Apprentissage Few Shot et méthode d'élagage pour la détection d'émotions sur bases de données restreintes

Kergann Le Cornec

Thèse : Soutenue le 03 novembre 2020

Apprentissage *Few Shot* et Méthode d'Élagage pour la Détection d'Émotions sur Bases de Données Restreintes

---

Auteur
Kergann Le Cornec

Rapporteurs
Stéphane Canu ; *Prof. Institut National des Sciences Appliquées de Rouen*
Antoine Cornuejols ; *Prof. AgroParisTech*

Examinateurs
Violaine Antoine ; *Prof. Université Clermont Auvergne*
Gérard Bailly ; *Prof. Gipsa Lab Grenoble*
Engelbert Mephu Nguifo ; *Prof. Université Clermont Auvergne*

Directeur de thèse
Vincent Barra ; *Prof. Université Clermont Auvergne*

Résumé

La détection d'émotions joue un rôle majeur dans les relations humaines : une bonne compréhension de l'état émotionnel de l'interlocuteur mène à une bonne compréhension de son discours. De fait, elle est aussi très importante dans les relations humain-machine. Dans le domaine de la détection des émotions par ordinateur, l'apprentissage profond s'est imposé comme l'état de l'art. Cependant, les techniques classiques d'apprentissage profond ne fonctionnent plus lorsque la base d'apprentissage est petite. Cette thèse explore deux pistes de réponse : l'élagage et quelques méthodes d'apprentissage *few shot*.

De nombreuses techniques d'élagage existent, mais se concentrent souvent sur un équilibre entre le pourcentage des poids supprimés et la précision. Nous proposons une nouvelle méthode d'élagage, améliorant le choix des poids à supprimer. Cette technique est basée sur la mise en compétition de deux réseaux : le réseau original et un réseau que nous nommons rival. L'idée est de partager les poids entre ces deux réseaux dans le but de maximiser la précision du modèle. Pendant l'apprentissage, les poids ayant un impact négatif sur la précision vont être supprimés, optimisant ainsi le modèle, tout en améliorant la précision. Cette technique est testée sur différents réseaux et différentes bases de données et atteint l'état de l'art en améliorant la précision tout en supprimant un pourcentage de poids non négligeable.

La seconde contribution concerne l'utilisation de réseaux correspondants (réseaux siamois et triple), comme réponse à l'apprentissage sur bases multimodales restreintes. Les deux modalités son et image sont combinées afin d'apprendre leurs caractéristiques principales, pour la détection d'émotions. Nous montrons qu'en se limitant à 200 instances d'entrainement par classe, le réseau triple atteint l'état de l'art (appris sur des centaines de milliers d'instances) sur certaines bases de données.

Nous montrons aussi que, pour la classification d'émotions, les réseaux triples offrent une meilleure représentation des émotions, et par conséquent présentent de meilleurs résultats que les réseaux siamois.

Une nouvelle fonction de perte basée sur le *triplet loss* est introduite facilitant l'appren-

tissage de ces réseaux. Différentes méthodes sont aussi appliquées, offrant des éléments de comparaisons du modèle et plus précisément de la représentation vectorielle.

A terme, ces deux méthodes pourront être combinées pour proposer des modèles légers et performants. Comme le nombre de paramètres à apprendre sera plus faible, un réseau triple élagué donnera possiblement de meilleurs résultats.

Mots clés : Détection, Émotion, Apprentissage Profond, Élagage, *Few Shot*, Réseau Siamois, Réseau Triples.

Abstract

Emotion detection plays a major part in human interactions, a good understanding of the speaker's emotional state leading to a better understanding of his speech. It is de facto the same in human-machine interactions. In the area of emotion detection using computers, deep learning has emerged as the state of the art. However, classical deep learning technics perform poorly when training sets are small. This thesis explores two possible ways for tackling this issue, pruning and few shot learning.

Many pruning methods exist but focus on maximising pruning without losing too much accuracy. We propose a new pruning method, improving the choice of the weights to remove. This method is based on the rivalry of two networks, the original network and a network we name rival. The idea is to share weights between both models in order to maximise the accuracy. During training, weights impacting negatively the accuracy will be removed, thus optimising the architecture while improving accuracy. This technic is tested on different networks as well as different databases and achieves state of the art results, improving accuracy while pruning a significant percentage of weights.

The second area of this thesis is the exploration of matching networks (both siamese and triple), as an answer to learning on small datasets. Sounds and Images were merged to learn their main features, in order to detect emotions. We show that, while restricting ourselves to 200 training instances for each class, triplet network achieves state of the art (trained on hundreds of thousands instances) on some databases. We also show that, in the area of emotion detection, triplet networks provide a better vectorial embedding of the emotions than siamese networks, and thus deliver better results.

A new loss function based on triplet loss is also introduced, facilitating the training process of the triplet and siamese networks. To allow a better comparison of our model, different methods are used to provide elements of validation, especially on the vectorial embedding.

In the long term, both methods can be combined to propose lighter and optimised networks. As the number of parameters is lowered by pruning, the triplet network should learn more easily and could achieve better performances.

Keywords : Emotions Detection, Deep Learning, Pruning, Few Shot, Siamese Network,

Triplet Network.

Table des matières

Table des figures

Liste des tableaux

Notations

Cette section présente les notations utiliséees tout au long du document.

Ensemble

$\mathbb{A}$	Un ensemble
$\mathbb{R}$	L'ensemble des réels
$\{0, 1\}$	l'ensemble contenant 0 et 1
$\{0, 1, \ldots, n\}$	L'ensemble de tous les entiers entre 0 et n
$[a, b]$	L'ensemble des réels entre a et b (a et b inclus)

Algèbre Linéaire

a	Un scalaire (entier ou réel)
$\boldsymbol{a}$	Un vecteur
$\boldsymbol{A}$	Une matrice
$\boldsymbol{A}^\top$	La transposée de la matrice $\boldsymbol{A}$
$\boldsymbol{A} \odot \boldsymbol{B}$	Le produit d'Hadamard de $\boldsymbol{A}$ et $\boldsymbol{B}$
$\boldsymbol{I}_n$	Matrice identité avec n colonnes et n lignes
$\text{diag}(\boldsymbol{a})$	Une matrice carrée diagonale, dont les valeurs sont données par le vecteur $\boldsymbol{a}$

Indice

a_i	L'élément i du vecteur $\boldsymbol{a}$
$A_{i,j}$	L'élément i, j de la matrice $\boldsymbol{A}$

Calcul

$\frac{dy}{dx}$	La dérivée de y selon x
$\frac{\partial y}{\partial x}$	La dérivée partielle de y selon x
$\nabla_{\boldsymbol{x}} y$	Le Gradient de y selon $\boldsymbol{x}$

Fonctions

$f : \mathbb{A} \to \mathbb{B}$	Une fonction f dont l'ensemble de départ est $\mathbb{A}$ et l'ensemble d'arrivée est $\mathbb{B}$
$\log x$	Logarithme de x
$\sigma(x)$	Sigmoide de x, $\sigma(x) = \frac{1}{1+\exp(-x)}$
$\|\|\boldsymbol{x}\|\|_p$	L^p norme de $\boldsymbol{x}$
$\|\|\boldsymbol{x}\|\|$	L^2 norme de $\boldsymbol{x}$

Datasets

$\mathbb{X}$	Un ensemble d'entrainement
$\mathbf{y}$	Le vecteur label associé a l'ensemble d'entrainement $\mathbb{X}$
$\mathbf{x^i}$	L'élément i d'un ensemble d'entrainement $\mathbb{X}$
y^i	Le label associé a $\mathbf{x^i}$
x	Un élément de l'ensemble d'entrainement $\mathbb{X}$
y	Le label associé a x

Acronymes

AC Informatique Affective (Affective Computing). 2

ACP Analyse en Composantes Principales. 34, 148

AE Auto-Encodeur. 145, 146

ANN Réseau de Neurones Artificiels. 128

CNN Réseaux de Neurones Convolutifs. 137, 146

FER Reconnaissance d'Émotions Faciales. 3, 33

FSL Apprentissage Few Shot. 8, 14

GAN Réseaux Antagonistes Génératifs. vii, x, 17, 147

HOG Histogramme de Gradient Orienté. 39

LBP Motif Binaire Local (Local Binary Patterns). 39

PMC Perceptron Multicouche. 131, 146

SIFT Transformation de caractéristiques Visuelles Invariante à l'Échelle. 39

SVM Machine à Vecteurs de Support. 8, 33

t-SNE t-distributed Stochastic Neighbor Embedding. 149

UMAP Uniform Manifold Approximation and Projection. vii, 150

VAE Auto-Encodeur Variationnel. 146

1 Introduction

Selon Paul Watzalawick et al. [6] le langage non verbal représenterait plus de 75% du contenu global de communication.

Cette communication non verbale fait référence à l'étude du langage corporel, qui correspond de manière non exhaustive : aux expressions faciales, aux gestes, aux postures, voire même aux distances interpersonnelles.

Par ailleurs, dans les travaux d'Albert Mehrabian et al. [7], [8], la communication est séparée en trois catégories, elle est :

— Verbale (par la signification des mots) à 7%

— Vocale (intonation de la voix) à 38%

— Visuelle (expression du visage et langage corporel) à 55%.

Ces chiffres sont, bien entendu, controversés mais les psychologues s'accordent à dire que le langage non verbal représente une immense partie de la communication.

La reconnaissance des émotions joue un rôle prépondérant dans la communication, puisqu'elle aide à comprendre les messages portés par nos interlocuteurs et permet d'adapter nos réponses.

Le terme *émotion* est un terme couramment utilisé, et difficile à définir avec précision. Intuitivement, l'émotion est un phénomène à la fois physique et physiologique. Mais que caractérise une émotion ? Il n'y a pas de concensus parmis les psychologues permettant de répondre à cette question. Les travaux de Frijda et Scherer [9] offres tout de même différentes caractéristiques capitales des émotions, partagées par de nombreux psychologues.

Nous avons décidé de nous baser sur les six émotions de base décrites par Paul Ekman en 1972 [10]. Il les annonce universelles et beaucoup de bases de données se fondent sur cette définition. Les six émotions sont : la colère, le dégoût, la peur, le bonheur, la tristesse et la surprise. À ces six émotions peut être ajoutée l'expression neutre, indiquant l'absence d'émotion qu'il peut être utile de détecter.

Plus récemment, en 2007, Simon Baron-Cohen introduit dans [11], 412 émotions dif-

ferentes organisées dans 24 groupes. Le but est l'apprentissage de la reconnaissance des émotions. Ces différentes émotions sont très précises et très interressantes, cependant les bases de données classiques sont fréquemment labélisées avec la description plus classique de Paul Ekman.

Pour le détection d'émotions, l'informatique affective (AC) apporte différentes pistes de réponses. Elle trouve son origine en 1995 dans [12] et déclare que, comme l'émotion est fondamentale dans les relations humaines, elle doit aussi être présente dans les nouvelles technologies.

L'AC étudie la conception de systèmes capables de reconnaître, de modéliser et de synthétiser les émotions. Elle est aujourd'hui en plein essor pour répondre à différents besoins. Par exemple dans les domaines nécessitant une relation humain-machine comme les assistants virtuels, les chat-bots ou la robotique où l'humanisation d'un robot permet de favoriser les échanges. Le domaine de la santé présente aussi des besoins dans ce domaine, afin de détecter la souffrance des patients dans des flux vidéos par exemple.

La présence de nombreuses compétitions (Audio/Visual Emotion Challenge [13], Interspeech Computational Paralinguistics Challenge [14], Emotion Recognition in the Wild [15], ...) souligne les besoins dans ces différents domaines.

De nombreuses contraintes sont associées à ces domaines. Afin de détecter les émotions, ces systèmes se basent sur différents types de données, qui peuvent être textuelles, vocales, visuelles, audio-visuelles, etc [16].

Ces différentes sources d'informations peuvent être utilisées afin de déduire l'état émotionnel de l'interlocuteur. Nous nous plaçons dans le cadre de cette thèse dans une situation de conversation et choisissons de combiner des images faciales avec des phrases sonores afin de détecter l'émotion portée par l'interlocuteur. Un de nos buts est de combiner ces informations multimodales complémentaires afin d'améliorer la détection d'émotions.

De nos jours, L'AC utilise très couramment l'apprentissage automatique et en particulier l'apprentissage profond.

L'apprentissage profond est l'une des branches de l'apprentissage automatique (machine learning) qui vise à construire automatiquement des connaissances à partir de grandes quantités d'information. Pour ce faire, ces techniques modélisent avec un haut niveau d'abstraction des données, grâce à des architectures complexes composées d'un très grand nombre de transformations non linéaires. Nous en faisons une introduction détaillée en annexe .1 et .2

Elle est de nos jours très performante, et très utilisée dans de nombreux domaines,

comme par exemple les voitures autonomes, les diagnostics médicaux, la détection de fraudes et pour notre cas, la reconnaissance d'image et de son.

1.1 Buts et motivations

Bien que performantes, les techniques classiques d'apprentissage profond nécessitent d'énormes bases de données pour le bon fonctionnement du processus d'apprentissage. Au contraire, les humains peuvent facilement apprendre à classifier des données en utilisant peu de données d'exemples et peuvent généraliser et apprendre rapidement de nouveaux concepts [17].

Par exemple, les réseaux convolutifs présentent les meilleurs résultats en reconnaissance d'émotions faciale (FER), mais le processus d'apprentissage nécessite des centaines de milliers d'exemples avant d'atteindre de telles performances.

Depuis 2012, les architectures des réseaux profonds ont évoluées. Bien que les nouveaux réseaux soient très performants, ils comportent souvent énormément de paramètres. Or plus le réseau comporte de paramètres, plus le besoin en données d'apprentissage grandit [18].

Deux pistes se dessinent alors :

— garder les nouvelles architectures d'apprentissage profond, et les alléger en enlevant des paramètres au sein du modèle. Ces techniques sont nommées méthodes d'élagage (pruning) et deviennent de plus en plus courantes.
— utiliser des approches d'apprentissage *few shot* proposant des moyens d'apprentissage sur bases de données restreintes.

L'élagage est une technique, couramment utilisée sur les arbres, permettant de réduire le nombre de noeuds [19]. LeCun [20] en propose une adaptation aux réseaux de neurones en 1990, permettant de supprimer les poids inutiles dans les réseaux profonds. Bien qu'ayant été améliorées au cours des années, le but de ces méthodes reste le même : l'optimisation de la vitesse d'entrainement (et d'exécution) et la réduction de la taille en mémoire du réseau.

Les techniques d'apprentissage *few shot* consistent à apprendre différentes classes avec uniquement quelques données par classe. Elles sont très utiles lorsque peu de données sont disponibles pour l'apprentissage. Cela arrive très fréquemment, lorsque les expériences sont coûteuses par exemple, que les cas sont rares, ou qu'il n'existe pas d'expert pour l'étiquetage des données.

Lorsque le nombre de données d'apprentissage se réduit à zéro (respectivement une) donnée par classe, on parle de *zero-shot* (resp. *one-shot*) *learning*.

1.2 Solutions proposées

Les deux pistes citées précédemment sont explorées dans ce manuscrit.

Nous proposons dans un premier temps une nouvelle approche d'élagage, basée sur un réseau, que nous nommerons "rival". Le mécanisme d'élagage que nous proposons permet de choisir de manière efficace les poids à enlever, tout en impactant positivement la précision du modèle. Nous comparons ce modèle à l'état de l'art et établissons que ce choix donne des meilleurs résultats, lorsque nous favorisons la précision.

Nous explorons aussi les mécanismes des réseaux dit correspondants (réseaux siamois et triples), lorsque les informations sonores et visuelles sont combinées. Nous présentons ainsi un réseau triple et un réseau siamois multimodaux pour la détection d'émotions sur des bases de données multimodales restreintes. Une nouvelle fonction d'erreur permettant un apprentissage plus rapide est présentée. Ces réseaux sont testés sur différentes bases de données, des éléments de référence sont donnés et les réseaux sont comparés à l'état de l'art.

1.3 Structure

Cette thèse est structurée en 5 parties.

Le chapitre 2 présente succinctement les méthodes de base de l'apprentissage profond, puis les méthodes et les réseaux utilisés durant la thèse.

Puis chaque chapitre est séparé en deux parties, une partie sur l'élagage des réseaux, et une partie sur les méthodes *few shot* pour la détection d'émotions.

Le chapitre 3 expose une revue sur l'état de l'art de l'élagage des réseaux ainsi que la détection d'émotions.

Puis le chapitre 4 présente nos approches sur l'élagage ainsi que leurs résultats. La section 4.1 introduit, dans un premier temps, une nouvelle manière d'élaguer les réseaux, avec la présentation d'un réseau rival. Les résultats de cette aproche, testés sur différentes bases et différents réseaux, sont par la suite présentés en section 4.2.

Enfin le chapitre 5 l'approche d'apprentissage *few shot* ainsi que leurs résultats. La section

5.1 présente l'utilisation de réseaux correspondants pour la reconnaissance d'émotions sur des bases de données multimodales restreintes. Puis, la section 5.2 détaille les résultats obtenus par les réseaux correspondants sur différentes bases de données restreintes.

La figure 1 présente la structure globale du manuscrit.

Apprentissage profond

Chapitre 2 : Méthodes Utilisées

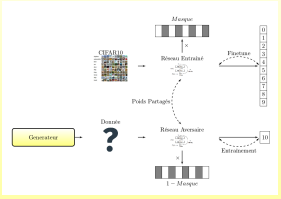
2.1 : Réseaux Utilisés

2.2 : Apprentissage Few Shot

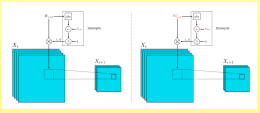
Chapitre 4 : Élagage

Section 4.1 : Approche

-Réseau rival sur base opposée



-Réseau rival sur base identique



Section 4.2 : Résultats

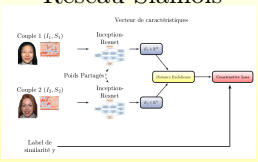
-Réseau rival sur base opposée

-Réseau rival sur base identique

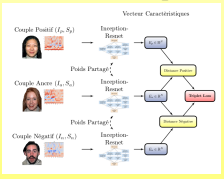
Chapitre 5 : Apprentissage Few Shot

Section 5.1 : Approche

-Réseau Siamois



-Réseau Triple



Section 5.2 : Résultats

-Réseau Siamois

-Réseau Triple

Chapitre 6 : Discussion

Figure 1: Structure du manuscrit

2 Réseaux profonds utilisés et apprentissage few shot

L'apprentissage automatique se place comme sous-domaine de l'intelligence artificielle. Son objectif est d'extraire et d'exploiter les informations présentes dans un jeu de données. Il diffère des approches informatiques traditionnelles en un point majeur.

Les algorithmes traditionnels sont des instructions explicitement programmées pour résoudre des problèmes. L'apprentissage automatique permet de construire des systèmes informatiques apprenant seuls, à partir de données qu'ils utilisent. Puis des méthodes statistiques sont utilisées pour répondre à un problème donné.

L'apprentissage automatique est de nos jours utilisé dans de nombreux domaines comme la reconnaissance faciale, la reconnaissance optique de caractères, les moteurs de recommandation, les voitures autonomes et obtiennent des performances parfois comparables, voire supérieure, à l'expertise humaine.

Parmi toutes les taxonomies possibles, les algorithmes d'apprentissage automatique peuvent être classés en trois grandes catégories : l'apprentissage supervisé, l'apprentissage non-supervisé et l'apprentissage par renforcement.

L'apprentissage par renforcement consiste, pour un agent (robot, ordinateur, etc.) à apprendre les actions à effectuer pour optimiser une fonction de récompense. Durant l'entrainement, l'agent cherche la stratégie optimale maximisant les récompenses obtenues. Ce type d'apprentissage est beaucoup utilisé en robotique ainsi que dans les jeux (go, échecs, ...).

Dans le cas de l'apprentissage non supervisé, le modèle apprend de manière totalement autonome. Aucune information ne lui est préalablement communiquée. Les données sont non étiquetées de sorte que l'algorithme d'apprentissage trouve lui même des points communs parmi les données d'entrée. Il est couramment utilisé pour les données transactionnelles (en finance), surtout lorsque la taille des données est conséquente (données difficilement étiquetables), ou lorsque plus généralement, aucun superviseur n'est présent

pour étiqueter les données d'entrée.

Dans l'apprentissage supervisé, le résultat attendu est transmis au modèle afin qu'il apprenne de ses erreurs. Les données d'entrainement sont étiquetées des résultats attendus. On dit aussi que les données d'entrainement sont *labélisées*. Le modèle va alors apprendre de chaque exemple en modifiant ses paramètres, de façon à diminuer l'écart entre le résultat obtenu et la vérité. Il existe de nombreux algorithmes d'apprentissage supervisés, parmi lesquels on peut citer les machines à vecteurs de support (SVM), la méthode des K plus proches voisins, les arbres de décision, les réseaux de neurones, etc. Le lecteur pourra consulter [21] présentant en détails les méthodes supervisées.

Dans cette thèse nous nous intéressons à l'apprentissage supervisé et notamment aux réseaux de neurones. Ce choix à été influencé par leurs performances dans le domaine de la classification d'images. Nous explorons plus particulièrement les réseaux de neurones profonds, se plaçant au sein de l'apprentissage profond, qui sont l'état de l'art dans beaucoup de problèmes de vision par ordinateur. Contrairement aux méthodes d'apprentissage peu profond, les algorithmes d'apprentissage profond présentent un nombre conséquent de transformations (d'opérations), appliquées de l'entrée à la sortie de l'algorithme.

Les réseaux profonds sont connus depuis des dizaines d'années. Cependant avant 2012, et la mise à disposition pour les problèmes liés aux images de la base de données ImageNet [22], ces réseaux étaient souvent surpassés par des techniques moins coûteuses et des architectures plus simples. Aujourd'hui, les masses de données importantes, ainsi que les progrès en puissance de calcul, font de l'apprentissage profond un outil performant dans de nombreux domaines.

Cependant, l'apprentissage profond dans sa forme générale ne se comporte pas de manière optimale en présence de bases d'apprentissage restreintes.

En effet, dès que la base d'apprentissage devient petite, ce qui arrive fréquemment dans de nombreux domaines (neurosciences, maladie rare, expériences coûteuses, ...), les réseaux profonds ne généralisent plus correctement et n'obtiennent pas les résultats escomptés. En réponse à ce problème, de nombreuses techniques d'apprentissage *few shot* (FSL) ont émergé ces dernières années. Ces techniques proposent des pistes pour l'apprentissage de réseaux de neurones profonds, sur un nombre de données réduit.

Dans ce chapitre, les réseaux utilisés dans la suite du manuscrit sont, dans un premier temps présentés. Pour une bonne compréhension de ces réseaux, une description des méthodes utilisées en apprentissage profond, ainsi que dans cette thèse, est présentée en annexe .1 et .2. Le lecteur pourra également se reporter aux ouvrages de GoodFellow

et al. [23] et de Cornuejols et al [24], présentant ce sujet plus en profondeur. Puis, les différentes méthodes d'apprentissage *few shot* utilisées lors de cette thèse sont décrites.

2.1 Réseaux utilisés

Dans cette section, les types de réseaux utilisés dans la suite de la thèse sont décrits. Ces réseaux ont été choisis soit pour permettre une comparaison plus simple avec l'état de l'art (LeNet5, Resnet, Wide-Resnet), soit pour leurs performances dans les différents domaines étudiés (BigGan pour la génération d'images, Inception-Resnet pour la reconnaissance faciale).

2.1.1 LeNet-5

LeNet-5 est un réseau de neurones convolutif multicouche présenté par LeCun et al. [25].

Son architecture est présentée dans la figure 2. C'est une architecture de nos jours bien connue et considérée comme classique, avec une succession de convolutions et de pooling, pour finir par des couches complètement connectées (*fully connected*). Comparée aux réseaux plus actuels présentés par la suite, cette architecture est très petite, elle ne contient que 60,000 paramètres. Elle donne encore aujourd'hui de très bons résultats dans de nombreux domaines et est beaucoup utilisée comme référence, notamment pour l'élagage.

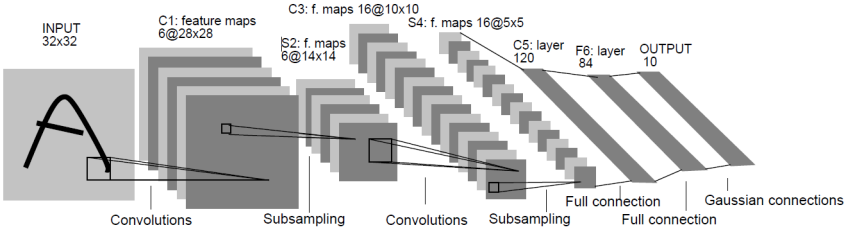


Figure 2: Architecture du réseau LeNet5. Source : [25]

2.1.2 Inception

Inception est une famille de réseau proposée par Google en 2014 [1] qui va changer l'architecture de base des CNN. L'idée est d'utiliser plusieurs filtres de tailles différentes, sur le même niveau, et de concaténer les résultats pour trouver une représentation plus robuste.

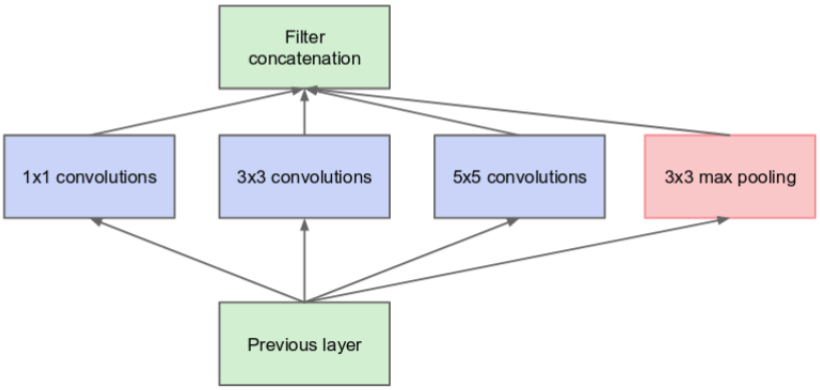


Figure 3: Le module Inception. Source : [1]

La figure 3 représente le module basique *inception*. Ce module calcule trois convolutions de tailles différentes (1×1,3×3 et 5×5) ainsi qu'un maxpooling. Ces résultats sont ensuite concaténés pour former la sortie du module.

Inception V1 est le vainqueur de la compétition ILSVRC en 2014. Il est schématisé sur la figure 4. Il contient 22 couches et de nombreux modules inception. Il reste néanmoins assez petit car il contient seulement 5 millions (5M) de paramètres.

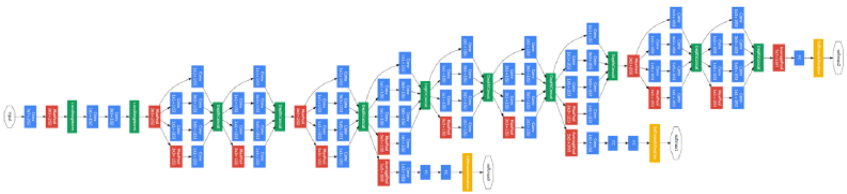


Figure 4: Architecture du réseau Inception v1. Source : [1]

Des versions successives v2, v3 et v4 du réseau initial ont permis d'améliorer la structure et les performances de cette famille de réseaux. Ces changements entrainent une augmentation considérable du nombre de paramètres, puisque ces réseaux contiennent

respectivement 7M, 24M et 43M de paramètres.

2.1.3 ResNet

En 2015, Kaiming et al. présentent le réseau Resnet [2] à la compétition ILSVRC. Ce réseau a 152 couches et utilise des connexions résiduelles. L'idée est de rajouter des raccourcis entre les couches, pour préserver l'information importante dans les couches suivantes. Cela accélère le processus d'apprentissage et évite que le gradient ne disparaisse. Un module ResNet est présenté dans la figure 5.

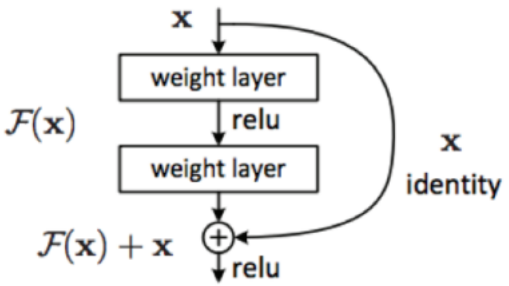


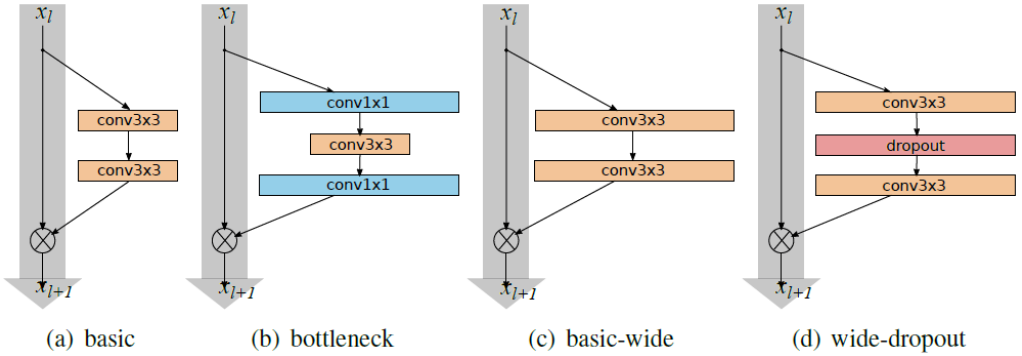
Figure 5: Connexion résiduelle utilisée par Resnet. Source : [2]

Cela permet aussi d'accélérer la propagation du gradient. En effet, durant l'apprentissage, le gradient de la fonction d'activation est calculé, ce qui peut être coûteux. Lorsque des raccourcis : $y = \sigma(x) + x$ sont ajoutés, le gradient peut transiter directement sans passer par la fonction d'activation.

Beaucoup d'améliorations ont été proposées depuis. La figure 6 en présente quelques unes. L'idée est non pas d'ajouter l'identité, mais de trouver les parties importantes (grâce à des convolutions) à donner aux autres couches lors de la connexion résiduelle.

Par la suite, le réseau wide-resnet [3] à été utilisé pour ses performances dans les problèmes de classification sur les bases de données CIFAR10 [26] et MNIST [27].

Comme montré dans le tableau 1, un hyperparamètre k est ajouté afin de contrôler la largeur des blocs au sein des connexions résiduelles, avec N le nombre de blocs dans la couche k . Plusieurs types de blocs résiduels peuvent être utilisés, un bloc est noté $B(M)$ avec M la taille du noyau de convolution dans le bloc. $B(3,1)$ dénote un bloc résiduel avec une couche de convolution $(3,3)$ et une couche de convolution $(1,1)$. $B(3,3)$ à été utilisé car il fournit les meilleurs résultats sur CIFAR10 [26].



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

Figure 6: Différentes améliorations du réseau ResNet. Source : [3]

nom du groupe	taille des sorties	Blocs B=(3,3)
conv1	32×32	$[3 \times 3, 16]$
conv2	32×32	$N \times \begin{bmatrix} 3 \times 3 & 16 \times k \\ 3 \times 3 & 16 \times k \end{bmatrix}$
conv3	32×32	$N \times \begin{bmatrix} 3 \times 3 & 32 \times k \\ 3 \times 3 & 32 \times k \end{bmatrix}$
conv4	8×8	$N \times \begin{bmatrix} 3 \times 3 & 64 \times k \\ 3 \times 3 & 64 \times k \end{bmatrix}$
avg-pool	1×1	$[8 \times 8]$

Table 1: Structure du réseau *wide resnet*. N est le nombre de blocs dans la couche. k est un hyperparamètre, contrôlant la profondeur du réseau. Les dernières couches sont omises pour plus de clarté

Des connexions résiduelles plus larges permettent une meilleure sélection des éléments importants à garder en mémoire. Le cas $k = 1$ correspond au réseau ResNet basique. Pour la suite, $N = 16$ et $k = 8$ ont été choisis pour une bonne comparaison avec l'état de l'art . Ce réseau est parfois noté WRN-16-8-B(3,3).

2.1.4 Inception-ResNet

Inspirés par ces connexions résiduelles, Szeged et al. proposent en 2017 Inception-Resnet [4]. Le principe est présenté dans la figure 7. L'idée est de remplacer, dans le

module, la couche de pooling par des connexions résiduelles.

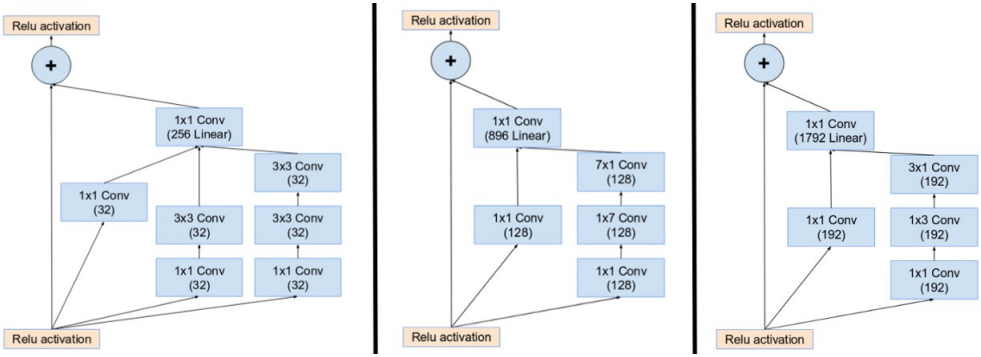


Figure 7: Le module Inception Resnet. Source : [4]

Les auteurs montrent qu'en rajoutant un module résiduel le modèle apprend plus rapidement. Le réseau, bien qu'ayant un nombre important de paramètres (56M), obtient une meilleure précision beaucoup plus tôt, ce qui est dans notre cas, très intéressant.

2.1.5 BigGan

L'architecture BigGan est l'une des architectures les plus performantes dans le domaine de la génération d'image. Elle a été introduite par Andrew Brock et al. en 2018 [28].

L'architecture BigGan128 a été utilisée par la suite, pour ses performances sur les bases de données utilisés, elle est présentée sur la table 2 et dans la figure 8.

Comme son nom l'indique, l'architecture BigGan est une architecture contenant beaucoup de paramètres. L'idée est de créer une grande architecture permettant de générer des images de grande taille (128×128). Pour ce faire, les auteurs rajoutent de nombreux paramètres au sein de l'architecture. Ils introduisent une architecture profonde contenant des connexions résiduelles (b et c de la Figure 8), basée sur l'architecture Resnet [2] et présentent des multiplicateurs (qu'ils nomment ch) sur la largeur des différents blocs afin de régler le nombre de paramètres.

2.2 Apprentissage *Few Shot*

L'apprentissage profond connaît un succès dans beaucoup de domaines, cependant, il est limité dès que les bases de données d'apprentissage sont petites. Récemment les

$z \in \mathbb{R}^{120} \sim N(0, I)$ $Embed(y) \in \mathbb{R}^{120}$
$Linear(20 + 128) \rightarrow 4 \times 4 \times 16ch$
Resblock up $16ch \rightarrow 16ch$
Resblock up $16ch \rightarrow 8ch$
Resblock up $8ch \rightarrow 4ch$
Resblock up $4ch \rightarrow 2ch$
Non-Local Block (64×64)
Resblock up $2ch \rightarrow ch$
BN,ReLU, $3 \times 3 Convch \rightarrow 3$
Tanh

Table 2: Architecture du générateur du BigGan128, ch représentent les multiplicateurs sur la largeur.

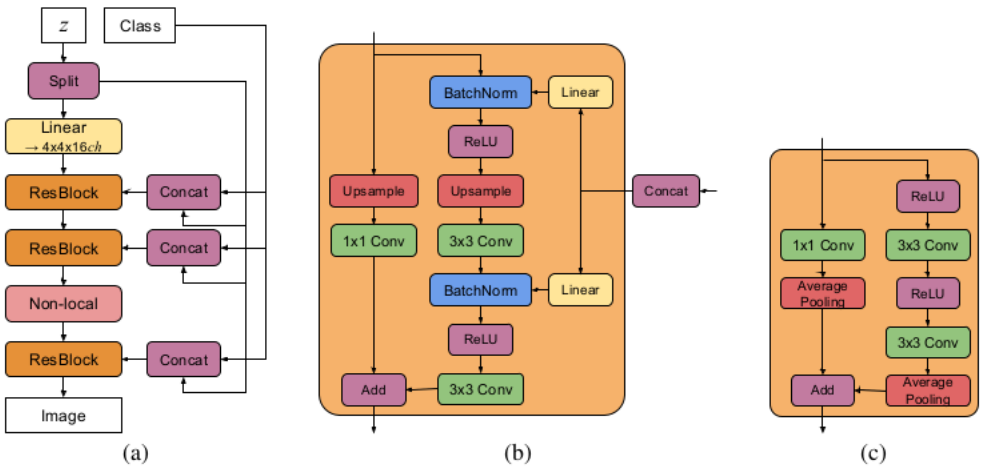


Figure 8: Description des resblocks au sein de l'architecture BigGan : a) Architecture typique des générateurs BigGan b) Un bloc résiduel du générateur (ResBlock up) basé sur l'architecture Resnet c) Un bloc résiduel du discriminateur (ResBlock down). Source [28].

méthodes d'apprentissage *Few Shot* (FSL) proposent des pistes intéressantes pour pallier ce problème. Les méthodes principales de FSL sont décrites ici. Pour plus d'information, le lecteur pourra se référer a [29].

Le FSL est un problème d'apprentissage où la base de données contient seulement quelques exemples labélisés de chaque classe. Cette technique est surtout utilisée en vision

par ordinateur car elle présente des approches et des résultats utilisables dans ce domaine.

Apprendre avec peu de données peut être abordé soit par l'intermediaire des données elles-mêmes (2.2.1), soit par l'utilisation d'algorithmes dédiés (2.2.2), soit grâce à l'adaptation directe d'un modèle (2.2.3). Nous présentons dans cette section, les méthodes utilisées durant la thèse.

2.2.1 Modification des données

Transformer les données pour enrichir la base d'apprentissage est une méthode très facile à mettre en place, et dans beaucoup de cas, très efficace. Cette méthode est, cependant, dépendante des données d'apprentissage et donc non facilement généralisable.

Lors de l'apprentissage sur des bases de données restreintes, le surapprentissage arrive souvent très vite. Le réseau n'a pas assez de données pour bien généraliser le problème. Ce phénomène bien connu est souvent résolu en augmentant la taille de la base de données et/ou réduisant le nombre de paramètres du modèle. L'idée est de générer de nouvelles données labéllisées, à partir des données d'entrainement déjà disponibles, ce qui est une solution relativement facile à mettre en place. Il existe une multitude de techniques d'augmentation de données, nous exposons succintement celles utilisées dans la suite. Pour plus d'information sur l'augmentation de données, le lecteur pourra se réferer par exemple à [30].

2.2.1.1 Augmentation à partir des mêmes données

Inversion

L'inversion est l'une des méthodes les plus simples pour augmenter la base de données. Elle consiste tout simplement à faire des symétries axiales. La première ligne de la figure 9 montre trois inversions effectuées sur des données provenant de la base de données d'images faciales Nimstim [31].

Rotation

La rotation est aussi une méthode simple permettant l'augmentation des données. Elle consiste tout simplement à faire tourner l'image autour de son centre. La deuxième ligne de la figure 9 montre trois rotations à titre d'exemples. En général, on utilise des rotations entre $[-20, 20]$ degrés, le risque d'utiliser une rotation trop élevée est d'introduire des images qui ne correspondent pas au label original (le réseau ne pourra alors pas généraliser).

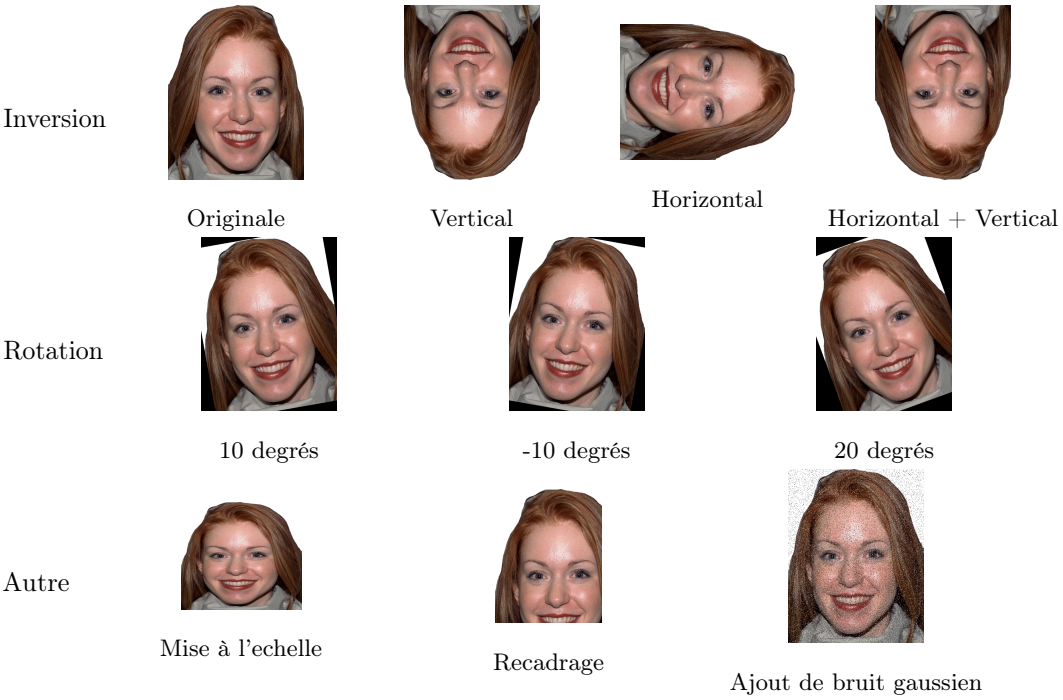


Figure 9: Techniques classiques d'augmentation de données utilisées par la suite.

Mise à l'échelle

Une mise à l'échelle consiste à déformer l'image pour lui donner des dimensions voulues. Une déformation trop importante de l'image poura introduire une image trop différente de la base de données et le réseau n'arrivera pas à généraliser. Un exemple de mise à l'échelle est montré dans la dernière ligne de la figure 9.

Recadrage

Le recadrage consiste à choisir une portion de l'image comme nouvelle donnée (dernière ligne de la figure 9). Si le recadrage supprime des parties importantes de l'image, les convolutions ne trouveront pas les bonnes caractéristiques pour un bon apprentissage.

Ajout de Bruit

Cette technique consiste à ajouter aléatoirement des valeurs, d'une distribution de probabilité (dans la suite, la distribution gaussienne à été utilisée [32]), sur une image. Ajouter trop de bruit conduira à une dénaturation trop forte de l'image et le réseau n'arrivera pas à apprendre correctement. Une addition de bruit sur l'image originale est présentée dans la dernière ligne de la figure 9.

2.2.1.2 Création de nouvelles données avec un réseau antagoniste génératif GAN

Le principe des GAN a été présenté paragraphe .2.2. Ils peuvent aussi être utilisés pour de l'augmentation de données. Par exemple, les auteurs de [33] génèrent (à l'aide d'un GAN) une fausse base de données pour la détection de cancer. Ils montrent que l'entrainement sur cette base de données (composée uniquement d'images créées) donne de meilleurs résultats que l'entrainement sur la petite base de données originale. Sur des bases de données restreintes, les GAN peuvent générer des images correctes (non dénaturées) et labélisées. En effet, après entrainement d'une classe (quand le discriminateur peine à distinguer), le générateur créé des images semblables aux données originales de cette classe . Ce générateur peut être utilisé dans le but de créer des couples (image, label), afin d'enrichir (ou de créer) la base d'entrainement.

2.2.2 Modification de l'algorithme

Pour s'adapter au faible nombre d'exemples, il est possible de modifier le réseau considéré.

2.2.2.1 Apprentissage par transfert

L'apprentissage profond nécessite beaucoup d'images et de temps. Il faut, pour entrainer les réseaux les plus complexes, plusieurs semaines de calculs avec des machines possédant plusieurs GPUs et des bases de données de plusieurs millions d'images. Notre sujet est basé sur l'apprentissage avec peu de données. Une première approche a été d'utiliser des techniques d'apprentissage par transfert. Ces techniques sont ainsi nommées car elles exploitent la connaissance acquise sur un problème de classification général pour l'appliquer de nouveau à un problème particulier. Ici, deux stratégies d'apprentissage par transfert sont présentées : l'extraction automatique de caractéristiques et le finetuning.

Extraction automatique de caractéristiques

L'extraction automatique de caractéristiques exploite uniquement la partie convolutive d'un réseau préalablement entrainé. Elle l'utilise comme extracteur de caractéristiques des images. Il faut noter que cela fonctionne d'autant mieux si l'on travaille sur un problème proche du problème initial, sur lequel le réseau a été entrainé. Lorsque le problème est très similaire, des poids déjà entrainés peuvent être utilisés pour permettre de gagner du temps. Dans cette stratégie, la dernière couche du modèle déjà entrainé est enlevée, les poids des couches restantes sont fixés et un simple perceptron multicouche est appliqué à partir des résultats des couches du réseau pré-entrainé. Chaque donnée de l'ensemble d'entrainement est ainsi transformée en un vecteur de caractéristiques, qui est utilisé pour entrainer le classifieur. Cette méthode présente de nombreux intérêts pratiques. Tout d'abord, l'image est transformée en un vecteur de petite dimension, qui extrait des caractéritiques pertinentes, cela réduit la dimension du problème. L'extraction de caractéristiques permet également d'apprendre beaucoup plus rapidement car tous les poids des couches de convolution sont gelés.

Finetuning

Dans cette stratégie, le réseau déjà entrainé est affiné sur les nouvelles données en continuant la rétro-propagation du gradient. On peut, soit le faire sur tout le réseau (dans ce cas, si les poids trouvés auparavant ne correspondaient pas, le temps gagné est très faible), soit fixer quelques poids, ou même quelques couches. L'intérêt est double :

— On utilise une architecture optimisée
— On profite des capacités d'extraction de caractéristiques apprises sur un jeu de

données de qualité.

Le finetuning, consiste en quelque sorte à prendre un système déjà entrainé sur une tâche de classification, pour le raffiner sur une autre tâche.

Les réseaux pré-entrainés possèdent déjà des coefficients optimisés avec soin sur un grand jeu de données. Il s'agit de les modifier faiblement à chaque itération, pour s'adapter au nouveau problème, sans écraser les connaissances déjà acquises. Pour l'entrainement, il est possible de geler les couches initiales du réseau de neurones et d'adapter seulement les couches finales pour le nouveau problème de classification. Dans ce cas, on fait l'hypothèse que les premières couches extraient des caractéristiques communes à toute image (bords, coins, ...), et que les dernières couches s'intéressent plus à des caractéristiques spécifiques du domaine étudié. Geler toutes les couches convolutives correspond à la première méthode présentée.

2.2.3 Modification du modèle

Des modifications peuvent être appliquées directement dans le réseau. Dans cette partie, deux réseaux correspondants (*Matching Network*) sont présentés : les réseaux siamois et les réseaux triples. Ces réseaux offrent de bonnes performances lors de l'apprentissage sur base de données restreinte. En effet, l'apprentissage de ces réseaux est différent de celui sur les réseaux classiques. Comme nous allons le voir par la suite, ces réseaux ne vont pas apprendre sur une instance mais sur des couples ou des triplets d'instances, cela a pour effet d'obtenir une base de couples ou de triplets plus importantes et ainsi de retarder le surapprentissage.

2.2.3.1 Réseau Siamois

Les réseaux siamois ont été introduits dans les années 90 par Bromley et al. [34], pour répondre à un problème de vérification de signature. Le principe était d'utiliser deux réseaux utilisant des poids partagés (poids identiques), appris sur des images de signatures, dans le but de vérifier si une signature correspond à une autre (si le résultat du premier réseau est proche du résultat du deuxième). Cette architecture est très intéressante car elle permet d'apprendre avec peu de données.

Architecture

Le but d'un réseau siamois est d'apprendre la similarité entre deux images. L'architecture

de base d'un réseau siamois est présentée dans la figure 10.

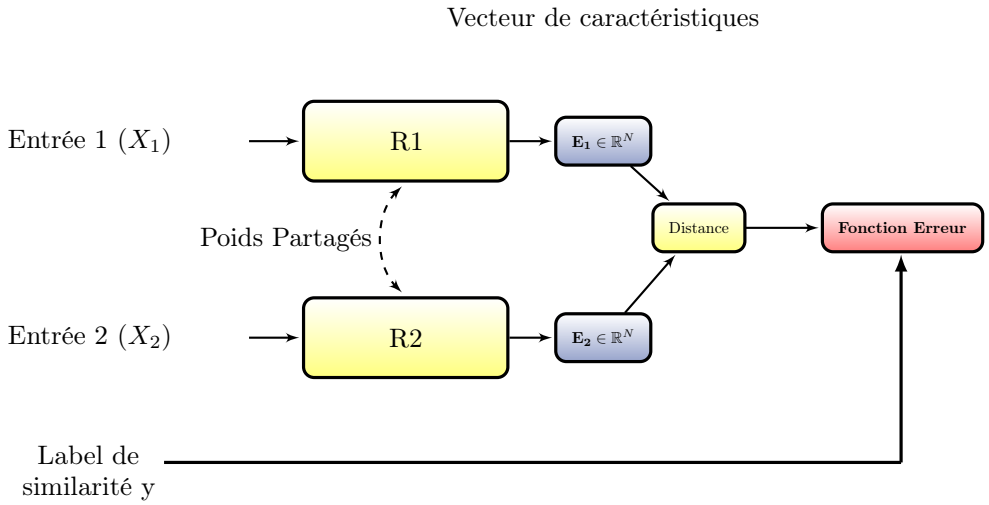


Figure 10: Principe des réseaux siamois R1 et R2 partagent les poids et biais. La fonction d'erreur prend en entrée la distance entre les deux vecteurs de sortie. La fonction de perte va pénaliser les petites ou grandes distances relativement au label de similarité y.

Un réseau siamois est un réseau de neurones composé de deux réseaux identiques, les poids sont partagés. Ces deux réseaux prennent chacun une donnée différente X_1 et X_2 en entrée, et calculent leur vecteur de caractéristiques, respectivement $\mathbf{E_1}$ et $\mathbf{E_2}$. La distance entre ces deux vecteurs est alors calculée, la distance euclidienne est souvent choisie : $d(E_1, E_2) = \sqrt{||\mathbf{E_1} - \mathbf{E_2}||^2}$. Puis le réseau est optimisé, le choix de base pour les réseaux siamois est de minimiser le *Contrastive Loss*.

Contrastive Loss

Le *Contrastive Loss* est défini comme :

$$L = \frac{1}{2}(1-y)d(\mathbf{E_1}, \mathbf{E_2})^2 + \frac{1}{2}(y)max(0, m - d(\mathbf{E_1}, \mathbf{E_2}))^2$$

avec :

— y le label de similarité, $y = 0$ si les deux entrées X_1 et X_2 appartiennent à la même

classe, $y = 1$ sinon.

— $m > 0$ une marge, qui indique que pour deux entrées provenant de classes différentes, si la distance est grande (si $d(\mathbf{E_1}, \mathbf{E_2}) > m$) l'optimisation est finie (pour ces entrées) et ne se fait donc pas ($L = 0$).

L'idée est de minimiser la distance entre deux entrées du même label ($y = 0$) et de maximiser la distance entre deux entrées de labels différents.

À la fin de l'entrainement, toutes les données d'une même classe seront proches. Mais les classes seront éloignées les unes des autres. L'avantage des réseaux de ce genre est qu'aucune information sur le nombre de classe n'est donnée. Le réseau va lui-même trouver le nombre de classes.

Exemples d'utilisation

Les réseau siamois sont beaucoup utilisés en vison par ordinateur. Nous pouvons par exemple citer [35], utilisant un réseau siamois convolutif afin de classifier les bases de données de caractères Omniglot [36]. Avec 30,000 images de la base de données, le modèle atteint une précision de 90.61%.

Dattaraj J et al. [37] proposent un réseau siamois détectant les anomalies sur des images de rails de train. En comparant des rails de train défectueux avec des références, les auteurs prédisent les anomalies à plus de 95% sur leur propre base de données.

Un autre exemple d'utilisation est la reconnaissance faciale. Le réseau proposé dans [38] utilise un réseau siamois pour classifier les visages de la base AT&T [39].

2.2.3.2 Réseau Triple

Aussi très performant quand le nombre de données d'entrainement est restreint, Wang et al. présentent dans [40] une amélioration du réseau siamois : les réseaux triples. Cette section explique le fonctionnement du réseau et l'erreur utilisée.

Architecture Le principe du réseau triple est schématisé sur la figure 11.

Le principe est de donner au réseau un triplet d'instances au lieu d'une donnée seule. Le triplet est composé de trois données :

— une ancre d'une classe aléatoire, notée I_a

— une donnée positive de la même classe que l'ancre, notée I_p

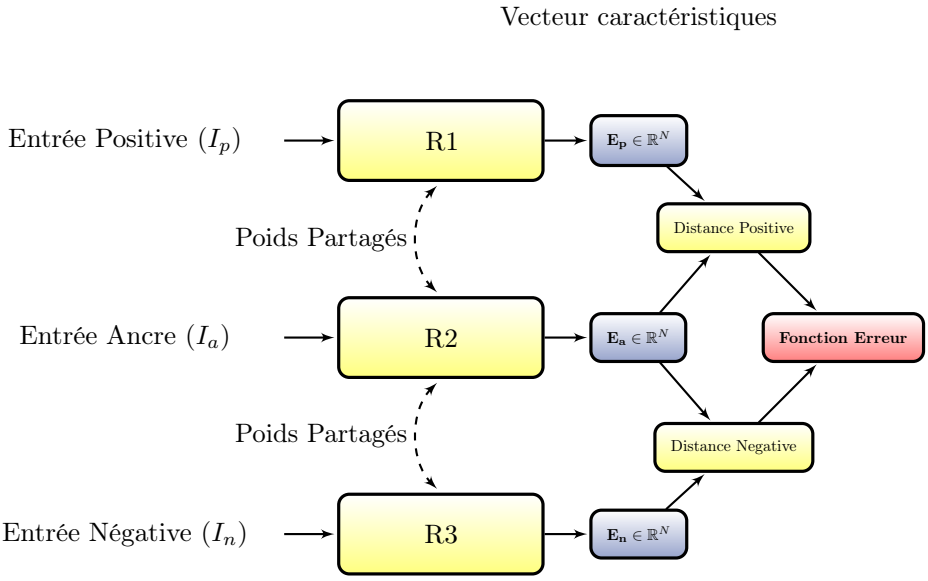


Figure 11: Principe des réseaux triples. R1,R2 et R3 partagent les poids et biais. La fonction de perte prend en entrée les distances entre les vecteurs de sorties. La fonction d'erreur va pénaliser les petites ou grandes distances relativement au label négatif ou positif.

— une donnée négative d'une classe différente de celle de l'ancre, I_n.

Le réseau va représenter ces trois instances puis modifier ses poids pour faire en sorte d'éloigner le négatif de l'ancre, et de rapprocher le positif de l'ancre. Comme dans tout processus d'apprentissage d'un réseau de neurones, ceci est réalisé en minimisant une fonction de coût, et celle utilisée ici est le *Triplet loss* (paragraphe 2.2.3.2).

La seule information donnée au réseau est que les deux premières entrées sont de la même classe et la troisième d'une classe différente, nous notons cette information **Positif/Négatif**.

Triplet Loss

Le *Triplet Loss* est défini comme :

$$Tl(\mathbf{E_a}, \mathbf{E_p}, \mathbf{E_n}) = \max(d(\mathbf{E_a}, \mathbf{E_p}) - d(\mathbf{E_a}, \mathbf{E_n}) + m, 0) \ ,$$

où $\mathbf{E_a}$ est la représentation (*embedding*) de l'ancre après le réseau, $\mathbf{E_p}$ (respectivement $\mathbf{E_n}$) est la représentation de la donnée positive (resp. négative), m est une marge et d est une métrique. Dans la suite, la norme euclidienne $||\ ||_2$ est utilisée.

La fonction objectif est :

$$max(||\mathbf{E_a}, \mathbf{E_p}||_2^2 - ||\mathbf{E_a}, \mathbf{E_n}||_2^2 + m, 0).$$

Cette fonction entraine le réseau sur un triplet d'instances au lieu d'une instance à la fois, et permet de :

— Rapprocher deux données de la même classe, cela revient à diminuer la distance entre une ancre et une donnée positive $||\mathbf{E_a}, \mathbf{E_p}||_2$.
— Éloigner deux données de classes différentes, cela revient à augmenter la distance entre l'ancre et une donnée négative $||\mathbf{E_a}, \mathbf{E_n}||_2$.

Pour que les classes soient bien séparées, une marge m est ajoutée. Pour un triplet (C_a, C_p, C_n), la distance $||\mathbf{E_a}, \mathbf{E_n}||_2$ doit être supérieure à $||\mathbf{E_a}, \mathbf{E_p}||_2$ plus une marge. En minimisant cette fonction de coût, la distance $||\mathbf{E_a}, \mathbf{E_p}||_2$ sera donc diminuée et la distance $||\mathbf{E_a}, \mathbf{E_n}||_2$ sera augmentée.

La fonction *max* est utilisée afin de ne plus déplacer les triplets bien placés, les triplets possédant la condition suivante :

$$||\mathbf{E_a}, \mathbf{E_p}||_2^2 \geq ||\mathbf{E_a}, \mathbf{E_n}||_2^2 + m$$

Triplet Mining

Si la métrique précédente est choisie, le choix des triplets est capital pour la bonne convergence du modèle [41]. Il existe trois catégories de triplets :

— Les triplets simples : ceux dont la valeur du *triplet loss* est nulle car

$$d(\mathbf{E_a}, \mathbf{E_p}) + m < d(\mathbf{E_a}, \mathbf{E_n}).$$

— Les triplets complexes : ceux où le négatif est plus proche de l'ancre que le positif

$$d(\mathbf{E_a}, \mathbf{E_n}) < d(\mathbf{E_a}, \mathbf{E_p}).$$

— Les triplets semi-complexes : donnant une valeur du *triplet loss* positive

$$d(\mathbf{E_a}, \mathbf{E_p}) < d(\mathbf{E_a}, \mathbf{E_n}) < d(\mathbf{E_a}, \mathbf{E_p}) + m.$$

Les différents triplets sont illustrés dans la figure 12.

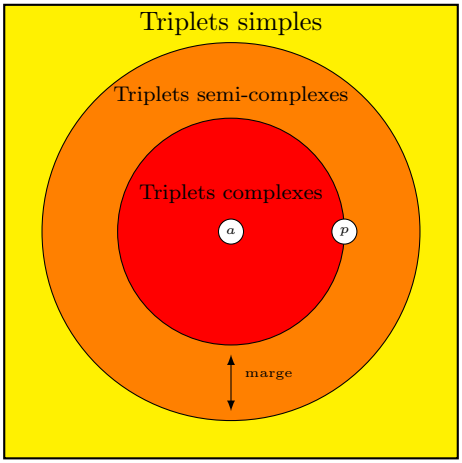


Figure 12: Types de triplets pour une ancre a, un positif p et une marge m fixés. Si l'instance négative se trouve dans la partie rouge, orange ou jaune, on nomme respectivement le triplet : complexe, semi-complexe ou simple.

Si le *Triplet Loss* est utilisé, les triplets simples ne sont pas utiles pour l'entrainement, car la valeur de la fonction de perte est nulle. Les triplets complexes et semi-complexes sont seulement alors choisis.

2.2.4 Relation à d'autres types d'apprentissage

Il est possible de mettre en perspective le FSL avec d'autres techniques d'apprentissage connexes :

— L'apprentissage semi supervisé, qui apprend une hypothèse optimale à partir de peu d'exemples labelisés, et d'exemples non labelisés. Dans le cas où seuls des exemples labalisés positifs sont donnés, on parle de Positive Unlabeled learning.
— L'apprentissage actif, qui sélectionne de l'information non labelisée pertinente pour interroger un oracle.
— L'apprentissage déséquilibré, où la distribution des sorties est fortement biaisée, et où il s'agit d'entraîner et de tester pour choisir la meilleure sortie entre toutes

les valeurs possibles. le FSL quant à lui entraîne et teste avec très peu d'exemples, utilisant éventuellement le reste des données comme connaissance a priori pour l'apprentissage

Par définition, le FSL peut être peut agir de manière supervisée, semi supervisée. Il peut même intervenir dans le cadre de l'apprentissage par renforcement. Il ne requiert pas nécessairement l'existence de données non étiquettées, ni d'un oracle.

3 État de l'art

Ce chapitre est consacré à l'état de l'art de l'élagage et de la détection d'émotions.

3.1 Élagage

De nos jours, les réseaux profonds atteignent l'état de l'art pour la classification d'image. Ces réseaux comportent cependant des millions de paramètres et leurs processus d'entrainement nécessitent des cartes graphiques très puissantes, et peuvent durer jusqu'à plusieurs semaines. La réduction de ces réseaux, pour gagner en temps, en puissance de calcul et en empreinte mémoire est capitale. De plus, élaguer un réseau au cours de l'apprentissage pourrait permettre une diminution du besoin en données d'entrainement. Afin de répondre à ce problème, plusieurs pistes sont étudiées parmi lesquelles :

- la quantification qui consiste à étudier et à réduire le nombre de bits nécessaire pour stocker un élément, avec l'apparition des réseaux binaires [42] (valeur des poids égale à 0 ou 1) comportant des poids stockés sur un bit, ou ternaires [43] (valeur des poids égale à 0,1 ou -1) utilisant des poids stockés sur deux bits. Choudhary et al. [44] proposent une revue sur la compression des modèles.
- Le partage de paramètres [45, 46, 47] qui consiste à partager des poids au sein d'une même architecture, cela divise de fait le nombre de paramètres à apprendre.
- Le dropout [48], présenté en annexe .2.0.5, réduisant le nombre de poids à apprendre.

Le domaine exploré dans cette thèse est l'élagage. La première technique d'élagage dans les réseaux de neurones a été introduite par LeCun [20]. Elle consiste à enlever les poids inutiles du réseau (une présentation plus détaillée est donnée paragraphe 4.1.1). Cette méthode est un peu trop simpliste et ne donne pas des résultats convaincants. Elle à été améliorée au cours des années.

Souhaitant favoriser un gain de précision, tout en supprimant un maximum de poids possibles, nous nous sommes surtout intéressés aux méthodes élaguant peu le réseau, sans une perte considérable (ou avec un gain) de précision.

Quelques méthodes élaguant davantage sont aussi décrites afin de garder une vision globale du domaine.

3.1.1 Élagage non structuré

Les techniques d'élagage que l'on nomme *non structurées*, se concentrent sur les matrices de convolution et les forcent à contenir un maximum de zéro possible pour qu'elles deviennent creuses.

Dans [49] les auteurs proposent de choisir les poids à supprimer en fonction d'un critère de pertinence des neurones par couche (Layer-wise Relevance Propagation LRP [50])).

L'idée est de calculer un critère de pertinence R pour chaque neurone, par couche l :

$$R_i^l = \sum_j \frac{z_{ij}}{\sum_k z_{kj}} R_j^{l+1}$$

avec $z_{ij} = x_i^l w_{ij}^{l,l+1}$ et i l'indexe du neurone sur une couche l.

Puis de supprimer les neurones de la couche non pertinents (avec le R le plus bas). L'approche globale est décrite sur la figure 13.

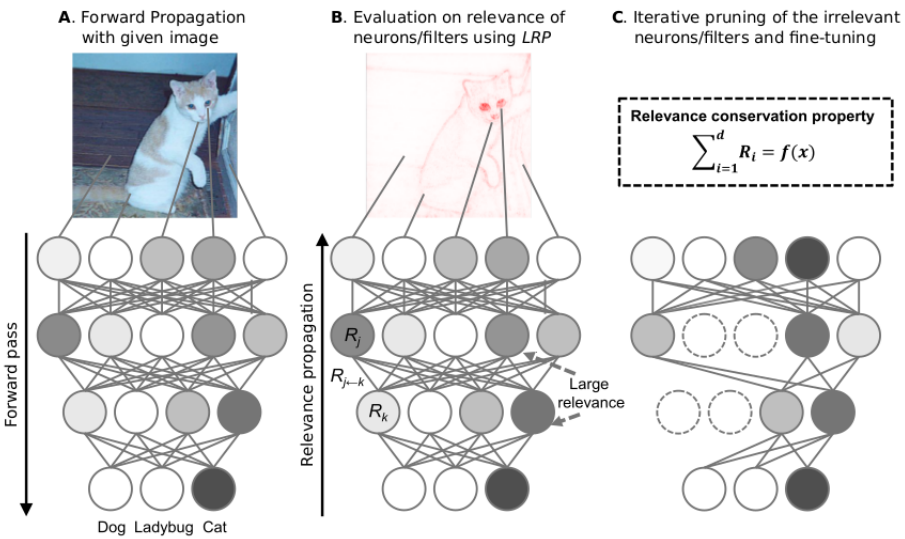


Figure 13: Élagage en fonction du critère de pertinence des neurones par couche (LRP). Source [49].

Les auteurs testent différents critères et plus de 30% de poids sont supprimés sur le réseau VGG16 [51], entrainé sur la base de données CIFAR10 [26] sans perte de précision.

Lee et al. proposent dans [5] de seuiller les valeurs des poids, et de les transformer en valeurs binaires, un exemple est donné sur la figure 14. En entrainant sur CIFAR10, 75% des poids de l'architecture Resnet32 sont supprimés avec une perte de 0.7 point en précision.

$$\begin{bmatrix} -0.1 & 0.9 & 1.2 & -0.2 & -0.6 \\ 1.8 & 0.2 & -0.7 & -1.6 & 0.6 \\ -0.1 & -1.7 & 0.1 & -0.3 & 1.2 \\ -0.4 & 1.4 & -0.9 & 0.6 & 1.4 \\ -1.1 & 0.5 & 1.0 & 1.0 & -0.3 \end{bmatrix} \Longrightarrow \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Figure 14: Binarisation d'une matrice de convolution 5×5 pour un seuil de 0.7. Exemple tiré de [5].

Liu et al. proposent dans [52] d'entrainer des valeurs (non binaires) contrôlant le choix de supprimer les poids ou non, durant le processus d'apprentissage. Ces valeurs sont, par la suite, seuillées par une fonction classique de seuil. En entrainant WideResnet-16-8 sur CIFAR10, le réseau atteint 95.36% de poids supprimés en perdant seulement 0.45 point de précision.

Lin et al. présentent dans [53] un choix de masques (contrôlant les poids) basé sur le gradient stochastique, permettant la réactivation de poids préalablement désactivés. Le réseau Wide-Resnet-28 est testé sur CIFAR10. 95% des poids sont enlevés, avec une perte de 0.31 point de précision.

Les auteurs dans [54] utilisent une série de Taylor du premier ordre pour évaluer l'importance des poids. Les poids jugés inutiles ne sont pas mis à jour dans l'entrainement par la minimisation de la fonction de perte, et convergent petit à petit vers 0, par l'habituelle régularisation par une norme L_2. Pendant l'entrainement, le modèle recalcule la liste des poids inutiles à chaque itération, il y a donc une probabilité de réactivation d'un poids inutile, avant qu'il n'arrive à 0. Grâce à ce choix, les auteurs ont supprimé 90% des poids du réseau Resnet-56 avec une perte de précision de 0.25, en entrainant sur la base de données CIFAR10.

Les méthodes d'élagage non structurées sont moins populaires que les méthodes d'élagage structurées. La raison principale est que ces méthodes ne réduisent pas forcément le temps d'exécution du modèle. La présence de matrices creuses dans l'architecture fournit un gain en espace, mais des librairies dédiées doivent être utilisées si le but est un gain en

temps d'exécution.

Notre objectif est d'élaguer le réseau afin d'obtenir un gain en précision du modèle. Notre méthode se place dans la catégorie des méthodes d'élagage non structurés, et le choix des poids est basé sur [52].

3.1.2 Élagage structuré

L'élagage structuré cherche à diminuer le nombre de convolutions, en s'intéressant à la suppression de groupes, ou même de couches de convolutions.

Les auteurs de [55] suppriment 23.5% des convolutions des réseaux Resnet-56 et Resnet-110 entrainés sur la base de données CIFAR10, avec respectivement un gain en précision de 0.19 et 0.41. Dans la même veine que [56], les auteurs utilisent une distance euclidienne sur les filtres, afin d'évaluer la similarité entre les différentes convolutions. Pour chaque couche, un nombre fixé de couches similaires est enlevé. Les auteurs de [57] utilisent eux, la similarité cosinus. L'erreur est alors réduite de 0.27 sur CIFAR10 avec 23.7% des convolutions supprimées.

Dans [58], les auteurs sélectionnent les filtres grâce à une norme. Après chaque epoch, les filtres choisis sont mis à 0, mais ne sont pas fixés. Les poids sélectionnés sont entrainés comme tous les autres poids et peuvent être mis à jour. Ce processus est testé sur différents réseaux et, sur Resnet56, les auteurs suppriment 52.6% des poids, avec une perte de précision de 0.24. Nous pouvons aussi noter le gain de 0.3 point de précision, pour 14.7% d'élagage.

Les auteurs de [59] proposent l'utilisation du développement de Taylor afin de minimiser la fonction de perte pour un meilleur choix des groupes de convolutions à supprimer. Une méthode d'élagage itératif qu'ils nomment Tick-Tock améliore le pourcentage de précision. Le modèle supprime 53.3% des convolutions de Resnet-56 avec un gain en précision de 0.33 point sur CIFAR10.

Salama et al. présentent dans [60] une nouvelle technique d'élagage, éliminant des filtres et des neurones selon leur norme L_1. L'idée globale est présentée sur la figure 15.

Dans un premier temps, les normes L_1 de chaque filtre sont calculées, puis ces normes sont normalisées en fonction du nombre de poids au sein du filtre. Les P% filtres avec la plus petite norme (normalisée) sont supprimés. Grâce à cette approche, 47.8% des poids sont supprimés du réseau ResNet-56 et 53% du réseau ResNet-110, sans perte de précision.

Zhao et al. proposent dans [61] une méthode permettant de sélectionner les canaux

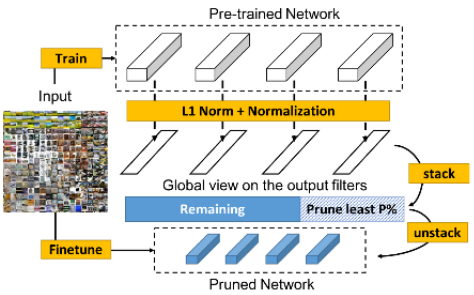


Figure 15: Élagage d'une architecture de [60]. Les filtres sont triés grâce à un calcul de norme L_1 ainsi qu'une normalisation. Puis, $P\%$ des poids sont élagués. Source [60].

(groupes de convolutions s'intéressant à une dimension de l'entrée) à supprimer au sein des réseaux convolutifs. Les auteurs proposent un choix probabiliste, basé sur une méthode bayésienne variationnelle. Entrainée sur CIFAR10, cette méthode permet d'atteindre 45% de canaux supprimés, pour une perte de 0.78 point de précision sur le réseau Resnet-56.

Les auteurs de [62] suppriment plus de 50% des convolutions de VGG16 avec un gain de précision de 0.19. Les auteurs entrainent le réseau de zéro (initialisation des poids aléatoire), et apprennent un contrôleur scalaire pour chaque canal, indiquant si ce canal est utile ou non.

Dans [63] les auteurs fixent un taux d'élagage (au début faible) qu'ils augmentent de façon asymptotique. L'idée est, au début de l'entrainement, de ne pas supprimer les poids pré-entrainés du réseau afin de laisser le temps au réseau de bien apprendre, puis d'élaguer par la suite, le réseau plus fortement.

Suivant cette méthode, les auteurs suppriment 52.6% des convolutions de Resnet-56 (0.85M de paramètres) en perdant 0.47 de précision. Cette méthode a aussi été testée sur Resnet-110 (1.7M de paramètres) et, bien que le nombre de poids triple, les résultats sont équivalents : 52.3% d'élagage pour une perte de précision de 0.58.

Dans [64], Xuanyi Dong and Yi Yang proposent la recherche d'une petite architecture basée sur une distillation des connaissances [65]. L'idée globale est représentée Figure 16.

L'idée est d'apprendre en même temps le réseau et ses hyperparamètres. La taille et le nombre de convolutions vont être appris, et minimisés dans la fonction de perte. Grâce à ce réseau, les auteurs suppriment 93.69% des convolutions sur Resnet56, en perdant 0.77 point de précision sur la référence.

Gao et al. présentent dans [66] une stratégie d'élagage pour les connexions résiduelles,

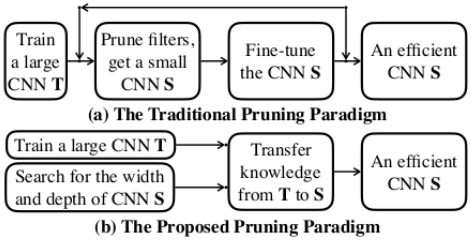


Figure 16: Recherche d'une petite architecture. La taille du réseau ainsi que le nombre de convolutions ne sont plus des hyperparamètres, mais sont minimisés au cours de l'apprentissage. Source [64].

basée sur une nouvelle régularisation qu'ils nomment VACL (Variance-Aware Cross-Layer). Au sein d'un groupe de convolutions, certaines valeurs vont être grandement augmentées, et d'autre grandement diminuées. Comme une moyenne est calculée afin de savoir si le groupe de poids est supprimé ou non, certaines fortes valeurs de poids vont augmenter grandement la moyenne, dans ce cas, le groupe de poids pourra être conservé, ce qui peut ne pas être efficace. Cette normalisation va permettre au poids de garder la même intensité et d'élaguer davantage des groupes inutiles. Grâce à cette stratégie, les auteurs élaguent 65.7% des poids de Resnet-56 avec un gain de 0.31 point sur l'erreur.

Généralement, l'élagage comporte trois parties : l'entrainement, la suppression des poids et l'optimisation des poids restants en ré-entrainant. La phase de ré-entrainement se fait souvent en initialisant les poids restant à leurs valeurs après la phase d'élagage, cela permet aux poids de se réadapter à la disparition des autres poids. Cependant les auteurs de [67] ont prouvé que ré-entrainer le réseau avec une initialisation aléatoire (avec les poids restant) menait, dans certains cas, à un gain en performance. [62] va plus loin et propose d'élaguer les réseaux depuis une initialisation aléatoire. Les auteurs expliquent qu'une initialisation aléatoire permet un choix plus divers des poids. Cette méthode permet de supprimer 93.69 ± 0.28 des poids de Resnet-110, et de gagner 0.20 point de précision par rapport à la référence.

Le tableau 3 récapitule les différentes méthodes d'élagage et résultats de l'état de l'art. A titre de comparaison, nous avons ajouté le réseau EfficientNet-B0 [68]. Il s'agit d'un petit réseau (contenant 4M de paramètres) atteignant une précision de 98.1% sur CIFAR10. La méthode utilisée se place dans le cadre des méthodes de *scaling*, le but est, à l'inverse du pruning, d'augmenter les hyperparamtres de profondeur, largeur et résolution pour une architecture donnée. Les auteurs trouvent une relation entre ces différents paramètres et donnent différentes architectures atteignant l'état de l'art.

Ce n'est pas une méthode de pruning mais il est bon de garder en tête un but à atteindre en terme de paramètres et de performance.

Architecture	# Paramètres	Structuré	Élagage %	Acc %	Gain/Perte
VGG16	138M	Non [49]	30%	93.42%	+0
		Oui [62]	50%	93.63%	+0.19
WideResnet-16	11M	Non [52]	95.36%	94.73%	−0.45
WideResnet-28	36M	Non [53]	95%	95.38%	−0.31
Resnet32	21M	Non [5]	75%	91.8%	−0.7
ResNet-56	24M	Oui [61]	45%	92.26%	−0.78
		Oui [63]	52.6%	93.12%	−0.47
		Non [54]	90%	93.80	−0.25
		Oui [58]	52.6%	93.35%	−0.24
		Oui [60]	47.8%	93.3%	+0
		Oui [55]	23.5%	93.83%	+0.19
		Oui [58]	14.7%	93.89%	+0.3
ResNet-110	42M	Oui [63]	52.3%	93.10%	−0.58
		Oui [60]	53%	92.9	+0
		Oui [62]	40%	93.69 ± 0.28	+0.20
		Oui [55]	23.5%	94.23	+0.41
EfficientNet-B0 [68]	4M	Non	0%	98.1	+0

Table 3: Récapitulatif des différentes méthodes d'élagage de l'état de l'art apprises sur la base de donnée CIFAR10, triées par réseau et par gain en précision.

3.2 Reconnaissance d'émotions

Ces dernières années, les réseaux de neurones profonds ont été optimisés pour la reconnaissance et la classification d'images et de sons. Ils présentent les résultats les plus avancés dans ces domaines. Dans cette partie, les modèles concernant la reconnaissance d'émotions sont présentés. Dans un premier temps, quelques méthodes n'utilisant pas d'apprentissage profond sont décrites. Malgré les différences avec notre approche, nous pensons qu'il est important de garder une vue globale de toutes les solutions présentées. Cela peut donner des informations utiles et des comparaisons. Puis, les méthodes d'apprentissage profond, par rapport à la taille des données d'entrainement, sont introduites. La table 4 récapitule les différentes bases de données, ainsi que leurs tailles, utilisées dans cette section.

Le lecteurs intéressé pourra consulter [69], [70] et [71], qui proposent une étude plus

complète sur le domaine de la reconnaissance d'émotions.

3.2.1 Approches n'utilisant pas d'apprentissage profond

Habituellement, dans la reconnaissance d'émotions, et en particulier dans la reconnaissance d'émotions faciales (FER), trois étapes majeures sont associées : (i) détection de visage, (ii) extraction de caractéristiques et (iii) classification des émotions. Ko décrit dans [69] un aperçu de ces méthodes et nous soulignons, dans la suite, quelques unes d'entre elles.

Siddiqi et al. [72] utilisent, afin trouver les caractéristiques importantes d'une image faciale, des analyses en composantes principales (PCA, cf. annexe .3.1) et indépendantes (ICA). Par la suite, les auteurs utilisent un modèle de Markov caché comme un module de reconnaissance, sur les caractéristiques extraites. Cette méthode atteint une précision de 98% sur les six émotions de base, sur une base de données privée.

Dans [73] les auteurs révèlent les régions importantes du visage afin de détecter chaque émotion, en utilisant des projections pondérées sur des motifs binaires locaux (LBP). Ils atteignent, par la suite, une précision de 98.51% sur la base de données Jaffe [74] en utilisant des SVM pour le processus de classification.

Biswas et Sil proposent dans [75] une méthode alliant les transformées de Contourlet et Curvelet pour l'extraction de caractéristiques. Utilisant par la suite une SVM, ils atteignent une précision de 98.63% sur les six classes de base.

Marcelo et al. [76] introduisent une méthode discriminative afin de différencier les six émotions de base. En utilisant un classifieur SVM sur des LBP et des descripteurs locaux de Weber, ils atteignent une précision de 99% en testant sur la base de données Jaffe [74].

Ces méthodes sont très performantes, cependant l'extraction de caractéristiques doit être conceptualisée par un expert au début et ne peut plus être optimisée durant l'étape de classification.

Ces algorithmes sont toutefois, encore beaucoup utilisés de nos jours, en raison de leur faible complexité et temps d'exécution.

3.2.2 Approches basées sur des petites bases de données.

Rao et al. présentent dans [77] une représentation des six émotions de base en utilisant un réseau siamois et le réseau Alexnet [78] pour l'extraction de caractéristiques. Le modèle

est présenté en figure 17. Il a été entrainé sur la base de données Nimstim [31].

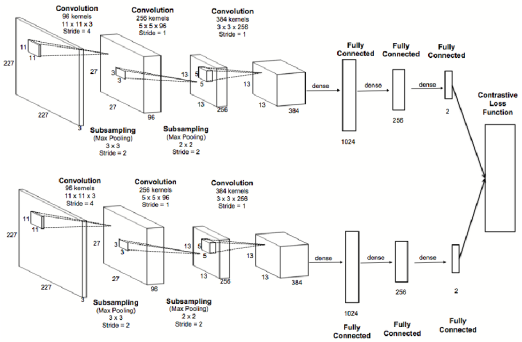


Figure 17: Modèle siamois, utilisant le réseau AlexNet, présenté dans [77]. Le réseau prend en entrée un couple d'images provenant de la base de données Nimstim

Une ACP est utilisée pour la réduction de dimension et la représentation vectorielle. Cette représentation est présentée dans la figure 18.

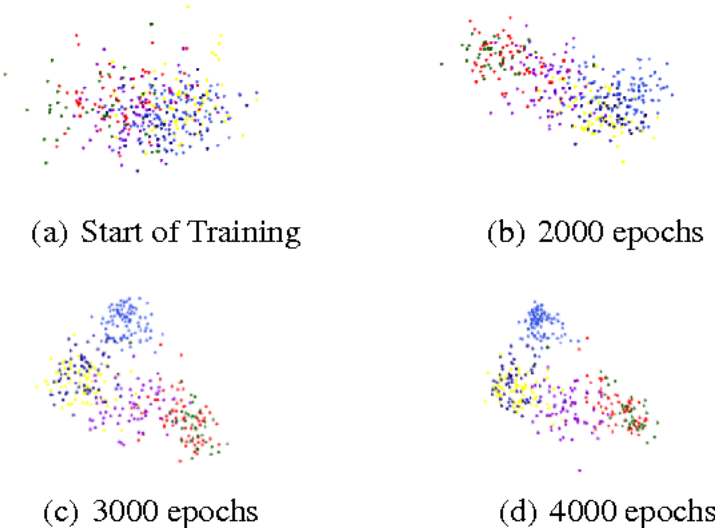


Figure 18: Représentation vectorielle, obtenue par ACP, des 6 émotions de base créée dans [77].

La comparaison avec notre modèle est très difficile car les auteurs ne donnent aucune information sur la précision et aucune analyse prouvant la justesse de leurs représentations vectorielles.

Dans [79], les auteurs utilisent un réseau profond entrainé sur la base de données audio-

visuelle Emotiw 2014 [80]. L'idée est de combiner un réseau convolutif pour l'image, un réseau *deep belief* [81] pour le son, un auto-encodeur pour apprendre les caractéristiques spatio-temporelles, et enfin un réseau peu profond pour extraire le mouvement des lèvres. Les auteurs obtiennent une précision de 47.7% sur les six émotions de base.

Ng et al. présentent dans [82], un réseau convolutif entrainé grâce à une combinaison de transfert de connaissance présentée en figure 19.

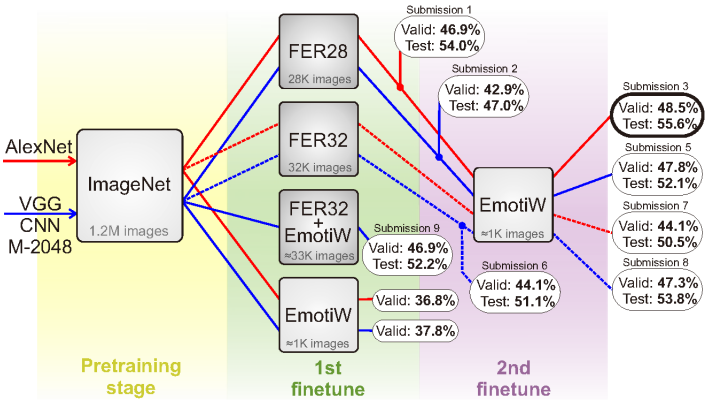


Figure 19: Apprentissage préalable sur différentes base de données. L'entraînement sur ImageNet, Fer28 puis EmotiW donne les meilleurs résultats. Source [82]

Grâce à la combinaison des trois stratégies de finetuning, testées sur 4 bases différentes, le réseau obtient un score de 55.6% sur la base de données EmotiW 2015 [15].

Kimet et al. introduisent dans [83] un réseau de convolution plus classique, présenté en figure 20, entrainé et testé sur la base de données SFEW 2.0 [84]. Ils atteignent une précision de 61.6% sur 7 classes.

Hasani et Mahoor présentent dans [85] un modèle convolutif profond 3D basé sur Inception-Resnet [4] et des couches de LSTM. L'approche globale est décrite dans la figure 21.

L'idée est d'utiliser une module Inception-3D afin de traiter, dans une première partie, des vidéos, et dans une seconde partie, des points caractéristiques du visage (*facial landmarks*).

Puis, comme les données sont des données temporelles, un LSTM ainsi qu'un réseau de neurones complètement connecté sont utilisés pour la classification.

Cette approche atteint une précision de 77.50% sur 6 émotions, en utilisant les vidéos et les repères faciaux de la base de données MMI [86] ; ainsi que 77.42% sur 5 émotions,

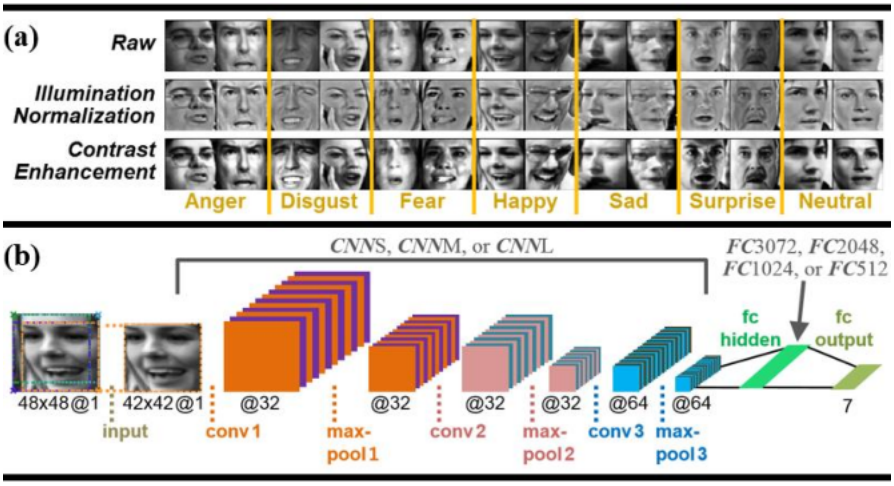


Figure 20: Présentation du réseau utilisé dans [83], CNNS, CNNM et CNNL sont des couches de convolution de différentes tailles (*Small*, *Medium* et *Large*). Source [83].

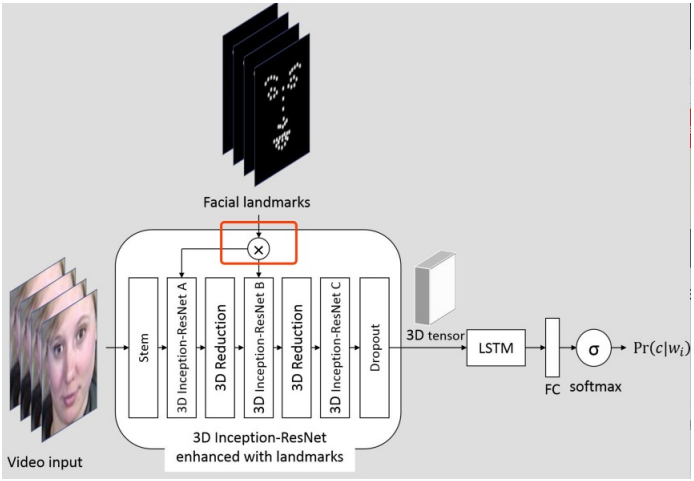


Figure 21: Approche décrite dans [85]. Les vidéos sont traitées en deux parties, un module inception traite en même temps les videos et les points caractéristiques du visage préalablement trouvés.

en s'entrainant et en testant sur la base de données Fera [87].

Globalement, les précisions que nous trouvons dans la littérature, pour l'utilisation de l'apprentissage profond sur des bases de données restreintes, sont très basses. Apprendre des modèles profonds sur de petites bases de données est un processus très complexe, et

les réseaux convolutifs classiques restent très limités dans ce domaine.

3.2.3 Approches basées sur grandes bases de données.

Même si ces méthodes ne sont pas semblables à notre approche, cela peut être utile de comparer les résultats avec l'état de l'art sur des grandes bases de données.

Hossain et Muhammad présentent dans [88] un modele convolutif multimodal entrainé sur une grande base de données de vidéos qu'ils ont créée et atteignent une précision de 86.4%.

Dans un premier temps, les auteurs traitent le son avec l'échelle Mel ; et transforment les images en niveau de gris puis les normalisent. Les différents traitements sont présentés en figure 22.

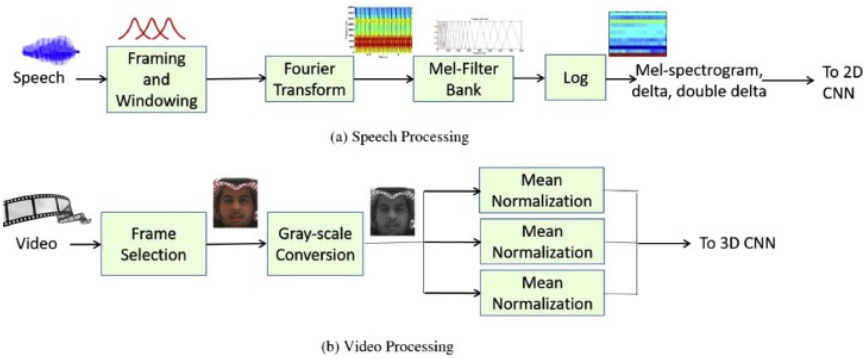


Figure 22: Les données audios et visuelles sont dans un premier temps prétraitées. Puis les caractéristiques sont apprises séparement grâce à l'utilisation de CNNs. Source [88]

Puis comme présenté sur la figure 23, les auteurs traitent indépendemment la recherche de caractéristiques avec un CNN 2D pour les sons, et un CNN 3D pour les images. Puis ils utilisent une Extreme Learning Machine [89] (ELM) sur le vecteur de caractéristiques, pour enfin le classifier grâce à un SVM.

Veminapli et Agarwala [90] ont utilisé un réseau triple, combinant le début de l'architecture Facenet [41] et DenseNet [91], pour la représentation vectorielle. Cette association est présentée sur la figure 24.

Les auteurs introduisent une nouvelle fonction d'erreur et obtiennent une précision de

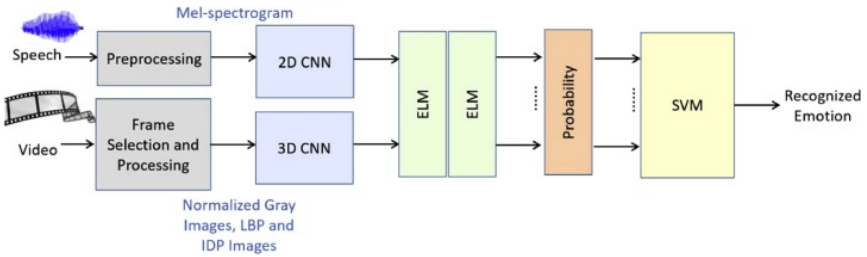


Figure 23: Utilisation d'une ELM et d'un SVM pour le processus de classification. Le modèle est appris sur une grande base de données créée par les auteurs. Source [88].

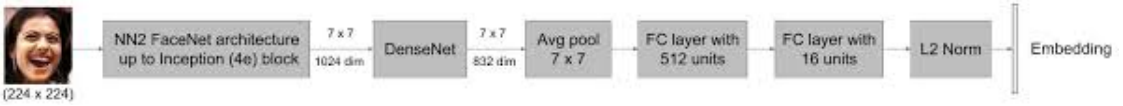


Figure 24: Réseau convolutif utilisé pour la représentation vectorielle, combinant le début de l'architecture Facenet [41] puis DensNet [91]. Source [90]

88.6% sur 8 émotions. Ce réseau à été entrainé sur la base de données AffectNet [92] sur 150K images différentes. Afin d'entrainer avec leur nouvelle fonction de perte, la base d'entrainement a été séparée en 3 types de triplets :

— Ceux contenant trois images de la même classe.

— Ceux avec deux images de classe différentes.

— Ceux avec trois images de classe différentes.

Harár et al. utilisent un réseau profond convolutif [93] entrainé sur la base de données sonores *Berlin Database of Emotional Speech* [94]. Après les couches de convolution, comme montré sur la figure 25, les auteurs appliquent d'un coté un pooling moyen et d'un autre un pooling max. Puis les deux résultats sont concaténés afin d'être classifiés par des couches complètement connectées.

Cette approche atteint une précision de 96.97% sur trois classes : colère, tristesse, neutre.

Zhao et al. [95] présentent une architecture générative (qu'ils nomment FERSnet), contenant un classifieur pour la reconnaissance d'émotions (FER) et une structure encodeur décodeur pour la synthèse d'image (FES) . Elle est décrite sur la figure 27.

Les auteurs introduisent aussi un nouveau module basé sur les GRUs, permettant de garder en mémoire, ou d'oublier des éléments. Elle est présentée en figure 27. Grâce à cette architecture, le réseau obtient un score de 97.85% sur les images de la base de données CohnKanade Extended (CK+) [96].

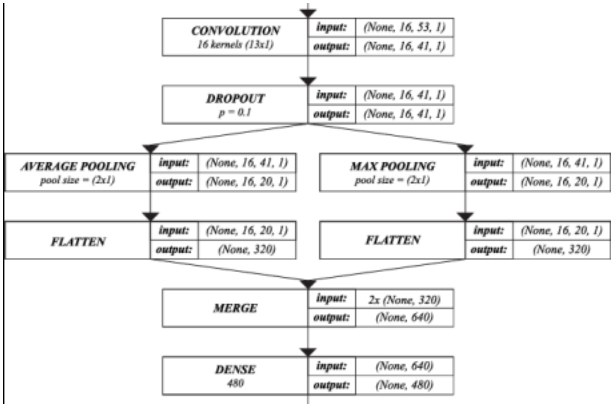


Figure 25: Combinaison du pooling max et moyen après les couches de convolution, suivi d'une concatenation. Source [93]

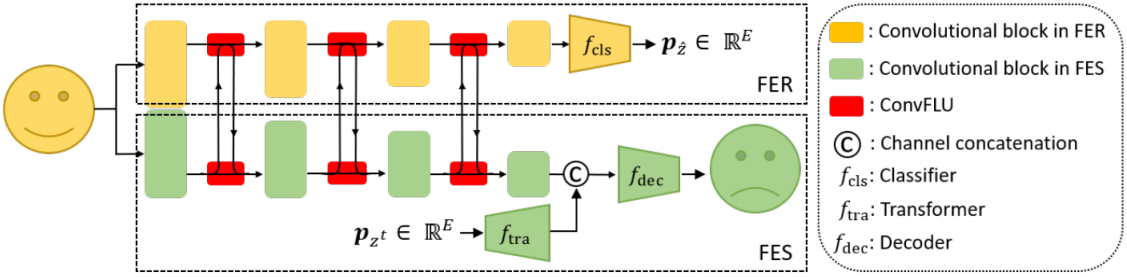


Figure 26: Architecture FERSnet utilisant un classifieur pour la reconnaissance d'émotions (FER) ainsi qu'une structure encodeur décodeur pour la synthèse d'image (FES). Source [95]

Heysem Kaya et al. présentent un modèle profond [97] entrainé sur la base de données *en conditions réelles*, EmotiW-2015 [15], atteignant un précision de 54.55%. Le modèle a aussi été testé sur la base de données audio-visuelles CohnKanade Extended (CK+) [96], pour laquelle une précision de 98.47% est atteinte. L'approche globale est présentée dans la figure 28.

Le processus est séparé en trois parties, dans un premier temps les images sont traitées grâce a un réseau convolutif classique pré-entraîné sur VGG FACE [98] et entrainé sur FER2013 [99]. Parallèlement, les images de visage sont traitées avec les descripteurs visuels classsiques d'imagerie : LBP [100], HOG [101], SIFT [102], afin de trouver les caractéritiques importantes des images manuellement.

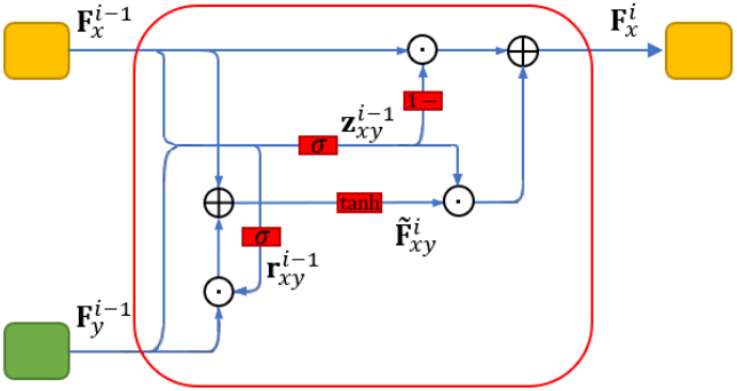


Figure 27: ConvFLu, nouveau module basé sur les GRUs, permettant une meilleure mémoire des éléments importants .Source [95]

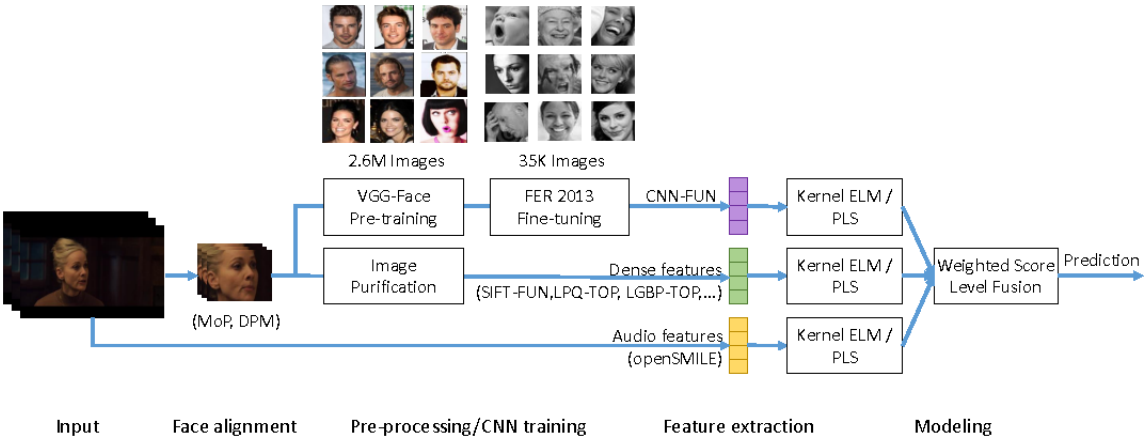


Figure 28: Approche Score Fusion [97]. Les images et les sons sont traités parallèlement et classifiés grâce à trois différents ELMs. Les trois vecteurs résultant sont ensuite combinés par un processus de fusion. Source [97]

L'audio est aussi traitée séparément et les caractéritiques sont trouvées grâce au programme OpenSmile [103]. Un programme très utilisé en reconnaissance d'émotions, permettant la détection de caractéristiques capitales. Puis les trois vecteurs de caractéristiques résultants sont traités par une *Extreme Learning Machine [89]* pour la classification.

Les trois vecteurs sont ensuite combinés grâce à un processus nommé fusion, ou des poids sont ajoutés (et appris) sur les vecteurs, afin de déterminer quel processus est le plus important.

Zhanpeng et al. [104] introduisent un réseau profond de convolutions atteignant une précision de 98.9% sur les 6 émotions de la base de données The Extended CohnKanade [96]. Afin d'améliorer leurs modèle, les auteurs utilisent un réseau siamois [105] pour trouver les caractéristiques partagées dans une classe. Ils prouvent que l'ajout de ce réseau siamois améliore le processus de classification.

Niu et al. [106] proposent une méthode afin d'augmenter les données de la base d'enregistrements audios IEMOCAP [107]. Cette méthode est basée sur le concept de symétrie des lentilles convexes. Le but est de créer des spectrogrammes plausibles, à partir des spectrogrammes existants. Puis, plus classiquement, un réseau profond de convolutions est entrainé et atteint une précision de 99.25%.

Bien que l'apprentissage profond offre de bons résultats pour la détection d'émotions, nous pensons que des travaux doivent être effectués dans le domaine de l'apprentissage sur base restreinte. Nous nous sommes, par conséquent, limités à des petites bases de données audio-visuelles, dans le but de créer un réseau performant grâce à des techniques d'apprentissage *few shot* et d'élagage.

Le tableau 4 décrit les différentes bases de données utilisées dans l'état de l'art, en fonction du nombre de données.

Nom	Catégorie	Nombre
Jaffe [74]	Image faciale	213
Nimstim [31]	Image faciale	672
SFEW 2.0 [84]	Image faciale	700
FERA [87]	Image faciale	7000
MMI [86]	Image faciale	11500
FER2013 [87]	Image faciale	35887
The extended CohnKanade [96]	Image faciale	72939
Affect [92]	Image faciale	150000
Berlin Database of Emotional Speech [94]	Enregistrement Audio	271 de 783 s
EmotiW 2015 [15]	Audio-Visuelle	1645 de 300-5400 ms
IEMOCAP [107]	Audio-Visuelle	10000 de 3-15 s

Table 4: Récapitulatif des bases de données utilisées dans l'état de l'art.

4 Élagage

Nous nous sommes interessés, dans un premier temps, à différentes techniques d'élagages dans le but d'obtenir un réseau plus petit et plus performant.

L'approche globale consiste à utiliser ces techniques afin d'obtenir le plus petit réseau possible (sans perte de performance) pour permettre l'apprentissage sur des bases de données multimodales restreintes, grâce à des techniques d'apprentissage *few shot*. L'idée est de détecter des émotions, sur des bases de données d'images faciales et de phrases sonores, en utilisant un réseau correspondant, préalablement élagué.

L'approche globale est schématisée dans la figure 29.

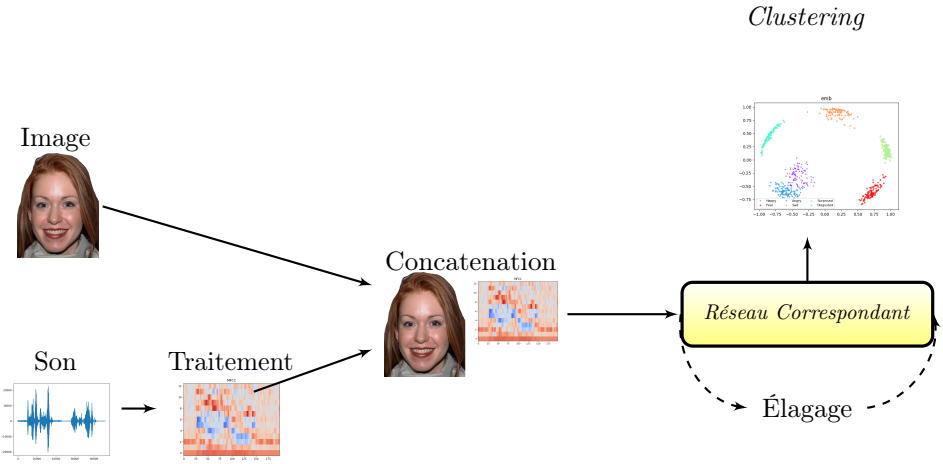


Figure 29: Approche proposée : un réseau corréspondant, préalablement élagué, traite les données et calcule un plongement vectoriel. Une classification non supervisée est ensuite effectuée dans l'espace ainsi créé.

Nous nous concentrons, dans ce chapitre, sur la partie élagage.

4.1 Approche

4.1.1 Motivations

Face à l'essor de l'utilisation des téléphones portables, des besoins en robotique et plus généralement des besoins d'optimisation, il est essentiel que les modèles d'apprentissage profonds évoluent vers plus de portabilité. Parmi les techniques possibles, nous nous sommes tournés vers des approches d'élagage (*pruning* [20]). L'idée originale est d'enlever les poids inutiles pour la classification, pour avoir un modèle plus léger et plus rapide. Le principe est schématisé sur la figure 30 et décrit plus en détail dans l'algorithme 1.

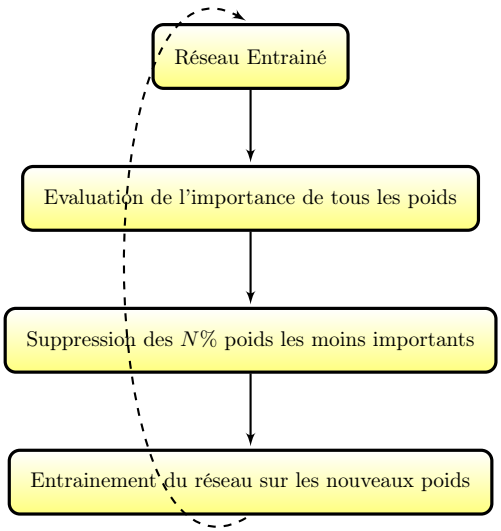


Figure 30: Concept du pruning classique.

Dans un premier temps, le réseau est entrainé, puis une évaluation de l'importance de chaque poids est effectuée. L'hypothèse dans [20] est que plus le poids est grand, plus il aura d'impact dans les couches suivantes.

algorithme 1 Pruning de $P\% > 0$ des poids **W** du réseau Res

Entrées : P : pourcentage voulu de pruning ; **W** : poids du réseau Res

Entrainement de Res
Pour Tous les poids w dans **W Faire :**
    Calculer la pertinence des poids w

Mettre à 0 les P poids les moins pertinents

Entrainement de Res

Les $p\%$ poids les moins importants sont supprimés, puis le réseau est ré-entrainé (*finetuned*) pour adapter les autres poids au nouveau modèle.

Description du problème

Durant l'élagage, il est nécessaire de toujours trouver un équilibre entre le pourcentage de poids supprimés et la précision du modèle. Si l'objectif est la portabilité, le réseau sera élagué un peu plus, au dépend de la précision. Si par ailleurs, la précision est le plus important, seulement les poids n'impactant pas la précision seront supprimés. Dans ce deuxième cas, le choix des poids est majeur dans le processus d'élagage.

Nous avons souligné de nombreux problèmes dans l'approche de base :

— Il faut entrainer le réseau au préalable.
— Le pourcentage d'élagage maximal (sans influencer la précision) est inconnu. Il faut donc répéter la méthode un certain nombre de fois ce qui est très long (il faut ré-entrainer à chaque fois) et très coûteux.
— Si un poids est choisi pour être supprimé, il est enlevé et il n'y aura aucun moyen de le récupérer.
— Le choix de l'importance de chaque poids est crucial, et se baser seulement sur la valeur du poids n'est pas optimal.
— Le choix d'un hyperparamètre afin de contrôler le pourcentage d'élagage est empirique.

Les techniques d'élagage ont bien entendu évolué depuis [20], cependant de nombreux problèmes persistent. Nous avons exploré des pistes de réponse pour les trois derniers problèmes.

Contributions

Nous présentons un modèle d'élagage sans hyperparamètre, dans la fonction de perte, pour le contrôle des poids. Le réseau n'est pas obligé de supprimer des poids afin de minimiser l'erreur. Nous proposons une nouvelle manière de choisir les convolutions en présentant un réseau rival influençant le choix des poids. Grâce à ce réseau, l'état de l'art est atteint sur la base de données CIFAR10.

4.1.2 Approches

Cette section présente les différentes approches effectuées basées sur le réseau rival. Notre technique d'élagage se fonde sur l'utilisation de masques binaires.

4.1.2.1 Masques

L'idée est d'utiliser des masques binaires dont le but est de contrôler le choix des poids. Si la valeur du masque est égale à 0, le poids est enlevé, si elle est égale à 1 le poids est gardé.

Le résultat est donc le produit de Hadamard du masque par la matrice locale des poids. Les poids multipliés par 0 seront les poids supprimés.

Pour le choix des poids à supprimer, plusieurs méthodes s'offrent à nous. Nous pouvons utiliser :

— une valeur binaire pour chaque poids au sein d'une convolution. C'est un processus très long (deux fois plus de paramètres à apprendre) et nécessitant une base d'apprentissage très grande.
— une valeur binaire pour chaque convolution. Cette méthode est moins longue, mais nécessite encore beaucoup de paramètres à apprendre.
— une valeur par sortie sur une couche. Cette valeur va contrôler toutes les convolutions permettant d'atteindre cette sortie. Sur une couche à $\mathbf{s} = (s_1, ..., s_n)$ sorties, k convolutions et la profondeur j pour chaque sortie, la valeur binaire, que nous nommons c_1, va contrôler le groupe de kj convolutions pour la sortie 1. En pratique, une valeur va être apprise puis seuillée pour obtenir une valeur binaire. Si $c_s > seuil$ alors les convolutions sont gardées ($\times 1$) sinon elles sont supprimées ($\times 0$).
— une valeur binaire par couche, très rapide pour déterminer si une couche entière est inutile ou non. Mais cette approche ne peut pas supprimer les convolutions à

l'intérieur des couches et déterminer les convolutions inutiles.

De nos jours, les réseaux possèdent un grand nombre de couches, chacune contenant un nombre différent de convolutions. Le nombre de convolutions à supprimer sera différent dans chaque couche. La quatrième option n'est donc pas optimale car elle contrôle pour chaque couche un trop grand nombre, et un nombre trop différent de convolutions. La troisième option nous a paru être un bon compromis entre temps d'apprentissage, performance et nombre de convolutions contrôlées par une valeur, que l'on nomme contrôleur.

Ces masques sont appliqués au sein de chaque couche de convolution (figure 31), avant de donner le résultat à la couche suivante. Nous notons m un masque.

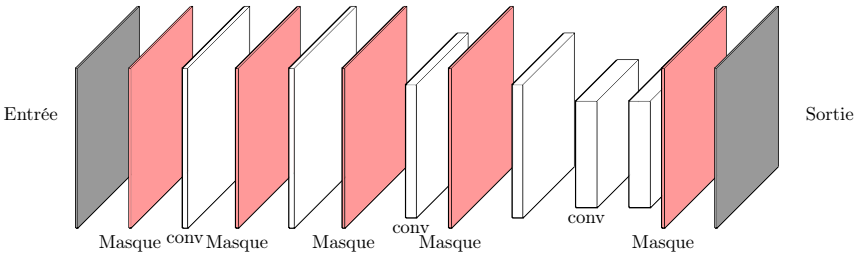


Figure 31: Masques appliqués sur chaque couche de convolutions, une valeur binaire contrôle toutes les convolutions utilisées pour une sortie.

Durant l'apprentissage, les masques vont être appris. Souvent, si l'on veut forcer le modèle à supprimer davantage de poids, la norme du masque $-\alpha||m||$ est ajoutée à la fonction erreur, pour favoriser plus ou moins l'élagage.

α est un hyperparamètre, choisi empiriquement. Si α est petit, la norme du masque ne va pas beaucoup changer, s'il est grand, le masque va être minimisé et le modèle va être davantage élagué, souvent au dépend de la précision. Un de nos buts est de s'affranchir de ce paramètre

4.1.2.2 Réseau rival utilisant une base de données opposée

Idée d'origine

Nous appelons un réseau rival, un réseau qui prend les poids supprimés (inutilisés) du réseau original $(1 - masque)$.

Notre première idée, décrite dans la figure 32, était d'apprendre le même réseau sur une base opposée à la base de données de départ. On appelle base opposée, une base

de données contenant des motifs (formes) inexistants de la base d'origine, qui seront détectés par les convolutions inutiles pour la base d'origine. L'idée est de supprimer ces convolutions inutiles à notre problème d'origine afin d'élaguer le réseau.

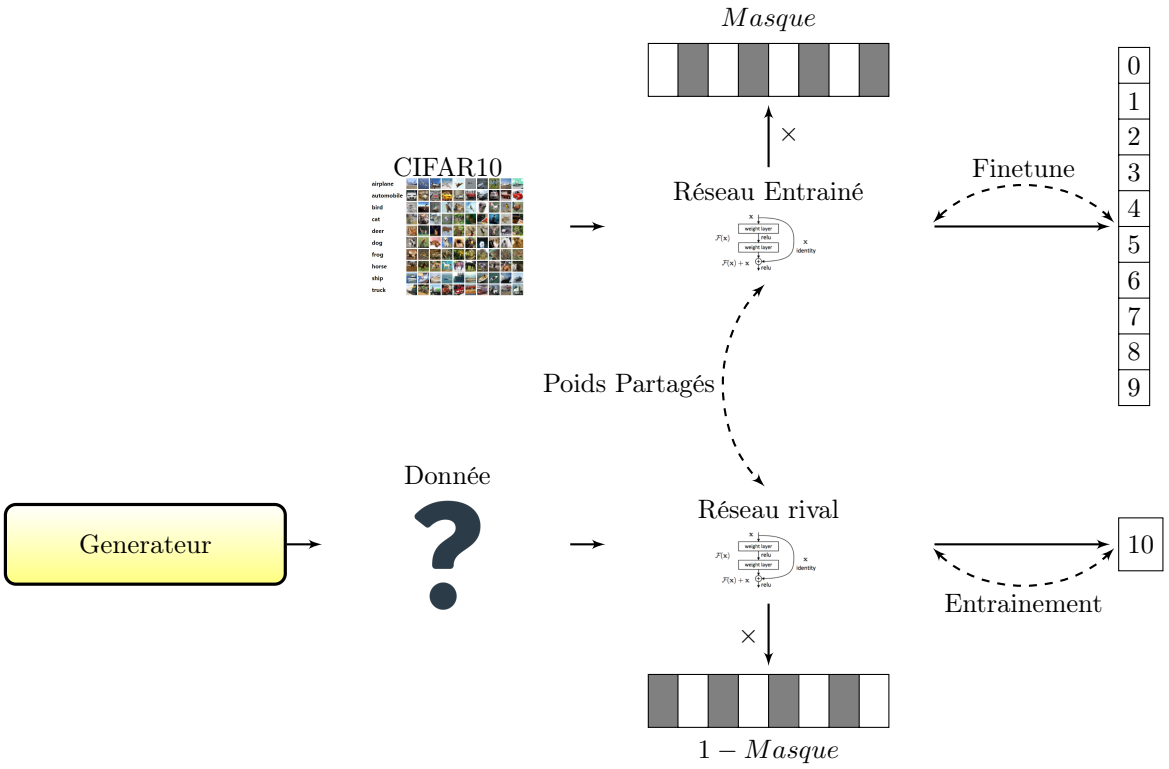


Figure 32: Réseau rival pour base de données opposée. Le générateur va créer une base opposée au problème de départ qui nécessitera l'utilisation des convolutions inutilisées du modèle. Le réseau rival apprend sur la base de données créée par le générateur et utilisera les convolutions inutiles pour la classification de la base de données de base (CIFAR10). Ces convolutions vont par la suite être supprimées dans le modèle de base. Les masques sont des valeurs binaires, apprises durant l'entraînement, indiquant le choix d'élaguer ou non les convolutions. Le réseau rival utilisera les convolutions élaguées du premier réseau.

— Le premier réseau utilise les poids des convolutions contrôlées par le masque (*masque*).
— Le réseau rival utilise les poids supprimés du premier réseau, les convolutions contrôlées par $(1 - masque)$.

— Les poids du classifieur ne sont pas partagés.

Le réseau apprend sur les deux bases en même temps. Le but est de partager les convolutions pour classifier les deux bases opposées. L'idée est de trouver les convolutions inutiles pour la base de données de départ, afin de les enlever définitivement. Le réseau a déjà appris sur les données de base, il va juste être ré-entrainé lorsque des poids sont enlevés.

Pour un bon fonctionnement du modèle, il faut que la base choisie soit opposée à la base de départ, qu'il y ait des formes nouvelles par rapport à la base d'apprentissage. Nous avons choisi de créer une telle base avec un réseau générateur.

Dans la figure 32 :
— Le but du premier réseau est de classifier les images de la base de données CIFAR10 [26], labélisées de 0 à 9.
— Le but du générateur est de créer une base de données opposée à CIFAR10.
— Le but du deuxième réseau (réseau rival) est de classifier les données opposées créées par le générateur que nous labélisons 10.

Générateur

Nous avons utilisé le générateur du réseau BigGan128 [28], que nous avons adapté à des images de taille 32×32.

Dans ce modèle le générateur peut créer les images qu'il souhaite, tant que le réseau rival arrive à les classifier. Il va donc créer des images très simples (figure 33), faciles à classifier et ne nécessitant que peu de poids. Le réseau ne va donc pas supprimer beaucoup de poids. Il faut ajouter de la complexité aux images créées.

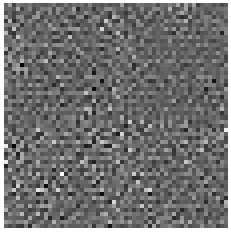


Figure 33: Image simple créée par le Gan, sans ajout de complexité. Les poids des générateurs et des classifieurs sont partagés.

Ajout de complexité .

L'idée est de créer une base de données opposée, possédant une complexité équivalente aux images de base. Dans la suite, nous utiliserons le mot *complexité* comme une mesure de formes, de couleurs, de coins, d'ombres, etc. En somme, toutes les caractéristiques présentes dans une image, utiles à l'apprentissage et détectées par les convolutions.

Le principe est schématisé sur la figure 34. L'idée est d'utiliser un encodeur variationnel E_2 (voir annexe .2.1) apprenant les caractéristiques importantes de l'image. Puis d'utiliser un réseau de neurones dont le but est de générer des images opposées aux données de base (label=10). Afin de forcer un certaine complexité dans la base de données générée, un deuxième encodeur variationnel E_1 est ajouté. Comme les poids des générateurs G sont partagés, des images différentes possédant une complexité semblable aux images de départ seront générées.

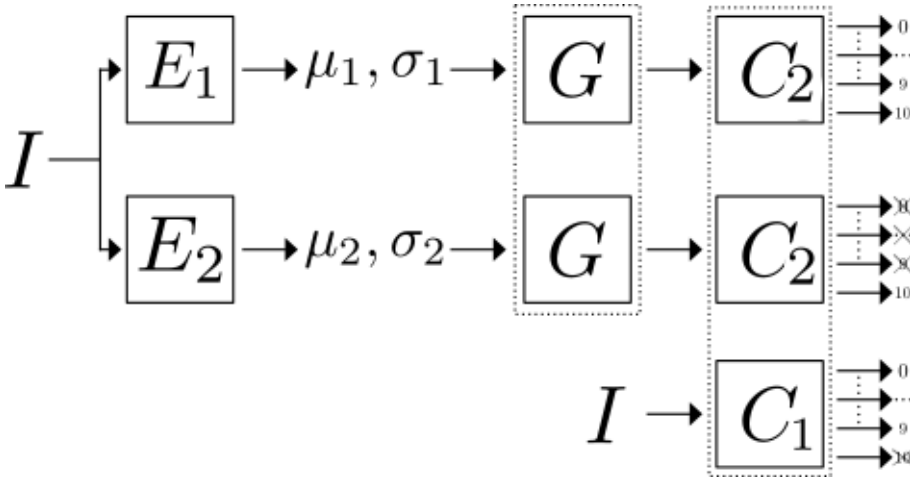


Figure 34: Ajout d'un réseau imposant une certaine complexité aux images créées. E_1 et E_2 sont des encodeurs variationnels, apprenant respectivement les distributions de paramètres (μ_1,σ_1) et (μ_2,σ_2) utilisées pour la génération d'images par le générateur G. C_1 est le réseau de base apprenant à classifier les images de départ. Par ailleurs C_2 classifie les images générées par G.

Dans la pratique, afin de forcer le générateur à créer cette complexité, il faut que les images créées puissent à la fois être détectées par le classifieur C_2 et qu'il arrive à classifier correctement les labels de base (0 à 9) sur les images générées. Le réseau de base C_1 apprend toujours sur les images de la base de données originale. Pendant que le réseau rival apprend sur les images créées par le générateur. Dans la même veine que précédemment,

les deux réseaux C_1 et C_2 vont se partager les poids du contrôleur qui génère les masques $masque$ et $(1 - masque)$.

L'idée est, comme précédemment, de détecter les convolutions inutiles à l'apprentissage du modèle de départ.

La figure 35 présente quatre images créées par le générateur.

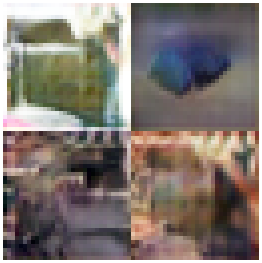


Figure 35: Quatre images créees par le générateur, avec ajout de complexité.

Le problème majeur de ce modèle réside dans l'utilisation du générateur. Pour obtenir des images correctes, le générateur doit être entrainé au préalable. Et même si des techniques de transfert de connaissance sont utilisées, le modèle global a un processus d'apprentissage très long.

4.1.2.3 Réseau rival utilisant la même base de données

Suite à ce constat, nous avons décidé d'enlever le générateur et d'élaguer les deux réseaux sur la même base de données. Les deux réseaux vont apprendre dans le but de classifier les mêmes images.

Rival

Un schéma du modèle est donné en figure 36 .

Durant l'apprentissage, les deux réseaux vont se partager les convolutions redondantes. On nomme convolutions redondantes, des convolutions qui ont le même but, répétant juste des informations qui ne sont pas utiles au réseau. Le but est de se partager la moitié des poids puis, si la perte en précision n'est pas élevée, recommencer avec le meilleur réseau élagué. Les deux modèles sont en compétition pour le choix des poids.

Comme les deux modèles ont le même but (classification de CIFAR10) ils vont tous les

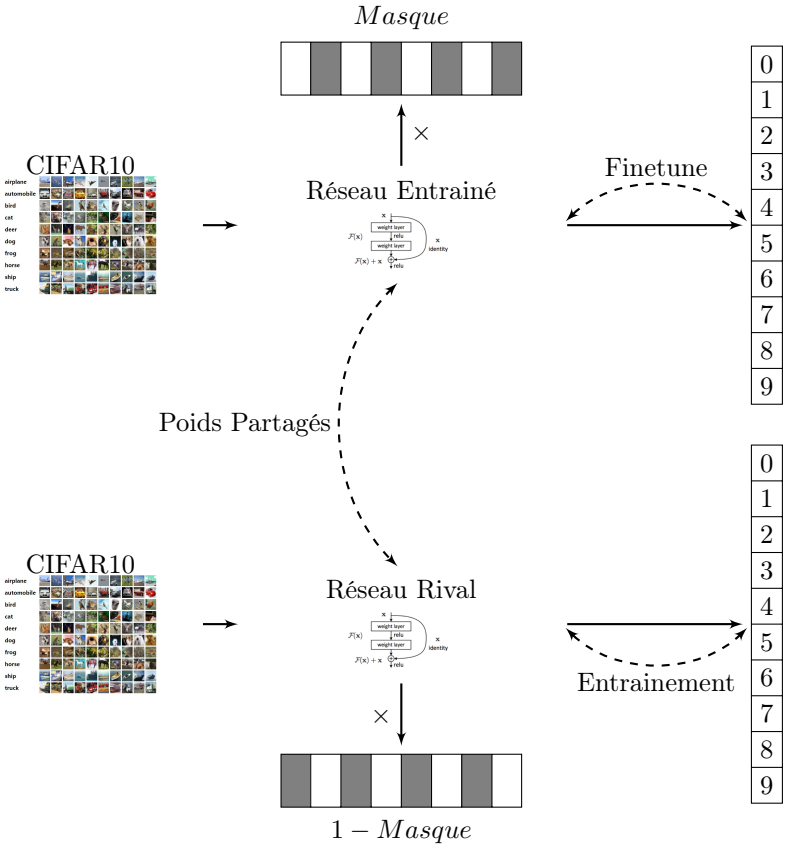


Figure 36: Elagage avec un réseau rival apprenant sur la même base, et dans le même but. Le but des deux réseaux est de classifier la base de donnée CIFAR10. Des masques binaires vont contrôler le choix de supprimer une convolution ou non. Le réseau adverse utilise les convolutions élaguées du réseau de base (1-Masque). Au cours de l'apprentissage, les convolutions redondantes vont être séparées dans les deux modèles. A la fin de l'apprentissage, le réseau nous convenant le mieux (meilleure précision ou meilleur élagage) sera choisi.

deux choisir des convolutions utiles à la classification, mais des convolutions différentes (imposé grâce au masque et 1-masque). Le partage de ces convolutions va permettre la détection des convolutions dites redondantes et donc leur suppression par la suite.

Rival et union

Afin de permettre au réseau de récupérer des poids perdus au cours de l'élagage, nous avons partagé un pourcentage des contrôleurs au sein du masque. Nous nommons cette partie *union* car les poids vont être utilisés par les deux modèles. Si une convolution est unique et sert pour l'apprentissage, elle va être placée dans l'union. Cela permettera un meilleur choix des convolutions, et une meilleure précision sur la base de données de départ des deux réseaux.

Un schéma du modèle est donnée en figure 37

Le masque union $masque_{union}$, est appris durant l'apprentissage. Nous l'initialisons aléatoirement.

Maintenant que l'architecture apprend correctement et rapidement, et afin d'améliorer les résultats, nous nous sommes concentrés plus en détails sur le choix de ces masques binaires.

Utilisation d'un contrôleur et d'une distribution de Bernoulli

Jusque là, les poids étaient choisis par seuillage, ce qui est une contrainte très forte. De plus, durant l'apprentissage, si une convolution est choisie pour être supprimée, elle ne peut plus être réutilisée dans la suite.

Nous avons donc opté pour une stratégie probabiliste de choix des poids.

Distribution de Bernoulli

La distribution de Bernoulli est une distribution discrète de probabilité. Elle prend la valeur 1 avec la probabilité p et 0 avec la probabilité $q = 1 - p$.

En d'autres termes :

$$P_B(X = x) = \begin{cases} p & \text{si } x = 1, \\ q = 1 - p & \text{si } x = 0 \\ 0 & \text{sinon} \end{cases}$$

L'espérance mathématique d'une variable aléatoire de Bernoulli vaut p : $\mathrm{E}(X) = p$

Sa variance vaut : $\mathrm{Var}[X] = pq = p(1 - p)$

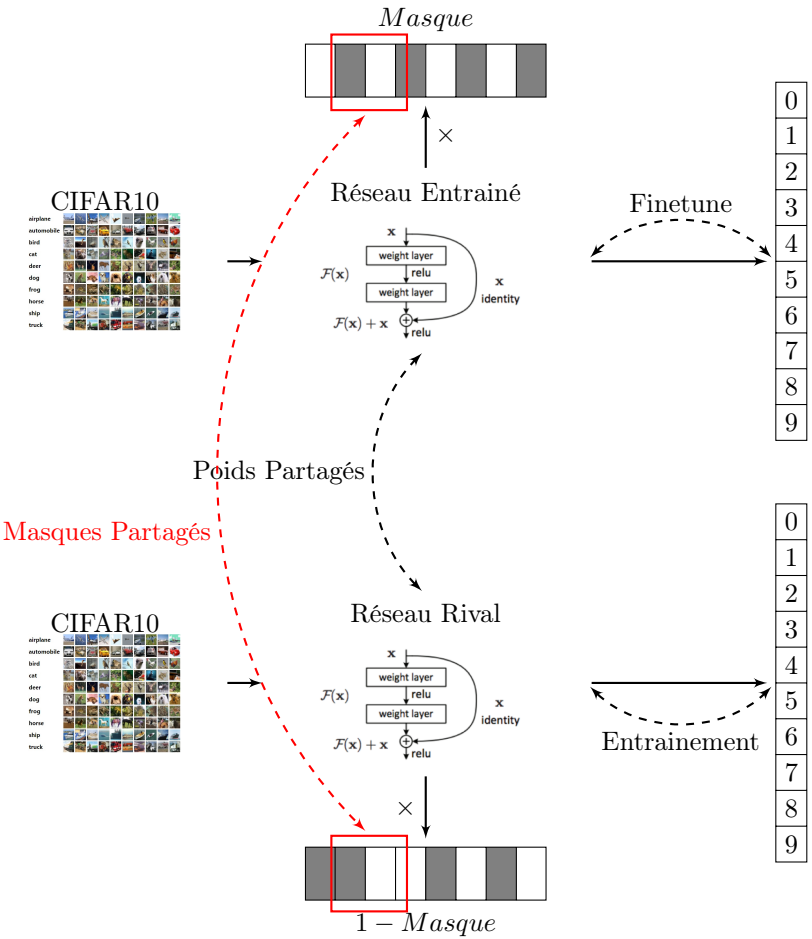


Figure 37: Ajout d'une union sur les masques. Une partie des masques est préalablement choisie et partagée dans les deux réseaux, on la nomme union. Si une convolution est capitale pour un bon processus d'apprentissage, elle sera choisie par les deux réseaux. Le principe est le même que pour la figure 36, les deux réseaux ont le même but, et vont se partager les convolutions redondantes.

Architecture

Le principe du choix des convolutions est décrit dans la figure 38.

Dans cette figure, on note :

— i : ième couche du réseau

— k : nombre de convolutions qui s'appliquent sur ces entrées

— j : profondeur

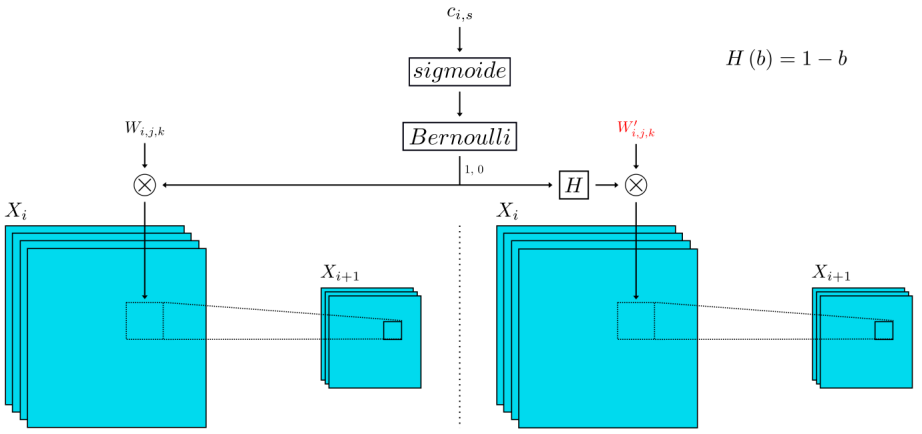


Figure 38: Gauche : modèle de base ; Droite : Modèle rival. Avec X_i : entrées de la couche i, $W_{i,j,k}$: k-ième convolution de la profondeur j de la couche i, du réseau original. Ici des contrôleurs $c_{i,s}$ sont appris, une fonction sigmoide est appliquée afin de transformer les contrôleurs entre [0,1], puis une distribution de Bernoulli est utilisée afin de déterminer les choix des convolutions. Le réseau adverse utilisera les convolutions inutilisées du réseau de base grace à la fonction H.

- s : nombre de sorties
- X_i : entrées de la couche i
- $W_{i,j,k}$: k-ième convolution de la profondeur j de la couche i, du réseau original.
- $W'_{i,j,k}$: k-ième convolution de la profondeur j de la couche i, du réseau rival.
- $c_{i,s}$: réel, contrôlant les kj convolutions de la couche i. Il y a pour une couche, s contrôleurs.
- $H(x) = 1 - x$: prenant les masques opposés pour le réseau rival.

Ici le réseau va apprendre une valeur réelle, le contrôleur. Une fonction sigmoïde va être utilisée dans le but de transformer les valeurs du contrôleur $\sigma(c) \in [0, 1]$. Puis nous allons déterminer si le poids est choisi ou non, en tirant aléatoirement une valeur suivant une distribution de Bernoulli. Si la valeur est supérieure à $\sigma(c)$ alors le poids sera pris dans le réseau de base, sinon il est choisi pour le réseau rival.

L'idée est que même si le contrôleur est petit, la convolution aura une chance d'être récupérée. Cela permettera de récupérer des convolutions perdues au début de l'apprentissage.

Une autre différence avec les modèles précédents, est que les poids des modèles ne sont pas partagés. Les deux modèles sont entrainés en même temps, sur la même base

de données et dans le même but, mais les poids vont être différents. Cela permet une meilleure adaptation des différents poids choisis, Á la fin de l'apprentissage, le meilleur réseau est choisi, c'est à dire celui qui correspond le plus aux objectifs voulus, soit :

— davantage d'élagage, au dépend de la précision.
— une meilleure précision, au dépend de l'élagage.

Notre choix est d'améliorer la précision au dépend de l'élagage.

Des problèmes persistent encore dans ce modèle ; nous pensons que le choix des convolutions est trop fort. La distribution de Bernoulli renvoie soit 0, soit 1 pour chaque groupe de convolutions. De plus comme le choix se base uniquement sur une valeur (le contrôleur), cela peut entrainer des choix non pertinents, surtout si l'apprentissage du modèle se fait en entier, sans méthode de transfert de connaissances.

Utilisation d'un contrôleur, architecture finale

En se basant sur l'article de Liu et al. [52], nous avons décidé d'ajouter la valeur des poids dans la sélection ou non d'une convolution. Le choix de l'élagage est aussi moins direct car les poids sont étudiés séparement, ils sont toujours contrôlés par le même contrôleur mais le choix va être différent selon leurs valeurs. Avec la distribution de Bernoulli, si le choix était de ne pas prendre le groupe de convolutions, elles étaient toutes fixées à 0. Ici le choix sur les différentes convolutions va être différent.

Le principe est décrit dans la figure 39 avec les mêmes notations que précédemment.

Pour le réseau de base le choix de la convolution se transforme en :

— Si $|W_{i,j,k}| - c_{j,s} \geq 0$ alors la convolution est choisie : $XiW_{i,j,k}$
— Sinon $X_i 0$

Le contrôleur n'est plus le seul à influencer les choix, la valeur du poids elle même joue aussi un rôle. Ici 4 cas se présentent :

— Si la valeur absolue des poids dans les convolutions est **petite** et que $|W_{i,j,k}| \leq c_{j,s}$, alors les convolutions vont être supprimées du modèle original. Comme les poids faibles ont souvent moins d'impact dans la suite de l'entrainement, ils vont souvent être supprimés.
— Si la valeur absolue des poids dans les convolutions est **petite** mais que $|W_{i,j,k}| \geq c_{j,s}$, alors les convolutions vont être gardées. Ici, les poids sont petits, mais le contrôleur pense qu'ils vont avoir un impact positif dans la suite de l'apprentissage. Il décide donc de les garder.

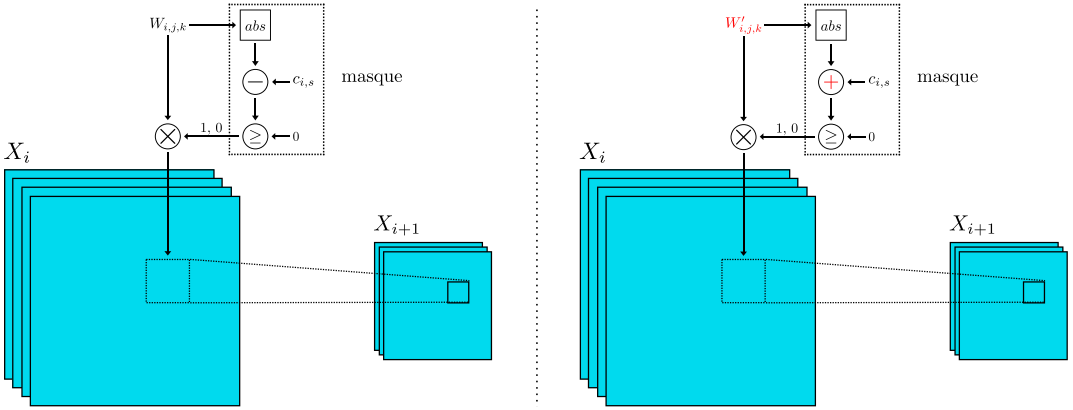


Figure 39: Architecture finale du réseau rival. Gauche : modèle de base ; Droite : modèle rival. X_i : entrées de la couche i, $W_{i,j,k}$: k-ième convolution de la profondeur j de la couche i, du réseau original. Le choix des convolutions est contrôlé par une valeur apprise (le contrôleur $c_{i,x}$) ainsi que la norme de la convolution $|W_{i,j,k}|$. Le réseau adverse utilisera les convolutions inutilisées dans le réseau de base.

— Si la valeur absolue des poids dans les convolutions est **grande** et que $|W_{i,j,k}| \geq c_{j,s}$, les convolutions sont gardées. Ici les poids sont grands et auront sûrement un impact positif par la suite.

— Si la valeur absolue des poids dans les convolutions est **grande** mais que $|W_{i,j,k}| \leq c_{j,s}$, les convolutions sont supprimées. Ici, les poids sont grands, mais le contrôleur pense qu'ils vont avoir un impact négatif dans la suite de l'apprentissage. Il décide donc de les supprimer.

Pour le réseau rival, les convolutions inutilisées du modèle de base sont choisies :

— Si $|W'_{i,j,k}| + c_{j,s} \geq 0$ alors $XiW'_{i,j,k}$

— Sinon $X_i 0$

Ici, la même idée que précédemment s'applique, les réseaux vont se partager les convolutions redondantes afin d'arriver au même but (la classification des images CIFAR10).

Un autre avantage de ce modèle, est que si $c = 0$, la convolution est prise dans les deux modèles. Si une convolution a un rôle unique et est importante au cours de l'apprentissage, elle va probablement être utilisée pour les deux modèles. Cela joue le rôle de l'union dans la section 4.1.2.3.

Fonctions d'erreur utilisées

Nous notons :

— La fonction de perte L_1 précédemment décrit comme cela : $L_1 = \sum_{i=1}^{n} |y_i - \hat{y}_i|$ avec y_i le label de vérité et $\hat{y}_i$ la prédiction.

— La fonction de perte L_2 précédemment décrit comme cela : $L_2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

— L'entropie croisée : $C = -\frac{1}{N} \sum_{i=1}^{n} (y_i.log(\hat{y}_i))$ utilisée comme mesure de précision du modèle.

Généralement, un terme de régularisation forçant les matrices de convolution à être creuse, est rajouté dans la fonction d'erreur. De nombreuses recherches ont soulignées la difficulté concernant le choix de ce terme. [108] et [109] montrent les difficultés ainsi que la versatilité des choix des hyperparamètres si les normes L_1 ou L_0 sont utilisées. Nous avons décidé d'enlever ce terme de la fonction d'erreur et ne pas contraindre notre modèle à élaguer. Liu et al. [52] utilisent une entropie croisée ainsi que $\alpha \exp(\mathbf{c_i})$ comme fonction de régularisation, pour forcer l'élagage avec α un hyper paramètre et c_i les contrôleurs. Nous proposons de minimiser la fonction de perte suivante :

$$L_s = \min_{\theta} C(\mathbf{X}, \mathbf{Y}, \theta) + \lambda \|\omega\|_2 \tag{1}$$

avec

— θ l'ensemble des paramètres à entrainer (sans les contrôleurs).

— $\mathbf{X}$ le vecteur d'entrée.

— $\mathbf{Y}$ le vecteur de labels vérité.

— ω l'ensemble des matrices de poids W_i et contrôleur $\mathbf{c_i}$.

— λ hyperparamètre contrôlant la régularisation de l'ensemble des poids du réseau.

Cette fonction est séparée en deux parties. Tout d'abord une entropie croisée est appliquée afin d'entrainer les poids du modèle θ. Puis une régularisation usuelle de l'ensemble des poids ω. Avant l'élagage, le modèle n'est pas entrainé, les poids W_i sont initialisés aléatoirement, et $\mathbf{c_i}$ est préalablement fixé a 0 comme dans [52]. Pendant la rétropropagation $c_{i,j}$ est mis à jour et sa norme est ajustée par la régularisation. Le λ est un terme de normalisation utilisé pour que les contrôleurs (et l'ensemble des poids) ne deviennent pas trop grands.

Afin de tester les limites du modèle et de le comparer avec l'état de l'art, nous avons forcé l'élagage en utilisant :

— La fonction d'erreur maximisant l'élagage, en maximisant la norme globale des

contrôleurs $L_p = C - \alpha||\boldsymbol{C}||_1$ (avec $\boldsymbol{C}$ la matrice des contrôleurs et C l'entropie croisée)

— La fonction d'erreur forçant un élagage fort en maximisant directement les valeurs du contrôleur : $L_{pm} = C + \alpha exp(-\boldsymbol{C})$.

Réseaux rivaux Itératifs

Une fois qu'un équilibre est atteint, le réseau est élagué à un certain pourcentage et une précision est atteinte, si l'équilibre entre précision et élagage ne nous convient pas encore, nous pouvons réitérer le processus. Si par exemple, la précision nous convient encore et que pour notre problème, une perte de précision n'est pas capitale, nous pouvons autoriser d'avantage d'élagage et réitérer le processus.

Pour un canal donné, il y a au moins soit les poids du modèle de référence, soit les poids du modèle rival qui ne vont pas être élagués. Le taux d'élagage peut ne pas être directement optimal. Le processus d'entrainement du modèle rival est réitérable, en enlevant les poids élagués et en entrainant de nouveau de zéro. L'algorithme est présenté en 2.

algorithme 2 Itérations du processus d'entrainement d'un réseau rival. $acc_{min}\%$ minimum de précision admis

$masques \leftarrow 1$

$acc = 100\%$
Tant que $acc > acc_{min}$ **Faire :**
    $W \leftarrow$ initialisation aléatoire
    Élaguer avec le réseau rival
    Récuperer les nouveaux $masques$ du réseau élagué
    Calculer la précision acc du nouveau réseau.

Cette réitération est un peu longue, les fonctions L_p ou L_{pm} peuvent être utilisées afin de forcer le modèle à supprimer davantage de convolutions. Le but est de voir comment le modèle réagit lorsqu'un grand nombre de poids sont supprimés.

Dans les résultats, nous noterons Ours($\boldsymbol{\lambda = 1e^{-4}}$, $\boldsymbol{iter = 3}$), pour les résultats donnés par la troisième itération du modèle, en utilisant la fonction d'erreur L_s avec une valeur de $1e^{-4}$ pour λ.

4.2 Résultats expérimentaux

Cette section présente les bases de données utilisées pour les approches d'élagage, ainsi que les résultats des méthodes présentées dans la partie 4.1.

Dans cette section, plusieurs approches d'élagage ont été effectuées, toutes basées sur un réseau rival. Afin de se comparer à l'état de l'art, les architectures classiques Lenet5, Wide-Resnet et Resnet, présentées dans la section 2.1.3, ont été élaguées.

Deux bases de données, différentes, ont été utilisées afin de tester nos modèles rivaux. Nous allons dans un premier temps présenter ces bases de données CIFAR10 et MNIST.

4.2.1 Bases de données

MNIST

Les premiers réseaux testés ont aussi été entrainés et testés avec la base de données MNIST [27]. Il s'agit d'une base d'images noir et blanc, de taille 28×28, de chiffres manuscrits. Elle contient :

— $60,000$ images d'entrainement sur 10 classes (chiffres de 0 à 9) ;
— $10,000$ images de test (1000 par classes).

Quelques exemples sont présentés dans la figure 40.

Figure 40: Exemples d'images de la base MNIST.

Par la suite, la base de données CIFAR10 est aussi utilisée car c'est une base de données

complexe, correspondant davantage à notre problème final de détection d'émotions et permettant aussi une comparaison.

CIFAR10

Les réseaux présentés par la suite ont été entrainés et testés sur la base de données CIFAR10 [26].

Cette base de données contient 60000 images RGB de taille 32×32. Il y a au total 10 classes, labélisées de 0 à 9, avec 6000 images par classes. Elle est séparée en :

— $50,000$ images d'entrainement.

— $10,000$ images de test.

La base de données d'entrainement est divisée en 5 batchs de taille 10000 images et 1 batch de test composé de 1000 images aléatoires par classe. La figure 41 présente quelques exemples de cette base.

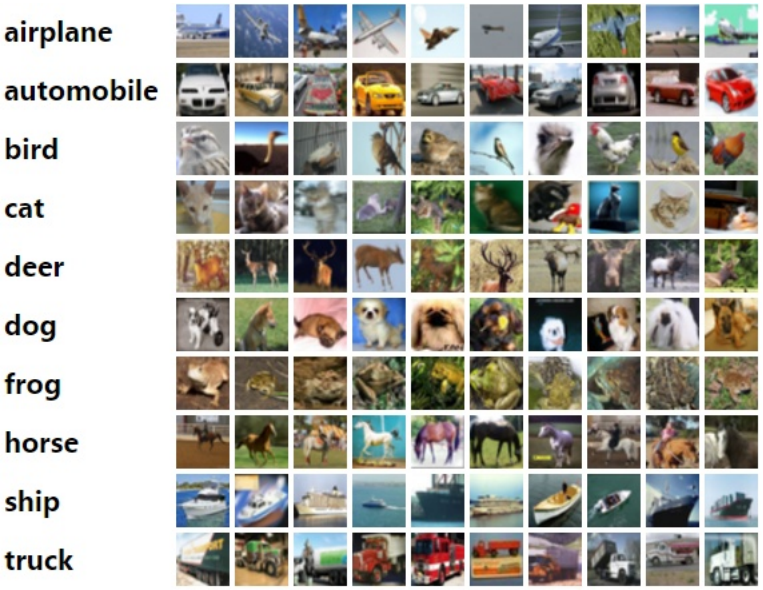


Figure 41: Exemples d'images de la base CIFAR 10

Le nombre d'images par classe est trop faible pour obtenir un bon élagage et une bonne précision. Nous avons donc augmenté ces données, grâce à des techniques d'augmentation de données présentées au paragraphe 2.2.1.1.

4.2.2 Résultats

Dans cette section nous présentons les résultats du modèle rival.

Nous avons testé les différentes approches sur les réseaux LeNet5 [25], Resnet [2] et Wide-Resnet [3] (présenté dans la section 2.1) sur les bases de données MNIST [27] et CIFAR10 [26] (présenté dans la section 4.2.1). Le choix de ces réseaux nous a permis de nous comparer à l'état de l'art afin de vérifier le bon fonctionnement des différentes approches. Pour vérifier la robustesse du modèle, d'autres réseaux et d'autres bases de données devront être testées.

Précision réseau rival avec une base opposée et un Vae-Gan

Dans un premier temps, nous avons entrainé les deux modèles sur une base que l'on nomme *opposée*. Le générateur de BigGan a été utilisé afin de créer cette base de données.

Nous présentons ici les résultats du pourcentage d'élagage en fonction de la précision du modèles présenté dans la partie 4.1.2.2.

Dans toute la suite on nomme référence, la précision atteinte du réseau sans processus d'élagage.

Les expériences effectuées sur LeNet5 avec utilisation d'un Vae-Gan sur la base de donnée MNIST nous ont permis d'observer les résultats suivants sur la précision en classification (référence avant élagage : 98.54%) :

— Sans utilisation d'un réseau rival [110], une précision de 98.54% (respectivement 97.5%) est obtenue avec un élagage de 95% (resp 99%)

— Avec utilisation d'un réseau rival, une précision de 98.3% est obtenue en élaguant 95.5% des poids du réseau LeNet5 (temps de calcul : 12h30)

Les résultats sont encourageants et comparables à l'état de l'art. En effet nous enlevons un peu plus de poids, 95.5% en perdant seulement 0.2 point en précision sur la base de données MNIST. L'apprentissage est cependant long même en entrainant préalablement le générateur. De plus, la base MNIST est une base assez facile à classifier. Les réseaux profonds, même très simples donnent des résultats très performants. Nous avons utilisé la base de données CIFAR10 par la suite. Afin de nous comparer à l'état de l'art nous avons décidé d'élaguer le réseau wide-resnet, présentant les meilleures performances de l'état de l'art sur la base de données CIFAR10.

Précision réseau rival sur la même base de données CIFAR10

Comme l'apprentissage avec un réseau génératif était trop coûteux, nous avons décidé de l'enlever et d'apprendre les deux réseaux sur la même base de données CIFAR10, afin de sélectionner et de supprimer les convolutions redondantes.

Les fonctions l_s, L_p et L_{pm}, définies paragraphe 4.1.2.3, sont utilisés dans cette section et nous les rappelons ici :

— $L_s = \min_\theta C(\mathbf{X}, \mathbf{Y}, \theta) + \lambda \|\omega\|_2$
— $L_p = C - \alpha||\boldsymbol{C}||_1$
— $L_{pm} = C + \alpha exp(-\boldsymbol{C})$.

Avec un contrôleur et une distribution de Bernoulli

Dans un premier temps, une distribution de Bernoulli sur le contrôleur a été testée pour le choix des convolutions.

Le tableau 5 présente les résultats du modèle présenté en partie 4.1.2.3 en comparaison avec l'état de l'art.

Rival	Enlevé %	Précision %	Gain/Perte	
non	21.00	94.99	-0.19	L_s
oui	**48**	**94.28**	**-0.9**	L_s
oui	**93.75**	**90.66**	**-4.52**	L_{pm}
non [52]	90.14 ± 0.22	95.05 ± 0.08	-0.13	L_{pm}
non [52]	95.36 ± 0.15	94.73 ± 0.11	-0.45	L_{pm}

Table 5: Résultat du réseau rival, avec utilisation d'une distribution de Bernoulli, sur Wide Resnet en fonction des différentes fonctions d'erreur utilisées : L_s et L_{pm}. La valeur $\lambda = 1e^{-4}$ a été utilisée pour la fonction de perte L_s. Référence : 95.18%. Cette approche apprend en 12h

Sans forcer l'élagage, juste en indiquant aux contrôleurs d'être différents de zéro, les deux modèles se partagent les poids presque équitablement (avec 2% d'union). Les résultats sont encourageants, surtout quand le taux d'élagage est bas. Cependant, dès que l'on rajoute une contrainte dans la fonction de perte ($\alpha exp(-c)$), l'élagage monte au dépend de la précision. Comme expliqué en section 4.1.2.3, nous pensons que l'utilisation de la distribution de Bernoulli impose une contrainte très forte et supprime des convolutions qui ne devraient pas être enlevées.

Avec un contrôleur, sans distribution de Bernoulli

Le modèle est présenté en partie 4.1.2.3, l'idée est de laisser la valeur du poids au sein des convolutions influencer dans le choix du poids ou non, plutôt que de laisser une seule valeur (le contrôleur) contrôler tout le groupe de convolutions.

Wide Resnet

Les résultats concernant la précision en fonction du pourcentage d'élagage, comparés à différentes fonctions de perte de l'état de l'art, sont présentés dans le tableau 6.

Rival	Enlevé %	Précision %	Gain/Perte	Nombre d'itération	
non	21.00	94.99	-0.19	1	
oui	**21.27**	**95.78**	**+0.6**	1	
oui	**36.56**	**95.84**	**+0.66**	2	
oui	**47.36**	**95.40**	**+0.2**	3	
oui	**55.58**	**95.31**	**+0.09**	4	
oui	**62.82**	**95.29**	**+0.07**	5	L_s
oui	**69.63**	**94.96**	**-0.26**	6	
oui	**73.71**	**94.70**	**-0.52**	7	
oui	**78.03**	**94.45**	**-0.77**	8	
oui	**81.82**	**94.11**	**-1.07**	9	
oui	**85.45**	**93.7**	**-1.48**	10	
oui	**45.18**	**95.43**	**+0.25**	1	L_p
non [52]	90.14 ± 0.22	95.05 ± 0.08	-0.13	1	
non [52]	95.36 ± 0.15	94.73 ± 0.11	-0.45	1	L_{pm}
oui	**98.43**	**94.33**	**-0.85**	1	

Table 6: Résultat de l'élagage par réseau rival, sur Wide Resnet, en fonction des différentes fonctions d'erreur utilisées. $\lambda = 1e^{-4}$ à été utilisé pour la fonction de perte L_s. Référence : 95.18%. Une itération prend 7h.

Ce tableau présente quatre résultats différents :

— Grâce au réseau rival, nous supprimons 21.27% des convolutions du modèle avec un gain en précision de 0.6 par rapport à la référence. Et ceci, sans indiquer à la fonction de perte d'élaguer (absence de contrôleur dans la fonction d'erreur). Le choix des poids est meilleur que pour l'état de l'art ; en effet, nous élaguons davantage (+0.27 point) et obtenons une meilleure précision (+0.79 point).

— Ce processus est donc réitérable (cela va enlever 21% des poids restants). Le résultat est encore meilleur que précédemment, le réseau rival aide à enlever les convolutions redondantes pendant l'apprentissage. 36.56% des poids sont supprimés pour arriver

à une meilleure précision que la référence (+0.66 points). Nous arrivons à supprimer jusqu'à 62.82% des convolutions en ayant un impact positif sur la précision. Ce qui veut dire que 62.82% des poids sont soit redondants, soit ont un impact négatif pendant le processus d'apprentissage. Puis, comme présenté sur la figure 42, la précision se dégrade, et la présence du contrôleur dans la fonction de perte devient indispensable. Une itération prend déjà 7h.

— Afin de tester les limites du modèle, nous avons aussi appris avec la norme du contrôleur dans le masque, afin de maximiser l'élagage. Nous arrivons en une fois à 45.18% de convolutions supprimées avec un gain encore positif de 0.25 points en précision.

— Nous avons ensuite utilisé une contrainte forte sur le contrôleur avec une fonction exponentielle qui force le modèle à élaguer 98.43% des poids avec cette fois-ci une perte de 0.85 points en précision. Ce qui reste tout de même encourageant. Empiriquement nous avons choisi $\alpha = 1e^{-5}$.

— Une comparaison peut ici être faite avec le réseau EfficientNet [68] contenant 4M de paramètres et atteignant une précision de 98.1%. A l'itération 6 le réseau WideResnet contient à peu près 4M de paramètres, en atteignant une précision de 95.29% (+0.07 par rapport au départ). Les résultats sont bien inférieurs mais ils est important de noter que notre approche est d'améliorer un réseau (ici le réseau WideResnet) en enlevant un certain nombre de paramètres tout en améliorant la précision. Le but n'est pas (comme EfficientNet) de créer une architecture optimisée pour la base de données CIFAR10. Les méthodes sont en soit difficilement comparables.

Resnet

Les résultats du réseau rival, sur Resnet-56 ainsi que Resnet-110 en comparaison avec l'état de l'art, sont présentés dans le tableau 7.

Cette comparaison est faite à deux niveaux, le premier correspond à la performance atteinte avec un élagage fort, le second se concentre sur une précision forte.

L'architecture rivale permet d'atteindre une précision plus forte sur CIFAR10, avec pour Resnet-56 un gain de 0.65 point pour 13.55% d'élagage. Ainsi qu'un gain de 0.27 point de précision pour 11.47% d'élagage sur Resnet-110.

4.2.3 Discussion

Nous remarquons que le réseau rival impacte énormément sur le choix des convolutions à supprimer. Grâce à ces choix, nous arrivons à enlever les convolutions inutiles ou ayant

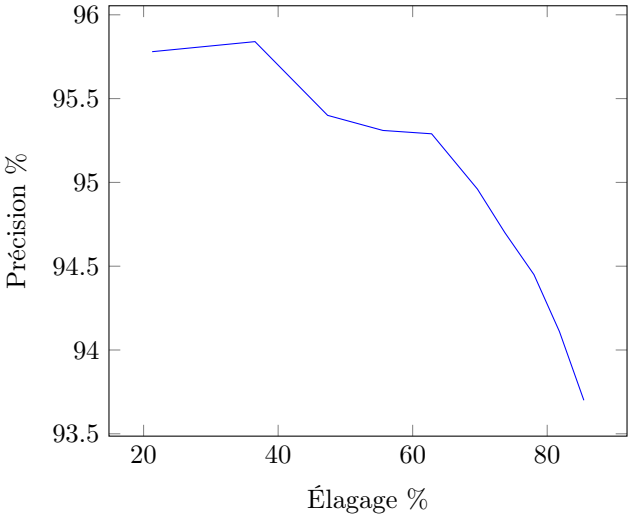


Figure 42: Limite de l'architecture ; à haut niveau d'élagage, la perte de précision est considérable.

Architecture	Méthode	Référence %	Élagage %	Acc %	Gain/Perte
ResNet-56	Zuo et al. [55]	93.64	23.50	93.83	+0.19
	You et al. [59]	93.10	53.50	93.43	+0.33
	He et al. [58]	93.59 ± 0.58	14.7	93.89	+0.30
	Gao et al. [66]	93.03	65.7	93.34	+0.31
	Ours($\lambda = 1e-4$, $iter = 1$)	$\mathbf{93.60 \pm 0.41}$	**13.55**	**94.25**	**+0.65**
ResNet-110	Wang et al. [62]	93.49	93.69 ± 0.28	93.97	+0.20
	Zuo et al. [55]	93.82	24.40	94.23	+0.41
	He et al. [56]	93.68 ± 0.32	40.00	93.85	+0.17
	He et al. [58]	93.68 ± 0.32	28.2	93.93	+0.25
	Ours($\lambda = 1e-5$, $iter = 1$)	94.05 ± 0.03	11.47	94.32	+0.27

Table 7: Résultat de l'élagage par réseau rival, sur Resnet-56 et Resnet-110. Ces réseaux sont entrainés et testés sur la base de données CIFAR10. Une itération sur Resnet-56 prend 8h, sur Resnet-110 : 9h30.

un impact négatif durant l'apprentissage et à gagner en précision en enlevant des poids. L'impact du réseau rival est surtout remarqué sur wide-resnet, avec un gain conséquent de précision grâce à la suppression d'un nombre de convolutions conséquent. Cependant, il se remarque aussi sur les réseaux Resnet, même si le gain en précision ainsi que le taux d'élagage sont plus faibles, les réseaux rivaux ont tout de même un impact positif.

Nous observons aussi que le contrôleur et sa *binarisation* (la manière de sélectioner les convolutions) est capital pour obtenir un élagage de qualité, le choix d'une contrainte

élevée va grandement impacter la qualité de l'élagage.

Nous constatons finalement que *wide-resnet*, un réseau très performant et très utilisé, contient presque 63% de poids soit inutiles (redondants), soit ayant un impact négatif pendant l'apprentissage de CIFAR10. L'ajout de couches afin d'obtenir une meilleure précision n'est pas nécessairement la meilleure solution. Nous pensons que les réseaux doivent, dans un premier temps, être optimisés.

Un problème cependant persiste : la réitération du processus d'élagage est assez longue et le choix du paramètre λ est empirique et capital. Une piste serait de le fixer en fonction du nombre de paramètres du modèle, mais aucun résultat n'a, pour le moment, été convaincant. Des travaux ultérieurs doivent être effectués dans ce sens.

5 Apprentissage *few shot* pour la détection d'émotions sur base de données restreintes multimodales

Dans ce chapitre, différentes techniques d'apprentissage *few shot* ont été explorées, avec notamment de nombreux tests sur les réseaux siamois et triples, des réseaux performants dans le domaine de l'apprentissage sur base de données restreintes.

5.1 Approche

5.1.1 Motivations

Description du problème

Nous avons choisi de détecter les 6 émotions de bases, décrite par Paul Ekman en 1972 [10] (cf. figure 43).

Comme décrit dans l'état de l'art (3.2.3), les réseaux convolutifs sont très performants dans le domaine de la détection d'émotions, si la base d'apprentissage est suffisamment grande. En effet, si le réseau apprend sur des centaines de milliers, voir des millions d'images, il n'aura aucun mal à généraliser les six émotions. Cependant, les méthodes classiques d'apprentissage profond ne peuvent pas généraliser sur des centaines ou des milliers d'images. Nous avons donc choisi de nous orienter vers des méthodes d'apprentissage *few shot*, notamment avec l'utilisation des réseaux correspondants.

Les méthodes d'apprentissage profond citées dans la section 3.2 présentent des pistes intéressantes pour la détection d'émotions. Cependant elles possèdent encore quelques limites.

— Certaines méthodes apprennent avec des bases de données d'apprentissage conséquentes.

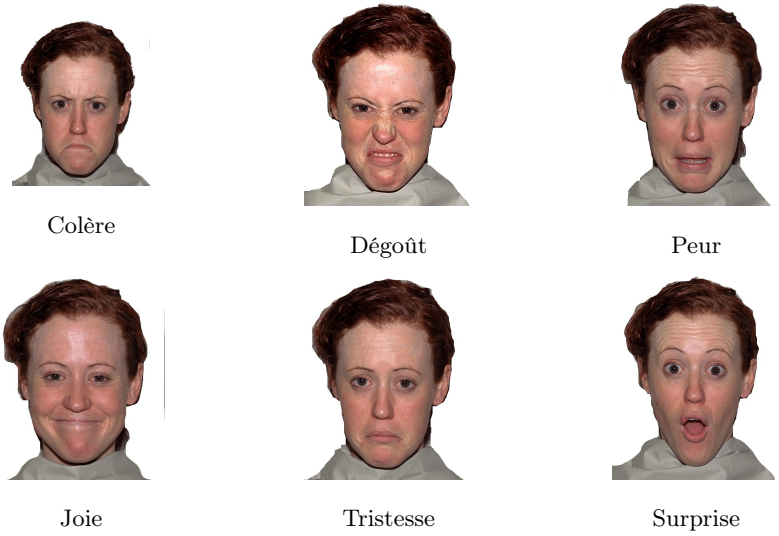


Figure 43: 6 émotions de bases décrite par Paul Ekman en 1972. Image prise dans la base de données Nimstim [31].

— Les comparaisons sont parfois impossibles.
— La précision est très faible sur certaines bases de données restreintes.
— Des méthodes de transfert de connaissance sont systématiquement utilisées, parfois même sur le même problème (détection d'émotions).

Nous avons fait l'étude de quelques méthodes d'apprentissage *few shot* sur des données multimodales, image et son. Nous pensons que ces données possèdent des informations complémentaires pouvant être apprises par le réseau.

Contributions

— Nous présentons un réseau triple et un réseau siamois multimodaux pour la détection d'émotions.
— Nous présentons une nouvelle fonction d'erreur permettant un apprentissage plus rapide pour les réseaux siamois et triples.
— Ces réseaux ont été testés sur différentes bases de données et des éléments de référence sont systématiquement donnés. L'état de l'art est atteint pour une de ces bases et se rapprochent des performances de méthodes nécessitant de grandes bases de données pour les autres.

5.1.2 Prétraitement

Dans la partie détection d'émotions sur données multimodales, des images et des sons sont utilisés. Pour un meilleur apprentissage, ces données ont été préalablement traitées.

5.1.2.1 Image

Peu de prétraitements ont été effectués, car les poids du réseau ont été pré-entrainés sur des images normalisées.

Normalisation La normalisation est une étape importante qui assure que chaque entrée (pixel ici) suit une distribution similaire. Le but est de contrôler le gradient lors de la rétropropagation. De plus, comme nous partageons les poids au sein des convolutions (cf .2.0.1) il ne faut pas que les intensités des pixels d'entrée diffèrent trop les unes des autres.

Tous les pixels ont été normalisés en soustrayant la moyenne des pixels et en divisant ce résultat par l'écart type.

Choix des images Comme les bases de données sont hétérogènes, le choix des images a parfois été différent et capital.

- Pour Nimstim Ravdess les sujets tests ont été choisis aléatoirement. La combinaison Image-Son a été effectuée aléatoirement sous la condition que le label soit identique et qu'une phrase prononcée par un homme (respectivement une femme) soit combinée avec le visage d'un homme (respectivement d'une femme).
- Pour CK+, les images sont séquentielles, nous avons choisi des sujets de test aléatoirement. Les images sont extraites de vidéos, deux images successives se ressemblent donc grandement. Afin de ne pas biaiser les tests ou donner des images semblables durant l'entrainement, nous avons attendu un certain laps de temps (500ms) avant de sélectionner une autre image.
- Pour Afew, les images sont tirées de vidéos et sont donc séquentielles. La méthode de sélection est la même que pour CK+. Cependant, comme il s'agit d'une base de données en *conditions réelles*, nous avons par la suite vérifié que chaque image était humainement classifiable dans le bon label (que le visage était présent dans l'image et présentait bien le bon label).
- Pour FER les images d'entrainement et de tests ont été choisies aléatoirement.

5.1.2.2 Audio

L'audio a été modifiée grâce aux Coefficients Cepstraux à Fréquences Mel [111] (MFCC) et à la transformée en ondelettes discrète de Haar (DWT) [112]. Nous allons donc les présenter ici.

MFCC

Les données temporelles sont traitées par analyse cepstrale, afin de fournir un ensemble de caractéristiques pertinentes en entrée du réseau profond. Plus précisément, les coefficients cepstraux de fréquence Mel (MFCC), introduits dans [111], sont utilisés dans la suite pour chaque enregistrement. Le calcul des coefficients MFCC est expliqué schématiquement sur la figure 44, et détaillé ci-après.

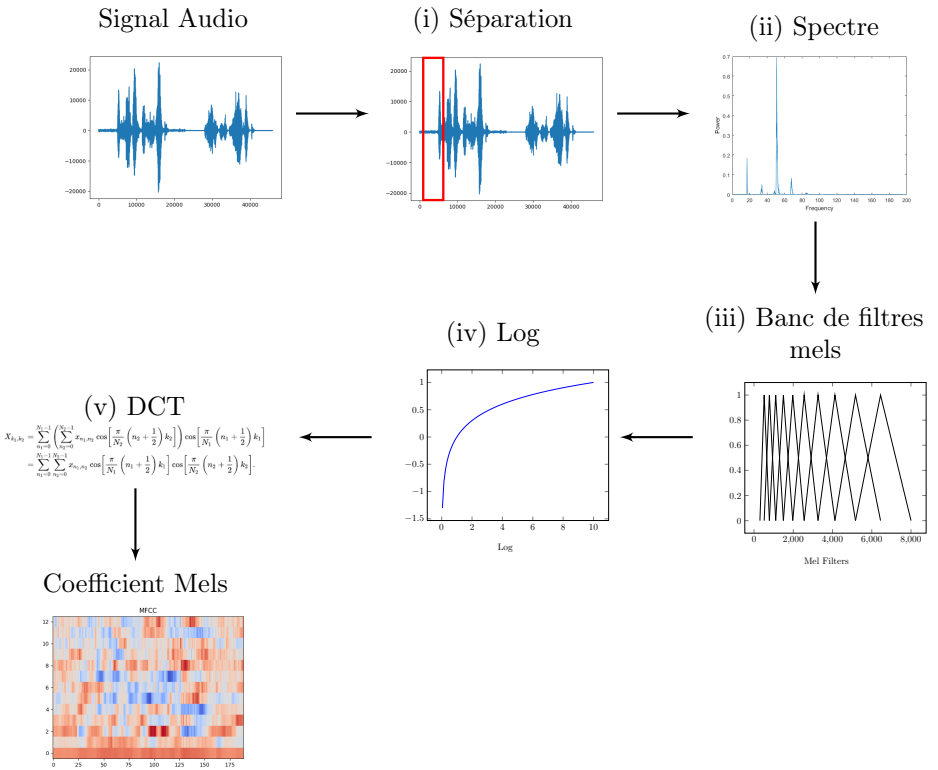


Figure 44: Principe des MFCCs, présentation des différentes étapes.

1. Un fenêtrage du signal temporel sur de petites fenêtres de temps (10-20 ms) est effectué ;
2. Le spectrogramme en puissance pour chaque fenêtre est calculé. Nous savons que l'oreille réagit différemment selon la fréquence et indique au cerveau les fréquences présentes dans le son. Le réseau utilisera cette information fréquentielle pour détecter les émotions présentes ;
3. Le spectrogramme est ensuite pondéré par un banc de filtres triangulaires espacés selon l'**échelle de Mel** [113]. Le ton est l'une des caractéristiques principales de la parole, nous pouvons le mesurer en fréquence. L'échelle de Mel est une échelle qui copie ce que l'humain perçoit ; elle adapte la mesure en fonction du ton pour être plus proche de ce que notre oreille entend. En effet, l'homme identifie mieux les petits changements dans les basses fréquences que les petits changements dans les hautes fréquences.

   Un banc de filtres Mel est appliqué aux signaux, ce qui permet d'extraire des bandes de fréquences du spectrogramme et de regrouper les données importantes. Les filtres sont schématisés sur la figure 45 [114] où l'on remarque un plus grand nombre de filtres dans les basses fréquences que dans les hautes fréquences. La conversion d'une fréquence f (en Hz) à l'échelle Mel se fait selon la formule

$$m = 2595 \log(1 + f/700).$$

   Pour chaque filtre, l'énergie totale est calculée par sommation, et donne une information sur la répartition énergétique dans chaque région fréquentielle de l'échelle Mel. Le premier filtre donne par exemple l'indication de l'énergie autour de la fréquence nulle ;
4. Un opérateur logarithme est appliqué pour amplifier les petites différences dans les faibles énergies ;
5. Une transformée en cosinus discret (DCT) est enfin effectuée. En effet, à cette dernière étape, les coefficients Mel sont corrélés puisque les filtres se recouvrent partiellement. La DCT permet de décorréler les coefficients calculés, pour aboutir aux MFCC. De manière usuelle, en reconnaissance vocale, seuls les coefficients cepstraux de 2 à 13 sont conservés, les autres coefficients représentant des changements trop rapides et ne contribuant pas à la reconnaissance vocale.

La figure 46 (droite) présente un exemple de spectrogramme MFCC calculé sur une phrase de la base de données.

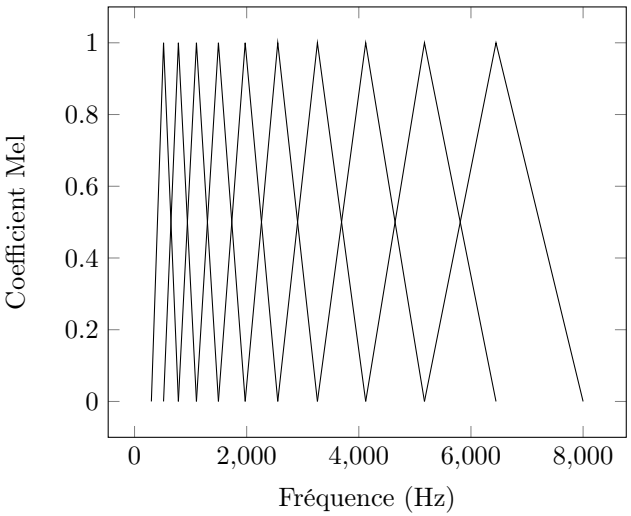


Figure 45: Exemple des 10 premiers filtres Mel.

Chaque ligne verticale représente 10 millisecondes de la phrase choisie. Les 13 lignes (0-12) représentent les coefficients MFCC. Pour la représentation de l'énergie, nous avons utilisé une représentation *Cool-warm*. La couleur représente l'énergie contenue dans chaque coefficient Mel, du bleu au rouge par niveau croissant d'énergie.

Transformée en ondelettes discrète (DWT)

Nous pensons que décomposer le son en différentes bandes de fréquences grâce aux ondelettes de Haar est approprié à des problèmes de classification. Nous allons dans cette section, expliquer la transformée en ondelettes discrète (DWT).

Soit $\psi(t)$ une fonction (ondelette mère) centrée en $t = 0$ définie sur $[-\frac{T}{2}, \frac{T}{2}]$.

La fonction $\psi_{s,u}(t) = \frac{1}{\sqrt(s)}\psi(\frac{t-u}{s})$ est alors centrée sur $t = u$ et est définie dans l'intervalle $[-s\frac{T}{2} + u, s\frac{T}{u} + u]$.

Nous utilisons comme valeur de déplacement (shift) $s = k2^{-j}$, et comme valeur de mise à l'échelle $u = 2^{-j}$ pour définir la base des ondelettes (où j et k sont des entiers strictement positifs). Cette base est alors définie par

$$\psi_{j,k}(t) = 2^{\frac{j}{2}}\psi(2^j t - k).$$

Un signal discret x peut alors être décrit par $f(x) = \sum_{j,k} c_{j,k}\psi_{j,k}(x)$. L'ensemble des

vecteurs de base étant orthogonaux, les coefficients cepstraux $c(j,k)$ sont obtenus grâce à la DWT :

$$c(j,k) = \sum_t f(t)\psi_{j,k}(t).$$

Dans notre thèse, nous avons utilisé la transformée de Haar [112] car c'est la plus rapide et la plus simple des ondelettes. Elle est définie comme :

$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2}, \\ -1 & \frac{1}{2} \leq t < 1, \\ 0 & \text{sinon.} \end{cases}$$

Comme toutes les transformées en ondelettes, Haar décompose le signal discret en deux signaux de taille égale. Le premier signal peut être vu comme une moyenne du signal global, alors que le second signal peut être vu comme une différence, ou une variation.

Cette transformée peut être utilisée à différents niveaux, la transformée de Haar au niveau 1 est noté H_1. Sur un exemple, prenons un signal s de taille n : $s = (s_0, s_1, \ldots, s_{n-1})$.

- La première partie de la transformée est la moyenne :

$$a_k = \frac{s_{2k-1} + s_k}{\sqrt{2}} \quad \text{où} \quad 2k \leq n-1 \ .$$

- L'autre partie de la transformée est une différence :

$$d_k = \frac{s_{2k-1} - s_{2k}}{\sqrt{2}} \quad \text{où} \quad 2k \leq n-1 \ .$$

Sur le signal original, le résultat de la transformée de Haar au niveau 1 est alors :

$$s \xrightarrow{H_1} (a_0, a_1, \ldots, a_m | d_0, d_1, \ldots, d_m) \quad \text{où} \quad m \leq \frac{n-1}{2} \ .$$

Le résultat de cette transformée sur une phrase est représenté sur la figure 46, au milieu.

Afin de présenter nos données au réseau, l'image faciale, ainsi que l'intensité des pixels des enregistrements sonores traités ont été concaténés dans un vecteur de taille 733440.

Lorsque nous traitons des données multimodales (ou provenant de multiples capteurs), deux stratégies sont classiquement appliquées.

— La concatenation de caractéristiques, traitant ces dernières uniformément, quel que soit le type des informations données pour chaque mode. Le but est de produire un

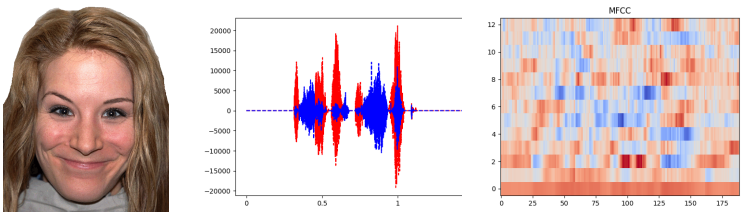


Figure 46: Gauche : image faciale, labélisée Heureux. Milieu : transformée de Haar de niveau 1, calculée sur une donnée de la base de données Mnist, labélisée Heureux. La fonction rouge représente a^1 et la bleu d^1. Axe-x : temps (ms), axe-y : Coefficient de Haar. Droite : MFCC représenté avec une échelle *Cool-Warm*. Axe-x : temps, axe-y : les 13 coefficients calculés.

seul vecteur de caractéristiques et de l'analyser.

— Un ensemble d'algorithmes traitant chacun séparement les caractéristiques d'un seul mode, dont les résultats sont par la suite combinés ensemble.

À notre connaissance, aucune des deux méthodes n'a été prouvée meilleure que l'autre. Comme nous ne favorisons aucun mode par rapport à l'autre et que les deux données expriment les mêmes émotions (même label), la concatenation des deux vecteurs a été choisie. Par conséquent, seul un réseau doit être entrainé (au lieu de deux), pour plus de facilités sur des petites bases de données.

5.1.3 Réseau siamois multimodal

Afin de classifier nos données traitées, un réseau siamois est utilisé. La figure 47 décrit le principe global de notre approche.

Deux couples Image-Son sont choisis aléatoirement (labélisés joie dans la figure 47). Dans la suite, nous appellerons instance un couple Image-Son. Les couples sont traités par le même réseau convolutif Inception-Resnet (décrit dans la section 2.1.4), donnant deux vecteurs de caractéristiques (E_1 et E_2). L'idée est de rapprocher ces vecteurs, s'ils sont de la même classe ($y = 0$) et de les éloigner s'ils sont d'une classe différente ($y \neq 0$).

Une fois que l'apprentissage est terminé, nous aurons pour résultat, une représentation spatiale contenant 6 clusters (représentant les 6 classes). Puis, grâce à cette représentation vectorielle, des techniques de clustering seront utilisées pour classifier chaque couple de test.

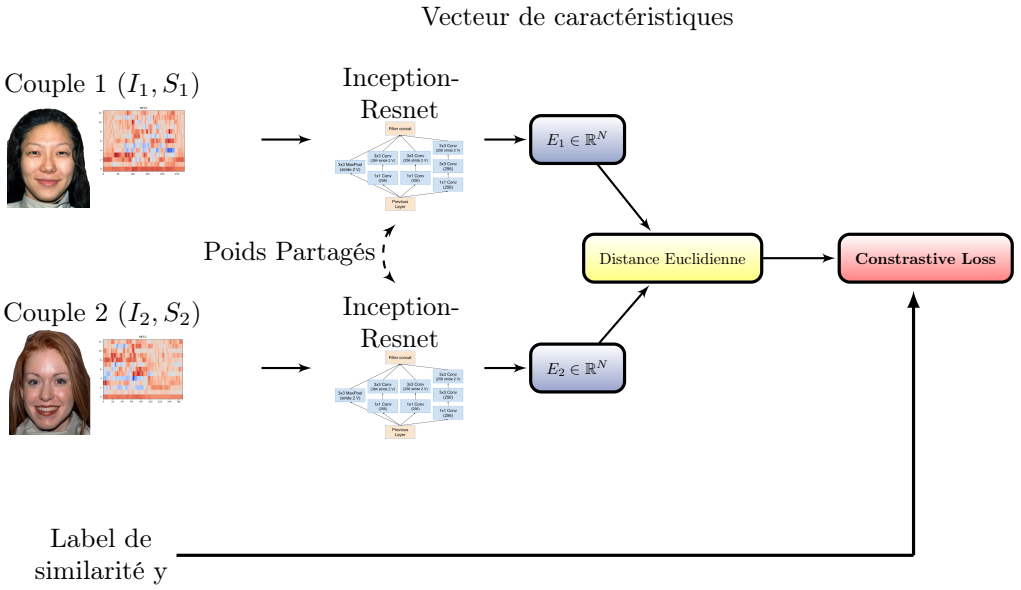


Figure 47: Réseau siamois multimodal, prenant en entrée deux couples images-sons et leur label de similarité. Le réseau construit un plongement vectoriel dans R^N dans lequel une métrique euclidienne est calculée, et qui sert de base au calcul au *contrastive loss*

5.1.3.1 Erreur

La fonction de perte utilisée est basée sur le *contrastive loss* vu dans la section 2.2.3.1 :

$$L = \frac{1}{2}(1-y)d(\mathbf{E_1}, \mathbf{E_2})^2 + \frac{1}{2}(y)max(0, m - d(\mathbf{E_1}, \mathbf{E_2}))^2.$$

Avec $d(\mathbf{E_1}, \mathbf{E_2})$ la distance entre les deux vecteurs caractéristiques calculés.

Lorsque les couples ne sont pas de la même classe, le label de similarité y est égal à 1. Dans ce cas, la fonction de perte est dépendante du choix de la marge m. Si m est trop faible, l'information des différentes paires est perdue. De plus, le choix de la marge n'est pas le même en fonction des données, des traitements choisis et de la normalisation.

Fort de ce constat, nous avons appliqué une couche sigmoïde à la fin du réseau *Inception-Resnet-v2*. Chaque valeur contenue dans le vecteur final sera donc entre 0 et 1. Si nous notons N la dimension de ce vecteur, sa norme euclidienne sera au maximum $\sqrt{N}$. Nous

pouvons alors transformer la fonction d'erreur en :

$$L = \frac{1}{2}(1-y)d(\mathbf{E_1}, \mathbf{E_2})^2 + \frac{1}{2}(y)(\sqrt{N} - d(\mathbf{E_1}, \mathbf{E_2})^2.$$

Cette fonction assure que l'erreur commise n'est jamais nulle et que toutes les paires seront utiles à l'entrainement.

5.1.4 Réseau triple multimodal

Le principe de base des réseaux triples et du triplet loss a été présenté dans la section 2.2.3.2. L'approche est résumée dans la Figure 48.

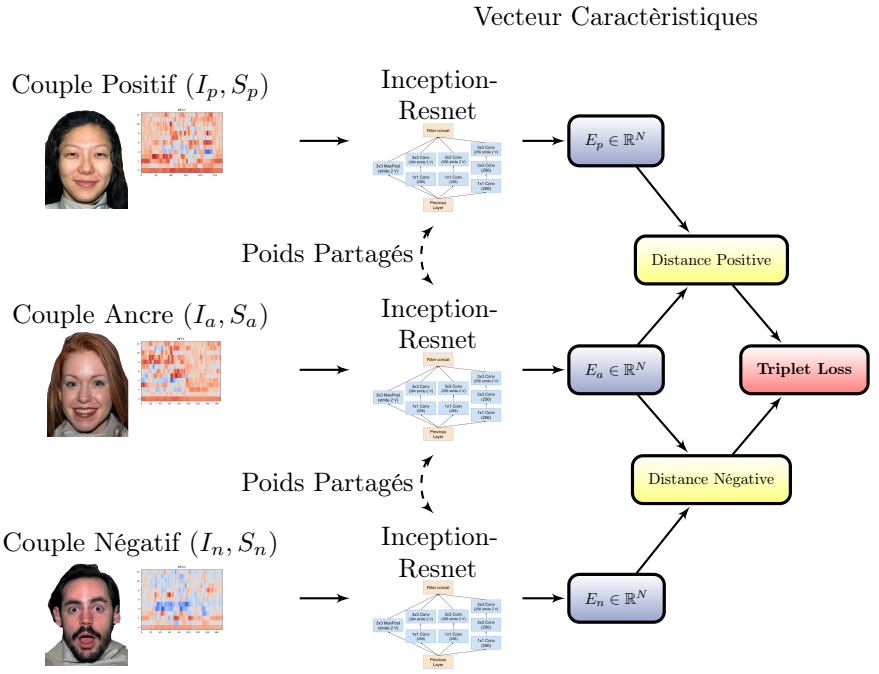


Figure 48: Réseau triple multimodal prenant trois couples images-sons. Le réseau construit un plongement vectoriel dans R^N dans lequel une métrique euclidienne est calculée, et qui sert de base au calcul du *triplet loss*

L'idée est de trouver les caractéristiques importantes de l'image faciale et du son, afin de distinguer les différentes classes. Le réseau de convolution va choisir trois instances,

que nous nommons un triplet (C_a, C_p, C_n). Un triplet est composé :

— d'un couple ancre $C_a = (I_a, S_a)$ que nous tirons aléatoirement (sur la figure 48 labélisé : joie)
— d'un couple positif $C_P = (I_p, S_p)$ (même label que l'ancre : joie)
— d'un couple négatif $C_n = (I_n, S_n)$ (un autre label, ici : surprise).

Pour le choix du réseau, Alexnet, Resnet50, Inceptionv3 et Resnet-Inception-V2 ont été testés.

Alexnet est le réseau donnant de moins bon résultats, une des raisons plausibles est son nombre conséquent de paramètres (60M). Même avec des méthodes de transfert de connaissances, le réseau généralise moins bien et apprend moins bien sur peu de données.

Resnet50 et Inceptionv3 sont à peu près au même niveau et donnent des résultats meilleurs qu'Alexnet. Le réseau retenu est Inception-Resnet-V2 [4] utilisant du *transfer learning* sur un problème de reconnaissance faciale. Grâce aux liens résiduels, le réseau apprend plus vite. Ce qui, dans notre cas, est très important. Le fait que la reconnaissance faciale est un problème proche du notre facilite l'apprentissage des convolutions. Les convolutions à bas niveau reconnaissant les différentes formes du visage sont déjà apprise. Cela accélère aussi le processus d'apprentissage.

Pour placer correctement les vecteurs de caractéristiques dans l'espace créé, une fonction d'erreur est minimisée.

5.1.4.1 Fonction de perte

Fonction de perte proposée

La fonction de perte est basée sur le triplet loss vu dans la section 2.2.3.2

$$Tl(\mathbf{E_a}, \mathbf{E_p}, \mathbf{E_n}) = \max(d(\mathbf{E_a}, \mathbf{E_p}) - d(\mathbf{E_a}, \mathbf{E_n}) + m, 0) \ .$$

Si cette fonction est utilisée, tous les triplets de la base de données ne donnent pas des informations utiles pour l'apprentissage (en particulier les triplets simples (paragraphe 2.2.3.2)).

Pour atteindre une convergence en un temps raisonnable, nous serions obligé de choisir les triplets utiles à l'apprentissage, en utilisant le *triplet mining* (paragraphe 2.2.3.2). De plus, comme dans le réseau siamois, le choix de la marge m est très versatile.

Sur la figure 49,

$$TL(\mathbf{E_a}, \mathbf{E_p}, \mathbf{E_n}) = 0$$

dans les deux cas, alors que le triplet de gauche est moins bien placé (a moins bien appris) que le triplet de droite.

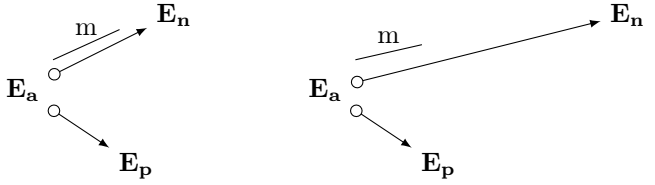


Figure 49: Problème avec le *triplet loss*, dans les deux exemples ci-dessus, $TL = 0$ car $m > D(\mathbf{E_a}, \mathbf{E_p})$. Le choix de la marge est capital.

Avec la même idée que dans les réseaux siamois, nous avons appliqué une fonction sigmoïde après le réseau convolutif *Inception-Resnet-v2* pour borner les valeurs du vecteur de caractéristiques dans $[0, 1]$.

Avec les mêmes notations que précédemment, nous pouvons alors transformer la fonction d'erreur en :

$$Loss(\mathbf{E_a}, \mathbf{E_p}, \mathbf{E_n}) = d(\mathbf{E_a}, \mathbf{E_p}) - d(\mathbf{E_a}, \mathbf{E_n}) + \sqrt{N}.$$

Cette fonction assure que l'erreur ne sera jamais nulle et que tous les triplets seront utiles à l'entrainement. Pour rendre cette fonction non-linéaire, sensible aux petites erreurs et améliorer l'entrainement, nous appliquons une fonction log sur chaque distance.

$$LLoss(\mathbf{E_a}, \mathbf{E_p}, \mathbf{E_n}) = \log(d(\mathbf{E_a}, \mathbf{E_p})) - \log(d(\mathbf{E_a}, \mathbf{E_n})) + \log(\sqrt{N}) \ .$$

Autre fonctions de pertes

Comme nous proposons une fonction de perte, nous avons comparé la vitesse d'entraînement ainsi que le précision globale sur différentes fonctions trouvées dans l'état de l'art. Nous les introduisons dans cette section.

Les auteurs de [115] considèrent l'entraînement comme une classification binaire, où la fonction objectif permet d'affirmer si les couples sont d'une même classe ou non. Dans ce cas, les auteurs remplacent le *triplet loss* par une simple erreur quadratique :

$$Loss(\mathbf{E_a}, \mathbf{E_p}, \mathbf{E_n}) = ||(d_p, d_n - 1)||_2^2$$

avec

$$d_p = \frac{exp\{||\mathbf{E_a} - \mathbf{E_p}||_2\}}{exp\{\mathbf{E_a} - \mathbf{E_p}||_2\} + exp\{||\mathbf{E_a} - \mathbf{E_n}||_2\}}$$

et

$$d_n = \frac{exp\{||\mathbf{E_a} - \mathbf{E_n}||_2\}}{exp\{||\mathbf{E_a} - \mathbf{E_n}||_2\} + exp\{\mathbf{E_a} - \mathbf{E_n}||_2\}}$$

Qian et al. introduisent dans [116] l'erreur *SoftTriple*, une fonction combinant un SoftMax et le *triplet loss* :

$$L_{SortTriple}(x_i) = -log\frac{exp(\lambda(S_{i,y_i} - \gamma)}{exp(\lambda(S_{i,y_i} - \gamma)) + \sum_{j \neq y_i} exp(\lambda S_{i,j})}$$

Avec

— $S_{i,x}$ une mesure de similarité entre l'instance $\mathbf{x}$ et la i^{em} classe.

— λ un terme de régularisation.

— γ une constante.

5.1.5 Représentation vectorielle, classification et métriques

5.1.6 Représentation vectorielle

Durant l'apprentissage, le réseau convolutif se termine par des couches complètement connectées. La sortie est un vecteur $\mathbf{E} = (y_1, ..., y_N)^\top$.

L'espace vectoriel créé a comme dimension le nombre de neurones choisi à la fin de ces couches. Il est ici de dimension $N = 512$. Afin de représenter l'espace vectoriel en deux dimensions, pour une analyse graphique des résultats, trois méthodes différentes ont été utilisées : ACP, Umap et l'ajoût d'une couche FC de dimension 2. Ces trois méthodes sont présentées en annexe .3 et donnent des résultats différents sur notre problème.

L'obtention d'une représentation vectorielle la plus correcte possible est capitale dans le processus final de classification.

5.1.6.1 Classification

Durant l'apprentissage, les vecteurs de la même classe vont être proches et vont former des groupes (*clusters*). À la fin de l'apprentissage, un espace vectoriel de dimension N (la taille du vecteur de caractéristiques choisi) sera créé. En utilisant cet espace vectoriel, nous pourrons visualiser nos instances de test, afin de vérifier le bon déroulement du processus d'entrainement .

1. Les centroïdes de chaque cluster d'entrainement sont calculés.
2. Le vecteur de caractéristiques du couple test est calculé en le passant au réseau entrainé.
3. La distance entre le vecteur test et chaque centroïde est calculée.
4. La distance minimale correspond à la classe choisie.

En d'autres termes, pour k classes avec $\mathbf{C} = (c_1, ..., c_k)$ les centroïdes des clusters, la classe $\hat{c}$ trouvée est la classe correspondante à la distance :

$$\hat{c} = ArgMin_{C_i} D(\mathbf{E_{test}}, c_i).$$

5.1.6.2 Métriques

Afin de comparer nos modèles à l'état de l'art, ainsi que fournir des éléments de référence, différentes métriques ont été utilisées.

Coefficient de silhouette

Les coefficients de silhouette ont été introduits dans [117] comme mesure de qualité d'une représentation d'un ensemble de données. Nous l'avons utilisée pour mesurer le bon fonctionnement du clustering. Son principe est expliqué ici.

Pour chaque élément $\mathbf{E_i}$ de la classe c_i nous notons :

— $a(\mathbf{E_i})$ la distance moyenne entre $\mathbf{E_i}$ et tous les points de la même classe :

$$a(\mathbf{E_i}) = \frac{1}{|C_I| - 1} \sum_{j \in C_I, j \neq i} d(\mathbf{E_i}, \mathbf{E_j})$$

— $b(\mathbf{E_i})$ la plus petite distance moyenne entre la classe i et les points des autres classes :

$$b(\mathbf{E_i}) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(\mathbf{E_i}, \mathbf{E_j}).$$

Le coefficient de silhouette d'une donnée E_i est défini par :

$$s(\mathbf{E_i}) = \frac{b(\mathbf{E_i}) - a(\mathbf{E_i})}{\max\{a(\mathbf{E_i}), b(\mathbf{E_i})\}}$$

Le coefficient de silhouette varie entre -1 (représentation aléatoire des données) et 1 (meilleure séparation des clusters et meilleure classification).

Précision, rappel et accuracy

La précision, l'accuracy et le rappel ont été utilisés pour présenter les résultats.

Notons dans la suite :
— TP_i : le nombre d'éléments de la classe i qui ont bien été prédits (vrais positifs) ;
— TN_i : le nombre d'éléments qui ont correctement été identifiés comme n'appartenant pas à la classe i (vrais négatifs) ;
— FN_i : le nombre d'éléments de la classe i déclarés comme appartenant à une autre classe (faux négatifs) ;
— FP_i : le nombre d'éléments déclarés à tort comme appartenant à la classe i (faux positifs) ;
— $|C_i|$: le nombre d'éléments de la classe i.
Ces éléments sont représentés sur la matrice de confusion (Figure 50 du point de vue de la classe C1).

— La précision pour une classe i est la proportion des éléments pertinents parmi l'ensemble des items proposés. Elle compte combien d'éléments ont été bien classés parmi tous les éléments. Elle est définie par : $precision_i = \frac{TP_i}{TP_i + FP_i}$.
— Le rappel (recall) pour une classe i prend en compte les faux négatifs, et précise à quel point le modèle est juste. S'il y a beaucoup de faux négatifs, cela veut dire que le modèle a surappris. Il est défini par : $recall_i = \frac{TP_i}{TP_i + FN_i}$.
— L'accuracy est une mesure globale du modèle, elle mesure à quel point le modèle est correct sur toutes les classes. Elle est généralement utilisée au cours de l'apprentissage. Elle est définie par : $accuracy = \frac{\sum_i TP_i}{\sum_i |C_i|}$.

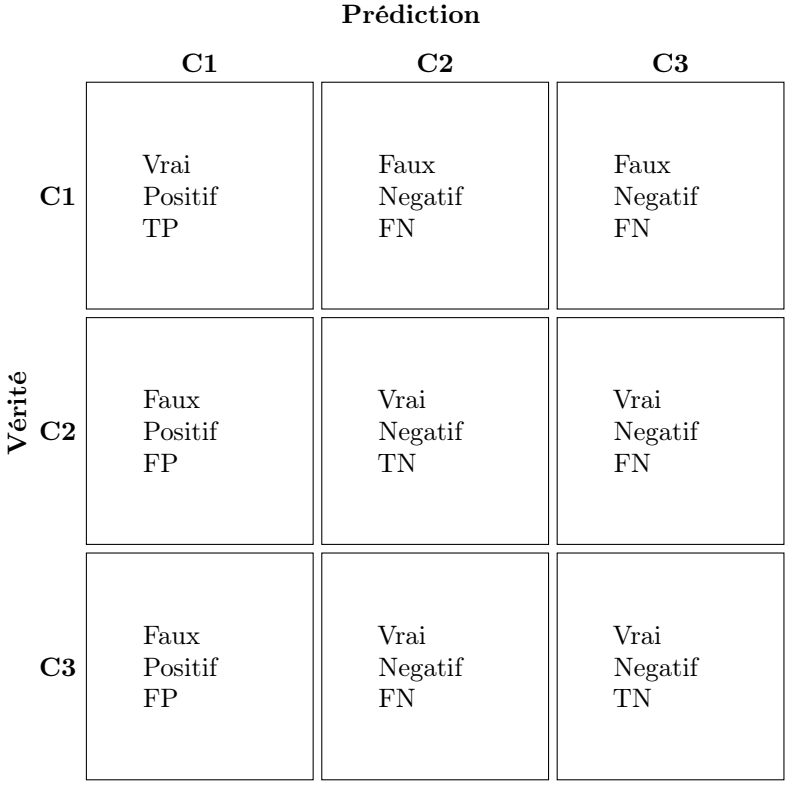


Figure 50: Matrice de confusion sur 3 classes, du point de vue de la classe C1.

Dans le cas général (k classes), une moyenne de la précision et du recall sur chaque classe est faite :

— $precision_{globale} = \frac{1}{k} \sum_i^k precision_i$;
— $recall_{globale} = \frac{1}{k} \sum_i^k recall_i$.

Comme l'accuracy est une mesure globale du modèle, rien ne sert d'en calculer la moyenne.

5.2 Résultats expérimentaux des méthodes d'apprentissages *Few Shots*

Cette partie présente les bases de données utilisées et les résultats de la détection d'émotions sur bases de données restreintes.

5.2.1 Bases de données

Dans le but de comparer le modèle avec l'état de l'art, ce dernier a été entrainé sur les quatre bases de données suivantes : Nimstim [31]-Ravdess [118], Cohn Kanade [96], Afew [15] et FER [99]. Pour chaque base de données, les triplets (C_A, C_P, C_N) ont été choisis aléatoirement avec la condition $C_A \neq C_P$ (i.e $I_A \neq I_P$ et $S_A \neq S_P$) afin d'éviter de biaiser le modèle.

5.2.1.1 Nimstim-Ravdess

Nimstim

Pour les images faciales, nous utilisons la base de données NimStim, présentée dans [31] et détaillée dans le tableau 8. Dans ce tableau, nous indiquons également le nombre de triplets utilisés par le réseau décrit dans la section 5.1.4.

Émotion	Colère	Dégoût	Peur	Bonheur	Tristesse	Surprise
Images d'entrainement	63	70	68	113	68	25
Images de test	10	10	10	10	10	10
Triplets d'entrainement	671 832	813 855	772 242	1 860 432	772 242	114 600
Triplets de test	2.25K	2.25K	2.25K	2.25K	2.25K	2.25K

Table 8: Base de données faciale : NimStim.

Les données sont des images studio RGB de taille 506×650. Un exemple est donné figure 46, image de gauche.

Ravdess

Pour les sons, la base de données Ravdess est utilisée ; elle est détaillée dans le tableau 9, et introduite dans [118]. Ce sont des phrases de trois secondes enregistrées en studio, chacune prononcée six fois avec les intonations correspondant aux différentes émotions de

base. Dans le tableau 9, nous indiquons également le nombre de triplets utilisés par le réseau décrit en section 5.1.4.

	# par emotion
Sons d'entrainement	100
Sons de test	10
Triplets d'entrainement	2.475M
Triplets de test	2.25K

Table 9: Nombre de phrases sonores sélectionnées de la base de données RAVDESS, en fonction de l'émotion (colère, dégoût, peur, bonheur, tristesse, surprise).

Couples

Nous concaténons l'image de la personne et le spectrogramme du son dans une seule image. Un exemple est donné dans la figure 46.

Afin de tester les limites du modèle, ce dernier a été entrainé sur des bases de données de couples Image-Son de différentes tailles.

Le tableau 10 présente la taille de ces bases de données ainsi que le nombre de triplets utilisés.

Couples (Image-Son) (# par emotion)	10	50	100	200	500
Triplets d'entrainement (# par emotion)	2 250	306 250	2.475M	19.9M	311.875M

Table 10: Nombre de couples Image-Son sélectionnés ainsi que le nombre de triplets effectivement utilisés au cours de l'entrainement, en fonction de l'émotion (colère, dégoût, peur, bonheur, tristesse, surprise).

5.2.1.2 Afew

Le modèle à aussi été testé sur la base de données de vidéos Afew. Le son et les images de taille 720×576 ont été extraits de la vidéo, et nous avons sélectionné 200 couples d'entrainement, ainsi que 1000 couples de test pour chaque classe. Les mêmes traitements que pour la base de données Ravdess, ont été appliqués au son. Un des sujets est présenté en partie (b) de la figure 51.

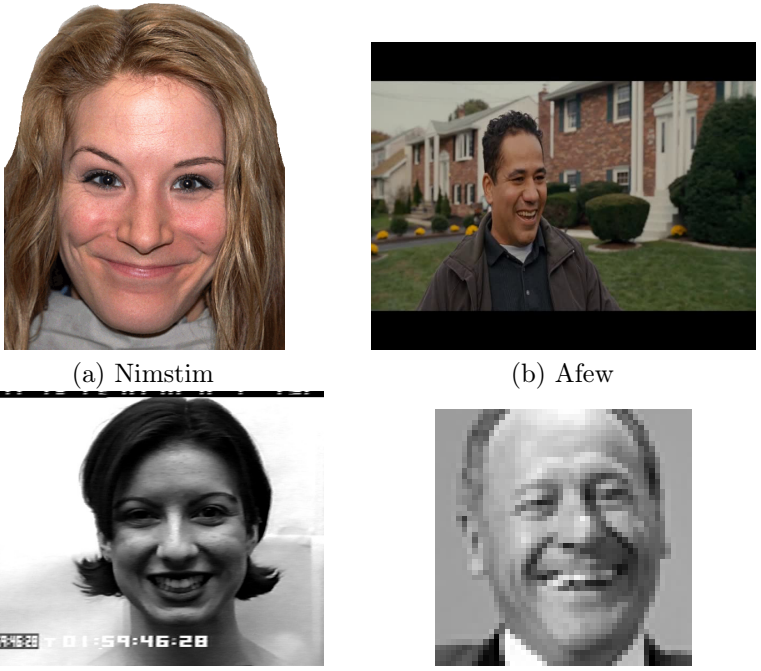
5.2.1.3 Cohn Extended

200 images d'entrainement, pour chaque émotion, ont été extraites des vidéos de la base de données CK+ [96]. Ces images sont des images en nuance de gris, prises dans un studio et de taille 640×490. Un sujet est présenté sur la partie (c) de la figure 51.

5.2.1.4 FER

La base de données FER contient des petites images faciales, en niveau de gris, de taille 48×48. Sur chaque image, le visage est centré et occupe la même place dans l'image. Cette base de données est un mélange d'images prises en studio, en conditions réelles ou même dessinées.

200 images d'entrainement pour chaque émotion, ainsi que 100 images de test, ont été sélectionnées. Un sujet est présenté dans la partie (d) de la figure 51.



(a) Nimstim (b) Afew

(c) Cohn Extended (d) FER.

Figure 51: Exemples d'images faciales labélisées *bonheur*, tirées des 4 bases de données utilisées.

5.2.2 Entrainement

Le réseau *Inception-Resnet-v2* [4] a, dans un premier temps, été entrainé sur la base de données de visages CASIA-WebFace [119] avec un objectif de reconnaissance faciale. Cette base de données est formée de 453 453 images de 10 575 sujets différents. Nous avons utilisé les poids de ce réseau entrainé pour l'initialisation du modèle. Il est très difficile d'entrainer correctement les poids des premières couches de convolution sur des données restreintes. Cette méthode de *finetuning* permet d'obtenir des poids acceptables sur les premières couches de convolution et d'améliorer la performance du réseau. Nous avons continué l'entrainement sur toutes les couches afin d'améliorer les poids pour correspondre à nos données.

Par souci de mémoire, nous avons sélectionné aléatoirement, pour chaque epoch, 20 images dans chaque classe et appris sur tous les triplets possibles. Une epoch représente donc tous les triplets possibles d'un sous groupe de la base de données entière.

Les réseaux ont été entrainés avec :
— l'optimiseur Adam ;
— un pas d'apprentissage décroissant exponentiellement avec comme valeur initiale 0.01 ;
— 40% de dropout sur les dernières couches complètement connectées afin d'éviter le surapprentissage.

Les réseaux ont été implémentés à l'aide du *framework* Tensorflow [1]. Pour la phase d'entrainement, une carte graphique TitanX Pascal ainsi que 64 GiB de RAM ont été utilisés.

5.2.3 Résultats

5.2.3.1 Réseau de neurones siamois

Dans cette section nous présentons la précision du réseau siamois, l'erreur et la représentation vectorielle des six émotions de base pendant l'apprentissage sur les quatre bases de données d'entrainement.

1. https ://tensorflow.org

Base de données	Réseau	Traitement	#Label	Acc
Afew (1400)	Siamese-Inc-Res	Image+Haar	7	38.6%
FER (1400)	Siamese-Inc-Res	Image	7	54.8%
Nimstim+Ravdess (1200)	Siamese-Inc-Res	Image+Haar	6	72.2%
CK+ (1200)	Siamese-Inc-Res	Image	6	74.6%

Table 11: Précision du modèle siamois sur différentes bases de données.

Précisions

La figure 11 présente la précision du réseau siamois avec notre fonction de perte, sur les cinq bases de données testées.

— La base de données Afew, est une base de données d'images complexes à classifier étant donné que les personnes d'intérêt peuvent se placer au second plan. Nous pensons qu'il faudrait davantage de données pour un meilleur apprentissage de ce réseau.

— La base de données Fer est une base de données de petites images, le manque d'information nuit considérablement à l'apprentissage. Cette base de données est aussi très versatile, elle contient des images en studio, des images en conditions réelles, des dessins. Elle est aussi très difficile à classifier.

— La base de données Nimstim+Ravdess est une base de données en studio. Le réseau atteint 72.2% en apprenant sur 1200 couples.

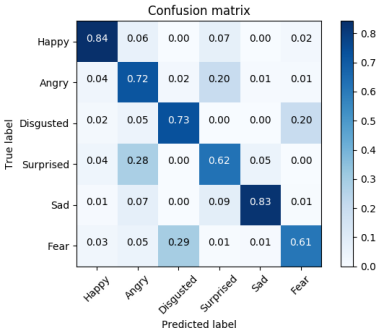
— La base de données CK+ est une base de données en studio de séquence d'images. Le réseau atteint 74.6% en apprenant sur 1200 couples.

Il faut prendre en compte que les bases de données Afew et Fer sont entraînées et testées sur 7 classes, cependant le modèle est loin d'être utilisable pour ces bases de données. Les deux derniers résultats sont encourageants, ils sont cependant inférieurs à l'état de l'art. Pour l'atteindre, il ne faudrait peut être pas se limiter à 200 instances par classes, mais en rajouter afin d'améliorer le processus d'apprentissage.

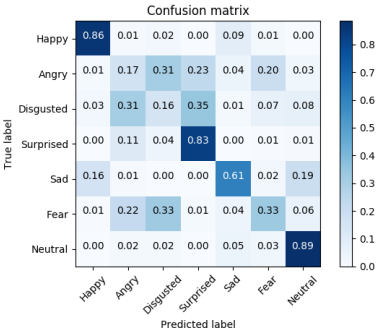
Matrice de confusion

La figure 52 présente les matrices de confusion sur chaque base de données. Le rappel et l'accuracy global pour chaque base de données ont aussi été calculés

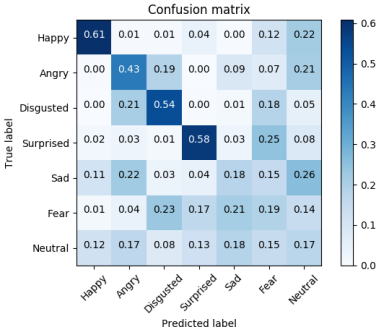
Les matrices de confusion de Nimstim-Ravdess et CK+ sont comparables en certains points. Nous pouvons en effet constater que le dégoût se confond par exemple avec la peur



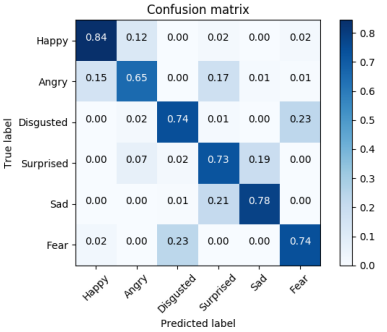
(a) Matrice de confusion sur Nimstim-Ravdess , 72.2% accuracy globale, 0.74 rappel global.



(b) Matrice de confusion sur FER, 54.8% accuracy globale, 0.53 rappel global.



(c) Matrice de confusion sur Afew, 38.6% accuracy globale, 0.31 rappel global.



(d) Matrice de confusion sur CK+, 74.6% accuracy globale, 0.75 rappel global.

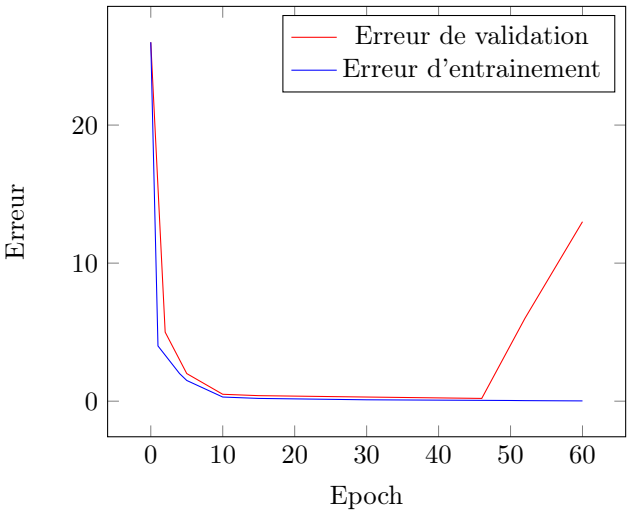
Figure 52: Matrices de confusion sur différentes bases de données, chaque modèle entrainé avec 200 couples, représenté avec une carte de couleur normalisée.

et que la surprise se confond avec la colère et la tristesse. Des émotions que même une personne aurait, dans certains cas, du mal à distinguer. Pour les deux autres matrices, comportant 7 classes, l'apprentissage n'a pas été assez performant. En effet certaines classes n'ont pas été apprises et donnent des résultats proches de l'aléatoire. Une comparaison est donc très difficile.

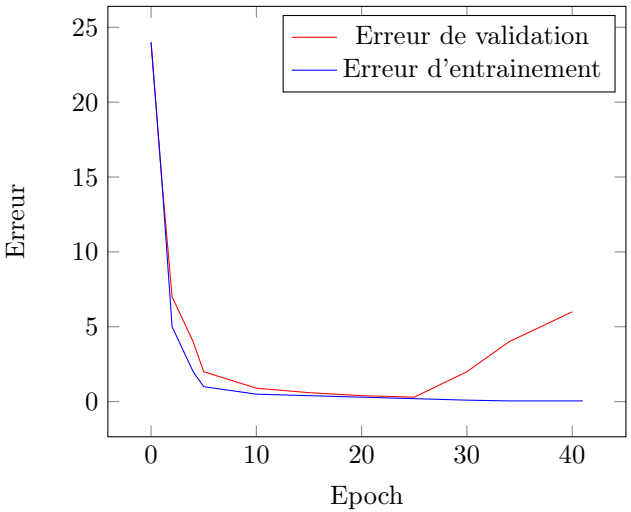
Erreur

Afin d'éviter le surapprentissage, un ensemble de validation a été utilisé. L'entrainement s'arrête lorsque l'erreur de validation augmente de manière statistiquement significative. La figure 57 présente l'évolution des erreurs d'apprentissage et de validation sur les 4

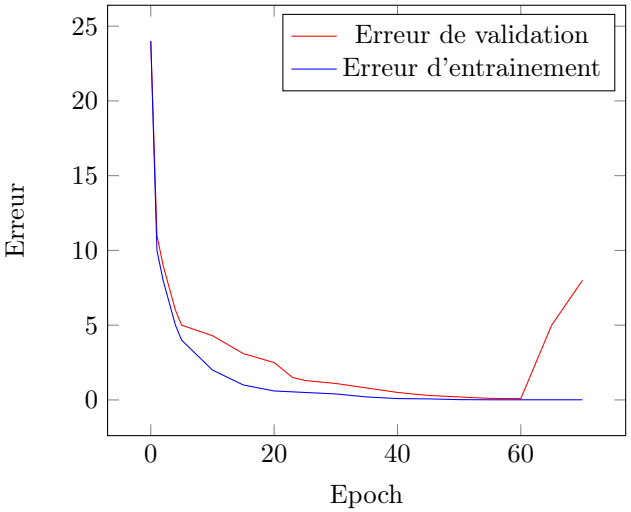
bases de données.



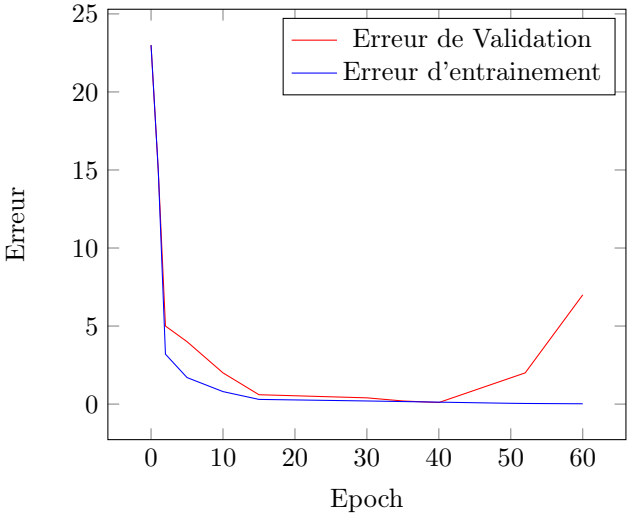
(a) Nimstim-Ravdess



(b) FER



(c) Afew



(d) CK+

Figure 53: Erreur en fonction du nombre d'epochs pour les ensembles d'entrainement et de validation.

Nous remarquons que l'apprentissage sur la base de données FER s'arrête plus tôt. Les images étant petites (48×48), le réseau à moins d'information pour généraliser, et il peut surapprendre plus rapidement.

Base de données	Score de silhouette moyen
Afew	0.43
FER	0.58
Nimstim+Ravdess	0.60
CK+	0.62

Table 12: Scores de silhouettes moyens sur les différentes bases de données.

Représentation vectorielle

Afin de vérifier le bon apprentissage du modèle, nous avons procédé à une réduction de dimension afin de projeter nos données dans un espace vectoriel de dimension 2.

La figure 54 présente l'entrainement sur la base de données Nimstim Ravdess, cette représentation vectorielle de deux dimensions a été obtenue en rajoutant une couche *fully connected* de taille 2, à la fin du réseau. L'entrainement est semblable sur trois des autres bases.

Nous remarquons sur la représentation finale (e) que les clusters représentant la peur et le dégoût sont très proches, ainsi que les clusters de la colère et de la surprise. Cela explique les erreurs commises aperçues sur les matrices de confusion.

SIL

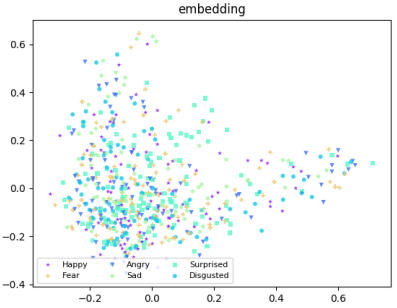
Afin de mesurer l'exactitude de la représentation, nous utilisons l'analyse par silhouette [117]. Les figures 55 et 56 montrent l'analyse par silhouette sur chaque classe et sur les différentes bases de données.

Les scores de silhouette moyens sur les différentes bases de données sont présentés dans le tableau 12.

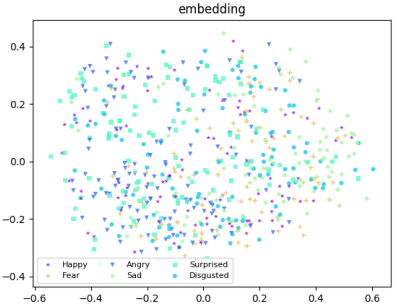
Ces scores sont encourageants pour les bases de données Nimstim-Ravdess et CK+.

Sur la base de données Fer, le modèle donne de moins bon résultats. Cela est probablement dû à la présence d'une 7ème classe (neutre) ainsi qu'à la taille des images (48×48). Pour obtenir de meilleurs résultats, davantage de données d'entrainement doivent sûrement être utilisées (afin d'obtenir la même *quantité* d'information que sur les autres bases de données).

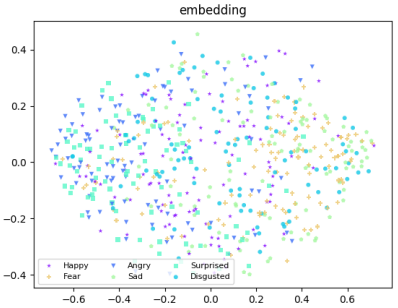
Le modèle échoue cependant sur la bases de données Afew probablement car les images sont complexes à classifier. Comme ce sont des images tirées de films, l'acteur peut être



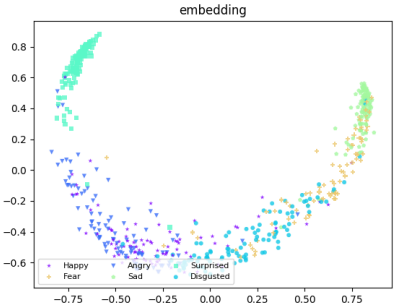
(a) Représentation initiale (début de l'apprentissage).



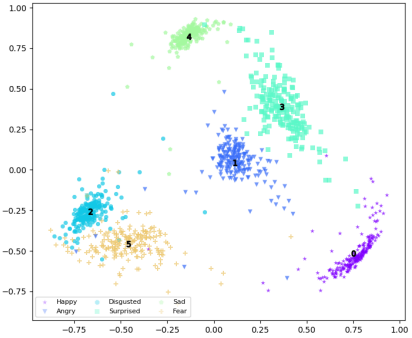
(b) Représentation après 5 epochs.



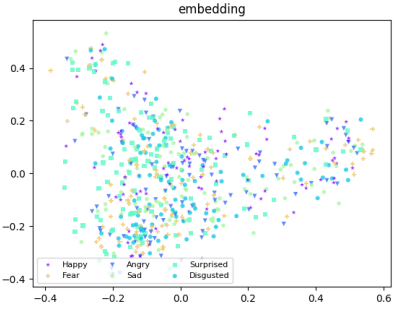
(c) Représentation après 15 epochs.



(d) Représentation après 30 epochs.

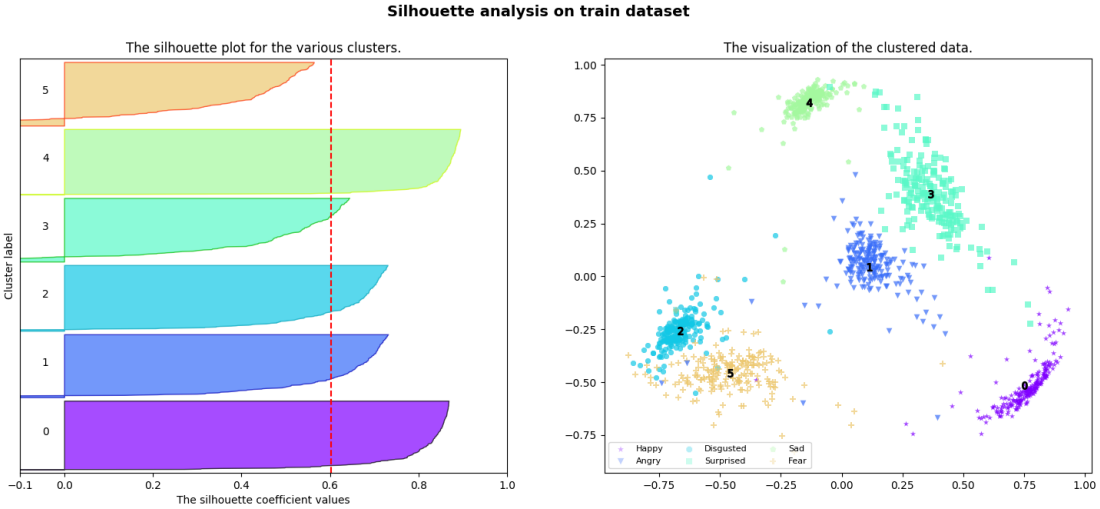


(e) Représentation finale après 43 epochs.

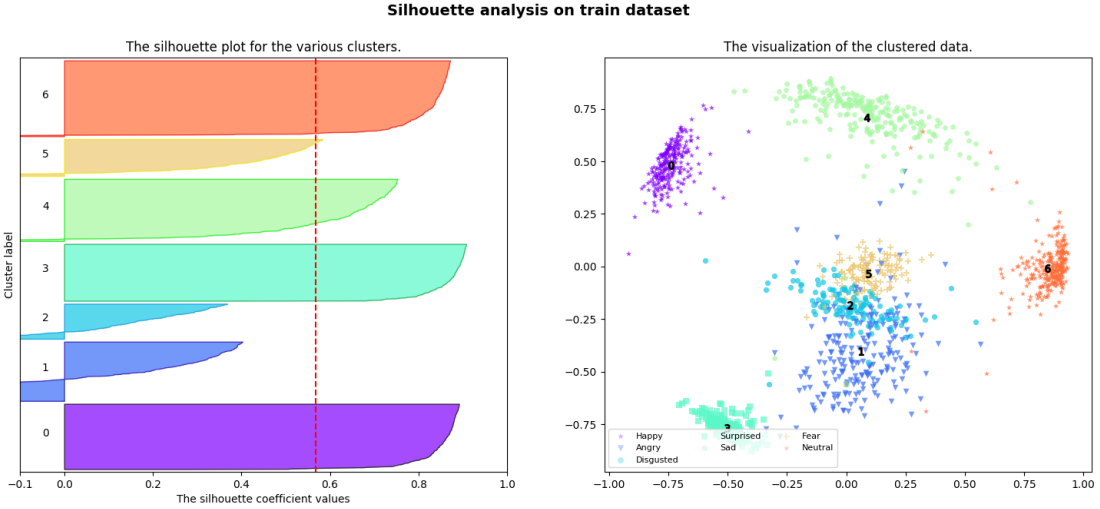


(f) Surapprentissage après 43 epochs.

Figure 54: Représentation vectorielle des six émotions de base, pendant la phase d'apprentissage sur la base d'entrainement d'images et de sons Nimstim-Ravdess du réseau siamois.



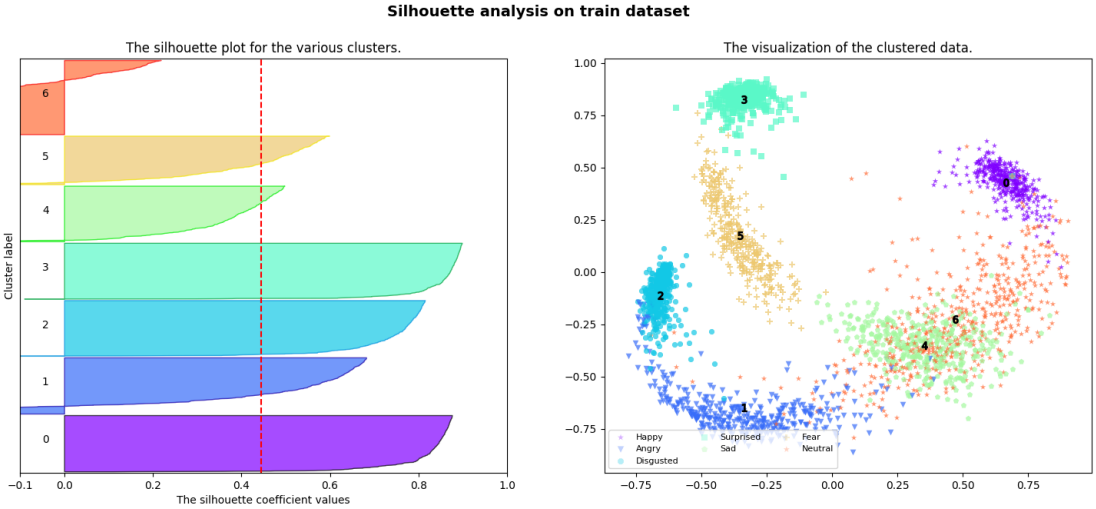
Nimstim-Ravdess



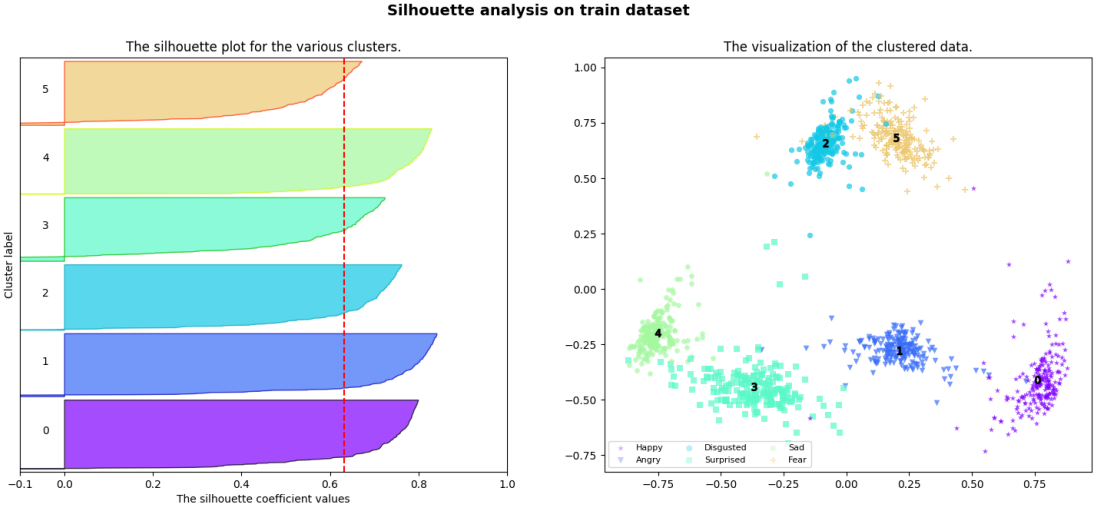
FER

Figure 55: Analyse par silhouette sur les différentes bases d'entrainement avec présentation du centroïde de chaque cluster.

au second plan de l'image ou non. Cette base de données contient beaucoup de variétés d'images différentes et 200 semblent trop peu pour l'apprentissage.



Afew



CK+

Figure 56: Analyse par silhouette sur les différentes bases d'entrainement avec présentation du centroïde de chaque cluster.

5.2.3.2 Réseau triples

Dans cette section nous présentons la précision du réseau triple, l'erreur et la représentation vectorielle des six émotions de base pendant l'apprentissage sur les quatre bases de données d'entrainement.

Erreur

Nous utilisons un ensemble de validation pour éviter le surapprentissage. L'entrainement est arrêté lorsque l'erreur de validation augmente. La figure 57 présente l'évolution des erreurs d'apprentissage et de validation sur les quatre bases de données.

Même remarque que pour les réseaux siamois : l'apprentissage sur la base de données FER s'arrête plus tôt que les autres. Un peu plus tard, tout de même que dans le réseau siamois. Cela est sûrement dû aux données en plus (provenant des triplets) et pourrait expliquer le gain en précision. Pour le reste, l'apprentissage est comparable au réseau siamois. Cependant, une epoch représente bien davantage de données pour les réseaux triples que pour les réseaux siamois ; en effet, le premier prend tout les triplets d'un batch et le second seulement les paires. Globalement, le réseau triple va apprendre sur davantage d'instances, avant le surapprentissage, ce qui explique les meilleures précisions par la suite.

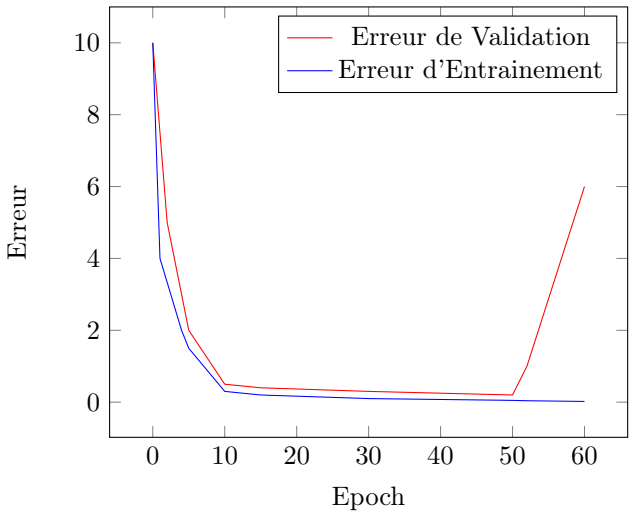
Fonction de perte

Afin de comparer notre approche à l'état de l'art, différentes fonctions d'erreur ont été implémentées. Les résultats sont présentés dans la table 13. La précision est calculée sur le nombre de triplet bien placés i.e si $d(E_a, E_p) < d(E_a, E_n)$. Le temps global d'apprentissage (contenant le *triplet mining* est aussi représenté.

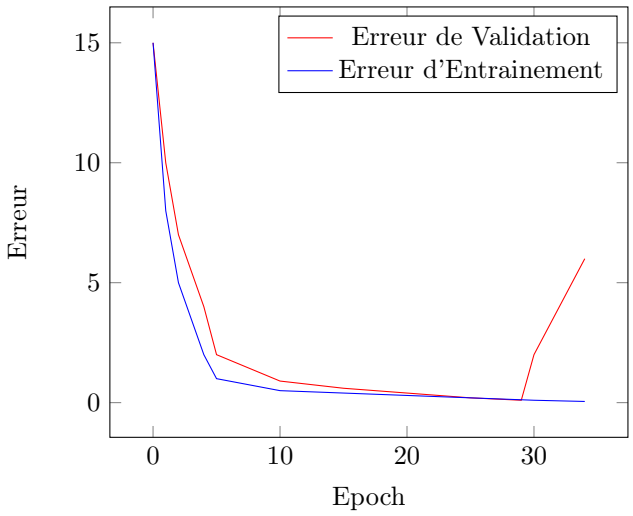
Fonction de perte	Précision	Temps
Triplet Loss + Data Mining	82%	46h
Loss [115] +Data Mining	81%	34h
SoftLoss [116]	85%	26h
Notre Loss	**86%**	**22h**

Table 13: Comparaison de différentes fonctions de perte, testées sur la base de données NimStim.

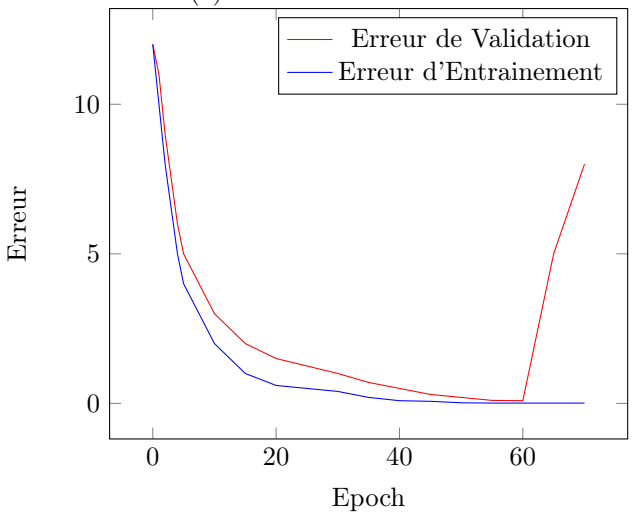
Nous voyons que grâce à l'absence de *Data Mining* notre fonction est plus rapide (22h),



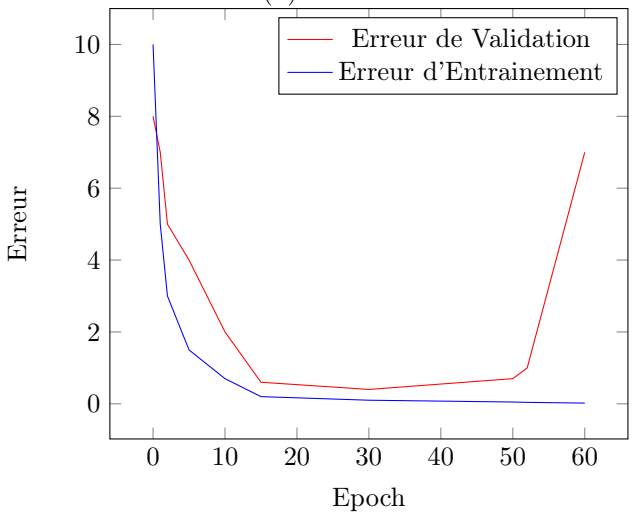
(a) Nimstim-Ravdess



(b) FER



(c) Afew



(d) CK+

Figure 57: Erreur en fonction du nombre d'epochs pour les ensembles d'entrainement et de validation.

et grâce a l'absence de triplets inutiles, la précision est légerement améliorée (+1 point).

Précisions

Précision des réseaux triples sur Nimstim-Ravdess

Nous présentons dans le tableau 14, les résultats de notre approche pour différents prétraitements et différents réseaux triples. Dans ce tableau, la précision est calculée sur le nombre de triplets que le réseau place correctement (i.e. tels que $d(A,P) < d(A,N)$).

Afin de comparer nos résultats sur cette métrique, nous avons aussi entrainé le réseau sur les images seules, et les sons seuls. Nous avons également entrainé le réseau sur le son pur et non le spectrogramme, puis utilisé d'autres réseaux pour la représentation. Les résultats sont meilleurs avec les spectrogrammes. De plus, le réseau est mieux adapté à des images et identifie mieux leurs régions significatives. Les tests ont été effectués sur la base de données Nimstim-Ravdess décrite dans la section 5.2.1.1. Les personnes (images faciales et phrases sonores) testées n'ont jamais été vues pendant la phase d'entrainement. Dans le tableau, la colonne *Temps* décrit le temps total d'entrainement du réseau. Le modèle avec le réseau Inception-Resnet-v2, permet de placer correctement les triplets de test avec une précision de **87%**. Nous notons que la fonction de perte proposée, améliore de 5 points cette précision par rapport au *triplet loss*. Nous remarquons aussi que le modèle apprend un peu mieux sur les sons traités avec la transformée de Haar qu'avec l'échelle de Mel.

Nombre de données	Réseau	Traitement	Précision	Temps
Image				
407	Triplet Alexnet	RGB	72%	12h
407	Triplet Resnet	RGB	75%	16h
407	Triplet *Inc-Res*	RGB	80%	16h
Audio				
600	Triplet *Inc-Res*	Son Pure	70%	12h
600	Triplet *Inc-Res*	MFCC	75%	12h
Image + Audio				
1200	Triplet *Inc-Res*	RGB+Son Pure	60%	24h
1200	Triplet *Inc-Res*	RGB+filtre lissant	70%	24h
Image + Spectrogram				
1200	*Inc-Res*+ Triplet Loss	RGB+Mel-Spec	82%	24h
1200	Linear Triplet Loss	RGB+Mel-Spec	84%	24h
1200	Non Linear Triplet Loss	RGB+Mel-Spec	85%	24h
1200	**Non Linear Triplet Loss**	**RGB+Haar-Wavelet**	**87%**	**24h**

Table 14: Comparaison de différentes approches sur la base de donnée Nimstim-Ravdess. On note Inc-Res le réseau Inception-Resnet-v2.

Base de données	Réseau	Traitement	#Label	Acc
Afew (1400)	**Triplet-Inc-Res**	**Image+Son**	**7**	**54.4%**
Afew (958)+FER (28709)	CNN et *Finetuning* [82]	RGB	7	55.6%
FER (28709)	CNN + Attention [120]	RGB	7	70.02%
FER (1400)	**Triplet-Inc-Res**	**Image**	**7**	**73.3%**
Nimstim+Ravdess (1200)	**Triplet-Inc-Res**	**Image+MFCC**	**6**	**80.2%**
Nimstim+Ravdess (1200)	**Triplet-Inc-Res**	**Image+Haar**	**6**	**80.5%**
CK+ (1200)	**Triplet-Inc-Res**	**Image**	**6**	**83.9%**
CK+ (450K)	Siamese Network + CNN [104]	Image	6	98.9%

Table 15: Précision du modèle en relation avec l'état de l'art sur différentes bases de données. Gras : Précision de notre algorithme

Précision couple Image-Son

Le tableau 15 montre une comparaison entre notre approche et l'état de l'art sur la détection d'émotions en utilisant l'apprentissage profond. La précision du modèle est le pourcentage de couples test Image-Son bien classifiés. Elle est mesurée grâce à la distance entre les centroïdes de la base d'entrainement et le vecteur de caractéristiques de tests.

En entrainant sur 200 couples pour chaque classe, 54.4% d'accuracy a été obtenue sur la base de données Afew, 70.5% sur la base de données FER, 80% sur Nimstim-Ravdess et 84.4% CK+.

Les résultats sont présentés par ordre croissant de précision.

Nous atteignons une meilleure précision sur la base de données FER, en utilisant 24 fois moins de données que [120]. Les résultats sur la base de données Afew sont comparables à [82] en utilisant un nombre de données semblable. On peut, par ailleurs, noter que [82] utilise de l'apprentissage par transfert sur un problème de détection d'émotions, en entrainant d'abord son réseau sur la base de données FER.

Avec 375 fois moins de données que dans [104], le modèle atteint 84.4% sur la base de données CK+, ce qui nous parait un résultat convenable, compte tenu du nombre de données d'entrainement.

Nous notons que les réseaux triples sont bien plus efficaces que les approches traditionnelles, lorsque le nombre de données est restreint. Il peut même rivaliser avec des modèles classiques entrainés sur une base bien plus grande.

Le réseau a aussi été entrainé sur un nombre différent de données, sur la base de données Nimstim-Ravdess. Les résultats sont présentés dans le tableau 16. Tous nos résultats sont présentés avec 200 couples (image-son) par émotion, car c'est avec ce nombre que le modèle

atteint plus de 80% de précision sur la première base de données testée, NimStim Ravdess.

Nombre de données d'entrainement (# par émotion)	Accuracy
10	35.2%
50	46.4%
100	64.5%
200	80.5%
500	85.2%

Table 16: Résultats du modèle sur un nombre différent d'instances de la base de données NimStim Ravdess.

Matrice de confusion

La figure 58 présente les matrices de confusion sur chaque base de données. Le rappel et l'accuracy global pour chaque base de données ont aussi été calculés.

Les deux matrices sur la base de données Nimstim-Ravdess et CK+ sont comparables, bien que les résultats sur CK+ sont globalement meilleurs, et que le modèle semble avoir surappris l'émotion *heureux* dans le premier cas.

L'ajout d'une 7ème classe détériore les résultats, mais le modèle n'arrive toujours pas à bien apprendre sur la base de données Afew. Nous pensons qu'il faudrait plus d'images afin d'obtenir une meilleure représentation des clusters et ainsi une meilleure précision.

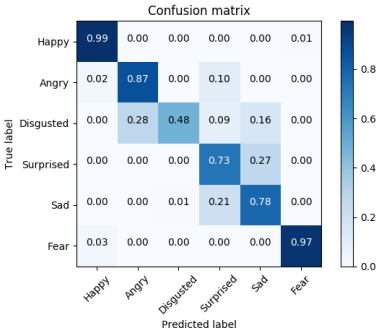
Sur la base Fer, le réseau triple apprend mieux que les réseau siamois, 73.3% devient maintenant un résultat très encourageant, surpassant [120] de 3 points, en utilisant 20 fois moins de données.

Représentation

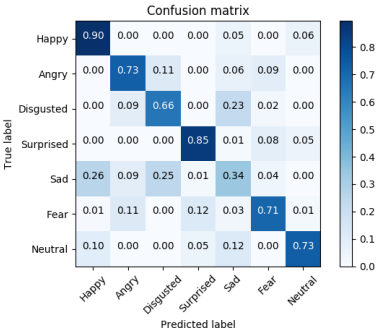
Pour obtenir une représentation vectorielle à deux dimensions, nous rajoutons une couche entièrement connectée (FC) de taille deux à la fin du réseau *Inception-Resnet-v2*. Les individus sont par conséquent projetés dans ce nouvel espace afin de déterminer si le réseau triple permet la distinction des classes.

Les algorithmes ACP, UMAP ainsi que t-SNE ont également été utilisés à cet effet.

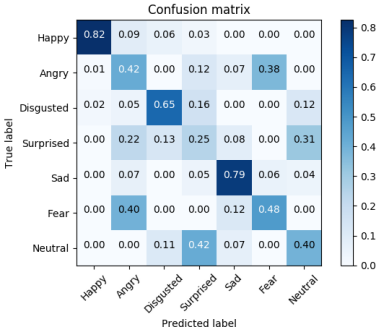
La figure 59 présente les projections des couples d'entrainement, en fonction de l'epoch, sur la base de données Nimstim-Ravdess. L'apprentissage sur les autres bases de données montre un apprentissage similaire.



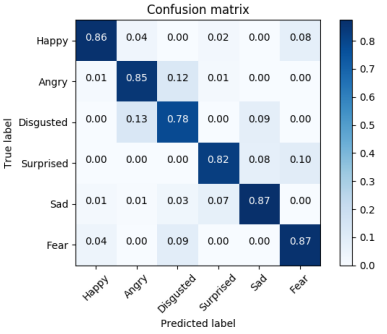
(a) Matrice de confusion sur Nimstim-Ravdess, 80.5% accuracy globale, 0.83 rappel global.



(b) Matrice de confusion sur FER, 73.3% accuracy globale, 0.7 rappel global.



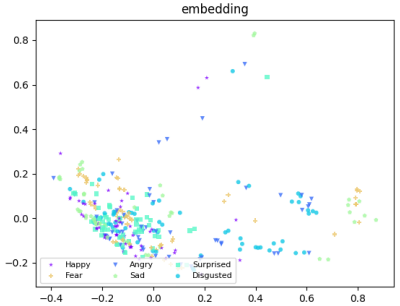
(c) Matrice de confusion sur Afew, 54.4% accuracy globale, 0.6 rappel global.



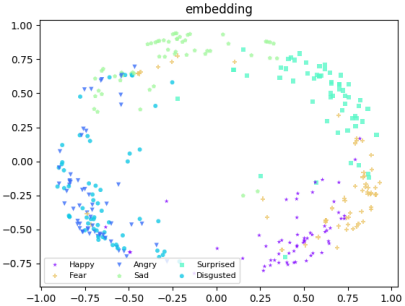
(d) Matrice de confusion sur CK+, 83.9% accuracy globale, 0.85 rappel global.

Figure 58: Matrices de confusion sur différentes bases de données, chaque modèle entrainé avec 200 couples, représenté avec une carte de couleur normalisée.

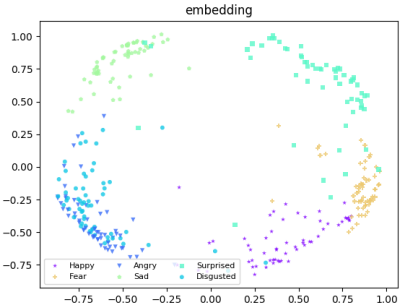
À la fin de l'apprentissage, le réseau génère six groupes homogènes dans l'espace vectoriel ; il nous offre une représentation des émotions de base. Il place la plupart des instances dans les bonnes régions et arrive à séparer les différentes classes sans indication explicite du nombre de classes ou du label de la donnée, uniquement avec l'indication *Positif/Négatif*. Lors de la phase d'entrainement, le modèle confond la peur et la surprise, ainsi que la colère et le dégoût. Le réseau met un certain temps à bien apprendre la différence entre ces classes car elles sont assez similaires.



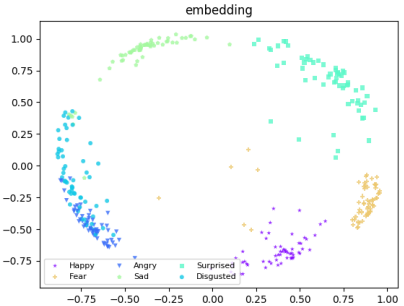
(a) Représentation initiale (début de l'apprentissage).



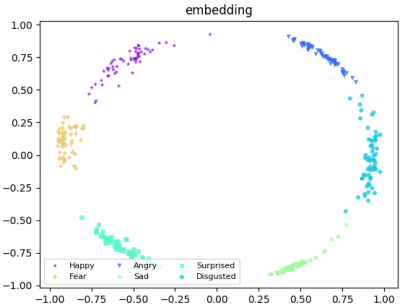
(b) Représentation après 5 epochs.



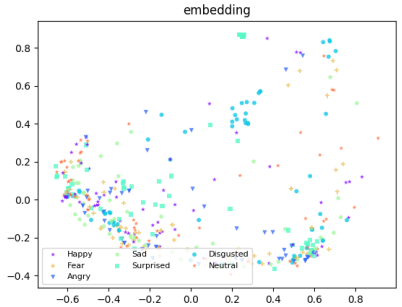
(c) Représentation après 15 epochs.



(d) Représentation après 30 epochs.



(e) Représentation finale après 50 epochs.



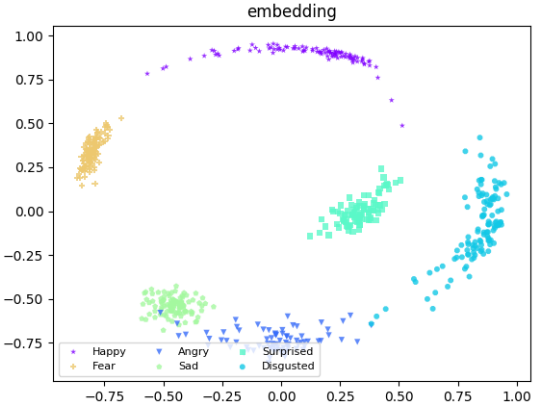
(f) Surapprentissage après 50 epochs.

Figure 59: Représentation vectorielle des six émotions de base, pendant la phase d'apprentissage sur la base d'entrainement d'images et de sons Nimstim-Ravdess.

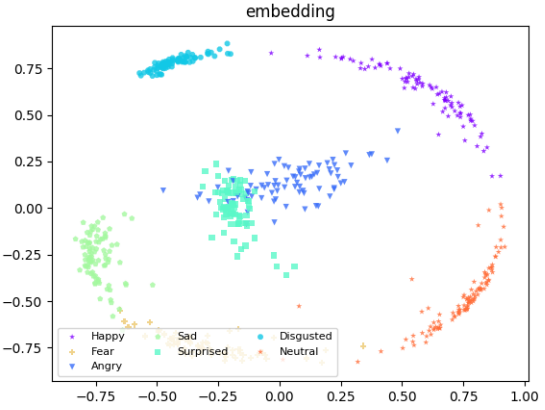
Réduction de dimension

Afin de vérifier le bon apprentissage du modèle, nos instances d'entraînement sont projetées dans un espace vectoriel de dimension 2. Les 4 méthodes présentées dans .3 sont ici testées.

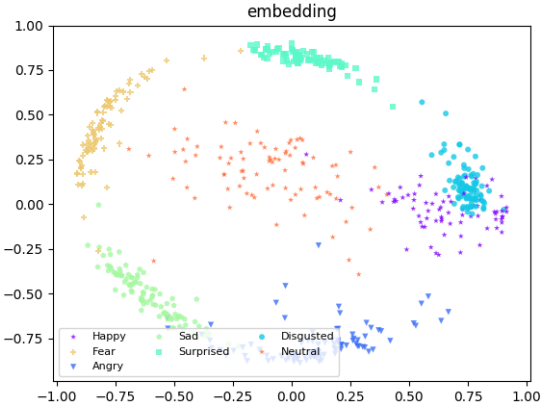
La figure 60 montre l'espace des composantes principales sur les quatre bases de données. Nous pouvons remarquer le placement des clusters sur un cercle, très semblable à la représentation apprise par le *fully connected* de taille 2.



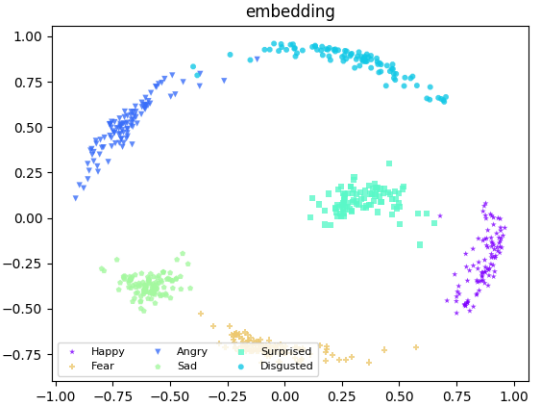
(a) Nimstim-Ravdess



(b) FER



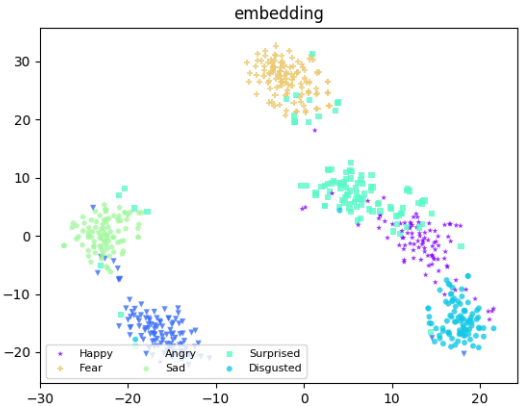
(c) Afew



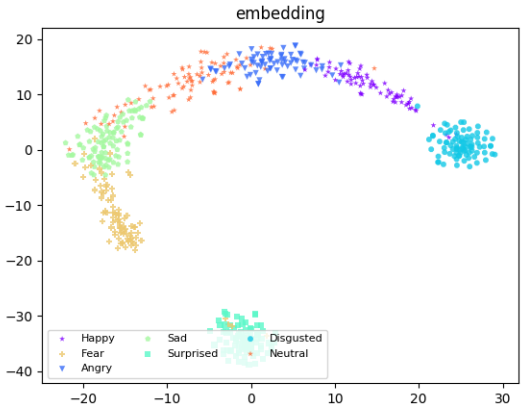
(d) CK+

Figure 60: Algorithme ACP sur les quatre bases de données

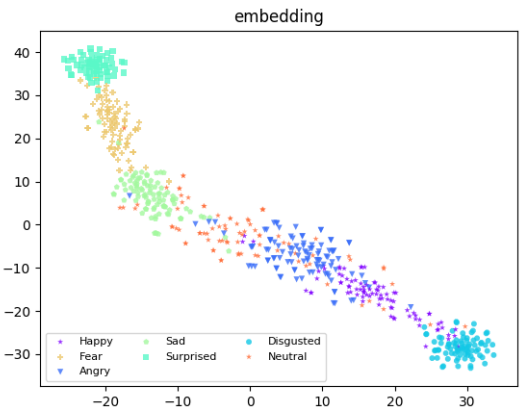
La figure 61 montre l'espace obtenu par t-SNE. À part pour la base de données CK+, l'algorithme t-SNE ne donne pas des groupes bien séparés. L'algorithme a été appliqué plusieurs fois, et la représentation est à chaque fois semblable. Certains groupes se mélangent même visuellement, l'algorithme t-SNE est celui séparant le moins bien les émotions.



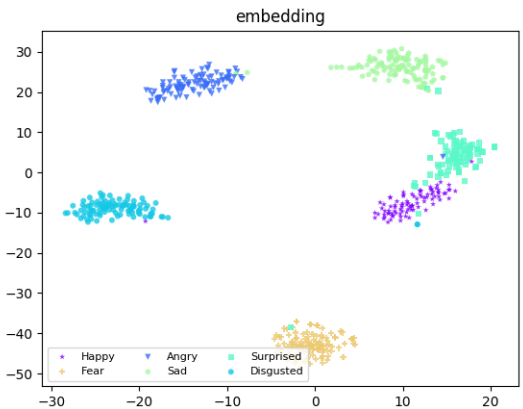
(a) Nimstim-Ravdess



(b) FER



(c) Afew



(d) CK+

Figure 61: Algorithme t-SNE sur les quatre bases de données.

Les figures 62 et 63 montrent l'espace obtenu par UMAP. Visuellement, la représentation avec UMAP possède une meilleure séparation pour chaque classe. Nous pouvons aussi noter le peu de connexions entre les clusters sur l'étude des connexions entre clusters,

Base de données	Score de silhouette moyen
Afew	0.56
FER	0.61
Nimstim+Ravdess	0.80
CK+	0.79

Table 17: Scores de silhouette moyens sur les différentes bases de données.

indiquant la justesse de cette représentation.

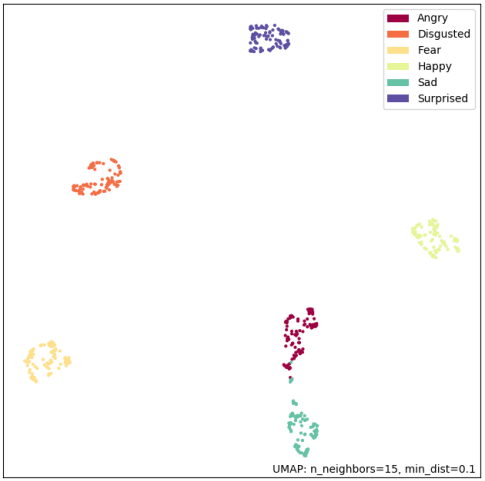
Les quatre méthodes de projection permettent de distinguer les 6 classes dans R^2, cependant, la nature aléatoire de certaines méthodes rend difficile l'explication des différents résultats. En revanche, il est intéressant de noter les liens entre les différentes classes donnés par l'algorithme UMAP.

SIL

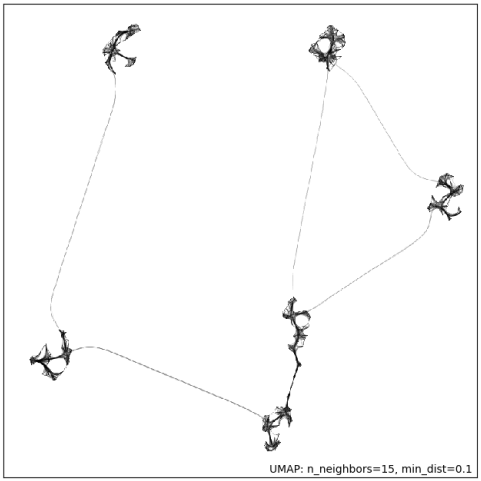
Les Figures 64 et 65 montrent l'analyse par silhouette sur chaque classe et sur les différentes bases de données.

Les scores de silhouette moyens sur les différentes bases de données sont présentés dans le tableau 17.

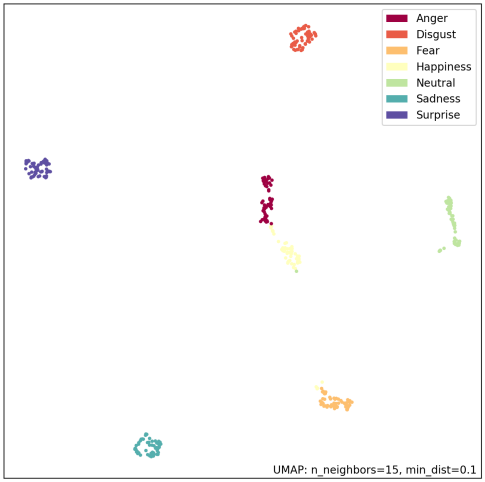
Nous notons que les scores ont globalement augmenté par rapport à l'utilisation des réseaux siamois. Les différentes représentations semblent maintenant utilisables sur les différentes bases de données ; cependant le modèle possède encore des défauts car il semble échouer partiellement (comme l'état de l'art) sur les bases de données *en condition réelle*.



(a_1) UMAP Nimstim-Ravdess



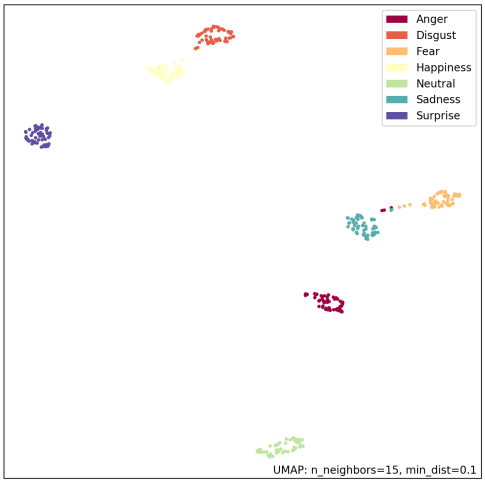
(a_2) Connexion des clusters Nimstim-Ravdess



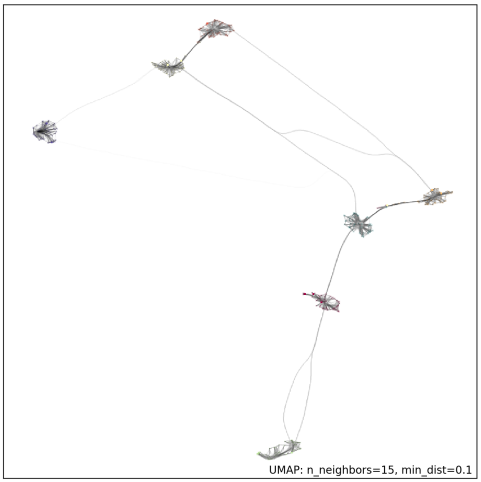
(b_1) UMAP Fer

(b_2) Connexion des clusters Fer

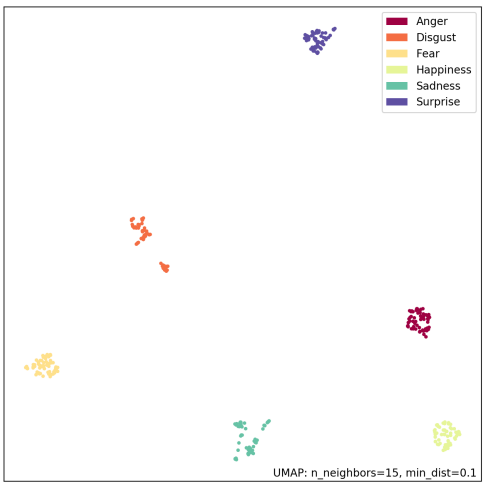
Figure 62: UMAP et *Cluster Connectivity* sur les données d'entrainement.



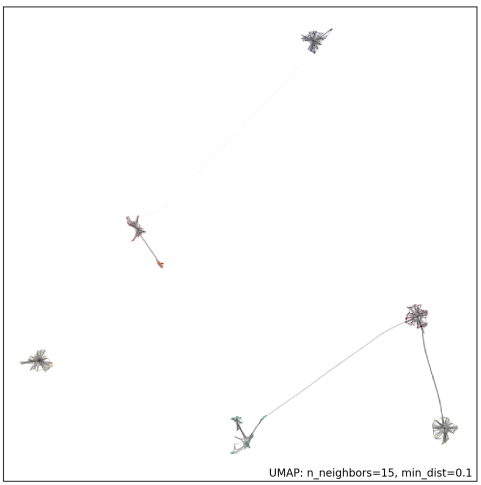
(a_1) UMAP EmotiW



(a_2) Connexion des clusters EmotiW

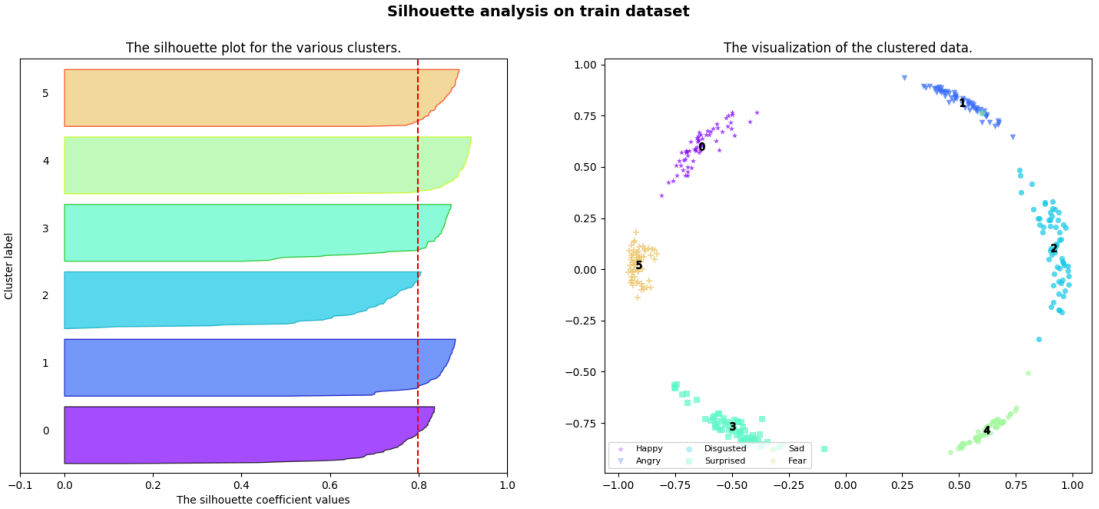


(b_1) UMAP CK+

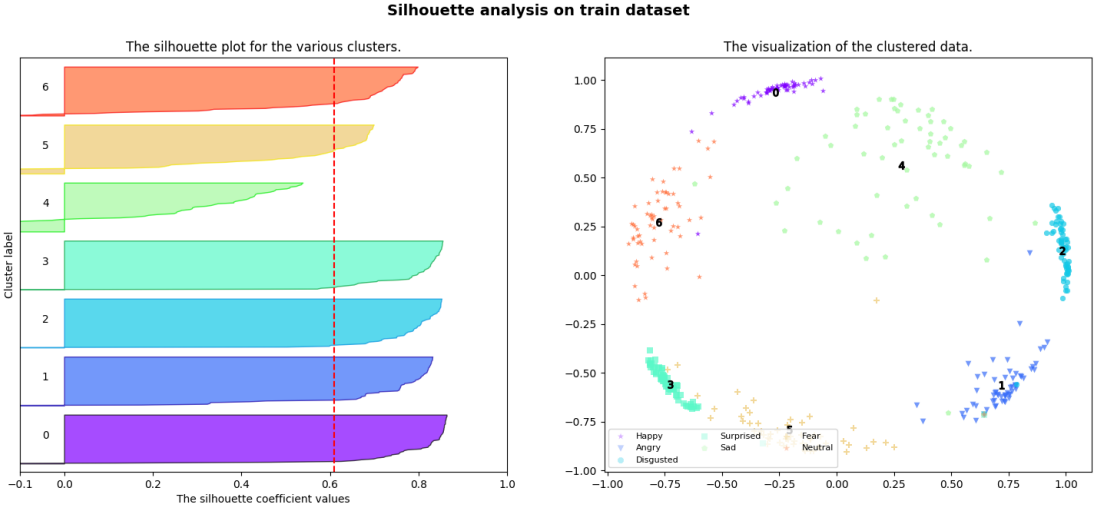


(b_2) Connexion des clusters CK+

Figure 63: UMAP et *Cluster Connectivity* sur les données d'entrainement.

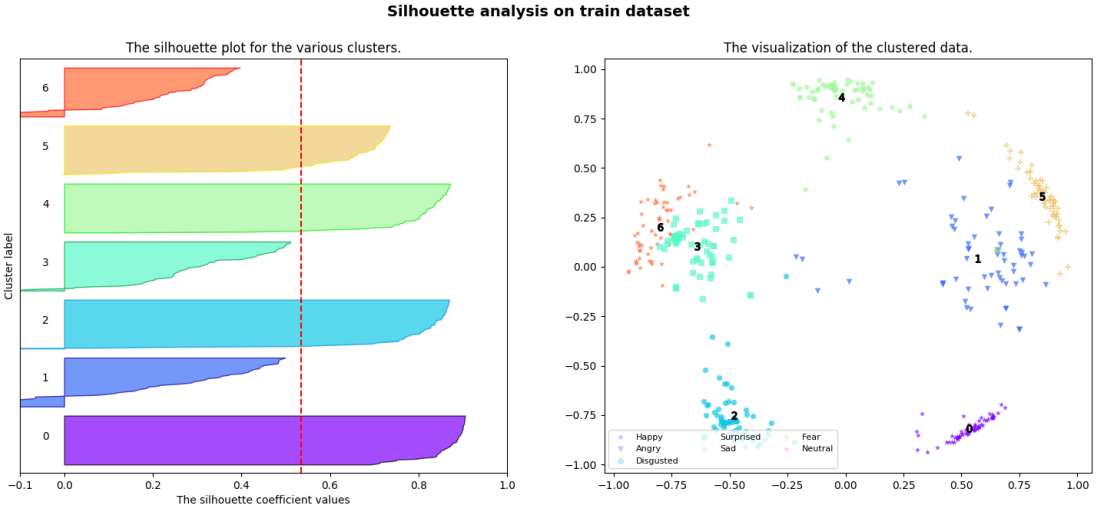


Nimstim-Ravdess

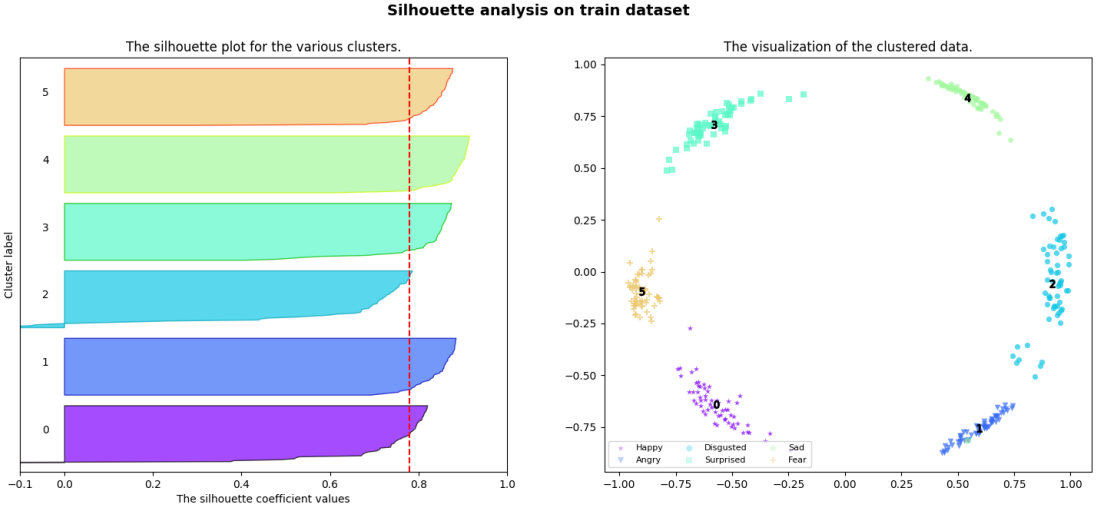


FER

Figure 64: Analyse par silhouette sur les différentes bases d'entrainement avec présentation du centroïde de chaque cluster.



Afew



CK+

Figure 65: Analyse par silhouette sur les différentes bases d'entrainement avec présentation du centroïde de chaque cluster.

5.2.4 Discussion

Nous pouvons, dans un premier temps, noter que les réseaux triples sont en général très performants lors de l'apprentissage sur base de données restreintes. Ils le sont d'autant plus avec l'utilisation de la fonction de perte proposée. Avec un gain de 5 points en précision par rapport au *triplet loss*, lors du placement des triplets. En effet, avec l'utilisation de seulement 200 couples Image-Son par émotion, le modèle atteint 84.4% sur la base de données CK+, 80.5% sur la base de données Nimstim-Ravdess, 70.5% sur FER et 54.4% sur la base *en condition réelle* Afew. Ils peuvent même rivaliser avec les techniques d'apprentissage profond traditionnelles, avec un nombre de données considérablement plus petit. En effet, sur la base de données FER, 3 points de précision sont gagnés en apprenant sur 20 fois moins d'instances.

Nous notons aussi que, pour notre problème, les réseaux triples donnent de bien meilleurs résultats que les réseaux siamois. En effet, les réseaux triples présentent une meilleure représentation vectorielle, comme en témoigne le gain de l'indice de silhouette moyen présenté dans le tableau 18.

		AFEW	FER	Nimstim	CK+
Précision	siamois	38.6%	54.8%	72.2%	74.6%
	triple	54.4%	73.3%	80.5%	83.9%
	gain	+15.8	+18.5	+8.3	+9.3
Silhouette	siamois	0.43	0.58	0.60	0.62
	triple	0.56	0.61	0.80	0.79
	gain	+0.13	+0.03	+0.2	+0.17

Table 18: Comparaison d'indice de silhouette et de précision entre réseau siamois et triple, en fonction de différentes bases.

Cette meilleure représentation fournit nécessairement de meilleurs résultats, les gains en précision sont aussi présentés dans le tableau 18, avec l'utilisation des mêmes données d'entrainement et de test.

Ces gains peuvent être dans un premier temps étonnants, vu que le mécanisme d'apprentissage des deux modèles est très semblable. Une piste de réponse pour ce constat est que le réseau triple apprend sur davantage d'instances différentes. En effet, il y a un nombre considérablement plus grand de triplets que de paires. Comme ce sont des instances que le réseau ne connaît pas, le réseau aura plus de temps, afin de mieux généraliser avant de surapprendre.

Une piste d'amélioration, serait de combiner l'élagage avec les réseaux triples. Un petit réseau apprendrait plus facilement sur des bases de données restreintes et donnerait probablement de meilleurs résultats.

6 Conclusion

6.1 Contributions

Dans cette thèse, nous avons exploré deux domaines donnant des pistes de réponse pour l'apprentissage de réseaux profonds sur des bases de données restreintes. Nous avons, dans un premier temps, proposé une nouvelle méthode d'élagage, se basant sur un réseau rival. Puis nous avons exploré des méthodes d'apprentissage *few shot* avec l'utilisation de réseaux correspondants pour des bases de données multimodales.

Élagage

Dans le chapitre 4.1 nous présentons un réseau rival forçant un meilleur choix des convolutions à supprimer. Grâce à ce réseau, nous enlevons 36.56% des convolutions du réseau Wide-Resnet. Le choix des convolutions à supprimer est meilleur que l'état de l'art, car nous atteignons une précision de 95.84% sur la base de données CIFAR10, soit 0.66 point de plus que la référence. Sur les réseaux Resnet, l'amélioration est moins nette vu que le taux d'élagage est faible ; nous pouvons cependant remarquer que la précision est améliorée grâce aux poids supprimés.

Une autre contribution a été la suppression de l'hyperparamètre indiquant le niveau d'élagage du réseau dans la fonction de perte. Si la précision est favorisée au lieu de l'élagage, ce paramètre n'est plus nécessaire. Si davantage d'élagage est souhaité, le processus est réitérable et 62.82% des poids peuvent être supprimés sans impact négatif sur la précision (95.29% soit un gain de +0.07 point sur la référence).

Si le but est d'élaguer davantage, un terme (forçant l'élagage) peut être ajouté dans la fonction de perte. Dans ce cas 98.43% des convolutions sont supprimées, en obtenant une précision de 94.33% ; soit un impact négatif de -0.85 point sur la référence.

Ces réseaux seront plus petits, et donc plus faciles à apprendre sur des bases de données restreintes.

Détection d'émotions

Dans le chapitre 5, nous présentons l'utilisation de réseaux correspondants comme piste de réponse à l'entrainement de réseaux profonds sur bases de données restreintes.

Nous nous sommes dans un premier temps concentrés sur le traitement des données audios et visuelles, avec notamment l'utilisation des MFCCs, de la transformée de Haar, et de leurs combinaisons.

Un réseau triple multimodal basé sur Inception-Resnet-v2 ainsi qu'une nouvelle fonction de perte ont été présentés. Entrainé sur 200 instances par classe, ce réseau calcule une représentation vectorielle permettant une classification de nos données de test.

Le réseau triple, entrainé avec la fonction de perte proposée, permet d'atteindre une précision de 84.4% sur la base de données CK+, 80.5% sur la base de données Nimstim-Ravdess, 70.5% sur FER et 54.4% sur la base Afew. L'état de l'art sur la base de données FER est atteint, et le modèle rivalise avec des réseaux entrainés sur des centaines de milliers d'instances.

Nous avons par la suite montré que malgré leurs ressemblances, les réseaux triples étaient globalement plus performants que les réseaux siamois dans le domaine de la détection d'émotions, sur les bases de données testées.

Nous avons testé ces deux types de réseaux avec les mêmes bases d'entrainement et de tests. Grâce au réseau triple, nous gagnons :

— 8.3 points de précision sur la base de données NimStim-Ravdess ;
— 18.5 points de précision sur la base de données FER ;
— 15.8 points de précision sur la base de données Afew ;
— 9.3 points de précision sur la base de données CK+.

Ces gains ne sont pas négligeables, et nous pensons que cela est dû au fait que les réseaux triples apprennent sur des triplets d'instances et non des couples. Il y a donc davantage de données nouvelles, et le surapprentissage est retardé.

6.2 Travaux futurs

Avec les résultats encourageants du réseau rival et des réseaux triples, de nouvelles pistes se dégagent pour le futur.

Élagage

À court terme, le réseau rival sera testé sur d'autres architectures classiques et d'autres bases de données, pour une comparaison plus exhaustive avec l'état de l'art.

Des travaux seront effectués afin de fixer le terme de régularisation ; des pistes en fonction du nombre de paramètres du réseau sont pour le moment étudiées afin de le calculer de manière effective.

Le stockage des poids doit également être repensé. Des travaux en quantification peuvent être entrepris afin d'optimiser le stockage en mémoire du réseau, et favoriser leur portabilité.

Deux problèmes persistent sur notre approche : l'itération du processus d'élagage est assez longue et l'hyperparamètre dans la régularisation doit être fixé empiriquement. Des travaux pour répondre à ces deux problèmes doivent être effectués.

À long terme, nous voudrions tester la portabilité de ces réseaux élagués, en les exécutant sur différents smartphones ou robots. Cela pourrait être fait sur la détection d'émotions, où les besoins en robotique commencent à apparaître, en robotique sociale notamment, la détection d'émotions est capitale dans les interactions humain-machine. En effet, une meilleure compréhension de l'état émotionnel de l'humain, mènera à une meilleure réponse du robot.

Détection d'émotions

À court terme, d'autres méthodes de traitement des sons et des images peuvent être testées afin d'en voir l'impact sur l'apprentissage. Des méthodes de clustering peuvent aussi être examinées sur les différentes représentations créées.

Afin de valider la robustesse du modèle, d'autres types de bases de données devront aussi être utilisées, notamment des vidéos.

Nous devons tester notre approche d'élagage sur Inception-Resnet-v2 et analyser comment le nouveau réseau triple réagit. L'idéal serait d'enlever les méthodes de transfert de connaissances, afin d'apprendre le réseau à partir de poids aléatoires.

À long terme, d'autres modalités, comme des données textuelles, pourront être incluses au modèle. Le réseau pourra trouver diverses informations dans différents modes (comme dans le sens des mots), et les combiner afin de trouver l'émotion voulue. Une idée serait d'implémenter un chat-bot prenant en compte les émotions, fournies par le réseau triple,

et modifiant ses réponses en fonction. L'ajoût de données multimodales et l'apprentissage rapide sur base de données restreintes pourra améliorer les architectures existantes.

Bibliographie

[1] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Jun 2016.

[3] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016.

[4] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA* (S. P. Singh and S. Markovitch, eds.), pp. 4278–4284, AAAI Press, 2017.

[5] D. Lee, S. J. Kwon, B. Kim, P. Kapoor, and G. Wei, "Network pruning for low-rank binary indexing," *CoRR*, vol. abs/1905.05686, 2019.

[6] P. Watzlawick, J. Beavin, D. Jackson, and J. Morche, *Une logique de la communication*. Points (Paris), Éditions du Seuil, 1972.

[7] M. A and W. M, "Decoding of inconsistent communications," *Journal of Personality and Social Psychology*, vol. 6, pp. 109–114, 1967.

[8] M. A and F. S, "Inference of attitudes from nonverbal communication in two channels," *Journal of Consulting Psychology*, vol. 31, pp. 248–252, 1967.

[9] D. Sander and K. Scherer, *Oxford Companion to Emotion and the Affective Sciences*. Oxford, England, UK : Oxford University Press, Nov 2009.

[10] P. Ekman, *Emotion in the Human Face*. Cambridge University Press, 1972.

[11] S. Baron-Cohen, *Mind Reading : The Interactive Guide to Emotions 1.3*. Jessica Kingsley Publishers, May 2007.

[12] R. W. Picard, *Affective Computing*. MIT Press, Sep 1997.

[13] F. Ringeval, B. Schuller, M. Valstar, N. Cummins, R. Cowie, L. Tavabi, M. Schmitt, S. Alisamir, S. Amiriparian, E.-M. Messner, S. Song, S. Liu, Z. Zhao, A. Mallol-Ragolta, Z. Ren, M. Soleymani, and M. Pantic, *AVEC 2019 Workshop and Challenge : State-of-Mind, Detecting Depression with AI, and Cross-Cultural Affect Recognition*. New York, NY, USA : Association for Computing Machinery, Oct 2019.

[14] B. W. Schuller, A. Batliner, C. Bergler, E.-M. Messner, A. Hamilton, S. Amiriparian, A. Baird, G. Rizos, M. Schmitt, L. Stappen, H. Baumeister, A. D. MacIntyre, and S. Hantke, "The INTERSPEECH 2020 Computational Paralinguistics Challenge : Elderly emotion, Breathing & Masks," in *Proceedings of Interspeech*, (Shanghai, China), p. 5 pages, September 2020. to appear.

[15] A. Dhall, O. V. R. Murthy, R. Goecke, J. Joshi, and T. Gedeon, "Video and image based emotion recognition challenges in the wild : Emotiw 2015," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, Seattle, WA, USA, November 09 - 13, 2015* (Z. Zhang, P. Cohen, D. Bohus, R. Horaud, and H. Meng, eds.), pp. 423–426, ACM, 2015.

[16] A. Kołakowska, A. Landowska, M. Szwoch, W. Szwoch, and M. R. Wróbel, *Emotion Recognition and Its Applications*, pp. 51–62. Cham : Springer International Publishing, 2014.

[17] B. Lake, R. Salakhutdinov, and J. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, pp. 1332–1338, 12 2015.

[18] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[19] A. Aho and J. Hopcroft and J. Ullman, *Structures de données et algorithmes*. InterEditions, 1987.

[20] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 598–605, Morgan-Kaufmann, 1990.

[21] M. W. Berry, A. H. Mohamed, and B. W. Yap, *Supervised and Unsupervised Learning for Data Science | Michael W. Berry | Springer*. Springer International Publishing, 2020.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale

Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2012.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[24] A. Cornuejols, L. Miclet, and V. Barra, *Apprentissage artificiel - 3e édition : Concepts et algorithmes*. Eds. Eyrolles, 2018.

[25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[26] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.

[27] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges," May 2013. [Online ; accessed 28. Jun. 2020].

[28] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[29] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a Few Examples : A Survey on Few-shot Learning," *ACM Comput. Surv.*, vol. 53, pp. 1–34, Jun 2020.

[30] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, 12 2019.

[31] N. Tottenham, J. Tanaka, A. Leon, T. Mccarry, M. Nurse, T. Hare, D. Marcus, A. Westerlund, B. Casey, and C. Nelson, "The nimstim set of facial expressions : Judgments from untrained research participants," *Psychiatry research*, vol. 168, pp. 242–9, 07 2009.

[32] F. J. Moreno-Barea, F. Strazzera, J. M. Jerez, D. Urda, and L. Franco, "Forward noise adjustment scheme for data augmentation," in *IEEE Symposium Series on Computational Intelligence, SSCI 2018, Bangalore, India, November 18-21, 2018*, pp. 728–734, IEEE, 2018.

[33] F. H. K. dos Santos Tanaka and C. Aranha, "Data augmentation using gans," in *Machine Learning Applied*, 2019.

[34] J. Bromley, J. Bentz, L. Bottou, I. Guyon, Y. Lecun, C. Moore, E. Sackinger, and R. Shah, "Signature verification using a "siamese" time delay neural network,"

International Journal of Pattern Recognition and Artificial Intelligence, vol. 7, p. 25, 08 1993.

[35] G. R. Koch, "Siamese neural networks for one-shot image recognition," 2015.

[36] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "The Omniglot challenge : a 3-year progress report," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 97–104, Oct 2019.

[37] D. J. Rao, S. Mittal, and S. Ritika, "Siamese Neural Networks for One-shot detection of Railway Track Switches," *arXiv*, Dec 2017.

[38] S. Jadon and A. A. Srinivasan, "Improving Siamese Networks for One Shot Learning using Kernel Based Activation functions," *International Conference on Data Management, Analytics and Innovation*, Oct 2019.

[39] ""The Database of Faces" AT&T Laboratoris Cambridge," 2002.

[40] W. Hanyu, G. Jianwei, Y. D. Ming, Q. Weize, and Z. Xiaopeng, *Learning 3D Keypoint Descriptors for Non-rigid Shape Matching : 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII*, pp. 3–20. 09 2018.

[41] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet : A unified embedding for face recognition and clustering," pp. 815–823, 06 2015.

[42] M. Courbariaux and Y. Bengio, "Binarynet : Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[43] F. Li and B. Liu, "Ternary weight networks," *CoRR*, vol. abs/1605.04711, 2016.

[44] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, "A comprehensive survey on model compression and acceleration," *Artificial Intelligence Review*, 02 2020.

[45] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why M Heads are Better than One : Training a Diverse Ensemble of Deep Networks," *CoRR*, Nov 2015.

[46] G. Song and W. Chai, "Collaborative Learning for Deep Neural Networks," *arXiv*, May 2018.

[47] H. Chen and A. Shrivastava, "Group Ensemble : Learning an Ensemble of ConvNets in a single ConvNet," *Computer Vision and Pattern Recognition*, Jul 2020.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[49] S. Yeom, P. Seegerer, S. Lapuschkin, S. Wiedemann, K. Müller, and W. Samek, "Pruning by explaining : A novel criterion for deep neural network pruning," *CoRR*, vol. abs/1912.08881, 2019.

[50] L. Sebastian, B. Alexander, M. Grégoire, K. Frederick, M. Klaus-Robert, and S. Wojciech, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, p. e0130140, 07 2015.

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[52] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. H. So, "Dynamic Sparse Training : Find Efficient Sparse Network From Scratch With Trainable Masked Layers," *International Conference on Learning Representations*, May 2020.

[53] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic Model Pruning with Feedback," *International Conference on Learning Representations*, Jun 2020.

[54] X. Ding, g. ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum sgd for pruning very deep neural networks," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 6382–6394, Curran Associates, Inc., 2019.

[55] Y. Zuo, B. Chen, T. Shi, and M. Sun, "Filter Pruning Without Damaging Networks Capacity," *IEEE Access*, vol. 8, pp. 90924–90930, May 2020.

[56] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration," *Conference on Computer Vision and Pattern Recognition*, Nov 2019.

[57] B. O. Ayinde, T. Inanc, and J. M. Zurada, "Redundant feature pruning for accelerated inference in deep neural networks," *Neural Networks*, vol. 118, pp. 148–158, Oct 2019.

[58] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden* (J. Lang, ed.), pp. 2234–2240, ijcai.org, 2018.

[59] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator : Global filter pruning method for accelerating deep convolutional neural networks," *CoRR*, vol. abs/1909.08174, 2019.

[60] A. Salama, O. Ostapenko, T. Klein, and M. Nabi, "Prune Your Neurons Blindly : Neural Network Compression through Structured Class-blind Pruning," *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2802–2806, Dec 2017.

[61] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 2780–2789, Computer Vision Foundation / IEEE, 2019.

[62] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 12273–12280, AAAI Press, 2020.

[63] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic Soft Filter Pruning for Deep Convolutional Neural Networks," *IEEE Trans. Cybern.*, vol. 50, pp. 3594–3604, Aug 2019.

[64] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Advances in Neural Information Processing Systems 32 : Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 759–770, 2019.

[65] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[66] S. Gao, X. Liu, L. Chien, W. Zhang, and J. M. Alvarez, "VACL : variance-aware cross-layer regularization for pruning deep residual networks," in *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pp. 2980–2988, IEEE, 2019.

[67] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[68] M. Tan and Q. V. Le, "Efficientnet : Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (K. Chaudhuri

and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114, PMLR, 2019.

[69] B. Ko, "A brief review of facial emotion recognition based on visual information," *Sensors*, vol. 18, p. 401, 01 2018.

[70] S. Li and W. Deng, "Deep facial expression recognition : A survey," *IEEE Transactions on Affective Computing*, pp. 1–1, 2020.

[71] M. Soleymani, D. García, B. Jou, B. W. Schuller, S. Chang, and M. Pantic, "A survey of multimodal sentiment analysis," *Image Vision Comput.*, vol. 65, pp. 3–14, 2017.

[72] S. Muhammad, A. Rahman, S. Abdul, K. Adil, and L. Sungyoung, "Depth camera-based facial expression recognition system using multilayer scheme," *IETE Technical Review*, vol. 31, 07 2014.

[73] C. Biplab, B. Manas, and K. Sunil, "Extraction of informative regions of a face for facial expression recognition," *IET Computer Vision*, vol. 10, 02 2016.

[74] K. Miyuki, L. Michael, and G. Jiro, "The japanese female facial expression (jaffe) database," *Availble : http ://www. kasrl. org/jaffe. html*, 01 1997.

[75] S. Biswas and J. Sil, "An efficient expression recognition method using contourlet transform," in *Proceedings of the 2nd International Conference on Perception and Machine Intelligence, Kolkata, West Bengal, India, February 26-27, 2015* (S. K. Pal, M. K. Kundu, S. Chaudhury, S. Mitra, and D. Mazumdar, eds.), pp. 167–174, ACM, 2015.

[76] M. J. Cossetin, J. C. Nievola, and A. L. Koerich, "Facial expression recognition using a pairwise feature selection and classification approach," in *2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24-29, 2016*, pp. 5149–5155, IEEE, 2016.

[77] S. J. Rao, Y. Wang, and G. W. Cottrell, "A deep siamese neural network learns the human-perceived similarity structure of facial expressions without explicit categories," in *Proceedings of the 38th Annual Meeting of the Cognitive Science Society, Recogbizing and Representing Events, CogSci 2016, Philadelphia, PA, USA, August 10-13, 2016* (A. Papafragou, D. Grodner, D. Mirman, and J. C. Trueswell, eds.), cognitivesciencesociety.org, 2016.

[78] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[79] S. E. Kahou, X. Bouthillier, P. Lamblin, C. Gulcehre, V. Michalski, K. Konda, S. Jean, P. Froumenty, A. Courville, P. Vincent, R. Memisevic, C. Pal, and Y. Bengio, "Emonets : Multimodal deep learning approaches for emotion recognition in video," *Journal on Multimodal User Interfaces*, vol. 10, 03 2015.

[80] A. Dhall, R. Goecke, J. Joshi, K. Sikka, and T. Gedeon, "Emotion Recognition In The Wild Challenge 2014 : Baseline, Data and Protocol," in *Proceedings of the 16th International Conference on Multimodal Interaction*, (New York, NY, USA), Association for Computing Machinery, Nov 2014.

[81] G. E. Hinton and S. Osindero, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, p. 2006, 2006.

[82] H. Ng, V. D. Nguyen, V. Vonikakis, and S. Winkler, "Deep learning for emotion recognition on small datasets using transfer learning," in *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, Seattle, WA, USA, November 09 - 13, 2015* (Z. Zhang, P. Cohen, D. Bohus, R. Horaud, and H. Meng, eds.), pp. 443–449, ACM, 2015.

[83] B. Kim, J. Roh, S. Dong, and S. Lee, "Hierarchical committee of deep convolutional neural networks for robust facial expression recognition," *J. Multimodal User Interfaces*, vol. 10, no. 2, pp. 173–189, 2016.

[84] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon, "Static facial expression analysis in tough conditions : Data, evaluation protocol and benchmark," in *IEEE International Conference on Computer Vision Workshops, ICCV 2011 Workshops, Barcelona, Spain, November 6-13, 2011*, pp. 2106–2112, IEEE Computer Society, 2011.

[85] B. Hassani and M. H. Mahoor, "Facial expression recognition using enhanced deep 3d convolutional neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 2278–2288, IEEE Computer Society, 2017.

[86] M. Pantic, M. F. Valstar, R. Rademaker, and L. Maat, "Web-based database for facial expression analysis," in *Proceedings of the 2005 IEEE International Conference on Multimedia and Expo, ICME 2005, July 6-9, 2005, Amsterdam, The Netherlands*, pp. 317–321, IEEE Computer Society, 2005.

[87] M. Valstar, T. Almaev, J. Girard, G. McKeown, M. Mehu, L. Yin, M. Pantic, and J. Cohn, *FERA 2015 - second Facial Expression Recognition and Analysis challenge*, pp. 1–8. 2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), IEEE, 5 2015.

[88] M. S. Hossain and G. Muhammad, "Emotion recognition using deep learning approach from audio-visual emotional big data," *Information Fusion*, vol. 49, 09 2018.

[89] G.-B. Huang, Q.-Y. Zhu, and C. Siew, "Extreme learning machine : Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 12 2006.

[90] R. Vemulapalli and A. Agarwala, "A compact embedding for facial expression similarity," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 5683–5692, Computer Vision Foundation / IEEE, 2019.

[91] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 2261–2269, IEEE Computer Society, 2017.

[92] A. Mollahosseini, B. Hasani, and M. Mahoor, "Affectnet : A database for facial expression, valence, and arousal computing in the wild," *IEEE Transactions on Affective Computing*, vol. PP, 08 2017.

[93] P. Harár, R. Burget, and M. K. Dutta, "Speech emotion recognition with deep learning," in *2017 4th International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 137–140, 2017.

[94] F. Burkhardt, A. Paeschke, M. Rolfes, W. F. Sendlmeier, and B. Weiss, "A database of german emotional speech," in *INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005*, pp. 1517–1520, ISCA, 2005.

[95] R. Zhao, T. Liu, J. Xiao, D. P. K. Lun, and K.-M. Lam, "Deep Multi-task Learning for Facial Expression Recognition and Synthesis Based on Selective Feature Sharing," *arXiv*, Jul 2020.

[96] P. Lucey, J. F. Cohn, T. Kanade, J. M. Saragih, Z. Ambadar, and I. A. Matthews, "The extended cohn-kanade dataset (CK+) : A complete dataset for action unit and emotion-specified expression," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2010, San Francisco, CA, USA, 13-18 June, 2010*, pp. 94–101, IEEE Computer Society, 2010.

[97] H. Kaya, F. Gürpınar, and A. Salah, "Video-based emotion recognition in the wild using deep transfer learning and score fusion," *Image and Vision Computing*, 02 2017.

[98] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015* (X. Xie, M. W. Jones, and G. K. L. Tam, eds.), pp. 41.1–41.12, BMVA Press, 2015.

[99] I. Goodfellow, D. Erhan, P. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, and Y. Bengio, "Challenges in representation learning : A report on three machine learning contests," *Neural Networks*, vol. 64, 07 2013.

[100] T. Ojala, M. Pietikäinen, and D. Harwood, "Performance evaluation of texture measures with classification based on kullback discrimination of distributions," in *12th IAPR International Conference on Pattern Recognition, Conference A : Computer Vision & Image Processing, ICPR 1994, Jerusalem, Israel, 9-13 October, 1994, Volume 1*, pp. 582–585, IEEE, 1994.

[101] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, vol. 2, 06 2005.

[102] D. Lowe, "Distinctive image features from scale invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, 11 2004.

[103] F. Eyben, M. Wöllmer, and B. Schuller, *Opensmile : the munich versatile and fast open-source audio feature extractor*. New York, NY, USA : Association for Computing Machinery, Oct 2010.

[104] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "From facial expression recognition to interpersonal relation prediction," *International Journal of Computer Vision*, 09 2016.

[105] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," 2015.

[106] Y. Niu, D. Zou, Y. Niu, Z. He, and H. Tan, "A breakthrough in speech emotion recognition using deep retinal convolution neural networks," *CoRR*, vol. abs/1707.09917, 2017.

[107] C. Busso, M. Bulut, C. chun Lee, A. Kazemzadeh, E. Mower, S. Kim, J. N. Chang, S. Lee, and S. S. Narayanan, "Iemocap : interactive emotional dyadic motion capture database, language resources and evaluation," *Lang. Resour. Evaluation*, vol. 42, pp. 335–359, 2008.

[108] J. Yun, P. Zheng, E. Yang, A. Lozano, and A. Aravkin, "Trimming the ℓ_1 Regularizer : Statistical Analysis, Optimization, and Applications to Deep Learning," *PMLR*, pp. 7242–7251, May 2019.

[109] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VIII* (V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds.), vol. 11212 of *Lecture Notes in Computer Science*, pp. 191–207, Springer, 2018.

[110] S. Alford, R. A. Robinett, L. Milechin, and J. Kepner, "Training behavior of sparse neural network topologies," in *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019, Waltham, MA, USA, September 24-26, 2019*, pp. 1–6, IEEE, 2019.

[111] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *Acoustics, Speech and Signal Processing, IEEE Transactions On*, pp. 357–366, 1980.

[112] J. S. Walker, "A primer on wavelets and their scientific applications," *Journal of the American Statistical Association*, vol. 95, 09 2000.

[113] S. Stevens, J. Volkmann, and E. Newman, "A scale for the measurement of the psychological magnitude pitch," *Journal of the Acoustical Society of America*, vol. 8, pp. 185–190, 01 1937.

[114] Y. mohd ali, P. M P, S. Yaacob, R. Yusuf, and S. Abu bakar, "Analysis of accent-sensitive words in multi-resolution mel-frequency cepstral coefficients for classification of accents in malaysian english," *International Journal of Automotive and Mechanical Engineering*, vol. 7, pp. 1053–1073, 06 2013.

[115] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *Similarity-Based Pattern Recognition - Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015, Proceedings* (A. Feragen, M. Pelillo, and M. Loog, eds.), vol. 9370 of *Lecture Notes in Computer Science*, pp. 84–92, Springer, 2015.

[116] Q. Qian, L. Shang, B. Sun, J. Hu, T. Tacoma, H. Li, and R. Jin, "Softtriple loss : Deep metric learning without triplet sampling," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6449–6457, 2019.

[117] P. Rousseeuw, "Rousseeuw, p.j. : Silhouettes : A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 11 1987.

[118] S. R. Livingstone and F. A. Russo, "The ryerson audio-visual database of emotional speech and song (ravdess) : A dynamic, multimodal set of facial and vocal expressions in north american english," in *PloS one*, 2015.

[119] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *CoRR*, vol. abs/1411.7923, 2014.

[120] S. Minaee and A. Abdolrashidi, "Deep-emotion : Facial expression recognition using attentional convolutional network," *CoRR*, vol. abs/1902.01019, 2019.

[121] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

[122] A. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks : A systematic review," *IEEE Access*, vol. PP, pp. 1–1, 02 2019.

[123] P. Bhatt and I. Patel, "Optical character recognition using deep learning – a technical review," *National Journal of System and Information Technology (NJSIT)*, vol. 11, 06 2018.

[124] L. Gonog and Y. Zhou, "A review : Generative adversarial networks," in *2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 505–510, 2019.

[125] S. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain, "Machine translation using deep learning : An overview," in *International Conference on Computer, Communications and Electronics*, pp. 162–167, 07 2017.

[126] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, Dec 1943.

[127] "F. Rosenblatt, "The Perceptron A Perceiving and Recognizing Automaton," Report 85-60-1, Cornell Aeronautical Laboratory, Buffalo, New York, 1957. - References - Scientific Research Publishing," May 2020.

[128] K. You, M. Long, J. Wang, and M. I. Jordan, "How Does Learning Rate Decay Help Modern Neural Networks ?," *arXiv*, Aug 2019.

[129] D. A. Drachman and D. A. Drachman, "Do We Have Brain to Spare ?," *Neurology*, vol. 64, pp. 2004–2005, Jun 2005.

[130] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signal. Systems*, vol. 2, pp. 303–314, Dec 1989.

[131] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct 1986.

[132] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[133] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research (JMLR)*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[134] M. D. Zeiler, "Adadelta : An adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[135] D. P. Kingma and J. Ba, "Adam : A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[136] Y. Le Cun, L. Bottou, and Y. Bengio, "Reading checks with multilayer graph transformer networks," in *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, vol. 1, pp. 151–154, IEEE, 1997.

[137] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 807–814, 2020.

[138] S. Ioffe and C. Szegedy, "Batch normalization : accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, pp. 448–456, 2020.

[139] "Hinton, G.E. and Salakhutdinov, R.R. (2006) Reducing the Dimensionality of Data with Neural Net-works. Science, 313, 504-507. - References - Scientific Research Publishing," Jun 2020. [Online ; accessed 28. Jun. 2020].

[140] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[141] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[142] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal ? a large-scale study," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 700–709, Curran Associates, Inc., 2018.

[143] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," vol. 70 of *Proceedings of Machine Learning Research*, pp. 214–223, PMLR, 06–11 Aug 2017.

[144] "Hotelling, H. (1933) Analysis of a Complex of Statistical Variables into Principal Components. Journal of Educational Psychology, 24, 417-441, 498-520. - References - Scientific Research Publishing," Jun 2020. [Online ; accessed 24. Jun. 2020].

[145] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[146] L. McInnes, J. Healy, N. Saul, and L. Großberger, "UMAP : Uniform Manifold Approximation and Projection," *Journal of Open Source Software*, vol. 3, p. 861, Sep 2018.

7 Annexes

.1 Apprentissage Profond

L'apprentissage profond fait partie du domaine de l'intelligence artificielle. Il est, de nos jours, au coeur de la recherche dans ce domaine. En effet, il a déjà offert de nombreuses solutions sur divers problèmes ; que ce soit en vision par ordinateur [121, 1], en reconnaissance vocale [122], en reconnaissance de caractères [123], en génération d'images [124], ou encore en traduction automatique [125]. Cette partie explore les bases de l'apprentissage profond.

.1.1 Réseau de neurones artificiels

Historiquement, les premiers travaux sur les neurones artificiels (ANN) ont été effectués en 1943 quand McCulloch et Pitts [126] ont modélisé un neurone formel avec des circuits électriques. Il faut attendre 1957 pour que Rosenblatt propose le premier neurone artificiel, qu'il nomme perceptron [127], pouvant faire varier ses propres poids.

.1.1.1 Neurone artificiel ou Perceptron

Neurone artificiel

Le neurone est la structure de base des réseaux de neurones, son rôle est de déterminer l'importance d'une combinaison de signaux d'entrée (figure 66).

Le neurone calcule tout d'abord une somme pondérée des entrées, appelée potentiel post-synaptique :

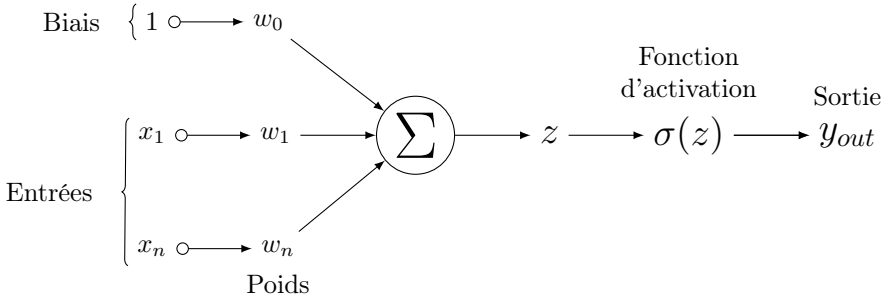


Figure 66: Schéma d'un neurone artificiel

$$z = \sum_{i=1}^{n} (w_i x_i) + b = w^\top x + b \tag{2}$$

Avec :

— $x^\top = (x_1, ..., x_n)$ la transposée du vecteur entrée de dimension n
— $w^\top = (w_1, ..., w_n)$ la transposée du vecteur de poids associé aux entrées x
— $b = w_0$ Le poids w_0 est souvent appelé b pour biais.

Ce potentiel est ensuite soumis à une fonction d'activation. Le terme de *fonction d'activation* vient de la biologie. Sur un neurone, le *potentiel d'activation* est le seuil de stimulation d'un neurone, il entraine une réponse. Les fonctions d'activation permettent aussi d'introduire de la complexité en amenant par exemple des propriétés non linéaires au réseau de neurones. En appliquant une transformation non linéaire aux valeurs d'entrée, elles permettent d'aborder une classe de problèmes plus large.

Dans un perceptron, les fonctions d'activation les plus simples sont les fonctions de seuillage, par exemple la fonction de Heaviside.

$$\forall x \in \mathbb{R},\ H(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0. \end{cases}$$

Pour cette fonction, le seuil est 0. Si l'activation est forte ($x > 0$) alors le neurone est activé.

On note

$$a(z) = \sigma(z)$$

avec σ la fonction d'activation.

La figure 71 présente une liste non exhaustive des fonctions d'activation couramment utilisées. Elles sont décrites dans la section .2.0.2.

Une des plus grandes contributions de Rosenblatt est sans doute l'algorithme permettant l'apprentissage des poids. Il est décrit dans l'algorithme 3 pour une epoch, c'est à dire le parcours de l'ensemble de la base d'entrainement durant la phase d'apprentissage.

algorithme 3 Algorithme d'apprentissage de Rosenblatt pour une epoch.

Entrée : une base d'entrainement $\mathbb{X} = (\mathbf{x^1}, ..., \mathbf{x^m})$, son vecteur de labels associés $\mathbf{y} = (y_1, ..., y_m)$, un vecteur de poids $\mathbf{W} = (w_0, ..., w_n)$, un taux d'apprentissage α
Pour Tout vecteurs d'entrainement $\mathbf{x^i}$ **Faire :**
  Récupérer la solution donnée par le perceptron : $\hat{y_i}$
  Pour Tous les poids w_j dans **W Faire :**
    $w_j = w_j - \alpha(\hat{y_i} - y_i)\mathbf{x^i}$

Le taux d'apprentissage permet de régler la vitesse d'apprentissage : plus il est grand, plus vite le poids va être modifié. Le régler correctement est une partie importante de l'apprentissage :

— si α est trop grand, la solution optimale (si elle existe) risque de ne pas être trouvée.
— si α est trop petit, des phénomènes d'oscillation pourront apparaître autour de la solution.

Habituellement, au début de l'apprentissage α est fixé avec une valeur initiale assez grande, afin d'accélérer le début de l'apprentissage, puis elle décroît au cours de l'apprentissage afin de gagner en finesse. You et al. [128] proposent des pistes d'explications du gain en performance sur les réseaux modernes.

Régression et Classification

La classification et la régression sont deux problèmes de prédiction majeurs traités au sein des réseaux de neurones.

— La classification permet de séparer des données en plusieurs classes. Pour une donnée test, l'appartenance à une catégorie est identifiée.
— La régression permet de distinguer des données en valeurs continues. En régression, on cherche à décrire une (ou plusieurs) variables quantitatives comme fonction de plusieurs autres variables. La régression sert à la prédiction de données et non à la

prédiction d'étiquettes.

Dans la suite de la thèse, le problème traité est la classification d'émotions.

Le perceptron peut par exemple être utilisé pour une classification binaire afin de décider si une entrée appartient à une certaine classe. Cette classification revient à trouver un hyperplan séparant les deux classes. La limite du perceptron linéaire est qu'il peut uniquement classifier des données linéairement séparables (voir figure 67).

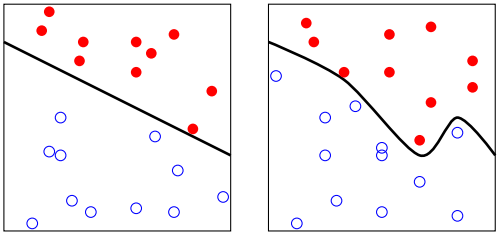


Figure 67: a) Classification linéairement séparable. b) Classification non linéairement séparable.

D'une manière générale, la classification consiste à trouver une fonction, dite fonction de décision, séparant les différentes classes.

Dès que le problème devient non linéairement séparable, il est nécessaire de recourir à des modèles plus complexes. Une idée est de combiner plusieurs perceptrons, avec des fonctions d'activation non linéaires, pour construire des modèles plus performants sur ces problèmes.

.1.2 Réseau de neurones artificiels ou perceptron multicouche

Les réseaux de neurones, ou perceptrons multicouches (PMC) se basent sur les connexions structurelles des neurones et synapses dans le cerveau. Ce dernier est composé d'un grand nombre de neurones massivement connectés les uns aux autres. Ce nombre est estimé à 10^{11} neurones, chacun connecté en moyenne à 10^4 de ses pairs [129]. Le niveau d'activité d'un neurone va soit exciter soit inhiber ses voisins, créant ainsi un réseau complexe d'interactions. Les réseaux de neurones artificiels sont inspirés de ce mécanisme. La figure 68 présente un exemple d'un tel réseau.

Dans les réseaux de neurones complètement connectés (*Full Connected*), le modèle simple de neurone (vu en annexe .1.1.1) est utilisé. Une structure composée de plusieurs

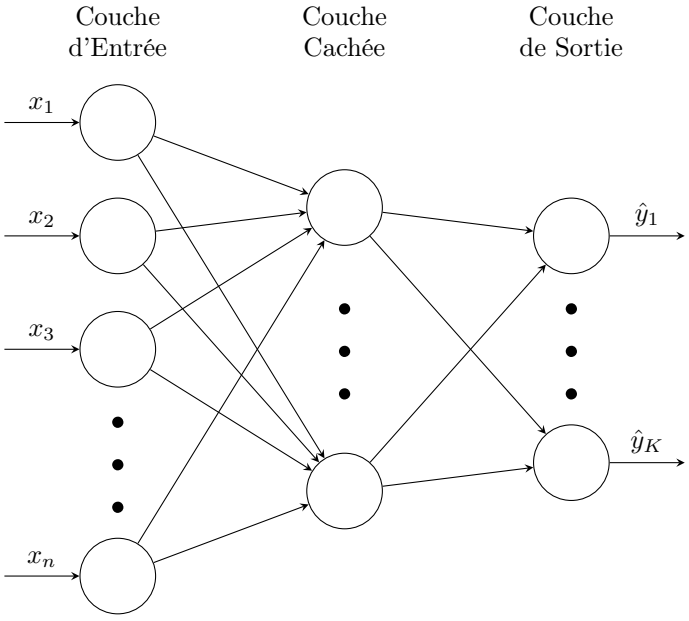


Figure 68: Réseau de neurones artificiels ; les biais (sur chaque neurone) ne sont pas représentés pour cause de visibilité.

neurones en structure de couches est alors créée. Chacune de ces couches va prendre en entrée la sortie des couches précédentes.

En d'autres termes, chaque neurone des couches cachées (couches intermédiaires) va effectuer une transformation affine des entrées. Notons z_i^l est le potentiel post synaptique du ième neurone de la couche l, c'est à dire la valeur calculée par un neurone avant de la passer dans la fonction d'activation.

$$z_i^l = \sum_{j=1}^{n} w_{ij}^l a_j^{l-1} + b_i^l = (\mathbf{W_i^l})^\top \mathbf{a^{l-1}} + \mathbf{b^l} \tag{3}$$

Avec :

— a_i^{l-1} la valeur d'activation du ième neurone de la couche $l-1$: $a_i^{l-1} = \sigma(z_i^{l-1})$
— w_{ij}^l le poids qui relie le ième neurone de la couche l au jème neurone de la couche (l-1)
— b_i^l le biais associé au ième neurone de la couche l.
— n le nombre de neurones de la couche l-1.

Le nombre de couches cachées dépend de la complexité du problème donné. En général, une couche cachée suffit.

Le théorème de Cybenko [130] affirme qu'un réseau de neurones avec une simple couche cachée, contenant un nombre fini de neurones, peut approximer toutes les fonctions continues de $\mathbb{R}^n$ (sous seulement quelques contraintes concernant la fonction d'activation).

Théorème .1. *Cybenko*
Soit σ une fonction sigmoïde non constante, bornée et continue. Soit I_m le cube unité de dimension m : $[0,1]^m$. L'espace des fonctions continues réelles est noté $C(I_m)$ Quel que soit $\varepsilon > 0$, et quelle que soit la fonction $f \in C(I_m)$, il existe un entier n (nombre de neurones), des réels v_i, b_i constants et un vecteur de poids $\boldsymbol{w}$ réel, qui définissent :

$$F(x) = \sum_{i=1}^{n} v_i \sigma \left(w_i x + b_i \right)$$

tel que l'on peut approximer la fonction f comme ceci :

$$|F(x) - f(x)| < \varepsilon$$

Pour tout $x \in C(I_m)$. En d'autres termes, les fonctions de la forme $F(x)$ sont denses dans $C(I_m)$.

Une démonstration est proposée dans [130].

Cybenko affirme aussi que le nombre de neurones croît exponentiellement en fonction de la complexité du problème. Au lieu d'utiliser une couche contenant un grand nombre de neurones, plusieurs couches cachées (contenant un nombre de neurones plus faible) sont souvent utilisées.

Pour la classification, la couche finale représente les valeurs attribuées à chaque classe. Sur la figure 68, il y a K valeurs pour K classes :

$$\hat{y}_1, ..., \hat{y}_K.$$

Habituellement, un softmax est utilisé pour obtenir une distribution de probabilité :

$$\sigma_i = \frac{e^{\hat{y}_i}}{\sum_{k=1}^{K} e^{\hat{y}_k}}$$

σ_i est la probabilité que l'entrée soit de classe i et donc $\sum_i(\sigma_i) = 1$.

Le modèle apprend des paramètres (poids et biais) en minimisant une fonction de coût basée sur la mesure d'une fonction de perte E entre la sortie théorique y_i et la sortie calculée $\hat{y}_i$ pour chaque exemple $E(y_i, \hat{y}_i)$. Le modèle minimise donc $\sum_i E(y_i, \hat{y}_i)$, auquel est éventuellement ajouté un terme de régularisation. La fonction de perte quadratique

$$E(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

est souvent utilisée en régression. En classification, l'entropie croisée est un choix possible :

$$E(y_i, \hat{y}_i) = -y_i log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i).$$

Pour entraîner les réseaux à une ou plusieurs couches cachées, Rumelhart, Hinton et Williams [131] ont proposé un algorithme permettant de mettre à jours les poids et biais de sorte à minimiser l'erreur du réseau.

.1.2 Algorithme de rétropropagation

Ce n'est qu'en 1986 qu'un algorithme de rétropropagation à été présenté par Rumelhart, Hinton et Wiliams [131], permettant un apprentissage des réseaux de neurones multicouches. Utilisant la descente de gradient, il relance un grand intérêt pour les réseaux de neurones.

Comme pour l'algorithme de Rosenblatt, l'idée principale est de modifier les poids du réseau relativement à l'erreur ($E = (\hat{y} - y)^2$ dans le perceptron).

La dérivée partielle de la fonction objectif est alors calculée en fonction de chaque poids. Nous supposons que l'erreur commise sur les données d'entrée est commise proportionnellement sur chaque poids. L'erreur est ensuite propagée en arrière, des dernières couches aux premières et chaque poids est mis à jour relativement à l'erreur totale. Plus l'erreur est grande, plus les poids sont modifiés. La dérivée donne la direction vers laquelle modifier le poids. C'est la descente de gradient :

$$w := w - \alpha \nabla E(w) = w - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla E_i(w)$$

avec :

— w un poids du réseau

— α reste le taux d'apprentissage.

— $\nabla E(w)$ le gradient de w .

Chacune des fonctions E_i est généralement associée avec la ième observation de l'ensemble des données (utilisées pour l'apprentissage). Il y a ici n données. Lorsqu'elle est utilisée pour minimiser cette fonction, la méthode standard de descente de gradient correspond à cette itération. Traditionnellement, la descente de gradient optimise la fonction de coût sur toute la base d'entrainement en même temps.

Cependant, cette approche est coûteuse en temps et en mémoire quand la base de données est grande. Une approche plus commune est d'utiliser la *descente de gradient stochastique* (SGD) [132], en effectuant cette descente sur une donnée d'entrainement, ou utiliser l'apprentissage par mini-batch, qui exécute cette descente sur une petite partie de la base de données (un batch).

Dans la descente de gradient stochastique, la descente de gradient est effectuée sur une seule composante de la somme (un seul exemple). Alors que l'algorithme parcourt l'ensemble d'apprentissage, il réalise l'étape de mise-à-jour pour chaque exemple :

$$w := w - \alpha \nabla E_i(w).$$

La modification des poids est faite grâce à la règle de la chaîne, par exemple, la modification du poids w_{11} s'effectue à l'aide de :

$$\frac{\partial E}{\partial w_{11}^l} = \frac{\partial E}{\partial a_1^l} \frac{\partial a_1^l}{\partial z_1^l} \frac{\partial z_1^l}{\partial w_{11}^l}.$$

L'algorithme est présenté dans la figure 4.

Puisque la descente s'effectue selon le gradient d'un seul exemple, selon les différents exemples, la variance de l'erreur va être grande. Cela va avoir un impact dans l'optimisation, et le modèle peut, si l'erreur est trop grande, ne pas converger vers la solution optimale. Le pas d'apprentissage est encore plus important quand on utilise SGD.

Pour améliorer cette optimisation, un *moment* est parfois utilisé. Une partie du terme précédent est rajoutée pour contraindre le gradient et éviter qu'il parte dans des directions non optimales. La méthode SGD avec moment conserve en mémoire la mise-à-jour à chaque étape (∇w)

algorithme 4 Algorithme de rétro propagation sur un exemple.

Entrée : une donnée d'entrainement **x** associée à son label y

Propagation Avant :

On propage le signal en avant dans les couches du réseau de neurones :

$x_k^{n-1} \rightarrow x_j^n$ ▷ n le numéro de la couche.

$x_j^{(n)} = \sigma^{(n)}(h_j^{(n)}) = \sigma^{(n)}(\sum_k w_{jk}^{(n)} x_k^{(n-1)})$ ▷ h la fonction d'agrégation entre les poids et les entrées du neurone (souvent le produit scalaire).

Calcul de l'erreur entre la sortie $\hat{y}$ et le vecteur vérité y

Pour Chaque neurone i dans la couche de sortie

$e_i^{sortie} = \sigma'(h_i^{sortie})(y - \hat{y})$ ▷ Calcule l'erreur de sortie

Fin

Propagation Arrière :

Pour toutes les couches n de m à 0 ▷ m nombre de couches

$e_i^{(n)} \mapsto e_j^{(n-1)}$ grâce à la formule suivante :

$e_j^{(n-1)} = \sigma'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij}^{(n)} e_i^{(n)}$

$e_i^{(n)} = e_i^{sortie} = (y - \hat{y}) \frac{\partial y}{\partial h_i^{(n)}}$

On met à jour tous les poids dans la couche m :

$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha e_i^{(l)} x_j^{(l-1)}$ ▷ α représente le taux d'apprentissage (compris entre 0 et 1)

Fin

$w := w - \alpha \nabla E_i(w) + \eta \nabla w.$

Avec $\eta \in [0, 1]$ déterminant la contribution de cette mémoire.

D'autres techniques sont aussi utilisées pour améliorer la convergence, comme l'utilisation d'un pas d'apprentissage basé sur le gradient calculé au préalable (*Adagrad* [133] ou *Adadelta* [134]) ou encore un terme de moment basé sur ces gradients (*Adam* [135]).

.2 Réseaux de neurones convolutifs

Les réseaux de convolution profonds sont au coeur de l'apprentissage profond. Bien qu'ils soient connus depuis plus d'une vingtaine d'années (Le Cun et al.[136]), surtout dans le domaine de la reconnaissance de caractères, leur popularité arrive en 2012, quand un réseau de convolution profond (Krizhevsky et al [78]) est vainqueur du challenge ILSVRC de classification ImageNet [22].

La quantité de données est en partie liée à cette attente. En effet, depuis 2012 l'accès à de grandes bases de données, indispensables pour le bon apprentissage des réseaux de neurones profonds, est devenu plus simple.

Un réseau de neurones convolutif CNN est un type de réseau de neurones artificiels basé sur une représentation des mécanismes de la vision humaine. En effet, dans le cerveau, les neurones permettant une bonne vision sont organisés hiérarchiquement, en couches. Les premières couches vont s'intéresser principalement à la détection de contours puis une généralisation va être faite ; détection de formes dans un premier temps, puis d'informations de plus en plus abstraites.

La figure 69 montre une architecture classique d'un réseau convolutif. Ces réseaux sont généralement composés d'une succession de couches de convolution et de pooling. Les parties suivantes expliquent ces différentes couches.

.2.0.1 Couches de Convolutions

La première couche d'un CNN est souvent une couche de convolution. La partie convolutive du réseau fonctionne comme un extracteur de caractéristiques des données. Une donnée est passée à travers une succession de noyaux de convolution, créant un vecteur appelé carte de convolutions, ou vecteur caractéristique. Les valeurs des noyaux sont apprises et changent au fur et à mesure de l'apprentissage. Ils vont servir à détecter des caractéristiques importantes pour la résolution du problème. Pour la classification

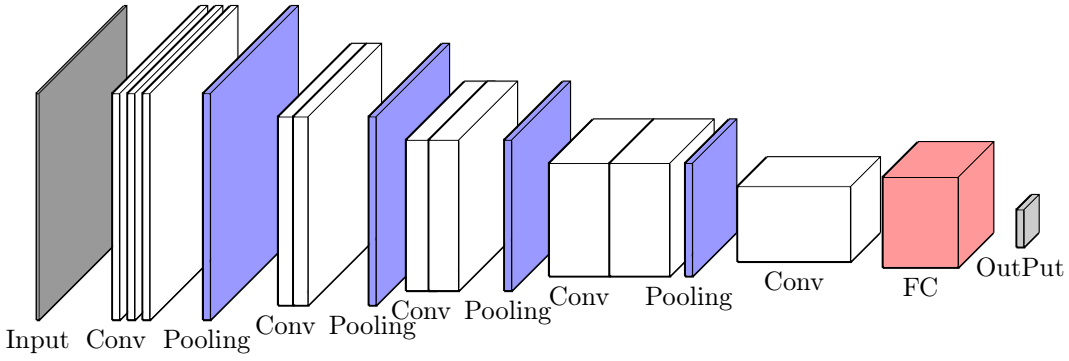


Figure 69: Architecture classique des réseaux de convolutions

d'une image par exemple, les caractéristiques importantes vont être, à bas niveau, des contours, des coins. Puis plus le nombre de couches augmente, plus le réseau va détecter des caractéristiques de plus haut niveau. Quelques éléments sont quand même à spécifier, notamment le nombre de noyaux par couche et leurs tailles.

L'expression générale d'une convolution est définie par :

$$g(x,y) = \omega f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

avec :

— $g(x,y)$ l'image filtrée

— $f(x,y)$ l'image originale

— w le noyau de convolution.

Un exemple de convolution discrète est présenté dans la figure 70

Le nombre de convolutions par couche est un hyperparamètre. Il correspond au nombre de filtres à utiliser sur les entrées.

Deux autres hyperparamètres doivent aussi être fixés :

— le pas (*stride*), s'il est de 1, la convolution se décale d'un pixel à chaque fois.

— le *padding* consiste à l'ajout de lignes ou de colonnes sur les bords de l'entrée ($f(x,y)$ sur la figure 70), afin de pouvoir calculer l'opération de convolution sur tous les pixels. Généralement un zéro-padding est effectué, signifiant un ajout de zéros sur le bord de la matrice.

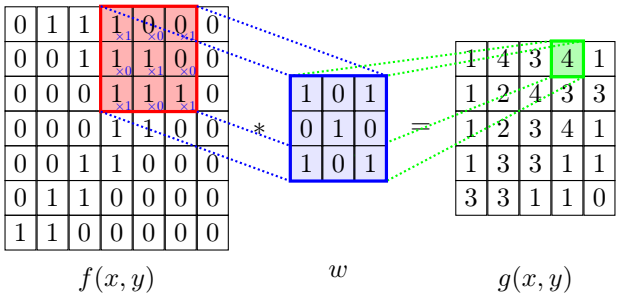


Figure 70: Exemple de convolution discrète

Partage des poids

Un problème est ici, le nombre de paramètres. Par exemple le réseau AlexNet [78] prend des images de tailles $[227 \times 227 \times 3]$. La première couche de convolution est de taille $[55 \times 55 \times 96]$ chacun des $55 \times 55 \times 96$ neurones est connecté a une partie de l'image de taille $[11 \times 11 \times 3]$.

Pour la première couche de convolutions il y a déjà $55 \times 55 \times 96 = 290,400$ neurones. Chaque neurone a $11 \times 11 \times 3 = 363$ poids et un biais. Il y a donc sur la première couche $290400 \times 364 = 105,705,600$ paramètres

Afin de contrôler le nombre de paramètres nous faisons l'hypothèse que si une convolution est utile à un endroit de l'image (x_1, y_1), il sera surement utile de la calculer sur une position différente (x_2, y_2)

Nous allons donc contraindre les neurones d'une même profondeur à avoir les mêmes poids et biais. Sur l'exemple, les 55×55 neurones vont utiliser les mêmes paramètres. Grâce à ce partage, nous arrivons donc à $96 \times 11 \times 11 \times 3 = 34,848$ ou $34,944$ paramètres ($+96$ biais).

Cette hypothèse est forte mais nécessaire pour le processus d'apprentissage.

.2.0.2 Couches de correction

On applique comme dans le perceptron (annexe .1.1.1), une fonction d'activation sur chaque sortie des couches de convolution. On appelle souvent cette opération, couche de correction. Les fonctions d'activation couramment utilisées sont présentées dans la figure 71, et décrites par la suite.

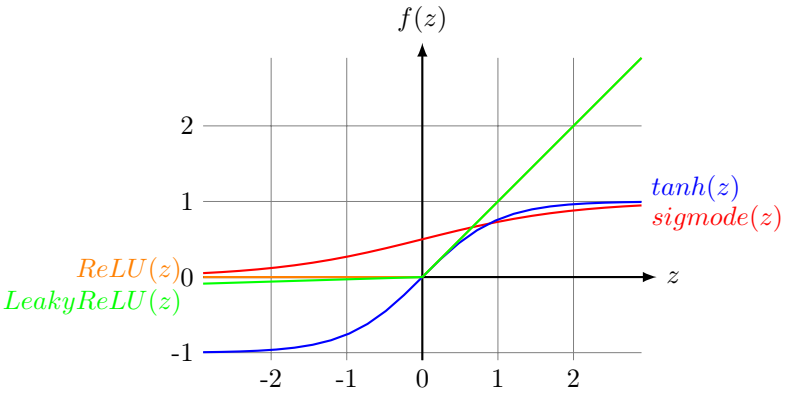


Figure 71: Fonctions d'activation couramment utilisées : sigmoïde, tanh, ReLU et Leaky ReLU. ReLU et Leaky ReLU se superposent pour $z \geq 0$.

Sigmoïde

La fonction sigmoïde est une fonction différentiable, strictement croissante.

Elle est définie par :

$$\begin{array}{rcccc} \sigma & : & R & \rightarrow & [0,1] \\ & & \mathbf{x} & \mapsto & \frac{1}{1+\exp(-\mathbf{x})} \end{array}$$

On la généralise souvent à l'aide d'un paramètre λ comme :

$$\begin{array}{rcccc} \sigma_\lambda & : & R & \rightarrow & [0,1] \\ & & \mathbf{x} & \mapsto & \frac{1}{1+\exp(-\lambda\mathbf{x})} \end{array}$$

Cette fonction a historiquement été la plus utilisée car elle est la version différentiable de la fonction Heaviside utilisée dans le perceptron. Quand le potentiel d'activation est bas, le neurone ne s'active pas (0), s'il est haut le neurone s'active (1). Une autre propriété intéressante est que la dérivée se calcule très vite.

Elle est cependant de moins en moins utilisée dans les réseaux profonds, car elle présente deux problèmes majeurs.

— *Disparition du gradient : (vanishing gradient)* Durant la phase de rétropropagation (annexe .1.2), le gradient de cette fonction va être calculé. Or ce gradient va souvent être très petit et dans les réseaux profonds, les gradients des couches inférieures (les plus proches de l'entrée) peuvent devenir extrêmement petits. Lorsque le gradient se rapproche de 0, les couches sont entrainées très lentement, voire pas du tout. Ce qui peut avoir pour effet de stopper la rétropropagation.

— *Non centré en 0 :* En effet, cette fonction est toujours positive. Il est admis que les fonctions centrées en zéro favorisent un bon apprentissage.

Tangente hyperbolique (tanh)

La tangente hyperbolique (*tanh*) est une fonction différentiable strictement croissante. Comme la sigmoïde, la tangente hyperbolique est sujette à la disparition du gradient.

Rectified Linear Unit (ReLU)

Depuis sa première utilisation par Nair et Hinton en 2010 [137], la ReLU est devenue la fonction d'activation par défaut dans beaucoup de cas. Elle est définie par

$$relu(\mathbf{x}) = max(0, \mathbf{x}). \tag{4}$$

Bien que très simple, elle possède des propriétés très intéressantes :

— *Pas de borne positive :* La ReLU vaut 0 quand les entrées sont négatives mais n'est pas bornée pour les entrées positives. Cela va permettre une bonne propagation quand les entrées sont positives, ce qui améliore les performances de l'apprentissage.
— *Non coûteuse :* Cette fonction est facile à calculer, son implémentation est juste un simple seuil à 0. Son gradient est aussi trivial et rapide à calculer.
— *Introduction des zéros :* Cette fonction introduit beaucoup de zéros dans les réseaux de neurones. Bien que cela améliore la vitesse d'apprentissage, les poids nuls ne peuvent plus être récupérés dans la suite de l'apprentissage. Pour pallier ce problème, la leaky ReLU est parfois utilisée.

Leaky Rectified Linear Unit (Leaky ReLU)

Les Leaky ReLUs ont été proposées pour permettre au gradient de se propager, même quand les données sont négatives. Cela permet de réactiver des poids potentiellement perdus (et utiles) que le ReLu aurait laissé à 0.

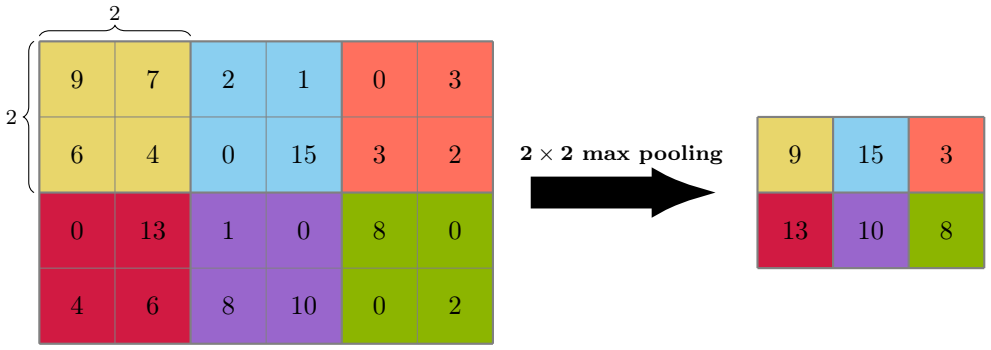
Elle est définie comme :

$$leaky\_relu(\mathbf{x}) = max(\beta\mathbf{x}, \mathbf{x}), \tag{5}$$

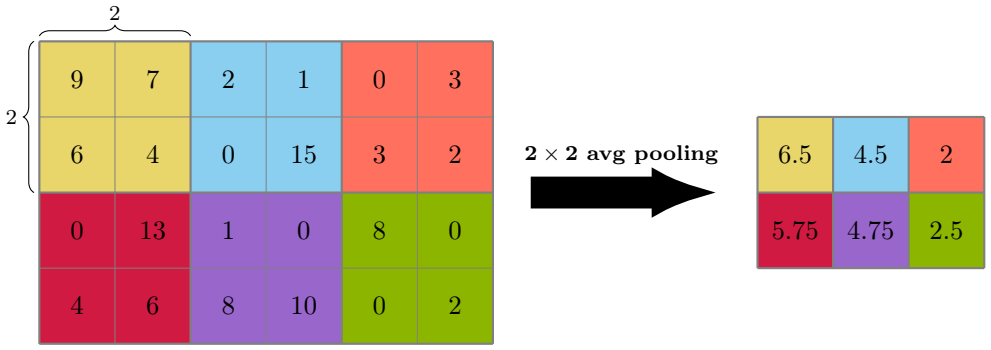
β est une constante positive.

.2.0.3 Pooling

Après l'obtention des caractéristiques importantes grâce aux convolutions, il faut les classifier, et donc utiliser un réseau de neurones (annexe .1.1.2). Cependant il serait trop coûteux de l'appliquer directement après des couches de convolution. Pour résoudre ce problème, des couches de pooling sont utilisées. Elles regroupent des parties, des régions, dans différentes localisations des caractéristiques. Il existe plusieurs types de pooling, les deux types de pooling les plus utilisés, max pooling et avg pooling (utilisé durant cette thèse) sont présentés dans la figure 72.



a) Pooling par maximum.



b) Pooling par moyenne.

Figure 72: Exemple de différentes techniques de pooling

L'utilisation du pooling a de nombreux avantages :

— Le pooling réduit la taille spatiale des vecteurs caractéristiques intermédiaires, réduisant ainsi la quantité de paramètres à apprendre du réseau.

— En apprentissage, on parle de surapprentissage lorsque le modèle apprend trop

précisément les caractéristiques des exemples fournis, et perd les informations de généralisation. Le pooling permet d'être moins atteint par le surapprentissage.
— Grâce au regroupement des différentes informations, l'utilisation du pooling permet aussi d'être localement moins sensible au bruit.
— Il crée aussi, localement, une invariance par translation.

.2.0.4 Couche complètement connectée (FC)

Une couche complètement connectée (FC) est un perceptron multicouche tel que décrit en annexe .1.1.2. Son but est d'apprendre les combinaisons non linéaires des valeurs données par le vecteur de caractéristiques. Ces couches sont fréquemment retrouvées à la fin du réseau, afin de répondre au problème donné, et par exemple de détecter l'appartenance à une classe.

.2.0.5 Dropout

Le dropout est une technique utilisée afin d'éviter le surapprentissage. Un schéma explicatif est donné Figure 73.

Le *Dropout* consiste à ignorer un certain nombre de neurones durant la phase d'apprentissage. A chaque batch d'entrainement, les neurones sont choisis aléatoirement et ne sont pas considérés pendant la propagation avant ou arrière. En pratique, à chaque phase d'apprentissage, chaque neurone a une probabilité p d'être choisi et une probabilité $1 - p$ d'être ignoré.

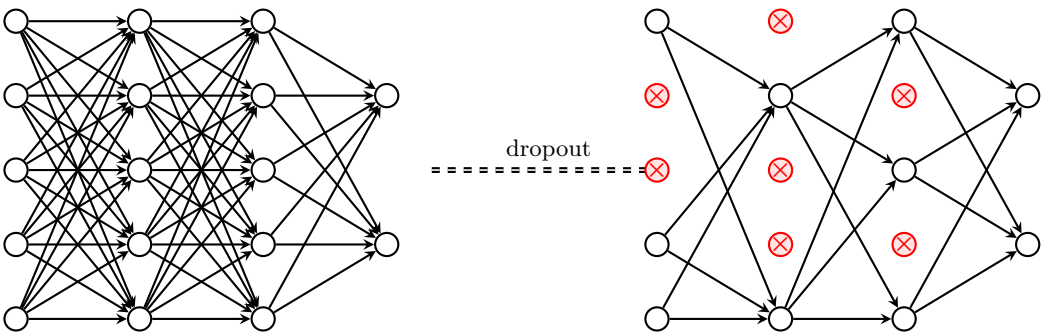


Figure 73: Schéma descriptif du dropout, les croix rouges représentent les neurones ignorés durant la phase d'entraînement.

.2.0.6 Batch Normalization

La *batch normalization* est une technique proposée par Ioffe et Szegedy [138] accélérant la convergence des modèles profonds.

L'idée est de normaliser les entrées de chaque couche. Les auteurs de [138] montrent que l'application de cette normalisation accélère la phase d'apprentissage du réseau.

Elle est appliquée sur un batch $\mathbb{B} = (\mathbf{x^i}, ..., \mathbf{x^m})$, comme ceci :

- La moyenne du batch est calculée, $\mu_{\mathbb{B}}$
- La variance du batch est calculée, $\sigma_{\mathbb{B}}$
- Une normalisation est faite sur chaque entrée, $\hat{\mathbf{x}}^\mathbf{i} \leftarrow \frac{\mathbf{x^i} - \mu_{\mathbb{B}}}{\sqrt{\sigma^2_{\mathbb{B}} + \epsilon}}$ avec $\epsilon > 0$.
- À cause de la normalisation, $\hat{\mathbf{x}}^\mathbf{i} \in [0, 1]^n$, ce qui n'est pas toujours désiré. Afin de garder l'intensité des valeurs (celle d'avant la normalisation), deux paramètres sont rajoutés : $\mathbf{y_i} \leftarrow \gamma \hat{\mathbf{x}}^\mathbf{i} + \beta = BN_{\gamma,\beta}(\mathbf{x^i})$ avec β et γ des paramètres initialisés aléatoirement, appris pendant la phase d'entrainement.

L'algorithme de normalisation par batch d'un réseau est présenté sur l'algorithme 5.

algorithme 5 Normalisation par batch d'un réseau Res

Entrée : un ensemble d'entrées $\mathbb{X} = (\mathbf{x^1}, ..., \mathbf{x^m})$
Pour i=1 à m **Faire :**
    Calculer $\mathbf{y^i} = BN_{\gamma,\beta}(\mathbf{x^i})$ comme précédemment
    Modifier dans chaque couche de Res : $\mathbf{x^i} = \mathbf{y^i}$
Entrainer Res pour optimiser les paramètres $(\gamma^i, \beta^i)_{1 \leq i \leq n}$
Pour i=1 à m **Faire :**
    Utiliser Res sur des batchs $\mathbb{B}$ de taille m
    Calculer la moyenne des moyennes μ^i et des variances $\sigma(\mathbf{x^i})$
    Remplacer dans Res, la transformation $\mathbf{y^i} = BN_{\gamma,\beta}(\mathbf{x^i})$ par
    $\mathbf{y^i} = \frac{\gamma^i}{\sqrt{(\sigma(\mathbf{x^i}) + \epsilon)}}\mathbf{x^i} + (\beta^i - \frac{\gamma^i \mu^i}{\sqrt{(\sigma(\mathbf{x^i}) + \epsilon)}})$

.2.0.7 Régularisation

Afin d'éviter le surapprentissage, un terme de régularisation est souvent ajouté à la fonction de perte. Lors du surapprentissage, l'erreur sur l'ensemble d'entrainement diminue alors que l'erreur sur l'ensemble de validation augmente ; en d'autre terme, le modèle se spécialise sur l'ensemble d'entrainement, on peut dire qu'il le *mémorise*.

Pour un réseau paramétré par θ, un terme $\lambda||\theta||_p^2$ est souvent ajouté à la fonction de perte (où λ est un hyperparamètre). La régularisation consiste fréquemment à pénaliser la norme des poids, pour empêcher qu'ils atteignent des valeurs trop extrêmes, cas correspondant souvent au phénomène de surapprentissage. Courament, les cas $p = 1$ ainsi que $p = 2$ sont utilisés. Le cas $p = 2$ est nommé *weight decay* car il force la valeur des poids à diminuer vers 0, $p = 1$ est par exemple utilisé en compression de modèle (et plus largement en élagage), car il force la valeur des poids à 0.

.2.1 Auto-encodeur

Auto-encodeur

Les auto-encodeurs (AE) ont été introduits par Hinton et al. [139] pour répondre à un problème de réduction de dimension. C'est une technique d'apprentissage non supervisé.

Un auto-encodeur est un réseau de neurones qui apprend à reproduire ses entrées. Le principe de base est schématisé sur la figure 74.

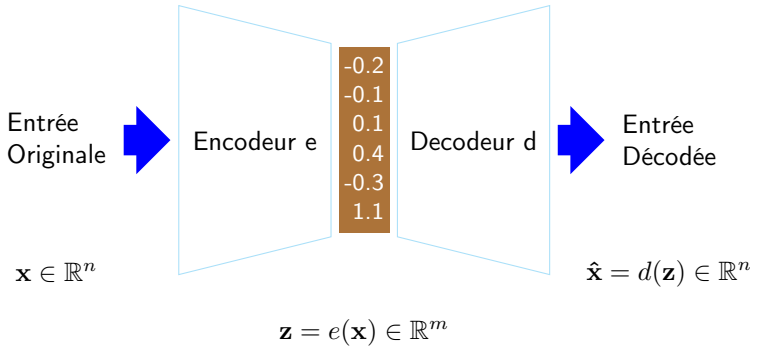


Figure 74: Schéma descriptif d'un auto-encodeur.

L'encodeur est un réseau de neurones dont le but est de trouver les caractéristiques importantes des données d'entrée. Souvent, ces caractéristiques sont *résumées* dans un vecteur de petite dimension. On appelle l'espace latent, l'espace où $\mathbf{z}$ (le vecteur de caractéristiques) vit. Le décodeur est un réseau qui essaie de reconstruire les données de départ en se servant uniquement du vecteur de caractéristiques donné par l'encodeur. Il est souvent entrainé en minimisant l'erreur quadratique entre les données reconstruites et les données d'entrée :

$$loss = ||\mathbf{x} - \hat{\mathbf{x}}||^2 = ||\mathbf{x} - d(\mathbf{z})||^2 = ||\mathbf{x} - d(e(\mathbf{x}))||^2$$

Avec :

— $\mathbf{x}$ l'entrée originale.

— e et d des réseaux choisis, peuvent être tout type de réseaux (PMC, CNN, ...)

— $\mathbf{z} = e(\mathbf{x})$ le vecteur de caractéristiques.

— $\hat{\mathbf{x}} = d(\mathbf{z})$ l'entrée reconstruite.

L'idée est de trouver les caractéristiques importantes, permettant de reconstituer l'image. L'encodeur est aussi utilisé afin de décrire les données d'entrée dans un espace vectoriel plus petit.

Les auto-encodeurs ne sont pas génératifs, ils ne génèrent pas de nouvelles données. Basés sur les auto-encodeurs, les auto-encodeurs variationnels ont été proposés comme modèle génératif.

Auto-encodeur variationnel

Les auto-encodeurs variationnels (VAE) ont été présentés par Kingma and Welling [140] afin de générer de nouvelles données.

Les VAE sont des modèles génératifs, empruntant une architecture similaire aux AE. Les VAE sont des modèles probabilistes. Plutôt que d'apprendre e et d, le VAE apprend des distributions $p_e(\mathbf{z}|\mathbf{x})$ et $p_d(\mathbf{x}|\mathbf{z})$

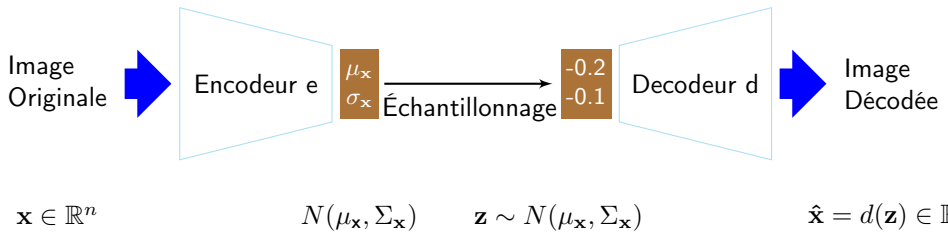


Figure 75: Schéma descriptif d'un auto-encodeur variationnel.

L'entrainement se fait selon ce processus :

— L'entrée est encodée comme une distribution sur l'espace latent.

— Un vecteur de l'espace latent est choisi par tirage selon la distribution.

— Le vecteur est décodé et l'erreur est calculée.

— La rétropropagation est appliquée. Afin d'apprendre les paramètres de la distribution, n valeurs sont échantillonnées (afin d'obtenir une bonne représentation) la moyenne de l'erreur sur ces n valeurs est ensuite effectuée, et une nouvelle distribution est calculée.

.2.2 Réseaux antagonistes génératifs (GAN)

Les GAN ont été introduits par Goodfellow et al. en 2014 [141]. Ils ont ensuite été grandement utilisés, notamment pour de l'augmentation de données. Leur principe est représenté dans la figure 76.

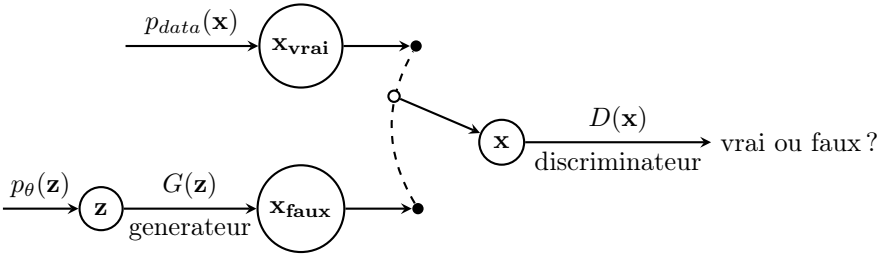


Figure 76: Réseaux antagonistes génératifs (GAN) ; G est le générateur ; D est le discriminateur

Dans un premier temps, nous échantillonnons un vecteur de bruit $\mathbf{z}$, selon une loi de probabilité multivariée (souvent normale ou uniforme).

$\mathbf{z} \sim N(O, I)$ ou $\mathbf{z} \sim U([-1, 1])$

Ici, $\mathbf{z}$ représente représente les caractéristiques latentes des données générées par la suite. L'idée est d'utiliser un générateur ($G(\mathbf{z})$, un réseau de neurones paramétré par θ) produisant des fausses images ($\mathbf{x_{faux}}$) (au début aléatoirement $p_\theta(\mathbf{z})$).

Ces images générées sont alors comparées aux vrais images par le discriminateur. Son rôle est d'identifier les vrais images des fausses, en testant à tour de rôle soit les vraies données ($\mathbf{x_{vrai}}$) soit les fausses $G(\mathbf{z})$. La sortie du discriminateur $D(\mathbf{x})$ est la probabilité que l'entrée $\mathbf{x}$ soit réelle, i.e. P(classe entrée = vraie image). Le discriminateur est entrainé comme un réseau de neurone profond, si l'entrée est réelle, $D(\mathbf{x})$ doit être égal à 1, sinon $D(G(\mathbf{z}))$ doit être égal à 0. À chaque donnée d'entrainement, le discriminateur fournit au générateur un vecteur de probabilité représentant la véracité ou non de la donnée.

Le générateur est entrainé par rétropropagation dans le but de créer des données pouvant tromper le discriminateur ($D(G(\mathbf{z})) = 1$).

Les deux réseaux sont entrainés l'un après l'autre, selon l'algorithme proposé dans [141] dans le but d'obtenir une meilleure discrimination et une meilleure génération des données.

Dans le modèle de base, pour l'entrainement, la fonction de perte *Minmax* est utilisée :

$$Min_G Max_D(\mathbb{E}_{\mathbf{x}}[log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}}[log(1 - D(G(\mathbf{z})))])$$

Avec $\mathbb{E}_{\mathbf{x}}$ l'espérance sur les vraies instances, et $\mathbb{E}_{\mathbf{z}}$ l'espérance sur les entrées aléatoires du générateur.

Le but du générateur est de minimiser cette fonction de perte, alors que le discriminateur veut la maximiser. Au final, le générateur pourra créer des données semblables à la base de données, utiles pour l'apprentissage d'un réseau.

Cette fonction possède des problèmes de convergence [142] et d'autres méthodes d'entrainement ainsi que d'autres architectures ont émergées ces dernières années. Nous pouvons par exemple citer les Wasserstein Gan, introduits dans [143] se basant sur la distance de Wasserstein ou les architectures BigGan [28].

.3 Réduction de dimension d'un espace

Les réseaux correspondants nous fournissent des vecteurs de caractéristiques des différentes instances. L'espace créé par ces vecteurs a comme dimension le nombre de neurones choisi à la fin de ces couches N. Afin de représenter cette espace en deux dimensions, pour une analyse graphique des résultats, quatres méthodes différentes ont été utilisées.

.3.1 Analyse en composantes principales

L'analyse en composantes principales (ACP) à été formalisée dans les années 1930 par Harold Hotelling [144]. Elle est de nos jours encore utilisée dans de nombreux domaines. Son but est de trouver une projection des données d'entrée sur un espace vectoriel de dimension inférieure. Elle a beaucoup d'avantages mais le principal et de décorréler les données. Pour une matrice X de dimension $K \times N$ sur un espace de dimension $N \times m$

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

Les variables aléatoires $\mathbf{X_i}$, dont $(X_{1,i}, \ldots, X_{K,i})$ sont des réalisations indépendantes, ont une moyenne $\bar{X}_i = (\bar{x}_1, ..., \bar{x}_N)$. L'ACP consiste à :

— Centrer les données :

$$\bar{X} = \begin{bmatrix} x_{1,1} - \bar{x}_1 & \cdots & x_{1,N} - \bar{x}_N \\ \vdots & \ddots & \vdots \\ x_{K,1} - \bar{x}_1 & \cdots & x_{K,N} - \bar{x}_N \end{bmatrix}$$

— Calculer la matrice de covariance $C = \frac{1}{N-1} X^\top X$

— Diagonaliser cette matrice : $C = PDP^\top$ avec $D = diag(\lambda_1, ..., \lambda_N)$

— Calculer les vecteurs propres, associés aux valeurs propres λ, de la matrice de covariance, qui formeront une nouvelle base. C étant symétrique, cette nouvelle base est orthonormée.

— Sélectionner les m valeurs et vecteurs propres les plus grandes de la nouvelle base, afin d'obtenir un espace de dimension $N \times m$.

.3.2 t-distributed stochastic neighbor embedding

L'algorithme t-distributed stochastic neighbor embedding (t-SNE) est une technique de réduction de dimension développée par Hinton et Maaten [145].

Pour une famille de m vecteurs $(\mathbf{x}^1, ..., \mathbf{x}^m)$, t-SNE consiste à :

— Choisir un point $\mathbf{x}^i$.

— Calculer la similarité entre $\mathbf{x}^i$ et tous les autres points $\mathbf{x}^j$. Dans un premier temps la distance euclidienne $||\mathbf{x}^j - \mathbf{x}^i||_2$ entre ces deux points est calculée. Les auteurs la transforment proportionnellement à une loi de probabilité gaussienne centrée en $\mathbf{x}^i$ et de variance σ_i :

$$p_{j|i} = \frac{\exp(-\|\mathbf{x^i} - \mathbf{x}^j\|_2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}^i - \mathbf{x}^k\|_2 / 2\sigma_i^2)}.$$

Pour les points très éloignés, la probabilité d'être choisi comme voisin sera presque nulle. La variance est choisie sous ce principe : elle doit être petite s'il y a une grande densité de population dans l'espace de départ et grande si les données sont éparpillées.

— Calculer la probabilité de choisir $\mathbf{x}^j$ comme voisin de $\mathbf{x}^i$:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2m}.$$

— Calculer une mesure de perplexité (mesure de qualité des modèles probabilistes)

comme contrainte afin de déterminer le σ optimal pour chaque base de données :

$$Perplexity = 2^{-\sum_j p_{j|i} log_2 p_{j|i}}.$$

Elle peut être vue comme une cible pour notre point central.

— Apprendre un espace à m dimensions afin de refléter les similarités p_{ij} de l'espace de base. La loi de Student est utilisée pour mesurer les similarités entre deux points $(\mathbf{y^i}, \mathbf{y^j})$ dans l'espace de faible dimension :

$$q_{ij} = \frac{(1 + \|\mathbf{y^i} - \mathbf{y^j}\|_2)^{-1}}{\sum_{k \neq i}(1 + \|\mathbf{y^i} - \mathbf{y^k}\|_2)^{-1}}.$$

— Apprendre les positions des nouveaux points y^i en minimisant, grâce à une descente de gradient, la divergence de Kullback-Leibler (KL) :

$$\mathrm{KL}\left(P \parallel Q\right) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

.3.3 Uniform Manifold Approximation and Projection (UMAP)

Présenté en 2018 par Lelan et al. [146], UMAP est une technique de réduction de dimension très performante. Elle se présente comme successeur à l'algorithme t-SNE. Si elles sont, en effet, très comparables, UMAP diverge en un certain nombre de points :

— Afin de calculer la similarité, UMAP utilise une distribution exponentielle de probabilité ; pas nécessairement la distance euclidienne. En utilisant les même notations que précédemment.

$$p_{i|j} = exp(\frac{d(\mathbf{x^i}, \mathbf{x^j}) - \rho_i}{\sigma_i}).$$

Avec ρ_i la distance du point i à son voisin le plus proche.

— Absence de normalisation (absence de la somme au dénominateur). Cela réduit drastiquement le temps de calcul.

— Utilisation du nombre de voisins le plus proche au lieu d'une mesure de perplexité :

$$k = 2^{\sum_i p_{ij}}.$$

— La probabilité de choisir $\mathbf{x^j}$ comme voisin de $\mathbf{x^i}$ est définie comme ceci :

$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j} p_{j|i}.$$

— Pour le nouvel espace à m dimensions, UMAP n'utilise pas la distribution de Student mais :

$$q_{i_j} = (1 + a(\mathbf{y^i} - \mathbf{y^j})^{2b})^{-1}$$

avec $a = 1.93$ et $b = 0.79$ des hyperparamètres.

— UMAP utilise une entropie croisée binaire au lieu de la divergence KL.

— UMAP utilise une descente de gradient stochastique au lieu d'une descente de gradient classique.

UMAP crée une représentation topologique sur l'espace de projection. En pratique, les données et leurs similarités peuvent être vues comme un graphe. Ce dernier est parfois représenté dans l'espace afin d'obtenir une illustration du nombre total de connexions. S'il y a beaucoup de connexions entre les classes, cela signifie que les données sont très similaires.

.3.4 Couche complètement connectée de dimension deux

L'idée est de rajouter une couche complètement connectée (*fully connected*, Figure 77) de dimension deux, et de représenter ces nouveaux vecteurs de caractéristiques sur un plan.

En bout du réseau, une couche complètement connectée de dimension 2 peut être ajoutée, pour permettre un réprésentation dans $\mathbb{R}^2$ des données. Des poids liant cette aux réseaux sont appris, permettant de calculer le plongement dans l'espace de représentation.

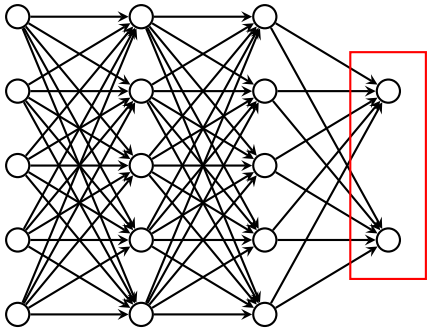


Figure 77: Réduction de dimensions grace à l'ajout d'une couche complètement connectée de dimension deux.

Maintenant que les bases de l'apprentissage profond et les méthodes d'apprentissage *few shot* utilisées, ont été décrites, l'état de l'art ainsi que les travaux connexes à nos

sujets sont présentés.