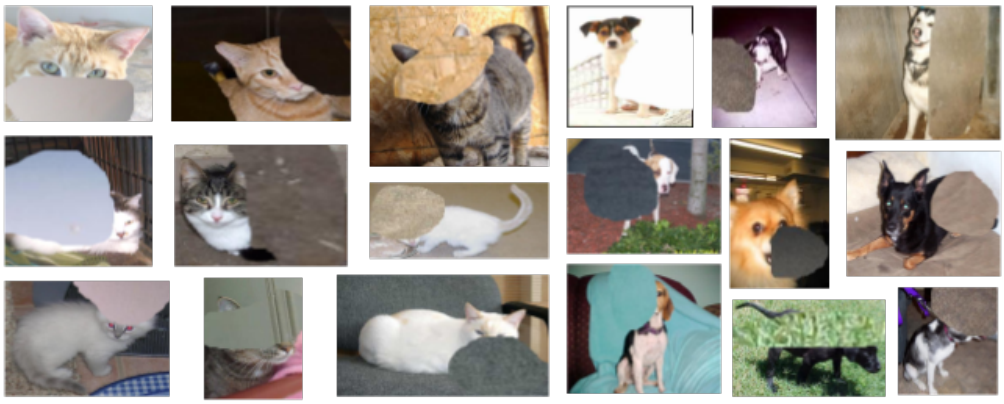


FIGURE 4.11 – Quelques objets partiellement occultés par l'arrière-plan de l'image

4.6 Conclusion

Dans ce chapitre, nous avons discuté l'approche proposée consistant en un modèle CNN guidé par les points SIFT. Nous avons commencé par expliquer la structure générale de notre approche composée de deux modèles de classification : le SVM permettant de sélectionner les points SIFT appartenant à l'objet, et le CNN ayant deux entrées pour la reconnaissance d'objet. Nous avons également détaillé le processus d'apprentissage pour chaque modèle. Nous avons comparé les performances de notre modèle CNN avec un modèle CNN simple (voir figure 4.9). Enfin, nous avons présenté le jeu de données utilisé ainsi que la façon dont nous l'avons divisé en ensemble d'entrainement, de validation et de test. Dans le prochain chapitre, nous verrons les outils que nous avons utilisés pour développer et affiner notre méthode ainsi que les résultats expérimentaux obtenus.

CHAPITRE 5

Expérimentation et Résultats

Dans ce chapitre nous présenterons les outils matériels et logiciels utilisés pour développer la méthode proposée, les interfaces utilisateurs graphiques, des bouts de codes implémentant notre système et surtout l'évaluation des performances du modèle SVM et de notre modèle CNN. Plusieurs expériences ont été menées, ces dernières ont prouvé l'efficacité de notre système basé CNN guidé par les caractéristiques SIFT à reconnaître les classes d'objets, dans le cas de forme globale similaire et en particulier lorsque les objets (e.g., chats et chiens) sont partiellement occultés. Des comparaisons ont été réalisées entre un modèle CNN simple et notre modèle.

5.1 Représentation des outils

5.1.1 Outils logiciels

Heureusement il y a beaucoup d'outils open source disponibles dans ce domaine qui peuvent facilement faciliter le processus de développement de notre application, et voici l'ensemble des outils utilisés pour le faire.

5.1.1.1 Python

Python est un langage de programmation interprété de haut niveau, multi paradigme et multi plateforme, la raison de sa grande renommée dans la société des développeurs est due à sa simplicité de développement d'applications dans différents domaines qui fonctionnent sur différentes plateformes et à la disponibilité d'une bibliothèque open source très bien conçue utilisée pour de multiples sous-domaines de l'informatique.

5.1.1.2 Opencv

Opencv (open source computer vision library) est une bibliothèque de fonctions de programmation destinées à la vision par ordinateur en temps réel, elle a été développée à l'origine par intel en 1999, aujourd'hui sa version stable est 4,5,1, et disponible pour une utilisation gratuite sous la licence open source apache , cette bibliothèque ne fournit pas seulement un code gratuit mais aussi très optimisé, ce qui en fait la bibliothèque la plus couramment utilisée dans les sous-domaines de la vision par

ordinateur tels que la détection et la reconnaissance d'objets, le suivi de mouvement, l'interaction homme-machine. etc.

5.1.1.3 Jupyter Notebook

Jupyter Notebook est un environnement de calcul interactif basé sur le web pour créer des documents Jupyter notebook, le nom de Jupyter est une référence aux trois principaux langages de programmation supportés par Jupyter, qui sont JUlia, PYThon et R, l'un de ses plus grands avantages est de permettre aux utilisateurs de visualiser les résultats du code en ligne sans la dépendance d'autres parties du code où chaque cellule du code peut être potentiellement vérifiée à tout moment pour dessiner une sortie, ce qui fournit un très bon avantage aux développeurs d'apprentissage automatique et d'apprentissage profond où ils ont à charger les données une seule fois contrairement à l'environnement auther où ils doivent les charger à chaque exécution.

5.1.1.4 Google collab

Google Colab est un environnement Jupyter notebook GRATUIT fourni par Google spécialement pour les tâches de Deep Learning. Il fonctionne entièrement dans le nuage et permet le partage de code, la sauvegarde dans google drive directement, et offre des ressources pour la puissance de calcul.

L'un des principaux avantages de Colab est qu'il offre un support GPU gratuit, mais il n'est ni garanti ni illimité, et les limites d'utilisation fluctuent parfois.

5.1.1.5 TensorFlow

TensorFlow est un cadre de bas niveau gratuit et open-source pour l'apprentissage automatique.Le nom TensorFlow dérive des opérations que de tels réseaux neuronaux effectuent sur des tableaux de données multidimensionnels, qui sont appelés tenseurs, il peut être utilisé à travers une gamme de tâches, mais a un accent particulier sur la formation et l'inférence des réseaux neuronaux profonds, Il a été développé et open source par Google et prend en charge le déploiement sur les CPU, GPU, et les appareils mobiles et périphériques ainsi. Il a été publié en novembre 2015 et a connu une augmentation considérable de son adoption dans l'industrie.

5.1.1.6 Keras

Keras est une API de réseaux neuronaux de haut niveau écrite en Python. Jusqu'à la version 2.3, Keras prenait en charge plusieurs backends, dont TensorFlow, Microsoft Cognitive Toolkit, Theano et PlaidML. Depuis la version 2.4, seul TensorFlow est pris en charge. Conçu pour permettre une expérimentation rapide des réseaux neuronaux profonds, Keras s'attache à être convivial, modulaire et extensible. Il a été développé dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), et son principal auteur et mainteneur est François Chollet.

Keras contient de nombreuses implémentations de blocs de construction de réseaux neuronaux couramment utilisés, tels que des couches, des objectifs, des fonctions d'activation, des optimiseurs, ainsi qu'un grand nombre d'outils permettant de travailler

plus facilement avec des données d'image et de texte afin de simplifier le codage nécessaire à l'écriture du code de réseaux neuronaux profonds. Il prend également en charge les réseaux neuronaux convolutifs et récurrents. Il prend en charge d'autres couches utilitaires courantes comme le dropout, la normalisation par lot et le pooling.

5.1.1.7 PyQt5

Qt est un ensemble de bibliothèques C++ multiplateformes qui mettent en œuvre des API de haut niveau pour accéder à de nombreux aspects des systèmes modernes de bureau et mobiles, PyQt5 est un ensemble complet de liaisons Python pour Qt v5. Il est mis en œuvre sous la forme de plus de 35 modules d'extension et permet d'utiliser Python comme langage de développement d'applications alternatif au C++ sur toutes les plates-formes prises en charge, y compris iOS et Android.

5.1.2 Matériel informatique

5.1.2.1 Machine virtuelle Google Colab

Pour l'apprentissage du modèle CNN, nous avons utilisé Google Colab avec cette configuration :

- ◇ **CPU :** Intel(R) Xeon(R) CPU @ 2.20GHz, (1 Core, 2 Threads).
- ◇ **GPU :** NVIDIA Tesla T4, 2560 CUDA Cores , 16GB GDDR6 VRAM.
- ◇ **RAM :** 12.6 Go Disponible.
- ◇ **Disk :** 38 Go Disponible.
- ◇ **OS :** Linux.

5.1.2.2 Matériel local

Pour l'apprentissage de modèle classiques SVM et la reconnaissance, nous avons utilisé un ordinateur avec cette configuration :

- ◇ **CPU :** Intel(R) Core(R) CPU i5-6300U @ 2.40GHz, (2 Core, 4 Threads).
- ◇ **RAM :** 8 Go.
- ◇ **Disk :** 256 Go.
- ◇ **OS :** Windows 10.

5.2 Interfaces Graphiques

Dans notre projet, nous avons utilisé PyQt5 à cause de sa facilité de créer des applications avec interface graphique.
Notre application compose les fenêtres suivantes :

Fenêtre principale

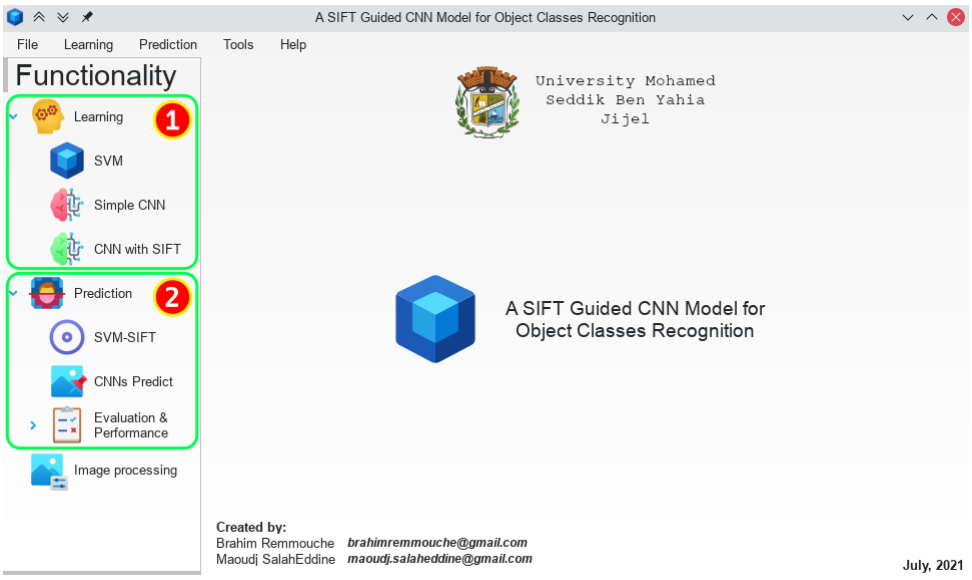


FIGURE 5.1 – Fenêtre principale

La fenêtre principale contient deux fonctionnalités :

1. Outils d'apprentissage des modèles.
2. Outils de prediction et évaluation des modèles.

Fenêtre de description SIFT

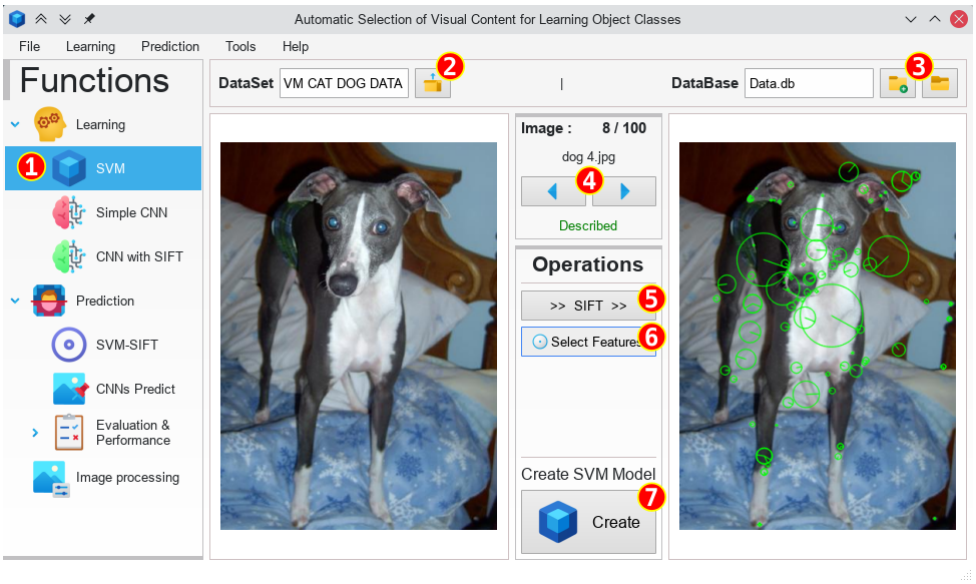


FIGURE 5.2 – Fenêtre de description SIFT

La fenêtre de description SIFT contient les fonctionnalités suivantes :

1. Un bouton pour ouvrir la fenêtre de prediction SIFT.
2. Un bouton pour ouvrir le dossier des jeux de données.
3. Un bouton pour ouvrir ou créer nouvelle base de données pour sauvegarder les descripteurs SIFT.
4. Des boutons pour la navigation entre les images du jeu de donées.
5. Un bouton pour appliquer SIFT sur l'image courante.
6. Un bouton pour ouvrir la fenêtre de création du descripteur d'image.
7. Un bouton pour créer un modèle SVM à partir de la base de données contenant les descripteurs SIFT.

Fenêtre de classification des points SIFT

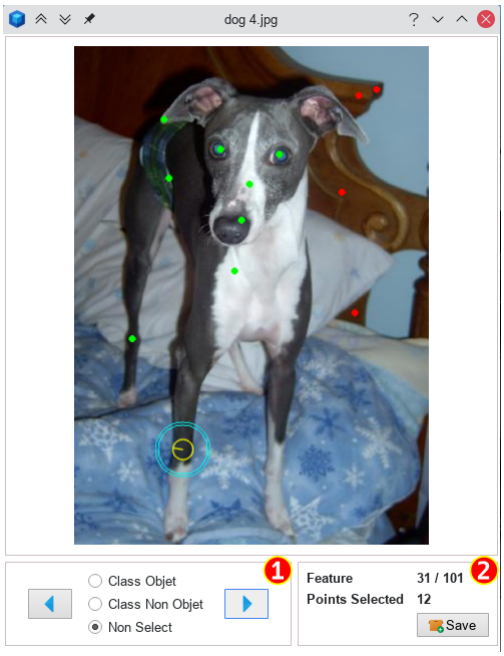


FIGURE 5.3 – Fenêtre de classification des points SIFT

La fenêtre de classification des points SIFT contient les fonctionnalités suivantes :

1. Partie navigation et classification des points SIFT.
2. Enregistrer les points sélectionnés dans la base de données.

Fenêtres d'apprentissage du modèle CNN simple et du modèle CNN proposé

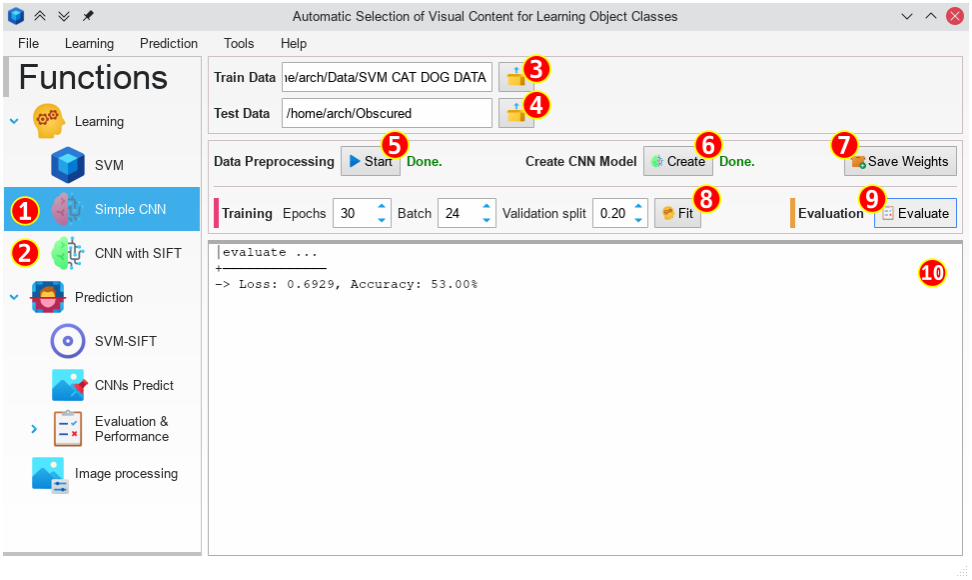


FIGURE 5.4 – Fenêtre d'apprentissage de modèle CNN simple

Fenêtres d'apprentissage du modèle CNN simple et du modèle CNN proposé contiennent les principales fonctionnalités suivantes :

1. Bouton pour ouvrir la fenêtre d'apprentissage du modèle CNN simple.
2. Bouton pour ouvrir la fenêtre d'apprentissage du modèle CNN proposé.
3. Bouton pour ouvrir le dossier du jeu de données d'entraînement.
4. Bouton pour ouvrir le dossier du jeu de données de test.
5. Bouton de prétraitement des données.
6. Bouton pour créer le modèle CNN.
7. Bouton pour enregistrer le modèle CNN sur le disque dur.
8. Bouton pour l'entraînement du modèle.
9. Bouton pour l'évaluation du modèle.
10. Panneau pour afficher les résultats.

Fenêtre de prédiction et d'évaluation du modèle SVM

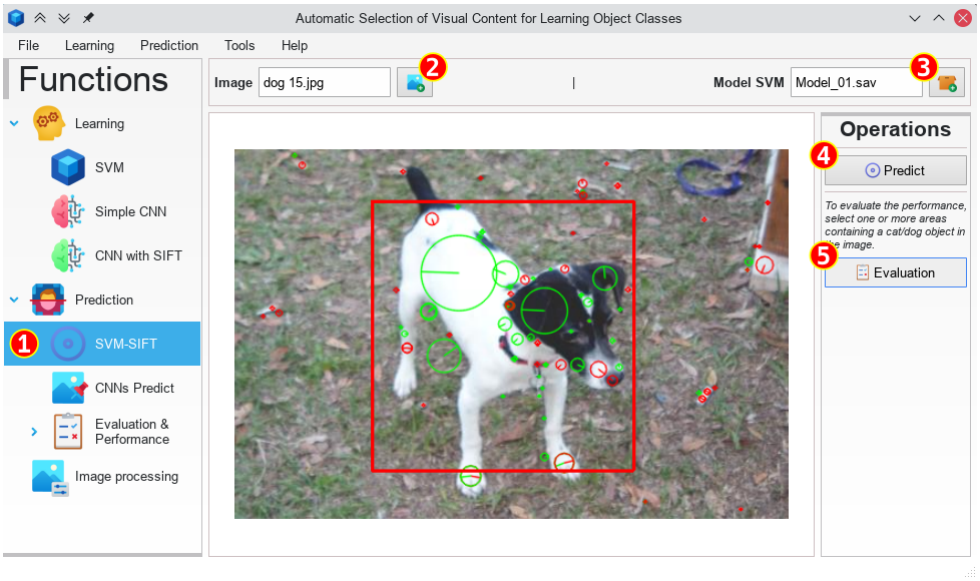


FIGURE 5.5 – Fenêtre de prédiction et d'évaluation du modèle SVM

La fenêtre de prédiction et d'évaluation du modèle SVM contient les fonctionnalités suivantes :

1. Bouton pour ouvrir la fenêtre de prédiction et d'évaluation du modèle SVM.
2. Bouton pour ouvrir l'image.
3. Bouton pour ouvrir le modèle SVM.
4. Bouton de prédiction des points SIFT (Objet / Non-objet).
5. Bouton pour évaluer la classification du modèle.

Fenêtre de classification

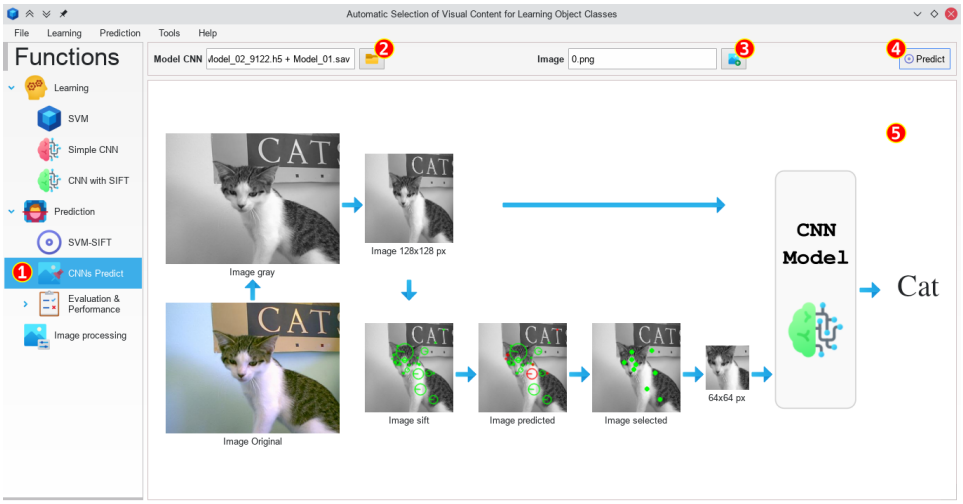


FIGURE 5.6 – Fenêtre de classification

La fenêtre de classification contient les fonctionnalités suivantes :

1. Bouton pour ouvrir la fenêtre de classification.
2. Bouton pour ouvrir le modèle CNN.
3. Bouton pour ouvrir l'image.
4. Bouton pour lancer la prédiction.
5. Panneau de résultats.

Fenêtre d'évaluation de la prédiction des images par les deux modèles CNN

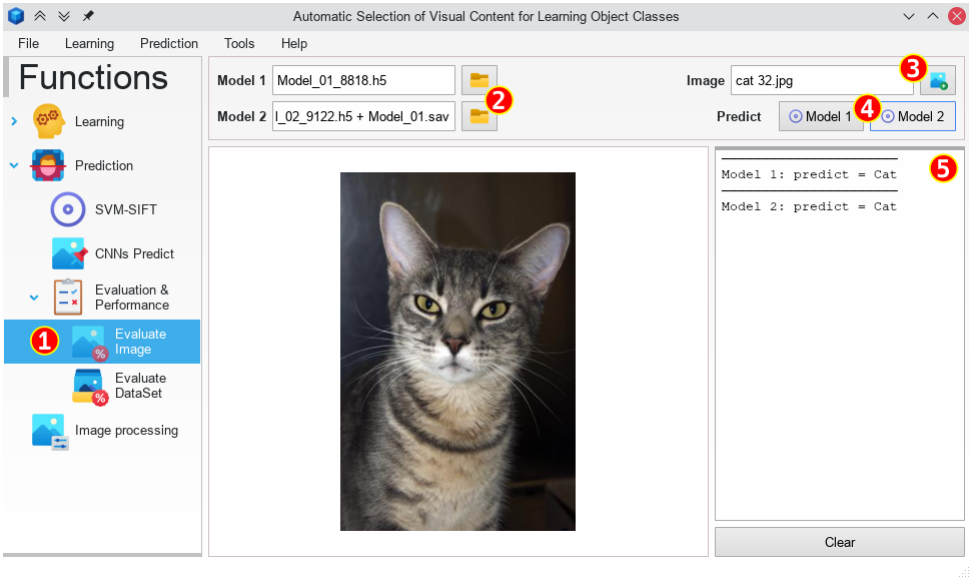


FIGURE 5.7 – Fenêtre d'évaluation de la prédiction des images par les deux modèles CNN

La fenêtre d'évaluation de la prédiction des images par les deux modèles CNN contient les fonctionnalités suivantes :

1. Bouton pour ouvrir cette fenêtre.
2. Boutons pour ouvrir les deux modèles CNN.
3. Bouton pour ouvrir l'image.
4. Boutons de prédiction.
5. Panneau de résultats de prédiction.

Fenêtre d'évaluation des performances des deux modèles CNN

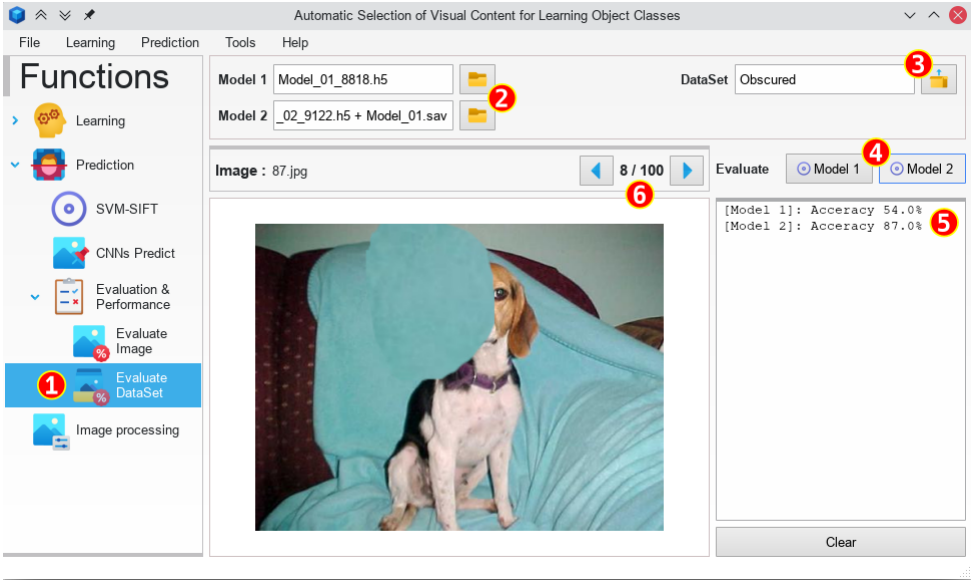


FIGURE 5.8 – Fenêtre d'évaluation des performances des deux modèles CNN

La fenêtre d'évaluation des performances des deux modèles CNN contient les fonctionnalités suivantes :

1. Bouton pour ouvrir cette fenêtre.
2. Boutons pour ouvrir les deux modèles.
3. Bouton pour ouvrir le dossier du jeu de données.
4. Bouton d'évaluation le jeu de données par chaque modèle.
5. Panneau pour afficher les résultats de l'évaluation.
6. Naviguer entre les images du jeu de données.

Fenêtre de traitement d'image

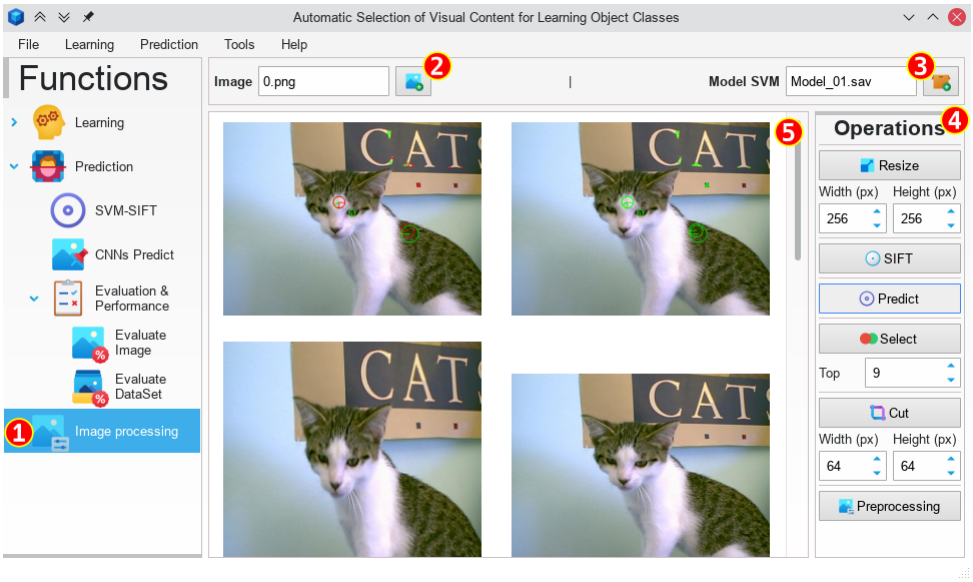


FIGURE 5.9 – Fenêtre de traitement d'image

La fenêtre de traitement d'image contient les fonctionnalités suivantes :

1. Bouton pour ouvrir la fenêtre de traitement d'image.
2. Bouton pour ouvrir l'image.
3. Bouton pour ouvrir le modèle SVM.
4. Panneau des opérations.
5. Panneau de résultat de l'opération.

5.3 Implementation

5.3.1 Code source de pré-traitement des images

cut_image Cette méthode permet à découper une partie de l'image à partir de points SIFT de plus grandes magnitudes.
Paramètres **image** : une image, **N** : nombre des points SIFT, **shape** : les dimensions de nouvelle image, **clf** : modèle SVM.
Retourner une image de dimension **"shape"**.

```
def cut_image(image, N, shape, clf):
    image = cv.resize(image, (128, 128))
    _x, _y = shape
    _, kp, des = SIFT(image, features=max(20, N * 3))
    y = clf.predict(des)
    points = select_points(y, kp, N)
    return image_by_point(image, calc_center(points), _x, _y)
```

FIGURE 5.10 – Le code source de la méthode *cut_image*

image_by_point Cette méthode permet de couper une partie de l'image à partir un point central.
Paramètres **img** : une image, **point** : point central, **height** : hauteur du nouvelle image, **width** : largeur du nouvelle image.
Retourner une image de dimension $width * height$.

```
def image_by_point(img, point, height, width):
    x, y = point
    h, w = img.shape[:2]
    x1 = x - (height // 2)
    y1 = y - (width // 2)
    x2 = x + (height // 2)
    y2 = y + (width // 2)
    if x1 < 0:
        x2 += -x1
        x1 = 0
    if x2 >= h:
        x1 -= x2 - h - 1
        x2 = h - 1
    if y1 < 0:
        y2 += -y1
        y1 = 0
    if y2 >= w:
        y1 -= y2 - w - 1
        y2 = w - 1
    return cv.resize(img[y1:y2, x1:x2], (height, width))
```

FIGURE 5.11 – Le code source de la méthode *image_by_point*

calc_center Cette méthode calcule le centre d'un ensemble de points.
Paramètres **points** : liste de points.
Retourner un point central (x, y).

```
def calc_center(points):
    P = np.array(points)
    x = np.mean(P[:, 0])
    y = np.mean(P[:, 1])
    return int(x), int(y)
```

FIGURE 5.12 – Le code source de la méthode *calc_center*

select_points Cette méthode permet de sélectionner les points SIFT de plus grandes magnitudes.
Paramètres **Y** : Liste de classe des points (Objet/Non-objet), **KP** : liste des points SIFT, **n** : nombre de points que nous voulons sélectionner.
Retourner une liste de points.

```python
def select_points(Y, KP, n):
    _KP = []
    for y, kp in zip(Y, KP):
        if y == 1:
            _KP.append(kp)
    _KP = sorted(_KP, key=lambda x: x.size, reverse=True)
    points = list()
    i = 0
    for kp in _KP:
        i += 1
        if (int(kp.pt[0]), int(kp.pt[1])) not in points:
            points.append((int(kp.pt[0]), int(kp.pt[1])))
        if len(points) == n:
            break
    return points
```

FIGURE 5.13 – Le code source de la méthode *select_points*

5.3.2 Code source de modèle CNN simple

```
model1 = Sequential()

model1.add(Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 1)))
model1.add(BatchNormalization())
model1.add(Dropout(0.2))
model1.add(MaxPooling2D(pool_size=(2,2)))

model1.add(Conv2D(64, (3,3), activation='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.2))
model1.add(MaxPooling2D(pool_size=(2,2)))

model1.add(Conv2D(128, (3,3), activation='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.2))
model1.add(MaxPooling2D(pool_size=(2,2)))

model1.add(Conv2D(128, (3,3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2,2)))

model1.add(Flatten())
model1.add(BatchNormalization())
model1.add(Dropout(0.4))
model1.add(Dense(512, activation='relu'))
model1.add(Dropout(0.4))
model1.add(Dense(1, activation='sigmoid'))

model1.compile(loss=tf.losses.binary_crossentropy,
               optimizer="adam", metrics=['accuracy'])
```

FIGURE 5.14 – Code source du modèle CNN simple

5.3.3 Code source de modèle CNN proposé

Le code contient 3 parties :
Réseau 1 : prendre en entrée une image $128 * 128$ px. (Figure 5.15)
Réseau 2 : prendre en entrée une image $64 * 64$ px. (Figure 5.16)
Concaténation des réseaux : MLP (Concaténation des deux réseaux). (Figure 5.17)

```
# first input model
input1 = Input(shape=(128, 128, 1))

conv2d_1_1 = Conv2D(32, (3,3), activation='relu')(input1)
batch_1_1 = BatchNormalization()(conv2d_1_1)
drop_1_1 = Dropout(0.2)(batch_1_1)
maxpol_1_1 = MaxPooling2D(pool_size=(2,2))(drop_1_1)

conv2d_1_2 = Conv2D(64, (3,3), activation='relu')(maxpol_1_1)
batch_1_2 = BatchNormalization()(conv2d_1_2)
drop_1_2 = Dropout(0.2)(batch_1_2)
maxpol_1_2 = MaxPooling2D(pool_size=(2,2))(drop_1_2)

conv2d_1_3 = Conv2D(128, (3,3), activation='relu')(maxpol_1_2)
batch_1_3 = BatchNormalization()(conv2d_1_3)
drop_1_3 = Dropout(0.2)(batch_1_3)
maxpol_1_3 = MaxPooling2D(pool_size=(2,2))(drop_1_3)

conv2d_1_4 = Conv2D(128, (3,3), activation='relu')(maxpol_1_3)
maxpol_1_4 = MaxPooling2D(pool_size=(2,2))(conv2d_1_4)

flat1 = Flatten()(maxpol_1_4)
```

FIGURE 5.15 – Code source du modèle CNN proposé : Réseau 1

```
# second input model
input2 = Input(shape=(64, 64, 1))

conv2d_2_1 = Conv2D(32, (3,3), activation='relu')(input2)
batch_2_1 = BatchNormalization()(conv2d_2_1)
drop_2_1 = Dropout(0.1)(batch_2_1)
maxpol_2_1 = MaxPooling2D(pool_size=(2,2))(drop_2_1)

conv2d_2_2 = Conv2D(64, (3,3), activation='relu')(maxpol_2_1)
batch_2_2 = BatchNormalization()(conv2d_2_2)
drop_2_2 = Dropout(0.1)(batch_2_2)
maxpol_2_2 = MaxPooling2D(pool_size=(2,2))(drop_2_2)

conv2d_2_3 = Conv2D(128, (3,3), activation='relu')(maxpol_2_2)
maxpol_2_3 = MaxPooling2D(pool_size=(2,2))(conv2d_2_3)

flat2 = Flatten()(maxpol_2_3)
```

FIGURE 5.16 – Code source du modèle CNN proposé : Réseau 2

```
# merge input models
merge = concatenate([flat1, flat2])

batch_0_1 = BatchNormalization()(merge)
drop_0_1 = Dropout(0.5)(batch_0_1)
dens_0_1 = Dense(1024, activation='relu')(drop_0_1)
batch_0_2 = BatchNormalization()(dens_0_1)
drop_0_2 = Dropout(0.5)(batch_0_2)
output = Dense(1, activation='sigmoid')(drop_0_2)

model2 = Model(inputs=[input1, input2], outputs=output)

model2.compile(loss=tf.losses.binary_crossentropy,
               optimizer="adam", metrics=['accuracy'])
```

FIGURE 5.17 – Code source du modèle CNN proposé : Concaténation des réseaux

5.4 Expérimentation et résultats

5.4.1 Évaluation de performance de modèle SVM

Dans cette section, nous allons évaluer la performance du modèle SVM en termes d'exactitude (accuracy), de précision, de rappel et de score F1. Pour ce faire, nous avons choisi au hasard 50 images que nous avons fournies au modèle. Ce dernier

génère pour chaque image 20 points SIFT, et cette procédure nous donne la matrice de confusion suivante.

	Predicter appartient à l'objet	Predicter n'appartient pas à l'objet
Appartient à l'objet	582	116
N'appartient pas à l'objet	89	243

TABLE 5.1 – La matrice de confusion pour l'évaluation de la performance de l'SVM

Nous indiquerons par N le nombre total de points SIFT, dans ce cas $N = 1000$, Nous utiliserons la terminologie suivante

- Vrais positifs (tp) : points qui ont prédit qu'ils appartenaient à l'objet, et qui y appartiennent réellement, et dans ce cas, ils sont égaux à 582.
- Vrais négatifs (tn) : points qui ont prédit que l'objet n'appartenait pas à l'objet, et qu'il n'y appartenait vraiment pas, et dans ce cas il est égal à 243.
- Faux positifs (fp) : points qui ont été prédits comme appartenant à l'objet , et qui ne le sont pas, et dans ce cas, ils sont égaux à 89.
- Faux négatifs (fn) : points pour lesquels l'objet prédit n'appartient pas à l'objet, alors qu'il y appartient, et dans ce cas il est égal à 116.

5.4.1.1 Evaluation en terme d'accuracy

La métrique la plus couramment utilisée pour tester la performance d'un modèle est l'accuracy [55], mais ce n'est pas toujours le meilleur choix, surtout dans le cas de données déséquilibrées, la précision d'un modèle est donnée par les points classés correctement sur le nombre total de points, et est donnée par la formule suivante

$$accuracy = \frac{tp + tn}{N} = \frac{825}{1000} = 82.5\% \tag{5.1}$$

5.4.1.2 Evaluation en terme de Precision

La précision d'un modèle est donnée par la proportion de points prédits correctement appartenant à l'objet par rapport à l'ensemble des points prédits appartenant à l'objet [57], et son donneur dans la formule suivante

$$Precision = \frac{tp}{tp + fp} = \frac{582}{671} = 86.7\% \tag{5.2}$$

5.4.1.3 Evaluation en terme de Rappel

Le rappel d'un modèle est donné par la proportion de points prédits comme appartenant correctement à l'objet par rapport à tous les points appartenant réellement à l'objet [57], et son donneur dans la formule suivante

$$rappel = \frac{tp}{tp + fn} = \frac{582}{698} = 83.3\% \tag{5.3}$$

5.4.1.4 Évaluation en termes de Score F1

Le score F1 est généralement utilisé lorsque l'on ne veut pas maximiser la précision ou le rappel seuls, mais que l'on veut trouver le meilleur équilibre entre les deux [56], et son donneur dans la formule suivante

$$F1 = \frac{2 * precision * rappel}{precision + rappel} = 84.96 \tag{5.4}$$

Comme nous pouvons le voir dans les résultats ci-dessus, le classifieur SVM fait un très bon travail dans la détection du point d'intérêt de l'objet avec un pourcentage élevé dans l'accuracy, le rappel, la précision et le score F1.

FIGURE 5.18 – Quelques images avec une bonne classification

Bien que le modèle ait montré de bons résultats dans différentes métriques, il y a quelques cas où il échoue dans la plupart des classifications de points SIFT. ces cas sont généralement lorsque l'objet (chat ou chien) est très petit ou qu'il y a d'autres objets à côté de lui.

FIGURE 5.19 – Quelques images avec une mauvaise classification

5.4.2 Comparaison du Modèle proposé avec un simple CNN

5.4.2.1 Apprentissage des modèles

Les deux modèles sont entraînés sur le jeu de données mentionné dans le chapitre précédent où nous avons divisé les données en 16000 pour l'entraînement et 4000 pour la validation, la figure suivante représente l'évolution d' accuracy et de loss dans l'entraînement et la validation pour les deux modèles.

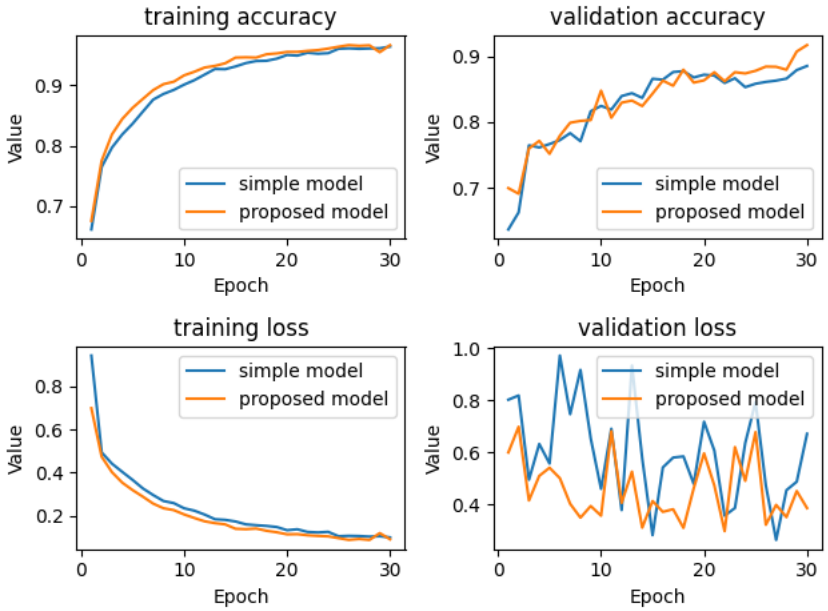


FIGURE 5.20 – L'évolution d'accuracy et de loss dans l'entraînement et la validation des deux modèles pour les 30 epoches

Comme nous le voyons dans les courbes ci-dessus, les modèles s'en sortent bien du côté de la sur-apprentissage où la valeur de l' accuracy de validation pour les deux modèles est proche de l' accuracy d'entraînement et augmente régulièrement, les deux figures suivantes représentent les accuracies et les losses d'entraînement et de validation en chiffres pour les 5 dernières des 30 époques pour les deux modèles.

```
Epoch 25/30
666/666 [==============================] - 21s 31ms/step - loss: 0.1045 - accuracy: 0.9599 - val_loss: 0.3591 - val_accuracy: 0.8850
Epoch 26/30
666/666 [==============================] - 21s 31ms/step - loss: 0.0936 - accuracy: 0.9617 - val_loss: 0.8354 - val_accuracy: 0.8054
Epoch 27/30
666/666 [==============================] - 21s 31ms/step - loss: 0.0930 - accuracy: 0.9635 - val_loss: 0.3712 - val_accuracy: 0.8778
Epoch 28/30
666/666 [==============================] - 21s 31ms/step - loss: 0.1023 - accuracy: 0.9610 - val_loss: 0.5279 - val_accuracy: 0.8858
Epoch 29/30
666/666 [==============================] - 21s 31ms/step - loss: 0.0836 - accuracy: 0.9665 - val_loss: 0.3023 - val_accuracy: 0.9076
Epoch 30/30
666/666 [==============================] - 21s 31ms/step - loss: 0.0729 - accuracy: 0.9740 - val_loss: 0.2454 - val_accuracy: 0.9118
```

FIGURE 5.21 – Les valeurs de loss et d'accuracy pour l'entraînement et la validation du modèle proposé dans les dernières 5 epoches

```
Epoch 25/30
666/666 [==============================] - 16s 24ms/step - loss: 0.1140 - accuracy: 0.9587 - val_loss: 0.5343 - val_accuracy: 0.8209
Epoch 26/30
666/666 [==============================] - 16s 24ms/step - loss: 0.1163 - accuracy: 0.9582 - val_loss: 0.6192 - val_accuracy: 0.8798
Epoch 27/30
666/666 [==============================] - 16s 24ms/step - loss: 0.1023 - accuracy: 0.9624 - val_loss: 0.7389 - val_accuracy: 0.8788
Epoch 28/30
666/666 [==============================] - 16s 24ms/step - loss: 0.1071 - accuracy: 0.9612 - val_loss: 0.6089 - val_accuracy: 0.8552
Epoch 29/30
666/666 [==============================] - 16s 24ms/step - loss: 0.0985 - accuracy: 0.9640 - val_loss: 0.4943 - val_accuracy: 0.8898
Epoch 30/30
666/666 [==============================] - 16s 24ms/step - loss: 0.0888 - accuracy: 0.9667 - val_loss: 0.3706 - val_accuracy: 0.8888
```

FIGURE 5.22 – Les valeurs de loss et d'accuracy pour l'entraînement et la validation du modèle simple dans les dernières 5 epoches

Comme nous le voyons dans les figures ci-dessus, notre modèle surpasse le modèle CNN dans l' accuracy de l'entraînement et la validation avec des résultats remarquables.

5.4.2.2 Cas des Objets non-Occultés

Pour tester la performance des modèles sur des données qu'ils n'ont jamais vues auparavant, nous avons réservé 5000 images pour l'ensemble de tests et les avons transmises aux modèles, les deux figures suivantes représentent l'accuracy et la loss en chiffres pour les modèles.

```
[ ]  1 loss , acc = model.evaluate([X_test], Y_test)
     2 print("model accuracy is : {:.2f} %".format(acc*100))

156/156 [==============================] - 31s 10ms/step - loss: 1.4125 - accuracy: 0.8818
model accuracy is : 88.18 %
```

FIGURE 5.23 – Les valeurs de loss et accuracy pour le TestSet du modèle simple

```
[ ]  1 loss , acc = model.evaluate([X_test, X_test2], Y_test)
     2 print("model accuracy is : {:.2f} %".format(acc*100))

156/156 [==============================] - 2s 11ms/step - loss: 0.2400 - accuracy: 0.9218
model accuracy is : 92.18 %
```

FIGURE 5.24 – Les valeurs de loss et accuracy pour le TestSet du modèle proposé

Comme nous le voyons, notre modèle surpasse le modèle CNN également dans la précision de l'ensemble de tests où il atteint 92% alors que l'autre modèle n'atteint que 88%, la figure suivante représente quelques images reconnues correctement par notre modèle alors que le modèle CNN échoue à le faire.

FIGURE 5.25 – Quelques images où notre modèle reconnaît correctement les objets et où le simple CNN échou

5.4.2.3 Cas d'Occultation Partielle des Objets

Les objets dans la réalité sont plus susceptibles d'apparaître partiellement dans la nature, ce qui rend l'opération de reconnaissance d'objet un peu difficile, nous prétendons que notre modèle peut être efficace dans ce cas parce que nous croyons que le point SIFT peut guider le modèle à la partie non occulter, donc nous avons alimenté les deux modèles avec 100 images avec des objets occulter.

```
[ ]  1 loss , acc = model1.evaluate(X, Y)
     2 print("model accuracy is : {:.2f} %".format(acc*100))

4/4 [==============================] - 30s 44ms/step - loss: 12.3352 - accuracy: 0.5400
model accuracy is : 54.00 %
```

FIGURE 5.26 – Les valeurs de loss et accuracy pour le TestSet du modèle simple dans le cas des objets occultés

```
[ ]  1 loss , acc = model2.evaluate([X, X2], Y)
     2 print("model accuracy is : {:.2f} %".format(acc*100))

4/4 [==============================] - 0s 33ms/step - loss: 0.3339 - accuracy: 0.8700
model accuracy is : 87.00 %
```

FIGURE 5.27 – Les valeurs de loss et accuracy pour le TestSet du modèle proposé dans le cas des objets occultés

Comme nous le voyons, notre modèle surpasse le modèle CNN même en cas d'objets partiellement occultés, il atteint une précision de 87% là où le CNN simple n'atteint que 54%.

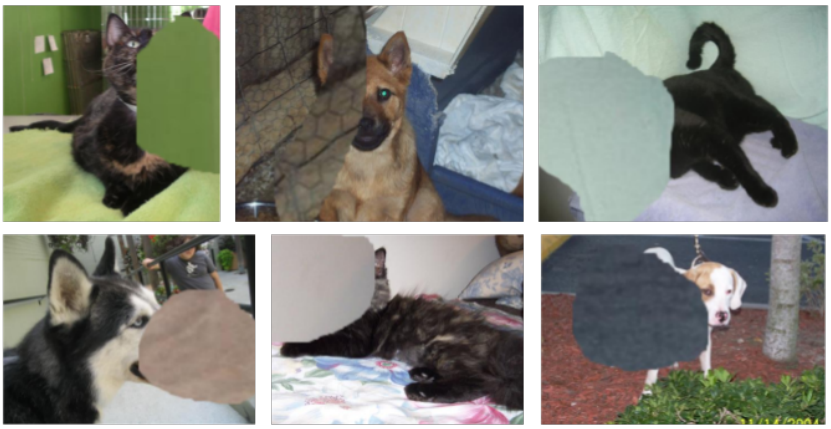


FIGURE 5.28 – Quelques images d'objets occultés où notre modèle les reconnaît correctement et le simple CNN échou

5.5 Interpretation

Notre approche surpasse le modèle CNN avec un pourcentage de 4% sur le test set où elle atteint 92% pour notre modèle et 88% pour le CNN simple. La raison pour laquelle notre modèle fonctionne mieux est que dans le jeu de données kaggle cats-vs-dogs les objets (chien et chat) sont similaires dans leur apparence globale. Ce surplus de performance de notre modèle a été acquis par la puissance discriminative des caractéristiques SIFT qui assure une dissemblance des classes d'objets en particulier des chats et des chiens. Ainsi, notre modèle peut apprendre des informations globales à partir de l' image et des informations locales de l' image partielle. Le modèle proposé fonctionne également bien dans le cas d'objets partiellement occultés où notre modèle surpasse le modèle CNN avec un pourcentage de 33% atteignant une accuracy de 87% pour notre modèle et 54% pour simple CNN, c'est parce que notre modèle a été guidé par les points d'intérêt SIFT à reconnaitre et discerner entre les objets sur les parties non occultées de l'objet.

5.6 Conclusion

Dans ce chapitre nous avons présenté le matériel et les outils logiciels utilisés dans le développement de la méthode proposée, l'interface graphique utilisateur, l'implémentation des différentes étapes impliquées dans le développement de cette méthode. De l'apprentissage et des tests expérimentaux ont été réalisés. L'évaluation du modèle SVM a montré que ce modèle était utile pour sélectionner les caractéristiques SIFT les plus discriminantes pour alimenter notre modèle CNN. Les résultats expérimentaux obtenus, sur les 5000 images réservaient pour les tests, ont montré que la performance de notre modèle CNN guidé SIFT atteint une performance de 92%. En revanche le simple CNN obtenu seulement 88%. Dans le cas des objets partiellement occultés, notre modèle atteint une performance acceptable sur 100 images du même test set contenant des objets occultés manuellement.

Conclusion Générale

Dans le cadre de ce mémoire, nous avons étudié la faisabilité de pouvoir guider un modèle CNN par des points d'intérêt SIFT afin d'améliorer le processus de la reconnaissance des classes d'objets quand un simple modèle CNN pourrait avoir du mal à le faire, en particulier dans le cas des objets occultés. Pour ce faire, nous avons développé un CNN avec deux entrées, la première entrée est pour l'image originale, la seconde est pour une partie de celle-ci recadrée sur la région qui a la plus grande densité de points d'intérêt SIFT appartient à l'objet. Pour détecter ces points, nous avons utilisé un classifieur SVM. Nous avons également utilisé un CNN simple pour comparer ses performances avec notre modèle. Les bons résultats expérimentaux obtenus sur le jeu de données kaggle Cats-Vs-Dogs ont montré que notre modèle offre une performance d'accuracy de 92% et surpasse le modèle CNN simple qui atteint un taux de précision de 88% sur ce même jeu de test.

Pour réaliser des tests de performances sur les objets occultés, nous avons choisi au hasard 100 images de l'ensemble de test kaggle Cats-Vs-Dogs pour lesquelles nous avons partiellement occulté les objets à reconnaître avec une partie de leur arrière-plan, puis nous les avons soumis aux modèles CNN. Les résultats ont montré l'efficacité de notre méthode proposée lorsqu'elle reconnaît 87 des 100 images où l'autre modèle reconnaît 54 seulement.

Revenant au volet théorique de notre mémoire, nous avons discuté des notions fondamentales nécessaires à la compréhension de l'approche proposée. Tout d'abord nous avons relaté la notion importante d'apprentissage, les détecteurs locaux et certains classifieurs dans le premier chapitre, puis nous avons présenté l'apprentissage profond et ensuite nous avons discuté en détail l'une de ses architectures d'apprentissage supervisé qui est le réseau de neurones convolutionnel.

Les limites que nous pouvons relever pour notre modèle sont les chutes drastiques de ses performances dans le cas de la reconnaissance multi-objets. La raison étant que le classifieur SVM ne peut pas distinguer si les points d'intérêts appartiennent aux différents objets, mais distingue seulement entre objet et non objet.

Pour les travaux futurs, nous proposons de construire un modèle purement basé sur des réseaux convolutifs profonds sans passer par un classifieur classique du type SVM et de le tester sur d'autres jeux de données dans le cas de la reconnaissance multi objets, où la phase de détection des objets est nécessaire.

Bibliographie

[1] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2) :91–110, 2004.

[2] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2017.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556*, 2014.

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

[6] Mingxing Tan and Quoc Le. Efficientnet : Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[7] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020.

[8] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V Le. Meta pseudo labels. *arXiv preprint arXiv :2003.10580*, 2020.

[9] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[10] Stuart Russell and Peter Norvig. Artificial intelligence : a modern approach. 2002.

[11] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[12] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf : Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[13] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[14] Evelyn Fix. *Discriminatory analysis : nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1985.

[15] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[16] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3) :273–297, 1995.

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133, 1943.

[18] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[19] Jürgen Schmidhuber. Deep learning in neural networks : An overview. *Neural networks*, 61 :85–117, 2015.

[20] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088) :533–536, 1986.

[22] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[23] Tom Mitchell and Machine Learning McGraw-Hill. Edition, 1997.

[24] Li Deng and Dong Yu. Deep learning : methods and applications. *Foundations and trends in signal processing*, 7(3–4) :197–387, 2014.

[25] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1) :106–154, 1962.

[26] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[27] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[28] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv :1609.04747*, 2016.

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

[30] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1) :1–48, 2019.

[31] Lutz Prechelt. Early stopping-but when ? In *Neural Networks : Tricks of the trade*, pages 55–69. Springer, 1998.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958, 2014.

[33] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv :1502.04156*, 2015.

[34] Ozgur Demir-Kavuk, Mayumi Kamada, Tatsuya Akutsu, and Ernst-Walter Knapp. Prediction using step-wise l1, l2 regularization and feature selection for small data sets with large number of features. *BMC bioinformatics*, 12(1) :1–10, 2011.

[35] Maria V Valueva, NN Nagornov, Pave A Lyakhov, Georgiy V Valuev, and Nikolay I Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, 177 :232–243, 2020.

[36] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[37] Aäron Van Den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Neural Information Processing Systems Conference (NIPS 2013)*, volume 26. Neural Information Processing Systems Foundation (NIPS), 2013.

[38] Ronan Collobert and Jason Weston. A unified architecture for natural language processing : Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[39] Oleksii Avilov, Sébastien Rimbert, Anton Popov, and Laurent Bougrain. Deep learning techniques to improve intraoperative awareness detection from electroencephalographic signals. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 142–145. IEEE, 2020.

[40] Yann LeCun. The mnist database of handwritten digits. *http ://yann. lecun. com/exdb/mnist/*, 1998.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25 :1097–1105, 2012.

[42] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[43] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[44] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet : Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv :1602.07360*, 2016.

[45] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[46] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment : Practical data augmentation with no separate search. *arXiv preprint arXiv :1909.13719*, 2(3), 2019.

[47] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[48] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3) :211–252, 2015.

[49] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[50] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv :1506.06579*, 2015.

[51] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam : Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[52] Ondrej Miksik and Krystian Mikolajczyk. Evaluation of local detectors and descriptors for fast feature matching. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 2681–2684. IEEE, 2012.

[53] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[54] Geoffrey Hinton. Neural networks for machine learning online course. `https://www.coursera.org/learn/neural-networks/home/welcome`.

[55] International Organization for Standardization. *ISO 5725-1 : 1994 : accuracy (trueness and precision) of measurement methods and results-part 1 : general principles and definitions*. International Organization for Standardization, 1994.

[56] Yutaka Sasaki et al. The truth of the f-measure. 2007. *URL : https ://www. cs. odu. edu/˜ mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf [accessed 2021-05-26]*, 2007.

[57] David L Olson and Dursun Delen. *Advanced data mining techniques*. Springer Science & Business Media, 2008.

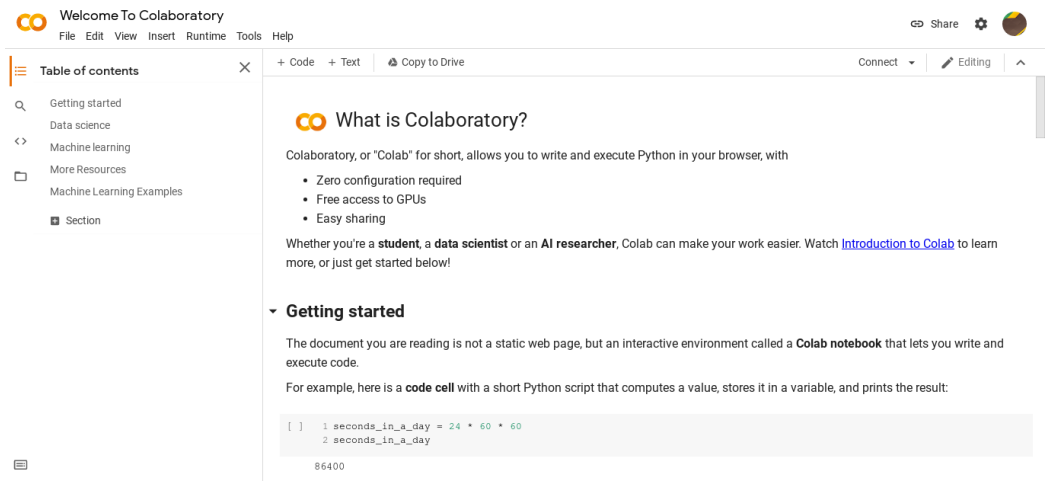
Annexe

Comment utiliser Google Colab

Si vous souhaitez créer un modèle d'apprentissage automatique mais que vous n'avez pas accès à un ordinateur capable de supporter la charge, Google Colab est la plateforme qu'il vous faut. Même si vous disposez d'un GPU ou d'un ordinateur puissant, la configuration d'un environnement local avec Anaconda, l'installation de paquets et la résolution des problèmes d'installation sont des procédures qui prennent du temps.

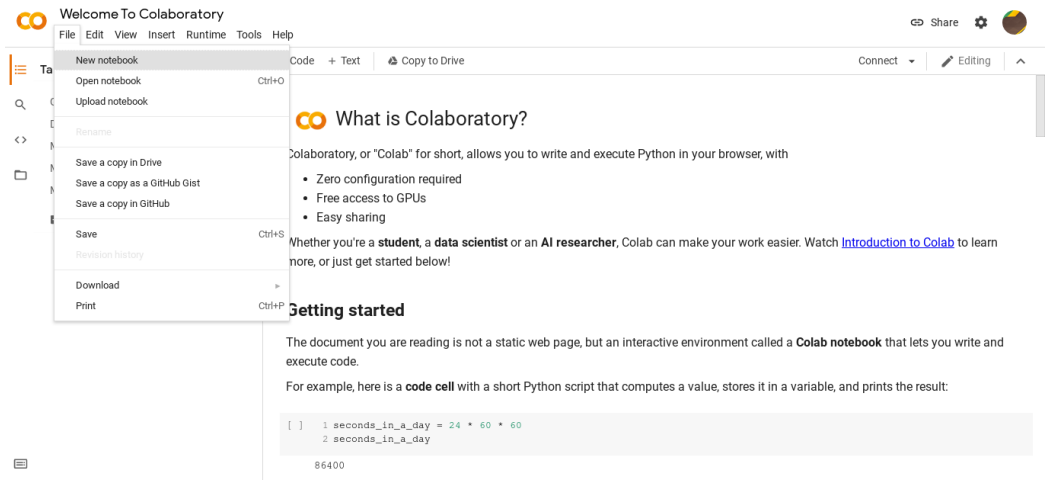
Colaboratory est un environnement de Jupyter notebook gratuit fourni par Google où vous pouvez utiliser des GPU et des TPU gratuits qui peuvent résoudre tous ces problèmes.

Pour commencer à travailler avec Colab, vous devez d'abord vous connecter à votre compte Google, puis aller à ce lien `http://colab.research.google.com/`



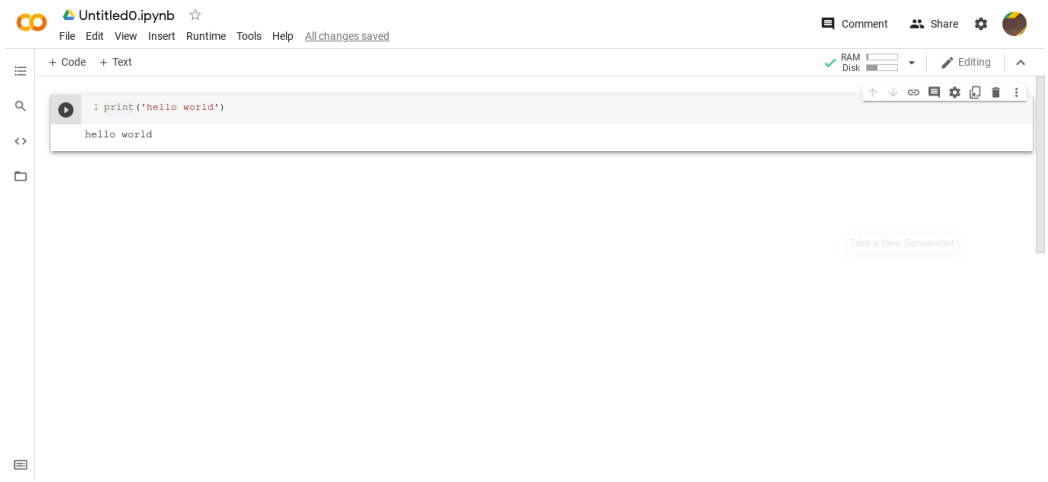
Créer un nouveau fichier Colab

Cliquez sur ***File > New Notebook*** pour créer un fichier colab.



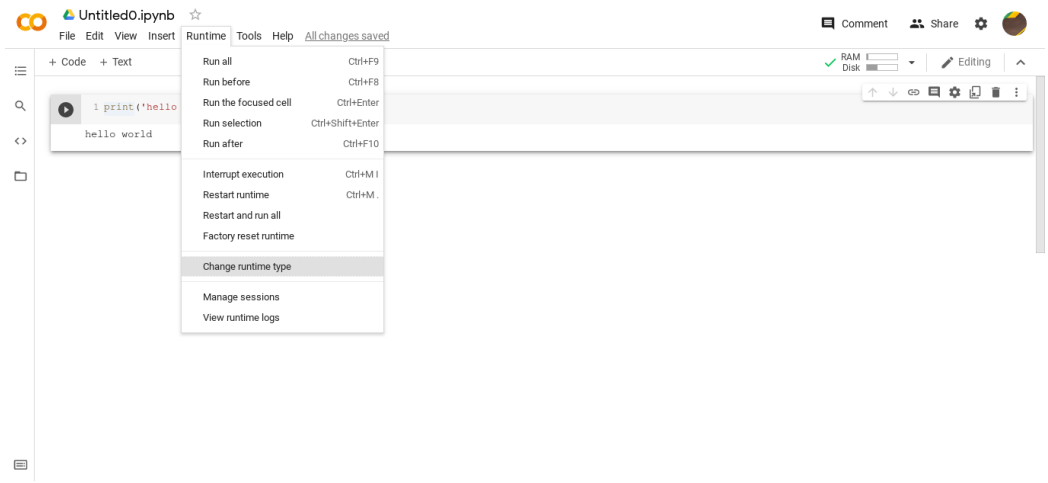
Lors de la création d'un nouveau notebook, il créera un notebook Jupyter avec le nom Untitled0.ipynb et l'enregistrera sur votre lecteur Google drive dans un dossier nommé Colab Notebook. Maintenant, comme il s'agit essentiellement d'un Notebook Jupyter, toutes les commandes de Notebook Jupyter fonctionneront ici.

Après cela, vous devriez voir qu'un notebook Jupyter a été ouvert. Vous voyez une cellule vide avec le bouton **exécution**. Une fois le code écrit, vous pouvez exécuter la cellule à l'aide du bouton d'exécution.

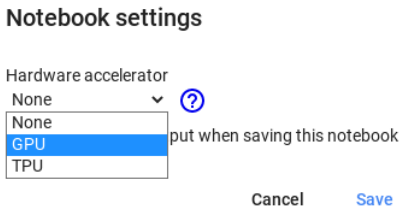


Changer l'environnement d'exécution

Cliquez sur ***Runtime > Change runtime type*** pour afficher les paramètres du notebook.



On ***Hardware accelerator*** sélectionnez **GPU** ou **TPU** pour accélérer l'apprentissage.

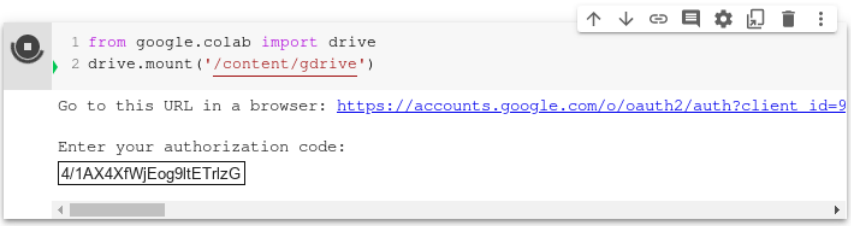


Utiliser les données de Google Drive

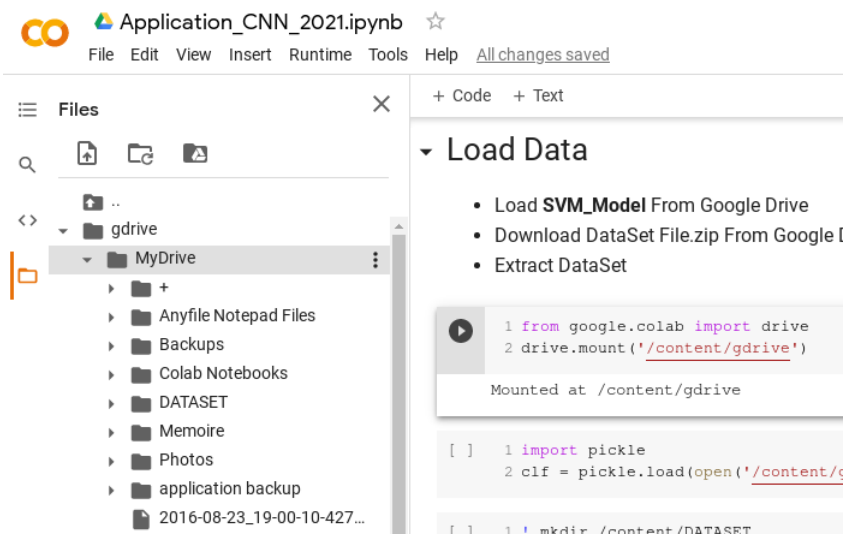
Avec deux lignes de code, vous pouvez installer Google Drive dans notre machine virtuelle.

from google.colab import drive
drive.mount('/content/gdrive')

Vous verrez un lien, cliquez dessus, accordez l'accès, copiez le code qui apparaît, collez-le dans la case et appuyez sur Entrée.



Les fichiers Google Drive font partie de la machine virtuelle Google Colab dans le dossier **/content**.



Vous pouvez maintenant écrire votre propre code Python, ajouter un bloc de code textbf [+ Code], écrire et entraîner votre modèle de réseau de neurones.