UNIVERSITE BADJI MOKHTAR-ANNABA
FACULTE DES SCIENCES DE L'INGENIORAT
DEPARTEMENT D'INFORMATIQUE

جامعة باجي مختار – عنابـــــــــة
كلية علـــــــوم الهندســـــــة
قسم الإعـــــــلام الآلـــــــي

Mémoire présenté en vue
de l'obtention du diplôme de Master

Intitulé

RESEAUX DE NEURONES CONVOLUTIFS POUR UNE CLASSIFICATION MULTI-LABEL

Domaine : Mathématiques-Informatique
Filière : Informatique
Spécialité : System d'information et décision

Par : Melle Feriel Zehar

Jury d'évaluation

Qualité	Nom et Prénom	Grade	Université
Président	Dr. Mehdi Heriati	MCB	Badji Mokhtar-Annaba
Rapporteur	Dr. Abdelouahid Alalga	MCB	Badji Mokhtar-Annaba
Examinateur	Dr. Sabri Ghazi	MCB	Badji Mokhtar-Annaba

REMERCIEMENTS

En tout premier lieu, je remercie dieu, tout puissant, de nous avoir donné la santé et la volonté d'entamer et de terminer ce mémoire.

Ce mémoire n'aurait jamais pu voir le jour sans le soutien actif des membres de ma famille, surtout mes parents qui m'ont toujours encouragé.

Enfin je tiens à exprimer vivement mes remerciements avec une profonde gratitude à toutes les personnes qui ont contribué de près ou de loin à sa réalisation, car un projet ne peut pas être le fruit d'une seule personne.

DEDICACES

Je dédie ce mémoire

A mes chers parents ma mère et mon père pour leur patience, leur amour, leur soutien et leurs encouragements

A mes frères.

A mes amies et mes camarades.

Sans oublier tous les professeurs que ce soit du primaire, du moyen, du secondaire ou de l'enseignement supérieur.

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

TABLE D'EQUATION

INTRODUCTION

La classification multi-label est une extension de la classification traditionnelle dans laquelle les classes ne sont pas mutuellement exclusives et où chaque exemple peut appartenir à plusieurs classes simultanément. Ce nouveau type d'apprentissage a reçu une attention d'importance croissante cette dernière décennie stimulée initialement par des travaux en catégorisation textuelle et est de plus en plus répandue en tant que technique d'exploration de données. La classification multi-label peut s'appliquer à un large éventail d'applications, notamment la catégorisation de documents textuels, la classification sémantique d'images, l'annotation de vidéos, la classification fonctionnelle de protéines etc.

Dans ce travail, nous nous intéressons à la classification sémantique des images numériques. L'image constituent une source d'informations très expressive et très abondante qu'il serait judicieux d'exploiter pour aider dans diverses activités humaines. D'autant plus que la démocratisation des systèmes d'acquisition d'images numériques, combinée à l'essor du Web ont permet d'avoir des quantités faramineuses de ce type de données. La classification sémantique d'images désigne le processus de caractérisation de l'image ou de son contenu par des concepts. Intuitivement, il s'agit d'étiqueter les images automatiquement par des concepts que l'on perçoit dans celles-ci : personne, train, montagne, paysage, etc.

Dans ce contexte, nous nous focaliserons plus en particulier sur la classification des affiches (posters) de film selon le genre. Dans la mesure où un film peut appartenir à plusieurs genres simultanément, la catégorisation de film est considérée comme un problème de classification multi-label. Dans l'industrie cinématographique, l'affiche de film prend une importance toute singulière. Cette dernière est le premier support publicitaire du film dont elle fait la promotion. De fait, nous allons procéder ici à leur catégorisation selon le genre afin d'aider des spectateurs à choisir entre les films en fonction de leur goût et de leur humeur.

CONTEXTE DE RECHERCHE

Les affiches ou les posters de films jouent un rôle primordial dans le choix du film. L'affiche est indissociable du film qu'elle promeut. Elle marque les esprits et forge l'empreinte visuelle qu'on lui associe, car elle constitue le premier contact avec le public sur lequel dépend la réussite ou l'échec du film. C'est souvent le moyen le plus utilisé pour présenter un film aux spectateurs, que ce soit dans les salles de cinéma, dans les journaux ou sur les pochettes de DVD. Cette affiche permet aux spectateurs d'avoir un aperçu rapide sur le contenu du film à visualiser.

Avec l'avènement des plateformes de streaming numérique, le rôle de l'affiche a gagné en importance. Pour les utilisateurs qui naviguent sur ces plates-formes numériques, la visualisation de l'affiche c'est ce qui permet de distinguer un film parmi tant d'autres. Malheureusement, cette expansion du contenu médiatique et des services de streaming sur internet sont devenus une source de problème pour les utilisateurs qui n'arrive plus à trouver ce qu'il cherche devant une telle explosion de contenu numérique. En effet, une enquête auprès des clients conclut qu'en général, les clients perdent leur enthousiasme en 60 à 90 seconde de recherche de film sur le web s'il ne trouvent pas le bon produit.

C'est donc tout naturellement qu'a émergé la nécessité de la recherche d'une solution informatique à ce problème. Il s'agit de mettre en place un système permettant de regrouper ces films en catégories explicites afin de réduire le temps passé par les clients à choisir un contenu à regarder. Nous proposons, dans ce projet, de réaliser une classification des films selon le genre à partir de leurs affiches. Dans cette optique, nous avons conçu un modèle qui prend l'image d'une affiche en entrée, qui va être traitée par un réseau de neurones convolutifs (CNN) et produit en sortie les différents genres cinématographiques auxquels l'affiche appartient. Un tel modèle serait très utile à l'industrie cinématographiques et pour les plates-formes de streaming. Il peut être utilisé dans des tâches telles que le tri rapide de bibliothèque de films et la mise en place de systèmes de recommandation.

CONTENU DU MEMOIRE

Notre mémoire est structuré comme suit :

- Dans le chapitre 01 : nous aborderons les différents aspects de la classification des données multi-labels.
- Le chapitre 02 : présente globalement les réseaux de neurone et détaille la structure, les différentes couches et la méthode d'entrainement pour les réseaux à convolution.
- Le dernier chapitre consiste à décrire en détail notre protocole d'expérimentation ainsi que les différents outils et technologies employés.

CHAPITRE 1 : CLASSIFICATION MULTI-LABEL

1.1 INTRODUCTION

Traditionnellement, il existe fondamentalement deux types de classification dans la reconnaissance des formes : la classification binaire et la classification multi-classes, la classification multi-classe généralise la classification binaire en permettant la présence de plusieurs classes. Cependant, l'émergence de nouvelles applications de domaine, où l'étiquetage d'objets du monde réel tels que les textes, les images, la musique et les vidéos implique l'attribution de plusieurs étiquettes en même temps. Ce qui nécessite un nouveau paradigme d'apprentissage. Dans ce contexte, la classification multi-labels est introduite pour répondre à cette nouvelle exigence. Par conséquent, de nombreuses méthodes ont récemment été proposées pour exploiter les données multi-labels. De manière générale, cette tâche peut être accomplie par deux approches principales : la transformation des données et l'adaptation des algorithmes. Dans ce chapitre, nous aborderons les différents aspects de la classification des données.

1.2 CLASSIFICATION

La classification est l'un des sujets les plus populaires du DM (Data Mining). Il s'agit d'une tâche prédictive, généralement réalisée à l'aide de techniques d'apprentissage supervisé [1].

La classification vise à apprendre à partir d'exemples étiquetés, un modèle capable de prédire les étiquettes (ou les classes) pour des échantillons de données jamais vu auparavant. Certains algorithmes de classification, tels que ceux fondés sur l'apprentissage par instance (instance-based learning) [2], peuvent permettre ce travail sans avoir préalablement à construire un modèle.

L'ensemble des attributs d'un jeu de données de classification est divisé en deux sous-ensembles. Le premier contient les caractéristiques d'entrée, c'est-à-dire les variables qui

serviront de prédicteurs. Le deuxième sous-ensemble contient les attributs de sortie, la classe ou l'étiquette assignée à chaque instance. Les algorithmes de classification induisent le modèle en analysant la corrélation entre les caractéristiques d'entrée et la classe de sortie. Une fois qu'un modèle est obtenu, il peut être utilisé pour traiter l'ensemble des varaibles de nouveaux échantillons de données obtenant ainsi une prédiction de classe.

Selon la nature du deuxième sous-ensemble d'attributs, celui-ci contenant la classe, plusieurs types de problèmes de classification peuvent être identifiés. Le tableau 1.1 résume les configurations les plus courantes, en fonction du nombre de sorties et de leurs types.

Nombre de sortie	Type de sortie	Type de classification
1 par instance	Binaire	Binaire
1 par instance	Valeurs multiples	Multi-class
n par instance	Binaire	Multi-label
n par instance	Valeurs multiples	Multidimensionnel
1 par instance	binaire/Valeurs multiples	Multi-instance

Tableau 1 : Problèmes de classification liés à la sortie à prévoir

1.2.1 CLASSIFICATION BINAIRE

C'est le problème de classification le plus facile auquel nous pouvons être confrontés [1]. Les instances d'un jeu de données binaire n'ont qu'une seule classe en sortie, et elle ne peut prendre que deux valeurs différentes. C'est généralement divisé en positifs et négatifs, mais peut également être interprété comme vrai et faux, 1 et 0, ou toute autre combinaison de deux valeurs. Un exemple classique de cette tâche est le filtrage de spam (le filtrage du courrier indésirable, voir figure 1), dans lequel le classificateur apprend, à partir du contenu des messages quels messages peuvent être considérés comme du spam.

Un classificateur binaire vise à trouver une limite capable de séparer les instances en deux groupes, l'un appartenant à la classe positive et l'autre à la classe négative. En pratique, en

fonction de l'espace des instances d'entrée, la distribution des échantillons peut être beaucoup plus difficile, nécessitant ainsi de dresser des frontières complexes pour séparer les instances.

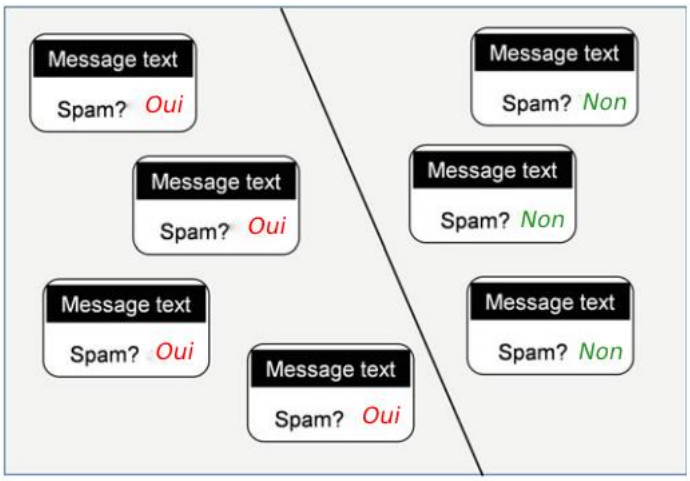


Figure 1 : le filtrage de spam est un problème typique de classification binaire.

Les applications actuelles de la classification binaire sont le filtrage des e-mails, pour éliminer spam, l'analyse des prêts bancaires, pour déterminer si un client est économiquement fiable ou non, le diagnostic médical pour déterminer si un patient a une certaine maladie ou non, et d'une façon générale, la reconnaissance de toute sorte de modèles binaires.

1.2.2 CLASSIFICATION MULTI-CLASSE

Un jeu de données multi-classe n'a qu'une seule classe en sortie, comme dans la classification binaire, mais cette classe peut prendre n'importe quelle valeur parmi un certain nombre de valeurs prédéfinies. La signification de chaque valeur et la valeur elle-même, sont spécifiques à chaque application. Dans le cas où l'ensemble des valeurs est défini dans un intervalle continue, la tâche considérée sera appelée régression. Les valeurs de classe peuvent avoir une relation d'ordre ou non. L'un des exemples de classification multi-classe le plus connu est l'identification des espèces d'iris (voir fig. 2). En utilisant les caractéristiques de la fleur, c'est-à-dire la longueur et la largeur des pétales et des sépales, le classificateur apprend à classifier les nouvelles instances dans la famille correspondante.

De nombreux algorithmes de classification multi-classes reposent sur la binarisation[3] : une méthode qui entraîne de manière itérative un classificateur binaire pour chaque classe en

considérant les autres classes comme la valeur négative, selon une approche One-vs-All (OVA), ou pour chaque paire de classes, selon une approche "One-vs-One" (OVO).

La classification multi-classe peut être considérée comme une généralisation de la classification binaire. Il n'y a qu'une seule sortie, mais elle peut prendre n'importe quelle valeur, alors que dans le cas binaire, la sortie est limitée à deux valeurs possibles.

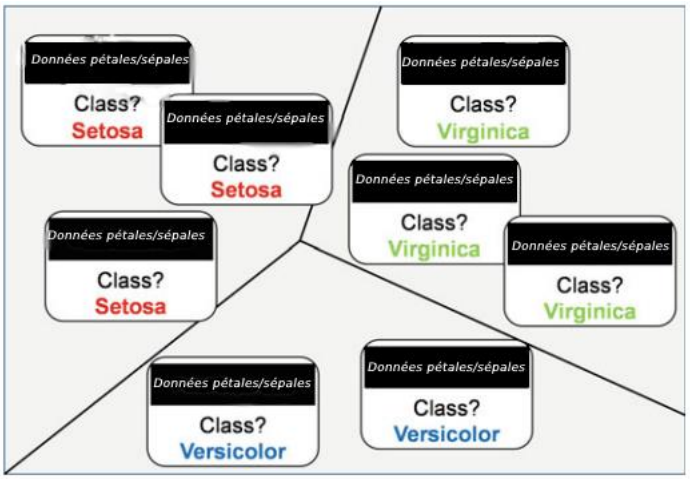


Figure 2 : La catégorisation des espèces d'iris.

1.2.3 CLASSIFICATION MULTI-LABEL

Contrairement aux deux modèles de classification précédents, dans la classification multi-label [4,5,6], chaque instance de données est associée à un vecteur de sorties, au lieu d'une seule valeur. La longueur de ce vecteur est fixée en fonction du nombre d'étiquettes différentes dans le jeu de données. Chaque élément du vecteur sera une valeur binaire, indiquant si l'étiquette correspondante est pertinente pour l'échantillon ou non. Plusieurs labels peuvent être actifs en même temps. Chaque combinaison distincte d'étiquettes est appelée *sous-ensemble d'étiquettes* (*labelset)*. La figure 3 illustre l'une des applications multi-labels des plus classiques, l'étiquetage des images. L'ensemble de données à quatre étiquettes au total et chaque image peut être attribuée à l'un d'entre eux, ou même tous à la fois s'il y avait une image dans laquelle les quatre concepts apparaissent.

La classification multi-label est actuellement appliquée dans de nombreux domaines, la plupart d'entre eux sont liés à l'étiquetage automatique des ressources multimédias des réseaux sociaux telles que les images, la musique, la vidéo, articles de blog, etc. Les algorithmes utilisés pour

cette tâche doivent être capables de faire plusieurs prédictions à la fois, que ce soit en transformant les ensembles de données originaux ou en adaptant les algorithmes de classification binaire/multi-classe existants.

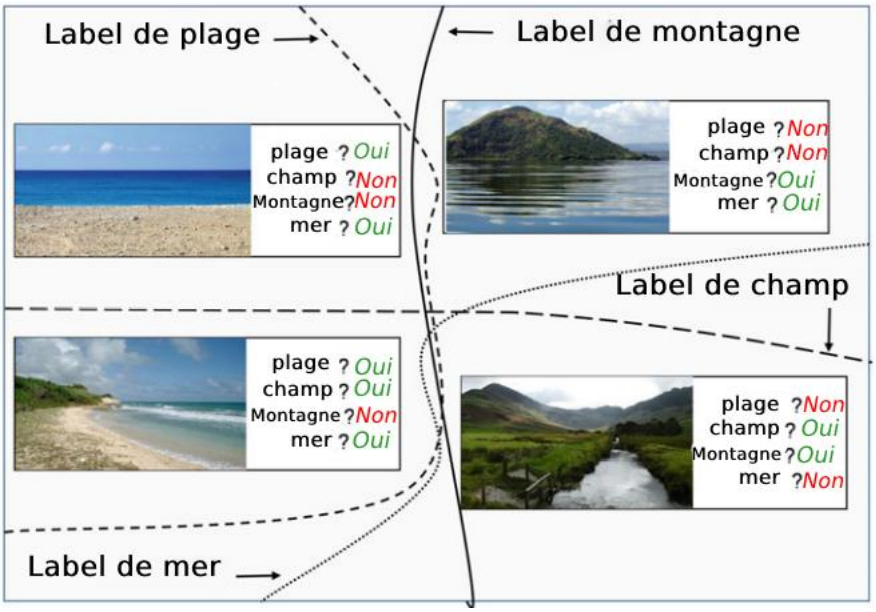
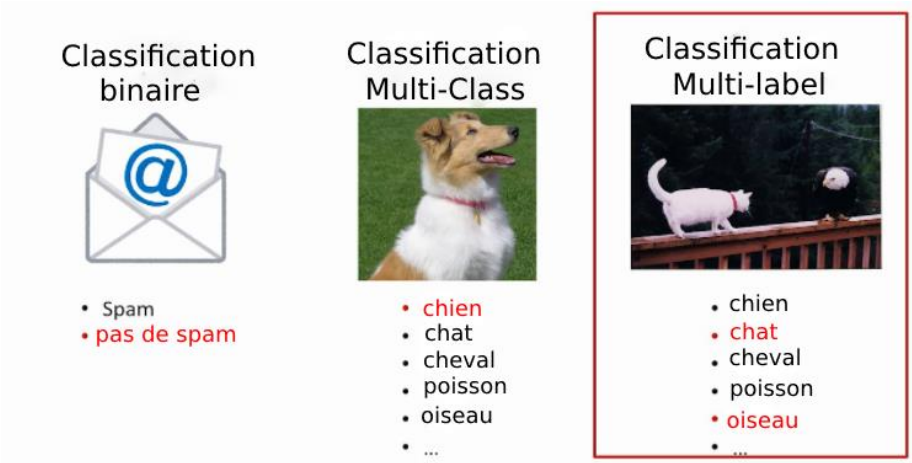


Figure 3 : L'étiquetage d'images par classification multi-label.

La figure suivante illustre la différence entre les différents types de classification



1.2.4 CLASSIFICATION MULTIDIMENSIONNEL

Ce type de tâche peut être considéré comme une généralisation du précédent modèle. Les ensembles de données multidimensionnels [7] ont également un vecteur de sortie associé à chaque instance, plutôt qu'une seule valeur. Cependant, chaque élément de ce vecteur peut prendre n'importe quelle valeur parmi un ensemble prédéfini, et qui n'est pas limité à être

binaire. Par conséquent, la relation entre la classification multidimensionnelle et la classification multi-label est essentiellement la même que celle mentionnée précédemment pour la classification multi-classes et la classification binaire.

Les techniques multidimensionnelles sont étroitement liées à celles utilisées dans le champ multi-label, avec des méthodes de transformation jouant un rôle important pour faire face à cette tâche. L'éventail des applications potentielles est également similaire. La classification multidimensionnelle est utilisée pour catégoriser les images, la musique, les textes et les ressources analogues, mais en prédisant pour chaque étiquette une valeur dans un ensemble plus large que la classification multi-labels.

1.3 APPRENTISSAGE A INSTANCES MULTIPLES

Contrairement à tous les autres types de classification décrits ci-dessus, le paradigme d'apprentissage à instances multiples [8], également connu sous le nom *d'apprentissage multi-instance,* apprend une étiquette de classe commune pour un ensemble de vecteurs de caractéristiques d'entrée. Chaque groupe d'instances est généralement appelé "*bag*". Et l'étiquette de la classe associée appartient au bag, au lieu des instances de données individuelles qu'il détient.

Parmi les applications de ce paradigme de classification, l'une des plus facile à comprendre c'est la catégorisation des images. Les images originales sont divisées en *régions* ou *patchs* ainsi chaque image est représentée par un ensemble de vecteurs, contenant chacun les données d'un patch ou d'une région. L'étiquette de classe associée peut dépendre de certains objets apparaissant dans certaines régions. L'apprentissage par instances multiples peut être étendu en considérant plusieurs étiquettes de classe comme sortie plutôt qu'une seule. D'autres variantes peuvent émerger telles que la classification multi-labels à instances multiples [17].

1.4 CLASSIFICATION MULTI-LABEL

La classification multi-label est une tâche d'exploration de données prédictive avec de multiples applications dans le monde réel : l'étiquetage automatique de nombreuses ressources telles que les textes, les images, la musique et la vidéo. L'apprentissage à partir de données multi-label peut être accompli par différentes approches, comme la transformation des données, l'adaptation de méthodes, et l'utilisation d'ensembles de classifieurs.

1.4.1 DEFINITION FORMELLE DU PROBLEME

La principale différence entre la classification traditionnelle (En général, nous ferons référence à la classification binaire et à la classification multi-classes, qui sont les types de classification les plus connus, en tant que classification *traditionnelle.)* et la classification multi-labels réside dans les résultats attendus des modèles formés. Alors qu'un classifieur traditionnel ne retourne qu'une seule valeur, un classifieur multi-label doit fournir un vecteur de valeurs. Ainsi, la classification multi-label peut être définie formellement comme suit.

1.4.2 DEFINITION

Définition 1 : Soit X l'espace des instances avec des échantillons de données x $\in A_1 \times A_2 \times ... \times A_f$, f étant le nombre d'attributs d'entrée et A_1, A_2,..., A_f des ensembles arbitraires. Chaque instance x sera obtenue par le produit cartésien de ces ensembles.

Définition 2 : Soit L l'ensemble de toutes les étiquettes possibles. $P(L)$ désigne l'ensemble de puissances de L, contenant toutes les combinaisons possibles d'étiquettes $l \in L$ y compris l'ensemble vide et L lui-même. $k = |L|$ est le nombre total d'étiquettes en L.

Définition 3 : Soit Y l'espace de sortie, avec tous les vecteurs possibles Y, $Y \in P(L)$. La longueur de Y sera toujours k.

Définition 4 : Soit D *l'*ensemble des données multi-labels, contenant un sous-ensemble fini de $A_1 \times A_2 \times ... \times A_f \times P(L)$. Chaque élément $(X, Y) \in D | X \in A_1 \times A_2 \times ... \times A_f$, $Y \in P(L)$ sera une instance ou un échantillon de données. $n = |D|$ sera le nombre d'éléments dans D.

Définition 5 : Soit F un classificateur multi-label, défini comme $F : X \rightarrow Y$. L'entrée de F sera toute instance $X \in X$, et la sortie sera la prédiction $Z \in Y$. Par conséquent, la prédiction du vecteur d'étiquettes associé à une instance quelconque peut être obtenue par $Z = F(X)$.

1.4.3 APPLICATIONS DE LA CLASSIFICATION MULTI-LABELS

Une fois que les principaux concepts liés à la classification multi-labels ont été introduits, la prochaine question qui se pose éventuellement est de savoir à quoi elle peut servir. Comme on l'a expliqué dans la section précédente, un classificateur multi-label vise à prédire un ensemble d'étiquettes pertinentes pour une nouvelle instance de données.

Dans cette section, plusieurs domaines d'application qui peuvent tirer parti de cette fonctionnalité sont présentés.

1.4.4CATEGORISATION DE TEXTE

La classification multi-labels est née d'une solution permettant d'organiser des documents textuels en plusieurs catégories non mutuellement exclusives. C'est pourquoi il y a tant de publications sur ce sujet [9,10,11,12].

Les documents textuels peuvent être trouvés n'importe où, des grandes entreprises qui stockent toutes sortes de rapports aux particuliers qui classent leurs factures et leurs messages électroniques. Tous les livres et magazines publiés, nos dossiers médicaux historiques, ainsi que des articles dans les médias électroniques, les articles de blog, les messages des forums (forum questions/réponses) etc., sont également des documents de texte. La plupart d'entre eux peuvent être classés dans plus d'une catégorie, d'où l'utilité de la classification multi-labels pour accomplir ce genre de travail.

Le processus habituel de transformation d'un ensemble de documents texte en jeu de données multi-labels repose sur des techniques d'exploration de texte. Les documents sources sont analysés, les mots non informatifs sont supprimés, et les vecteurs avec chaque occurrence de mot dans ces documents sont calculés. A la fin de ce processus, chaque document est décrit par une ligne, et les colonnes correspondent à des mots et à leurs fréquences ou tout autre type de représentation tel que TF / IDF (Fréquence du terme/Fréquence inverse du document) [13].

1.4.5 ÉTIQUETAGE DES RESSOURCES MULTIMEDIAS

Bien que les documents contenant uniquement du texte étaient les plus fréquents dans le passé, aujourd'hui, les images, les vidéos et la musique sont des ressources couramment utilisées en raison de l'énorme croissance des technologies de stockage et de communication. En suivant les statistiques YouTube 2015, on estime que 432000 nouvelles vidéos sont chargées chaque jour. Le nombre des nouvelles musiques et des clips sonores publiés chaque jour est également impressionnant, et les nouvelles images et photographies affichées partout, des blogs aux sites Web tels que Tumblr, atteint la barrière des millions par jour.

La classification multi-label a été utilisée pour étiqueter tous ces types de ressources [14,15,17,18,19,20] pour identifier les objets qui apparaissent dans des ensembles d'images, ou les concepts qui peuvent être dérivés des extraits de vidéos. Grâce à la classification multi-label toutes ces nouvelles ressources peuvent être correctement classifiées sans avoir besoin de l'intervention humaine, qui serait plutôt cher.

1.4.6 GENETIQUE/BIOLOGIE

La protéomique et la génomique sont des domaines de recherche qui ont connu un essor énorme au cours des dernières années. En conséquence, d'immenses bases de données liées aux séquences de protéines ont été produites, mais seule une petite fraction d'entre eux ont été

étudiés afin de déterminer leur fonction. L'analyse des propriétés des protéines et de l'expression des gènes est une tâche très coûteuse. Mais les techniques de DM (Data Mining) peuvent accélérer ce processus et le rendre moins cher.

L'application de la classification multi-labels dans ce domaine [21,22] vise à prédire les fonctions biologiques des gènes et des protéines. Chaque gène peut avoir plus d'une fonction à la fois, d'où l'intérêt d'utiliser les techniques de la classification multi-label pour faire face à ce problème.

1.5 APPRENTISSAGE A PARTIR DE DONNEES MULTI-LABEL

Le processus pour obtenir un modèle de classification multi-label est similaire à celui utilisé pour la classification traditionnelle, suivant généralement des techniques d'apprentissage supervisées. La plupart des algorithmes reposent sur une phase initiale d'apprentissage, cela dépend des échantillons précédemment étiquetés pour ajuster les paramètres du modèle. Une fois entraîné, le modèle peut être utilisé pour prédire l'ensemble des étiquettes pour les nouvelles instances.

Lorsqu'il s'agit d'apprendre à partir de données multi-label, deux approches principales ont été appliquées : **transformation des données** et **adaptation de méthodes.** La première est basée sur des techniques de transformation qui, appliquées sur des jeux de données mutli-labels, sont capables de produire un ou plusieurs ensembles de données binaires ou multi-classes. Ceux-ci peuvent être traités avec des classificateurs traditionnels, La seconde approche vise à adapter des algorithmes de classification existants, de sorte qu'ils puissent traiter des données multi-labels, en produisant plusieurs sorties au lieu d'une seule. Une troisième alternative est l'utilisation des méthodes ensemblistes.

1.5.1 L'APPROCHE DE LA TRANSFORMATION DES DONNEES

La classification multi-labels est une tâche plus difficile que la classification traditionnelle, puisque le classifieur doit prédire plusieurs sorties à la fois. L'une des premières approches apparues pour résoudre ce problème a été la transformation, qui permet de produire un ou plusieurs problèmes plus simples. L'idée de la transformation consiste à générer à partir de

données multi-label des ensembles de données qui peuvent être traités par des classifieurs binaires ou multi-classes. En général, la sortie produite par ces classifieurs doit être rétro-transformée afin d'obtenir la prédiction multi-label.

1.5.2 L'APPROCHE DE L'ADAPTATION DES METHODES

La classification automatique [23] est une tâche traditionnelle du DM depuis des années. Tout au long de cette période, plusieurs familles d'algorithmes [24] ont été progressivement développées, testées et affinées. Ces algorithmes constituent une base essentielle pour développer de nouveaux algorithmes plus spécifiques, y compris ceux destinés à travailler avec des données multi-labels.

Certains modèles de classification ont été initialement conçus pour résoudre des problèmes binaires, puis étendus pour prendre également en compte les cas multi-classes []. Un exemple de ceux-ci sont les SVM [25,26]. D'autres approches sont capables de traiter plusieurs classes avec une grande simplicité. Le classificateur kNN [27], par exemple, peut prédire la classe la plus fréquente parmi les voisins de la nouvelle instance, s'il y a deux ou plusieurs classes. De manière analogue, les arbres de décisions, les réseaux de neurones ont été utilisés pour traiter les classifications binaires et multi-classes.

1.6 CLASSIFIEURS BASES SUR L'ADAPTATION

Contrairement aux méthodes basées sur la transformation de problèmes, celles qui suivent l'approche d'adaptation visent à étendre les algorithmes de classification existants pour les rendre capables de gérer des instances avec plusieurs sorties. Les changements qui doivent être introduits dans les algorithmes peuvent être assez simples ou vraiment complexes, selon la nature de la méthode originale et aussi la façon dont l'existence de plusieurs étiquettes va être prise en compte.

Il existe des propositions de classifieurs multi-labels basés sur les arbres de décisions, les réseaux neurones, des machines à support vectoriel, des techniques d'apprentissage basées sur les instances et les méthodes probabilistes.

1.6.1 ARBRE DE DESISTIONS

Les arbres de décision (DT) sont parmi les modèles de classification les plus faciles à comprendre. Malgré leur apparente simplicité, certains algorithmes DT, tel que C4.5 [28], sont capable de réaliser des performances qui les rendent très compétitifs par rapport à d'autres approches d'apprentissage.

1.6.2 MULTI-LABEL C4.5, ML-C4.5

Les recherches menées en [29] visaient à classer les gènes selon leur fonction, en tenant compte du fait qu'un même gène peut accomplir plusieurs fonctions. La solution proposée est fondée sur le célèbre algorithme C4.5 modifié de façon appropriée pour traiter plusieurs étiquettes à la fois. Les deux points clés de la méthode adaptée sont les suivants :

- Les feuilles de l'arbre contiennent des échantillons qui sont associés à un ensemble d'étiquettes. Au lieu d'une seule classe.
- La mesure d'entropie d'origine est ajustée pour prendre en considération la probabilité de non-appartenance des instances à une certaine étiquette.

Le processus itératif pour construire l'arborescence partitionne les instances en fonction de la valeur d'un certain attribut, calculer la somme pondérée des entropies de chaque sous-ensemble potentiel d'étiquettes. Étant donné que chaque feuille représente un ensemble d'étiquettes, cela a un impact à la fois sur le processus d'étiquetage de chaque nœud de l'arbre et sur la tâche d'élagage supplémentaire inhérente à C4.5

1.7 METHODES BASEES SUR LES RESEAUX NEURONES

Les réseaux de neurones artificiels (ANN) en général, et en particulier les réseaux à base de fonctions radiales (RBFNs), ont prouvé leur efficacité dans les problèmes de classification, ainsi que dans la régression et la prédiction de séries chronologiques. Par conséquent, l'adaptation des ANN pour accomplir des tâches de classification multi-label est un sujet récurrent dans la littérature. L'objectif de cette section est de fournir une description succincte de plusieurs algorithmes multi-labels basés sur les ANN.

1.7.1 PROPAGATION ARRIERE MULTI-LABEL, BP-MLL

Considéré comme le premier ANN multi-label-adapté, BP-MLL [30], est fondé sur l'un des modèles ANN les plus simples, le perceptron. L'algorithme d'apprentissage choisi pour la création du modèle est également bien connu, la rétro-propagation.

L'aspect essentiel de BP-MLL est l'introduction d'une nouvelle fonction d'erreur utilisée lors de l'apprentissage de l'ANN, et calculée en tenant compte du fait que chaque échantillon contient plusieurs étiquettes.

Dans BP-MLL, la couche d'entrée a autant de neurones qu'il y a d'attributs d'entrée. Le nombre d'unités dans la couche de sortie est déterminé par le nombre d'étiquettes considérées. Le nombre de neurones dans la couche cachée est également influencé par le nombre d'étiquettes, et ils utilisent généralement une fonction d'activation sigmoïde.

L'algorithme BP-MLL produit un classement des étiquettes lors de la classification des nouvelles instances. Pour décider quelles étiquettes seront prédites comme pertinentes, un seuil de séparation doit être défini. La configuration de ce seuil est la seule difficulté dans l'utilisation de BP-MLL.

1.7.2 RESEAUX A BASE DE FONCTIONS RADIALES MULTI-LABEL, ML-RBF

L'algorithme proposé dans [31], est une méthode spécialisée pour concevoir des RBFN adaptés pour travailler avec des données multi-label. Il prend les échantillons associés à chaque étiquette et exécute ensuite un clustering avec K-means autant de fois qu'il y a d'étiquettes. De cette façon, les centres des RBF sont définis. Le nombre de clusters par étiquette est contrôlé par un paramètre α. Selon la valeur affectée à α, le nombre d'unités dans la couche cachée sera égal ou supérieur au nombre d'étiquettes dans le jeu de connées multi-label. Le ML-RBF utilise la méthode SVD (Décomposition en valeurs singulières) pour ajuster les poids des connexions aux unités de sortie, en minimisant la somme au carré de l'erreur calculée dans chaque itération d'apprentissage. L'activation de tous les neurones est définie sur 1 et un biais est défini pour chaque étiquette.

Il existe deux variantes de ML-RBF, appelées FPSO-MLRBF et FSVD-MLRBF []. Conçues comme l'hybridation de techniques qui visent à améliorer les résultats produits par l'algorithme

original. FPSO-MLRBF s'appuie sur la méthode PSO (*Particle Swarm Optimization*, optimisation à essaim de particules) dans le but de définir le nombre d'unités dans la couche cachée, ainsi que d'optimiser les poids entre les couches cachées et de sortie. L'algorithme FSVD-MLRBF, quant à lui recourt à l'utilisation d'un K-means flou avec SVD.

1.7.3 MACHINE D'APPRENTISSAGE EXTREME

Introduit dans [32], Cet algorithme suggère une méthodologie pour adapter un « extreme Learning Machine » (machine d'apprentissage extrême, ELM) afin qu'il puisse traiter des échantillons multi-labels. Les ELM sont un type d'ANN avec seulement une couche cachée, caractérisés par la vitesse à laquelle ils peuvent apprendre. Cette caractéristique les rend idéales pour traiter des ensembles de données multi-labels.

La méthode proposée commence par utiliser l'analyse de corrélation canonique (CCA) afin de détecter les corrélations potentielles entre les attributs d'entrée et les étiquettes. En conséquent, un nouvel espace qui combine les entrées et les étiquettes est généré, et utilisé pour entraîner l'ELM.

1.8 METHODES BASEES SUR LES MACHINES A SUPPORT VECTORIEL

Les machines à support vectoriel ont été traditionnellement appliquées pour résoudre des problèmes de classification binaire. Comme d'autres techniques, elles ont évolué au fil du temps, étant étendues pour faire face à d'autres types de tâches telles que la classification multi-classes et multi-labels. Dans cette section, plusieurs des méthodes MLC basées sur les SVM sont décrites.

1.8.1 MODEL-X

La tâche visée par les auteurs dans [33], était d'étiqueter des paysages naturels. Chaque image peut contenir plusieurs objets à la fois et, par conséquence, elle peut être étiquetée avec plus d'une classe. Certaines des étiquettes sont : urbaines, plage, champ, montagne, plage + champ,

champ + montagne, etc. La méthode proposée, appelée MODEL-x, entraîne un SVM par étiquette en utilisant une nouvelle approche appelée *entraînement croisé* "*cross-training*".

1.8.2 SVM MULTI-LABEL BASES SUR LE CLASSEMENT, RANK-SVM ET SCRANK-SVM

Comme l'indiquent les auteurs de Rank-SVM dans [22], les classifieurs binaires ne sont pas la meilleure option lorsqu'il existe des corrélations entre les étiquettes, puisque l'indépendance totale entre eux est supposée par la plupart des méthodes binaires. Pour atténuer ce problème Rank-SVM, une approche directe basée sur les principes SVM, s'appuie sur une nouvelle métrique à minimiser pendant l'entrainement, AHL. C'est une approximation linéaire de la perte de Hamming Loss.

Une fois que le modèle de type SVM a été entraîné pour produire un classement des étiquettes, une fonction spécifique est utilisée pour ajuster le seuil de coupe pour déterminer le sous-ensemble d'étiquettes les plus pertinentes.

Bien que Rank-SVM prenne en compte les corrélations entre les étiquettes, ainsi, il devrait théoriquement devrait être plus performant que les modèles binaires purs, Les tests expérimentaux montrent que son comportement est similaire lorsqu'on travaille avec des données de haute dimension.

De fait, de nouveaux algorithmes, Rank-CVM et SCRank-SVM, ont été conçus pour améliorer l'efficacité et les performances de Rank-SVM. L'objectif de Rank-CVM [34] est de réduire la complexité de calcul de Rank-SVM. Pour ce faire, un CVM (core vector machine, machine vectorielle à noyau) a été utilisé au lieu d'un SVM. La solution analytique des CVM est immédiate, et beaucoup plus efficace que celle des SVM. Le résultat est un classificateur MLC avec une performance prédictive similaire à celle du Rank-SVM mais plus efficient.

L'objectif de SCRank-SVM [35], était aussi d'améliorer l'efficacité du Rank-SVM, ainsi que ses performances en tant que classificateur MLC. La proposition comprend la reformulation du calcul de la limite de décision dans les SVM sous-jacentes, simplifier certaines des contraintes existantes pour maximiser la marge. Dans le processus, les auteurs se débarrassent d'un des paramètres habituels des SVM, réduisant ainsi la complexité des calculs.

1.9 L'ALGORITHME ML-KNN

L'une des approches les plus simples de la classification est celle du kNN. Pour chaque nouvelle instance, le classificateur kNN recherche ses k-plus proches voisins. Pour ce faire, la distance (dans un espace de dimension f) entre les caractéristiques de la nouvelle instance et toutes les instances du jeu de données est calculée. Une fois que les instances les plus proches ont été déterminées, leurs classes sont utilisées pour prédire celle de la nouvelle instance.

Étant donné que kNN ne crée aucun modèle ; ce n'est que lorsqu'un nouvel exemple est proposé que le classifieur effectue son travail, il est généralement considéré comme une méthode paresseuse [36]. Il est également fréquemment appelé apprentissage basé sur l'instance (instance-based learning) [37].

ML-kNN [] est une adaptation de la méthode kNN au scénario multi-label. Contrairement à l'algorithme kNN classique, ML-kNN n'est pas si paresseux. Il commence par construire un modèle limité qui se compose de deux éléments d'information :

- Les probabilités a priori pour chaque étiquette. Il s'agit simplement du nombre de fois que chaque étiquette apparaît dans le MLD divisé par le nombre total d'instances. Un facteur de lissage est appliqué pour éviter de multiplier par zéro.
- Les probabilités conditionnelles pour chaque étiquette, calculée comme la proportion d'instances avec l'étiquette considérée dans les k plus proches voisins.

Ces probabilités sont calculées indépendamment pour chaque étiquette. Par conséquent, les dépendances potentielles entre les étiquettes sont entièrement ignorées (rejeter) par cet algorithme.

Après ce processus d'entrainement limité, le classificateur est en mesure de prédire les étiquettes pour les nouvelles instances. Lorsqu'un nouvel échantillon arrive, il passe par les étapes suivantes :

- Tout d'abord, les *K*-plus proches voisins de l'échantillon donné sont obtenus. Par défaut, la norme L2 − (distance euclidienne) est utilisée pour mesurer la similarité entre l'instance de référence et les échantillons dans le MLD.

- Ensuite, la présence de chaque étiquette chez les voisins est utilisée pour calculer les probabilités maximales a posteriori (MAP) à partir des probabilités conditionnelles obtenues avant.
- Enfin, l'ensemble d'étiquettes du nouvel échantillon est généré à partir des probabilités MAP. La probabilité elle-même est fournie comme niveau de confiance pour chaque étiquette, permettant ainsi de générer également un classement des labels.

1.10 CONCLUSION

A travers ce chapitre, nous avons présenté les différents types de la classification en générale. La seconde partie de ce chapitre a été consacré à la classification multi-label et ses domaines d'application ainsi que l'apprentissage à partir des donnée multi-label.

CHAPITRE 2 : RESEAUX DE NEURONES CONVOLUTIFS

1.11 INTRODUCTION

Dans ce chapitre nous présentons brièvement tous les aspects liés aux réseaux de neurones et leur utilisation. Les réseaux de neurones à convolution seront abordés avec plus de détails.

1.12 LES RÉSEAUX DE NEURONES

1.12.1 HISTOIRE

L'histoire des NN remonte à 1943, lorsque Warren Mcculloch et Walter Pitts ont conçu un modèle mathématique inspiré de la biologie du système nerveux central des mammifères [38].

Cela a inspiré l'invention du Perceptron, créé en 1958 par Frank Rosenblatt. Le perceptron utilise un modèle très simple imitant le neurone biologique qui était basé sur le modèle mathématique de Pitts et Mcculloch.

Au début, le Perceptron semblait très prometteur, mais on s'est vite aperçu qu'il avait de sérieuses limitations. La voix la plus importante de la critique était Marvin Minsky. Minsky a publié un livre dans lequel il démontrait que le modèle Perceptron était incapable de résoudre des problèmes complexes [39]. Entre autres, le livre contenait notamment la preuve mathématique que le Perceptron est incapable de résoudre le simple problème XOR. Plus généralement, le Perceptron n'est capable de résoudre que des problèmes linéairement séparables. Même si, selon Minsky, cette critique n'était pas malveillante, elle a en fait étouffé l'intérêt pour les NN pendant plus d'une décennie.

L'intérêt pour les NN a été ravivé au début des années 80, lorsqu'il a été démontré que toutes les déficiences précédemment soulevées auraient pu être résolues par l'utilisation de plusieurs d'unités de calcul. Par ailleurs, l'invention de l'algorithme d'apprentissage par rétro-propagation a permis de rassembler les neurones en groupes appelés couches qui peuvent être empilé dans

des structures hiérarchiques pour former un réseau. Les NN de ce type étaient communément appelés Perceptron multicouches (MLP).

Dans les années 80 et 90, l'intérêt pour les NN s'est progressivement étiolé et la recherche en IA s'est concentrée sur d'autres techniques d'apprentissage automatique, tels que les SVM et les méthodes ensemblistes. Dans la même période, plusieurs autres paradigmes de NN, qui s'inspiraient également de la biologie de certains aspects du système nerveux central et adoptant des approches différentes, ont également été développés. Les exemples les plus importants étaient les SOM et le réseau neuronal récurrent (RNN).

En l'an 2000, il y avait très peu de groupes de recherche qui accordaient suffisamment d'attention aux NN. Il y avait également un certain dédain pour les NN dans les milieux universitaires et de la recherche en IA. Toutefois, le succès des NN, promis il y a près d'un demi-siècle, s'est finalement concrétisé vers 2009, lorsque les premiers réseaux dotés d'un grand nombre de couches cachées ont été entraînés avec succès. Cela a conduit à l'adoption par le grand public du terme générique d'apprentissage profond, qui désigne généralement les réseaux neurones profonds (Deep Neural Network, DNN). Le mot "profond" indique que les réseaux comportent un grand nombre de couches cachées.

La principale idée était que pour apprendre des fonctions compliquées représentant des abstractions de haut niveau (Par exemple, la reconnaissance de la vision, la compréhension du langage, etc.) il est nécessaire d'avoir une architecture profonde.

Les NN, avant les DNN, n'avaient que 1 ou 2 couches cachées. Ces réseaux sont aujourd'hui souvent appelés réseaux peu profonds (shallow networks). Les réseaux profonds typiques peuvent avoir un nombre de couches cachées de l'ordre de dizaines, voire dans certains cas des centaines [40].

Même si la progression des réseaux neurones vers des structures comportant un nombre élevé de couches cachées était évidente, son entrainement a été un problème technique non résolu pendant très longtemps. Il y a trois raisons principales pour lesquelles cette percée n'a pas eu lieu plus tôt :

1. Il n'existait aucune technique permettant de faire évoluer le nombre de couches cachées.
2. Il n'y avait pas assez de données étiquetées nécessaires pour entraîner le NN.
3. Le matériel de calcul n'était pas assez puissant pour entraîner des réseaux suffisamment grands et complexes de manière efficace.

Le premier problème a été résolu par l'invention des CNN [41]. Le deuxième problème a été résolu simplement quand il y a eu plus de données disponibles. Cela principalement grâce aux efforts des grandes entreprises (*Google, Facebook, YouTube, etc*.) mais aussi avec l'aide d'une grande communauté de professionnels et de passionnés des sciences de données.

Pour résoudre le troisième problème, il fallait à la fois innover dans le matériel de calcul et améliorer les méthodes.

L'une des percées techniques a été l'utilisation d'unités de traitement graphique (Graphics Processing Units, GPUs) pour les calculs exigés par la construction d'un réseau complexe. Grâce au fait que le processus d'apprentissage des NN consiste généralement en un grand nombre de calculs simples, il existe un grand potentiel de parallélisation.

1.12.2 STRUCTURE DES RESEAUX NEURONAUX

Le terme NN est très général et décrit une large famille de modèles. Dans ce contexte, le NN est un modèle distribué et parallèle qui est capable d'approximer des fonctions non linéaires complexes. Le réseau est composé de plusieurs unités de calcul, appelées neurones, assemblées selon une topologie particulière. La construction d'un NN implique généralement le choix d'une topologie, d'une fonction de coût et d'une procédure d'optimisation.

1.12.3 MODELE DE NEURONE

Comme nous l'avons déjà mentionné, le modèle de neurones est inspiré de la biologie. Les premières tentatives de création d'un modèle de neurones comportaient plusieurs éléments équivalents aux neurones du cerveau humain. Au fur et à mesure que la recherche a progressé, cette équivalence a cessé d'être aussi importante et les modèles NN modernes ne correspondent que superficiellement à leurs homologues biologiques. Théoriquement, un neurone est une unité de calcul qui effectue une transformation non linéaire de ses entrées.

$$\boldsymbol{y = g(w^t + b)} \quad \textbf{Équation 1}$$

L'argument $w^t x + b$ de la fonction g est souvent considéré comme z. Par conséquent, l'équation peut être réécrite comme suit :

$$\boldsymbol{y = g(z)} \quad \textbf{Équation 2}$$

Un schéma typique est présenté à la figure 3.1, qui représente les entrées, ls poids et la fonction d'activation.

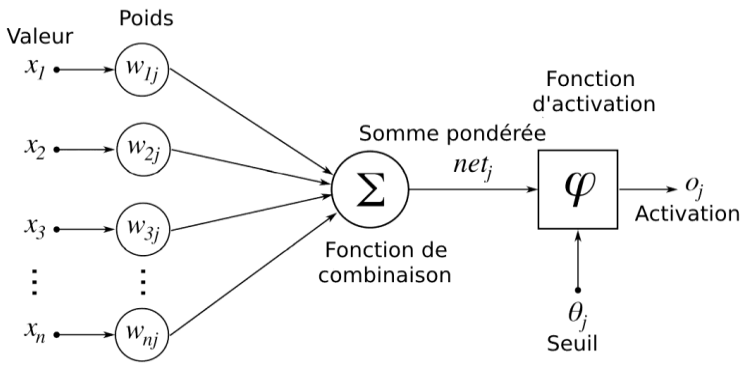


Figure 4 : Modèle de neurone artificiel

Entrées : Chaque neurone a plusieurs entrées x qui sont combinées ensemble pour exécuter une opération. Chaque entrée a un poids désigné qui lui est attribué.

Poids : Les entrées d'un neurone sont pondérées par des paramètres w qui sont modifiés pendant le processus d'apprentissage. Chaque poids donne de la force à chaque entrée individuelle dans le neurone. L'idée de base est que lorsque le poids est petit, l'entrée particulière n'influence pas beaucoup la sortie du neurone. Dans le cas contraire, son influence est importante.

Biais : Un autre paramètre modifiable est le biais b qui contrôle l'influence du neurone dans son ensemble.

1.12.4 FONCTION D'ACTIVATION

Pour que le NN puisse approximer une fonction non linéaire, chaque neurone doit effectuer une transformation non linéaire de son entrée. Ceci est fait avec la fonction d'activation $g(z)$ qui effectue une transformation non linéaire. Il existe plusieurs fonctions d'activation différentes couramment utilisées. Leur utilisation dépend du type de réseau et du type de couche dans laquelle elle est employée.

L'une des fonctions d'activation les plus anciennes et les plus utilisées est la fonction sigmoïde. Elle est définie par :

$$g(z) = \left(\frac{1}{1+e^{-z}}\right) \quad \textbf{Équation 3}$$

Le problème avec la sigmoïde est que son gradient devient vraiment plat sur les deux extrêmes, ce qui ralentit le processus d'apprentissage [42].

Une autre fonction d'activation est la tangente hyperbolique. Elle est définie comme suit :

$$g(z) = tanh - z \quad \textbf{Équation 4}$$

La fonction tangente hyperbolique est moins fréquente dans les réseaux NN à action directe, mais elle est largement utilisée dans les réseaux RNN.

Actuellement, la fonction d'activation la plus fréquemment utilisée est l'*unité linéaire restreinte* (Restricted Linear Unit, ReLU). Elle est définie comme suit :

$$g(z) = max\{0, z\} \quad \textbf{Équation 5}$$

Son inconvénient est qu'elle n'est pas différentiable pour $z = 0$, mais ce n'est pas un problème dans l'implémentation de l'algorithme d'apprentissage, d'un autre côté son plus grand avantage est qu'il aide à accélérer le processus d'apprentissage. Les trois fonctions d'activation sont illustrées dans la figure 5.

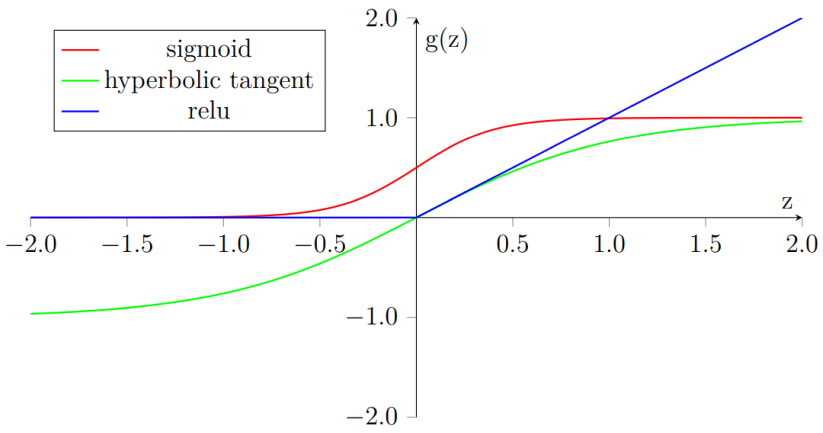


Figure 5 : Fonctions d'activation.

1.12.5 TOPOLOGIE DU RESEAU

Un autre aspect important dans l'apprentissage par réseaux de neurones et la topologie du réseau. Il existe plusieurs topologies différentes couramment utilisées. Les deux plus couramment utilisés dans l'apprentissage profond sont le feed-forward et le récurrent. Les réseaux de type feed-forward sont caractérisés par le fait que lors de l'activation l'information ne circule que dans le sens direct, des entrées vers la sortie.

Un autre critère de topologie est la manière dont les neurones individuels du réseau sont connectés. Le plus souvent, les NNs sont ordonnés en couches. Dans chaque couche, il peut y avoir de 1 à n neurones. Les couches sont empilées de manière hiérarchique. Dans la terminologie typique, la première couche est appelée couche d'entrée, la dernière couche est appelée couche de sortie et les couches intermédiaires sont appelées couches cachées.

La description du réseau repose sur les interconnexions entre les différentes couches. Le schéma le plus courant est le schéma entièrement connecté "fully connected" dans lequel chaque neurone de la couche cachée l est connecté à tous les neurones de la couche précédente l - 1 et sa sortie est connectée à de chaque neurone de la couche l + 1 suivante. La structure entière est illustrée à la figure 3.

Les types de neurones dépendent du type de couche. Actuellement, la principale différence réside dans leur fonction d'activation, ce qui n'a pas été le cas pendant longtemps. Historiquement, toutes les couches avaient des neurones avec une fonction d'activation sigmoïde. C'est principalement parce que la couche sigmoïde de sortie peut être facilement mise en correspondance avec une distribution de probabilité, puisqu'elle acquiert des valeurs entre 0 et 1.

Ce n'est que relativement récemment (Au cours de la dernière décennie) que l'on a découvert que les réseaux composés de neurones avec une fonction d'activation ReLU dans les couches cachées peuvent être entraînés très rapidement et sont plus résistants au sur-apprentissage.

Les neurones de la couche de sortie ont besoin d'une sortie qui puisse produire une distribution de probabilité, qui peut être utilisé pour estimer la probabilité des classes individuelles.

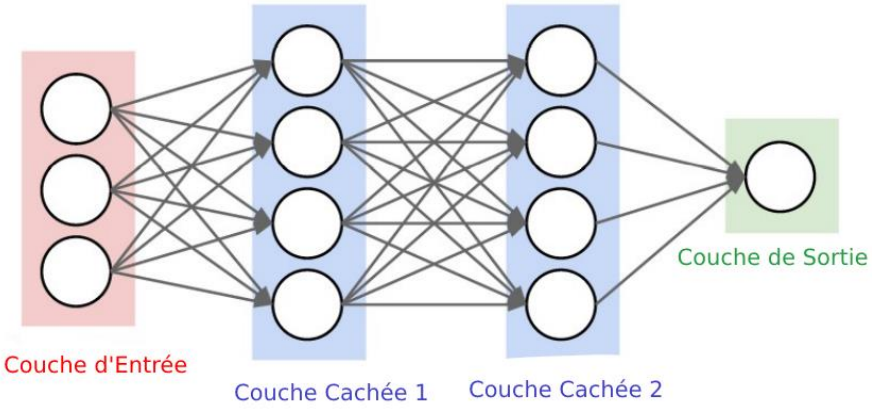


Figure 6 : Réseau de neurones Feed Forward entièrement connecté

Pour cette raison, la fonction d'activation la plus couramment utilisée pour la couche de sortie est appelée softmax. Softmax est une fonction exponentielle normalisée. Elle est utilisée pour représenter la probabilité qu'une instance soit membre de la classe j.

$$g(z) = ezj \sum k = 1 \, kezk \quad \textbf{Équation 6}$$

Où K est le nombre total des classes.

1.12.6 FONCTION DE COUT

La fonction de coût des NNs est un sujet très complexe. L'une des fonctions de coût les plus courantes dans les réseaux de neurones pour la classification multi-classes est *l'entropie croisée catégorielle*. Pour la fonction d'activation softmax de l'équation 6 la fonction de coût est définie comme suit :

$$C = -1n \sum ni = l \, y(i) lng(z(i)) + (1 - y(i)) ln\left(1 - g(z(i))\right) \quad \textbf{Équation 7}$$

Où y (i) si la classe de l'instance est correcte et n est le nombre total d'instances.

1.12.7 PROCEDURE D'OPTIMISATION

Chaque procédure d'optimisation pour les NN est basée sur la descente de gradient. En d'autres termes, il s'agit d'un processus itératif qui vise à réduire l'erreur d'apprentissage du réseau en différenciant la fonction de coût et en ajustant les paramètres du modèle en suivant le gradient négatif.

Le problème est que la fonction de coût de l'ensemble du réseau est très complexe et comporte de nombreux paramètres. La technique utilisée pour résoudre ce problème s'appelle la rétro-propagation.

1.13 RESEAUX NEURONES CONVOLUTIFS

Les RNC (en anglais ***CNN*** ou ***ConvNet*** pour *Convolutional Neural Networks*) sont un type spécialisé de RN qui était à l'origine utilisé dans les applications de traitement d'images. Ce sont incontestablement les modèles les plus réussis en IA inspirés de la biologie. Même s'ils étaient guidés par de nombreux domaines différents, les principes de conception de base ont été tirés des neurosciences. Depuis leur succès dans le traitement d'images, ils ont également été déployés avec beaucoup de succès dans les applications de traitement du langage naturel et de la vidéo.

L'inspiration susmentionnée en biologie était basée sur les travaux scientifiques des neurophysiologistes David Hubel et Torsten Wiesel. Ces derniers ont étudié le système de vision des mammifères depuis la fin des années 1950. Dans l'expérience, qui pourrait être considérée comme épouvantable pour les normes d'aujourd'hui, ils ont connecté des électrodes dans le cerveau d'un chat anesthésié et mesuré la réponse du cerveau aux stimuli visuels [43]. Ils ont découvert que la réaction des neurones du cortex visuel était déclenchée par une ligne de lumière très étroite qui brillait sous un angle spécifique sur l'écran de projection pour que le chat puisse le voir. Ils ont établi que les neurones individuels du cortex visuel réagissent uniquement à des modèles très spécifiques dans l'image d'entrée. Hubel et Wiesel ont reçu le prix Nobel de physiologie et médecine en 1981 pour leur découverte.

Dans ce qui suit, on suppose que la couche convolutive fonctionne avec des données d'entrée rectangulaires (*e.g.* des images). Même si les réseaux convolutifs peuvent également être utilisés pour classer une entrée unidimensionnelle (*e.g.* un signal sonore) ou tridimensionnelle (par exemple des tomodensitogrammes).

1.13.1 STRUCTURE DU CNN

La structure des réseaux convolutifs est généralement composée de trois types de couches différents. La couche peut être convolutive, en pool ou entièrement connectée. Chaque

type de couche a des règles différentes pour la propagation du signal d'erreur en avant et en arrière.

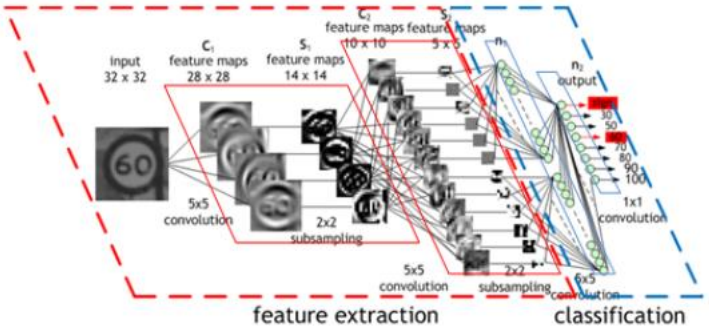


Figure 7 : Structure typique d'un réseau neuronal convolutif

Il n'existe pas de règles précises déterminant la façon dont la structure des couches individuelles doit être organisée. Cependant, les CNN sont typiquement structurés en deux parties. La première partie, généralement appelée extraction de caractéristiques,

Utilise des combinaisons de couches convolutives et de pooling.La deuxième partie appelée classification utilise des couches entièrement connectées. Ceci est illustré dans la figure 7.

1.13.2 COUCHE DE CONVOLUTION

Comme son nom l'indique, cette couche utilise une opération de convolution. L'opération de convolution est effectuée sur l'entrée avec un filtre spécifique, qui s'appelle **noyau**. La sortie de l'opération de convolution est généralement appelée carte de caractéristiques.

L'entrée de la couche convolutive est soit une image (dans le cas de la première couche réseau) soit une carte des caractéristiques issue de la couche précédente. Le noyau est généralement de forme carrée et sa largeur peut aller de 3 à N pixels. La carte des caractéristiques est créée par convolution du noyau sur chaque élément d'entrée spécifié.

Le processus de convolution peut produire une carte de caractéristiques de taille différente de celle de l'entrée. Lorsque la taille de la sortie doit être préservée, il est nécessaire d'employer un remplissage nul sur les bords d'entrée. C'est à dire qu'on doit ajouter des zéros autour des bords la matrice d'entrée. Le nombre des zéros à ajouter est déterminé par :

$$p = \left(\frac{(h-6)}{2}\right) \quad \textbf{Équation 8}$$

Où *h* est la largeur du noyau utilisé. La réduction de la taille de la carte des caractéristiques peut être dans certains cas souhaitable. Le remplissage (rembourrage nul) est illustré sur la figure 8.

La réduction de la carte de caractéristiques peut aller encore plus loin en cas d'utilisation de la foulée (stride). L'application de la foulée spécifie le nombre de points d'entrée traversés lors du déplacement vers la position voisine à chaque étape. Lorsque la foulée est de 1, le noyau est déplacé de 1 à chaque pas et la taille résultante de la carte des caractéristiques n'est pas affectée.

Chaque couche convolutive est typiquement composée de divers noyaux. Autrement dit, la sortie de cette couche est un tenseur contenant une carte de caractéristiques pour chaque noyau utilisé. Chacun d'eux est conçu pour souligner différentes caractéristiques de l'image d'entrée. Dans les premières couches, ces caractéristiques sont généralement des bords. Plus la couche est élevée, plus des entités complexes sont capturées. La fonction d'activation des CNN est généralement la fonction ReLU (unité linéaire rectifiée).

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

Figure 8 : Une matrice 4x4 avec un remplissage à zéro

1.13.3 COUCHE DE POOLING

Cette couche ne constitue généralement pas un processus d'apprentissage, mais elle est utilisée pour sous-échantillonner la taille de l'entrée. Le principe est que l'entrée est divisée en plusieurs éléments rectangulaires qui ne se chevauchent pas et que les unités de chaque élément sont utilisées pour créer une seule unité de sortie. Cela diminue la taille de la couche de sortie tout en préservant les informations les plus importantes contenues dans la couche d'entrée. En d'autres termes, cette couche permet de compresser les informations contenues dans l'entrée.

Il existe plusieurs types d'opération qui peuvent être utilisés dans cette couche. Ces opérations peuvent être la moyenne, la valeur maximale ou alternativement une combinaison linéaire. La combinaison linéaire introduit une forme d'apprentissage dans la couche, mais elle n'est pas très répandue.

La sélection de la valeur maximale est le type d'opération le plus courant et dans ce cas la couche est appelée Max-Pooling. Le principe du Max-Pooling est illustré sur la Figure 9.

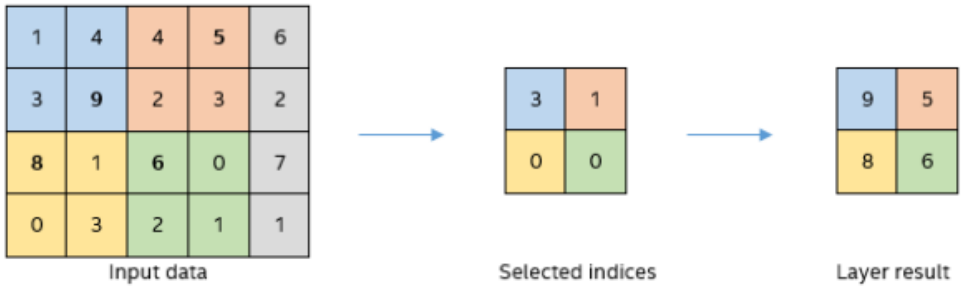


Figure 9 : Principe du Max-pooling

1.13.4 COUCHE ENTIEREMENT CONNECTEE

Après la couche de pooling vient la couche entièrement connectée (Fully-Connected layer), c'est cette couche qui permet à un CNN de classifier les instances.

1.14 ENTRAINEMENT DES CNN

Le processus d'optimisation de CNN est presque analogue à celui d'un réseau complètement connectée. Mais avec les CNN ce processus est un peu plus compliqué car le réseau est composé de différents types de couches. La propagation du signal vers l'avant et la propagation des erreurs vers l'arrière suivent des règles spéciales pour chaque couche.

La première phase est appelée propagation vers l'avant, où le signal est propagé des entrées des CNN à sa sortie. Dans la dernière couche, la sortie est comparée à la valeur souhaitée en utilisant la fonction de coût et l'erreur est estimée. Dans la deuxième phase, un algorithme de rétropropagation est à nouveau utilisé pour estimer la contribution aux erreurs des unités individuelles. Les paramètres variables du réseau sont à nouveau optimisés par un algorithme de descente de gradient.

1.15 AUTRES TYPES DE RESEAUX

1.15.1 RESEAU ANTAGONISTE GENERATIF

Un GAN ou Generative Adversarial Network (réseau antagoniste génératif) est une technique d'intelligence artificielle permettant de créer des imitations parfaites d'images ou autres données.

***Un GAN ou Generative Adversarial Network (réseau antagoniste génératif en français) est* une technique de Machine Learning. *Elle repose sur la mise en compétition de deux réseaux au sein d'un framework Ces** deux réseaux sont appelés " générateur " et " discriminateur ".

Le générateur est un type de réseau neuronal convolutif dont le rôle est de créer de nouvelles instances d'un objet. De son côté, le discriminateur est un réseau neuronal " déconvolutif " qui détermine l'authenticité de l'objet ou s'il fait ou non partie d'un ensemble de données.

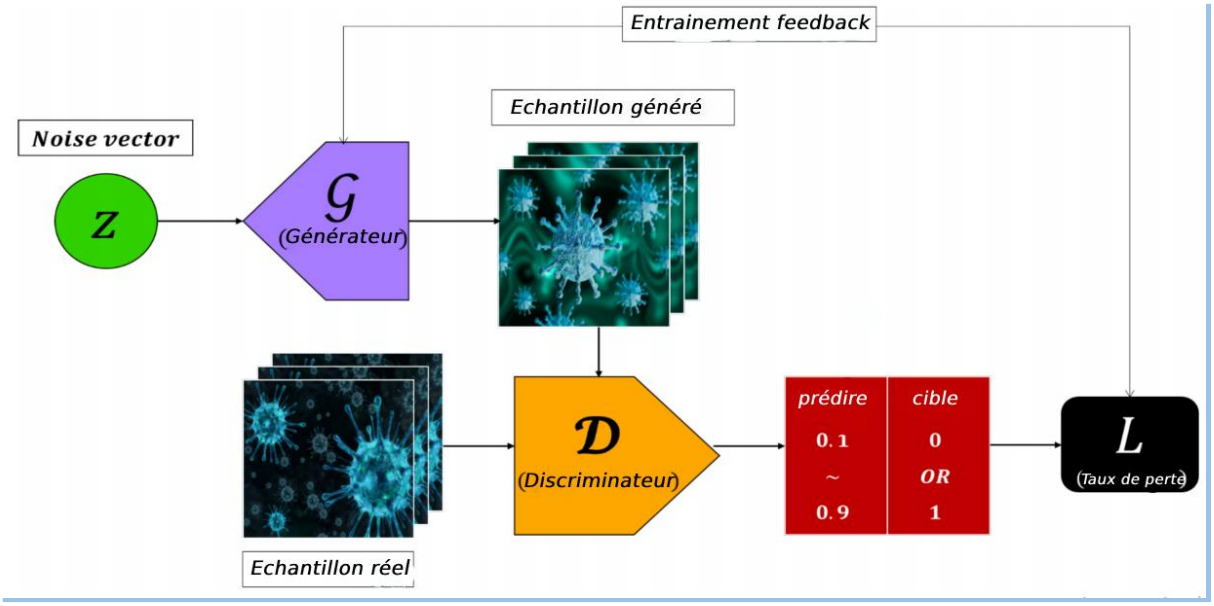


Figure 10 : Algorithme d'apprentissage des GAN (Generative Adversarial Networks).

Pendant le processus d'entraînement, **ces deux entités sont en compétition et c'est ce qui leur permet d'améliorer leurs comportements respectifs**. C'est ce que l'on appelle la rétro-propagation.

L'objectif du générateur est de produire des outputs sans que l'on puisse déterminer s'ils sont faux, tandis que l'objectif du discriminateur est d'identifier les faux. Ainsi, au fil du processus, le générateur produit des outputs de meilleure qualité tandis que le discriminateur détecte de mieux ne mieux les faux. De fait, **l'illusion est de plus en plus convaincante au fil du temps**.

Dans un premier temps, **il convient de déterminer ce que l'on souhaite que le GAN produise (l'output)**.

La seconde étape consiste à composer un ensemble de données basé sur ces paramètres. Ces données sont ensuite entrées dans le générateur jusqu'à ce qu'il commence à produire des outputs convaincants.

Les **images générées sont ensuite transmises au discriminateur**, aux côtés des véritables points de données. Le discriminateur filtre les informations, et établit une probabilité comprise entre 0 et 1 pour déterminer l'authenticité de l'image. Le chiffre 1 signifie que l'image est réelle, le 0 indique qu'il s'agit d'un faux. Ces valeurs sont ensuite vérifiées manuellement, et le processus est répété jusqu'à ce que le résultat souhaité soit atteint.

1.15.2 LSTM (Long Short-Term Memory)

Les réseaux LSTM (Long Short-Term Memory) sont une version modifiée des réseaux de neurones récurrents facilitant la mémorisation des données passées. Les LSTM sont bien adaptés à la classification et le traitement des séries temporelles. Il entraîne le modèle en utilisant la propagation arrière. Dans un réseau LSTM, trois portes sont présentes :

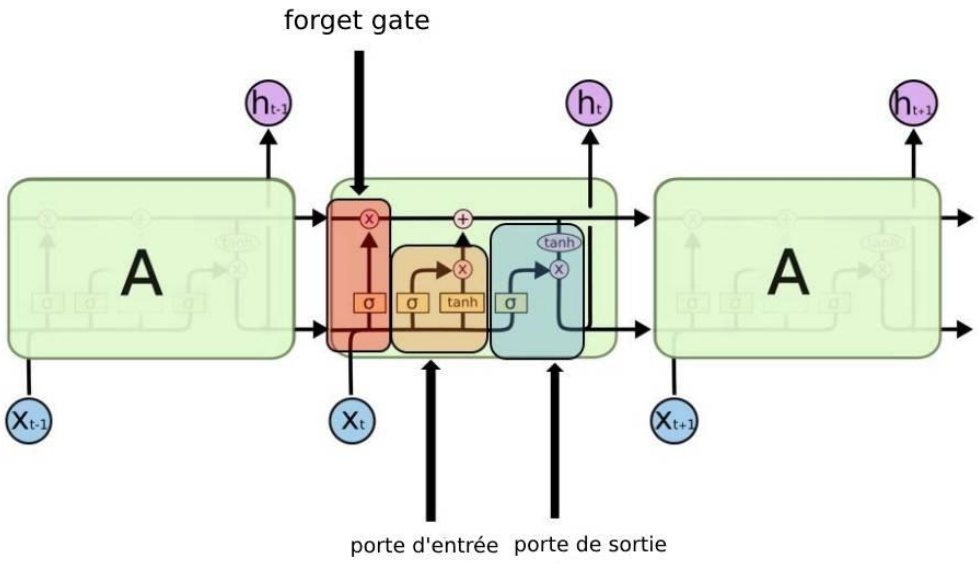


Figure 11 : Portes LSTM

1. **Porte d'entrée** : Découvrir quelle valeur de l'entrée doit être utilisée pour modifier la mémoire. La fonction **sigmoïde** décide des valeurs à laisser passer 0,1. La fonction **tanh** donne un poids aux valeurs qui sont passées en fonction de leur niveau d'importance allant de -1 à 1.

$$i_t = \delta(W_i[h_{t-1}, x_t] + b_i) \quad \textbf{Équation 9}$$

$$C_t = tan(W_i[h_{t-1}, x_t] + b_c) \quad \textbf{Équation 10}$$

2. **Porte d'oubli :** Permet de découvrir les détails à écarter du bloc. Elle est déterminée par la fonction **sigmoïde**. Elle examine l'état précédent (h_{t-1}) et le contenu de l'entrée (X_t) et génère un nombre compris entre 0 (omettre ceci) et 1(garder ceci) pour chaque nombre dans l'état de la cellule C_{t-1}.

$$f_t = \delta(W_f[h_{t-1}, x_t] + b_f) \quad \textbf{Équation 11}$$

3. **Porte de sortie :** l'entrée et la mémoire du bloc est utilisée pour décider de la sortie. La fonction **sigmoïde** décide des valeurs à laisser passer 0,1. La fonction tanh donne un poids aux valeurs qui sont passées en décidant de leur niveau d'importance allant de -1 à 1 et sont multipliées par la sortie de **Sigmoid**.

$$O_t = \delta(W_o[h_{t-1}, x_t] + b_o) \quad \textbf{Équation 12}$$

$$h_t = O_t * tanhC_t \quad \textbf{Équation 13}$$

1.16 CONCLUSION

Dans ce chapitre nous avons présenté globalement les réseaux de neurone, ensuite on a détaillé la structure et les différentes couches et la méthode d'entrainement pour les réseaux à convolution.

2CHAPITRE : IMPLEMENTATION

2.1 INTRODUCTION

Cette partie consiste à présenter le dernier volet de ce rapport qui sera consacré à décrire en détail notre protocole d'expérimentation ainsi que les différents outils et technologies employés.

2.2 PROBLEME

Dans la classification d'images multi-label on utilise une affiche de films, principalement hollywoodiens, et un réseau de neurones convolutifs, par exemple l'affiche du film le joker ci-dessous va être soumise à notre réseau connaitre son genre.

Figure 12 :Affiche du Film Joker

Ce film entre dans les catégories :

- Drame
- Policier
- Thriller

L’affiche suivante aussi peut être associée à plusieurs genres

Figure 13 : Affiche de film Iron Man

- Action
- Aventure
- Science-fiction

Et de la même manière les autres films aussi, par exemple si nous prenons le film The Avengers

Figure 14 : Affiche de film Avengers

Il est dans les catégories :

- Aventure
- Action
- Science-fiction

A cet effet nous avons choisi quelques affiches pour les tester, pour cela nous allons former notre propre modèle en se basant sur des affiches d’une collection de films hollywoodiens réalisé entre 1980 à 2015. Dans la mesure où chaque film peut être associée à plusieurs catégories simultanément, la tâche de catégorisation de films à partir de posters est considérée comme un classification multi-label.

Puisque la majorité des affiches contiennent plusieurs caractéristique visuel, la classification multi-label s’impose, si en prend les images ci-dessous comme référence on s’aperçois que

l'image qui contient à la fois un chat et un oiseau ne peut être classe que par la classification multi-label.

2.3 PLATE-FORME UTILISER (GOOGLE COLABORATORY)

Google Colaboratory (abréviation Colab) est un service en nuage basé sur Jupyter Notebooks pour la diffusion de l'enseignement et de la recherche sur l'apprentissage automatique.

Avant de présenter Google Colaboratory, nous présentons d'abord Jupyter Notebooks, la technologie sur laquelle repose Colaboratory. Jupyter est un outil open-source basé sur un navigateur qui intègre des langages interprétés, des bibliothèques et outils de visualisation []. Un Jupyter notebook peut fonctionner en local ou dans le cloud. Chaque document est composé de plusieurs cellules, où chaque cellule contient un langage de script ou un code de démarquage, la sortie est intégrée dans le document. Les sorties typiques comprennent du texte, des tableaux, des diagrammes et des graphiques. L'utilisation de cette technologie facilite le partage et la reproduction des travaux scientifiques puisque les expériences et les résultats sont présentés de manière autonome [11]. Google Colaboratory (abréviation Colab) est un projet qui a pour objectif de diffuser l'enseignement et la recherche sur l'apprentissage automatique [6].

Colaboratory notebooks est basés sur Jupyter et fonctionnent comme un objet Google Docs: il peut être partagé et les utilisateurs peuvent collaborer sur le même notebook. Colaboratory fournit des moteurs d'exécution Python 2 et 3 préconfigurés avec les bibliothèques essentielles d'apprentissage automatique et d'intelligence artificielle, telles que TensorFlow, Matplotlib et Keras. La machine virtuelle fonctionne sous le runtime (VM) qui se désactive après un certain temps, toutes les données et configurations de l'utilisateur seront perdues.

Alors que, le notebook sera conservé, il est également possible de transférer des fichiers du disque dur de la VM vers le compte Google Drive de l'utilisateur. Enfin, ce service de Google fournit un temps d'exécution accéléré par le GPU, qui est entièrement configuré avec les logiciels précédemment décrits. L'infrastructure de Google Colaboratory est hébergée sur la plateforme Google Cloud.

2.4 BIBLIOTHEQUES UTILISEES

2.4.1 TENSORFLOW

L'outil TensorFlow est une bibliothèque de logiciels open source publiée en 2015 par Google pour permettre aux développeurs de concevoir, créer et former plus facilement des modèles d'apprentissage en profondeur. TensorFlow est à l'origine une bibliothèque interne que les développeurs de Google utilisaient pour créer des modèles en interne, et nous nous attendons à ce que des fonctionnalités supplémentaires soient ajoutées à la version open source au fur et à mesure qu'elle est testée et vérifiée dans la version interne. Bien que TensorFlow ne soit qu'une des nombreuses options disponibles pour les développeurs, nous avons choisi de l'utiliser ici en raison de sa conception réfléchie et de sa facilité d'utilisation. Nous comparerons brièvement TensorFlow aux alternatives dans la section suivante.

À un niveau élevé, TensorFlow est une bibliothèque Python qui permet aux utilisateurs d'exprimer des calculs arbitraires sous forme de graphique de flux de données. Les nœuds de ce graphe représentent des opérations mathématiques, tandis que les arêtes représentent des données qui sont communiquées d'un nœud à un autre. Les données dans TensorFlow sont représentées sous forme de tenseurs, qui sont des tableaux multidimensionnels (représentant des vecteurs avec un tenseur 1D, des matrices avec un tenseur 2D, etc.). Bien que ce cadre de réflexion sur le calcul soit précieux dans de nombreux domaines différents, TensorFlow est principalement utilisé pour l'apprentissage en profondeur dans la pratique et la recherche.

Penser aux réseaux de neurones en tant que tenseurs et vice versa n'est pas anodin, mais plutôt une compétence que nous développerons au cours de ce texte. Représenter les réseaux de neurones profonds de cette manière nous permet de profiter des accélérations offertes par le matériel moderne (c'est-à-dire l'accélération GPU des opérations de tenseur parallèles) et nous fournit une méthode propre, mais expressive, pour la mise en œuvre des modèles.

2.4.2 KERAS

Keras est un framework (cadre) d'apprentissage en profondeur pour Python qui fournit un moyen pratique de définir et d'entraîner presque tout type de modèle d'apprentissage en

profondeur. Keras a été initialement développé pour les chercheurs, dans le but de permettre une expérimentation rapide. Keras a les caractéristiques clés suivantes :

- Il permet au même code de s'exécuter de manière transparente sur le UCT (Unité Centrale de Traitement) ou le PG (Processeur Graphique).
- Il dispose d'une IP (interface de programmation) conviviale qui facilite le prototype rapide de modèles d'apprentissage en profondeur.

- Il a un support intégré pour les réseaux convolutifs (pour la vision par ordinateur), les réseaux récurrents. (Pour le traitement des séquences) et toute combinaison des deux.

- Il prend en charge des architectures réseau arbitraires : modèles multi-entrées ou multi-sorties, partage de couches, partage de modèles, etc. Cela signifie que Keras est approprié pour construire essentiellement n'importe quel modèle d'apprentissage en profondeur, d'un réseau contradictoire génératif à une machine de Turing neuronale.

Keras compte plus de 200 000 utilisateurs, allant des chercheurs universitaires et des ingénieurs de startups et de grandes entreprises aux étudiants diplômés et aux amateurs. Keras est utilisé chez Google, Netflix, Uber, CERN, Yelp, Square et des centaines de startups travaillant sur un large éventail de problèmes. Keras est également un framework populaire sur Kaggle, le site Web du concours d'apprentissage automatique, où presque tous les récents concours d'apprentissage en profondeur ont été remportés à l'aide de modèles Keras.

2.4.3 NumPy

NumPy, abréviation de Python Numérique, est depuis longtemps la pierre angulaire des calculs numériques en Python. Il fournit les structures de données, les algorithmes, et la colle de bibliothèque nécessaire à la plupart des applications scientifiques impliquant des données numériques en Python. NumPy contient, entre autres :

- Un objet tableau multidimensionnel rapide et efficace *ndarray*.

- Fonctions permettant d'effectuer des calculs par éléments avec des tableaux ou des opérations mathématiques entre tableaux.
- Outils de lecture et d'écriture sur disque d'ensembles de données basés sur des tableaux.
- Opérations d'algèbre linéaire, Transformée de Fourier, et génération de nombres aléatoires.
- Une C API mature permettant aux extensions Python et au code C ou C++ natif d'accéder aux structures de données de NumPy et aux outils de calcul de NumPy.

Au-delà des capacités de traitement rapide des tableaux que NumPy ajoute à Python, l'une de ses principales utilisations dans l'analyse des données est de servir de conteneur pour les données à transmettre entre les algorithmes et les bibliothèques, Pour les données numériques, les tableaux NumPy sont plus efficaces pour le stockage et la manipulation des données que les autres structures de données intégrées à Python. De même, les bibliothèques écrites dans un langage de niveau inférieur, comme le C ou le Fortran, peut opérer sur les données stockées dans un tableau NumPy sans copier les données dans une autre représentation mémoire. Ainsi, de nombreux outils de calcul numérique pour Python supposent que les tableaux NumPy constituent la principale structure de données ou visent une interopérabilité transparente avec NumPy.

2.4.4 PANDAS

Pandas fournit des structures de données de haut niveau et des fonctions conçues pour faciliter le travail avec des données structurées ou tabulaires, Facile, et expressif. Depuis son émergence en 2010, il a contribué à faire de Python un environnement d'analyse de données puissant et productif.

Pandas et capable de mélanger les hautes performances, les calculs matriciel de NumPy avec les capacités flexibles de manipulation des données des feuilles de calcul et des bases de données relationnelles (comme SQL). Il offre une fonctionnalité d'indexation sophistiquée permettant de remodeler facilement, le découpage, l'agrégation et la sélection de sous-ensembles des données.

2.4.5 PYPLOT

Pyplot est un module Matplotlib qui fournit une interface de type MATLAB. Matplotlib est conçu pour être aussi utilisable comme MATLAB. Chaque fonction pyplot apporte des modifications à une figure par exemple : elle crée une figure, une zone de traçage dans une figure, trace des lignes dans une zone de traçage, tracé d'étiquettes, etc. Les différents graphiques que nous pouvons utiliser à l'aide de Pyplot sont le graphique linéaire, l'histogramme, le graphique de dispersion, le graphique 3D, l'image, le contour et le graphique polaire.

2.4.6 TRAIN_TEST_SPLIT

`train_test_split` est une fonction dans la sélection du modèle Sklearn pour diviser les tableaux de données en deux sous-ensembles : pour les données d'entraînement et pour les données de test. Avec cette fonction, nous n'avons pas besoin de diviser l'ensemble de données manuellement.

Par défaut, Sklearn **train_test_split** fera des partitions aléatoires pour les deux sous-ensembles. Toutefois, ont pout également spécifier un état aléatoire pour l'opération.

2.4.7 TQDM

tqdm est une bibliothèque Python qui nous permet d'afficher une barre de progression intelligente en entourant n'importe quel itérable. Une barre de progression tqdm nous indique non seulement le temps écoulé, mais aussi le temps estimé restant pour l'itérable.

2.4.8 IMPORTS DES BIBOTHEQUE

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization, Conv2D, MaxPool2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
print(tf.__version__)
```

Figure 15 : Importation des bibliothèque utiliser

```
2.5.0
```

Figure 16 : Version de tensorfliw utilisée

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tqdm import tqdm
```

Figure 17 : Importation des bibliothèques utilisées

2.5 TRAITEMENT DE DONNEES

2.5.1 PRESENTATION

L'ensemble des données utilisé a été collecté sur le site IMDB. Les images d'affiches employés ont été recueillis pour des films hollywoodiens (principalement) sortis entre 1980 et 2015. Chaque image d'affiche est associé à un film ainsi qu'à certaines métadonnées comme l'ID et les genres. L'ID de chaque image est défini comme son nom de fichier

2.5.2 VISUALISATION DES DONNEES

Nous lisons l'ensemble de données dans un dataframe **pandas** en utilisant **pd.read_csv()**. Le jeu de données contient 7254 lignes et 27 colonnes.

```
data = pd.read_csv('/content/Movies-Poster_Dataset/train.csv')
data.shape
```

```
(7254, 27)
```

Figure 18 : code pour lire les donnée

data.head() montre les 5 premières lignes de l'ensemble de données, le jeu de données contient l'**Id** de l'image. Les images sont stockées dans un dossier séparé avec l'Id comme nom du fichier image.

```
data.head()
```

Figure 19 ; Afficher 4 premier ligne

	Id	Genre	Action	Adventure	Animation	Biography	Comedy	Crime	Documentary	Drama	Family	Fantasy
0	tt0086425	['Comedy', 'Drama']	0	0	0	0	1	0	0	1	0	0
1	tt0085549	['Drama', 'Romance', 'Music']	0	0	0	0	0	0	0	1	0	0
2	tt0086465	['Comedy']	0	0	0	0	1	0	0	0	0	0
3	tt0086567	['Sci-Fi', 'Thriller']	0	0	0	0	0	0	0	0	0	0
4	tt0086034	['Action', 'Adventure', 'Thriller']	1	1	0	0	0	0	0	0	0	0

History	Horror	Music	Musical	Mystery	N/A	News	Reality-TV	Romance	Sci-Fi	Short	Sport	Thriller	War	Western
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 20 : les 4 premier ligne

2.5.3 NORMALISATION DU JEU DE DONNEE

Les images de nos données sont de tailles différentes. Pour les traiter, nous devons les convertir à une taille fixe. **tqdm()** affiche une barre de progression pendant le chargement des images. Nous allons bouclé chaque image en utilisant son chemin. Nous allons convertir chaque image à une taille fixe de 350×350. Les valeurs des images sont comprises entre 0 et 255. Les réseaux de neurones fonctionnent bien avec des valeurs comprises entre 0 et 1. Nous allons donc normaliser les valeurs en divisant toutes les valeurs par 255.

```
img_width = 350
img_height = 350

X = []
for i in tqdm(range(data.shape[0])):
  path = '/content/Movies-Poster_Dataset/Images/' + data['Id'][i] + '.jpg'
  img = image.load_img(path, target_size=(img_width, img_height, 3))
  img = image.img_to_array(img)
  img = img/255.0
  X.append(img)

X = np.array(X)
X.shape
```

Figure 21 : Code de normalisation de jeu de donnée

```
100%|██████████| 7254/7254 [00:24<00:00, 293.11it/s]
```

Figure 22 : Résultat du normalisation de jeu de donnée

X est un tableau NumPy qui contient 7254 images. Chaque image a une taille de 350×350 tridimensionnelle car il s'agit d'une image colorée.

```
X = np.array(X)
X.shape
```

```
(7254, 350, 350, 3)
```

Figure 23 : le tableau X

A présent nous allons voir une image de X.

```
plt.imshow(X[6])
```

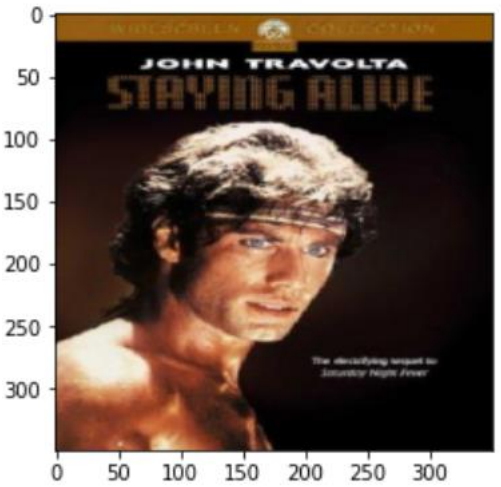


Figure 24 : Une affiche de film

Nous pouvons obtenir le Genre de l'image ci-dessus à partir des données.

```
data['Genre'][6]
```

```
'['Drama', 'Music', 'Romance']'
```

Figure 25 : Le genre de l'affiche

2.5.4 AFFICHAGE ALEATOIRE DE PLUSIEURS AFICHES

Nous avons pris aléatoirement de notre jeu de donnée quelque affiches avec leur genre.

```
fig=plt.figure(figsize=(15, 15))

columns = 4
rows = 4
for i in range(1, columns*rows +1):
    index = np.random.randint(len(X))
    img = X[index]
    fig.add_subplot(rows, columns, i)
    plt.title(data['Genre'][index])

    plt.imshow(img)
plt.show()
```

Figure 26 : Code pour afficher aléatoirement des affiches des films

Figure 27 : Des affiche des films afficher aléatoirement

2.6 CREATION DU MODELE

2.6.1 PREPARATION DES DONNEE

Pour préparer le jeu de données, Nous devons disposer d'un espace caractéristique et d'une cible. L'espace caractéristique nous l'avant déjà obtenu **X** ; Et afin d'obtenir la cible en **y**, nous devons supprimer les colonnes Id et Genre des données.

```
y = data.drop(['Id', 'Genre'], axis = 1)
y = y.to_numpy()
y.shape
```

```
(7254, 25)
```

Figure 28 : Préparation des donnée

2.6.2 ENTRAINEMENT ET TEST

Pour réaliser cette étape, les données seront divisées en un ensemble d'entraînement et un ensemble de test à l'aide de **train_test_split(). test_size = 0.15.** On conservera 15% des données pour les tests et 85% des données seront utilisées pour l'entraînement du modèle.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0, test_size = 0.15)
```

Figure 29 : code pour deviser les donnée

C'est la forme d'entrée que nous allons passer comme paramètre à la première couche de notre modèle.

```
X_train[0].shape
```

```
(350, 350, 3)
```

Figure 30 : La forme d'entre

2.6.3 ETAPE CONSTRUCTION DU CNN

Un modèle **Sequential()** est approprié pour une pile de couches simple où chaque couche a exactement un tenseur d'entrée et un tenseur de sortie.

Conv2D est une couche de convolution 2D, cette couche crée un noyau de convolution qui est enroulé avec les couches d'entrée, ce qui aide à produire un tenseur de sorties. Dans la première couche **Conv2D()** nous apprenons un total de 16 filtres avec la taille de la fenêtre convolutionnelle comme 3×3.

Le paramètre **input_shape** spécifie la forme de l'entrée. C'est un paramètre nécessaire pour la première couche de tout réseau neuronal. Nous allons utiliser la fonction d'activation **ReLu**. La fonction d'activation linéaire rectifiée, ou ReLU en abrégé, est une fonction linéaire par morceaux qui produit directement l'entrée si elle est positive, sinon, elle produit zéro.

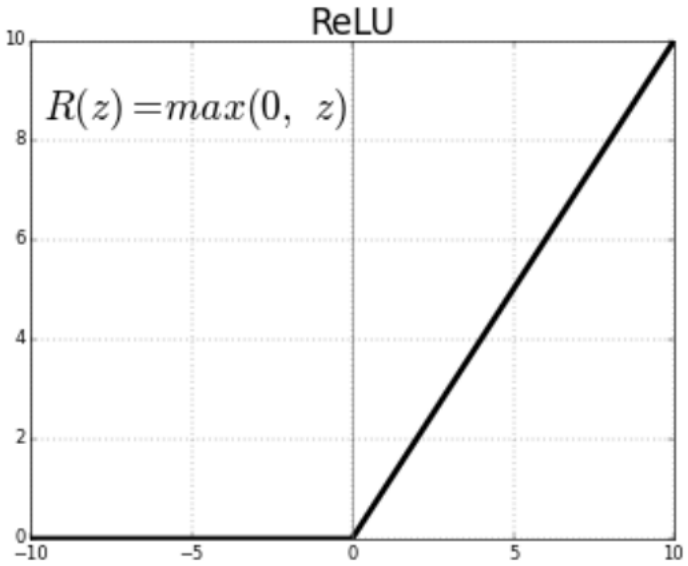


Figure 31 : Fonction ReLu

BatchNormalization() permet à chaque couche d'un réseau d'apprendre par elle-même un peut plus indépendamment des autres couches ; Pour augmenter la stabilité d'un réseau neuronal, la normalisation par lot normalise la sortie d'une couche d'activation précédente en soustrayant la moyenne du lot et en la divisant par l'écart-type du lot. Il applique une transformation qui maintient la moyenne de la sortie proche de 0 et l'écart type de la sortie proche de 1.

MaxPool2D : sous-échantillonne la représentation d'entrée en prenant la valeur maximale dans la fenêtre définie par **pool_size** pour chaque dimension le long de l'axe des caractéristiques. Le **pool_size** est 2×2 dans notre modèle.

Dropout() : est utilisé pour fixer aléatoirement les sorties des couche cachées à 0, à chaque mise à jour de la phase d'apprentissage. La valeur passée dans **dropout** spécifie la probabilité à laquelle les sorties de la couche sont abandonnées.

Flatten() : est utilisé pour convertir les données en un tableau à une dimension afin de les introduire dans la couche suivante.

Dense() : est la couche régulière du réseau de neurone profonde. La couche de sortie est également une couche **dense** avec 25 neurones car nous prédisons une probabilité pour 25 classes. La fonction **sigmoïde** est utilisée car elle existe entre (0 et 1) et cela nous facilite la prédiction d'une sortie binaire. Même si nous prédisons plus d'une étiquette, nous remarquant que dans la cible ***y*** toutes les valeurs sont soit 0 ou 1. La fonction Sigmoïde est donc la fonction appropriée pour ce modèle.

```
model = Sequential()
model.add(Conv2D(16, (3,3), activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))

model.add(Conv2D(32, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(2,2))
model.add(Dropout(0.4))

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(2,2))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))


model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))


model.add(Dense(25, activation='sigmoid'))
```

Figure 32 : Création du modèle

2.6.4 RESUME DU MODELE UTILISE

Nous allons afficher un résumé du modéle avec **summary().**

```
model.summary()
```

Figure 33 : Code pour afficher le résumé du modèle

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 348, 348, 16)      448
_________________________________________________________________
batch_normalization (BatchNo (None, 348, 348, 16)      64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 174, 174, 16)      0
_________________________________________________________________
dropout (Dropout)            (None, 174, 174, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 172, 172, 32)      4640
_________________________________________________________________
batch_normalization_1 (Batch (None, 172, 172, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 86, 86, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 86, 86, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 84, 84, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 84, 84, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 42, 42, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 42, 42, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 40, 40, 128)       73856
_________________________________________________________________
batch_normalization_3 (Batch (None, 40, 40, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 20, 20, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 51200)             0
_________________________________________________________________
dense (Dense)                (None, 128)               6553728
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)               512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 25)                3225
=================================================================
Total params: 6,672,889
Trainable params: 6,671,897
Non-trainable params: 992
_________________________________________________________________
```

Figure 34 : Résumé du modèle

Dans ce récapitulatif du modèle nous remarquons que nous avons au totale plus de 6 millions de paramètres.

2.6.5 ENTRAINEMENT DU MODELE

Pour compiler et ajuster le modèle nous utilisons 5 **époques** afin entraîner le modèle, chaque époque est une itération sur l'ensemble des données fournies.

validation_data c'est la partie de données sur laquelle on évalue l'erreur et toute métrique du modèle à la fin de chaque époque. Avec **metrics = ['accuracy']**, le modèle sera évalué en fonction de **la précision**.

```
model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))
```

Figure 35 : Code d'entrainement du modèle

```
Epoch 1/5
193/193 [==============================] - 606s 3s/step - loss: 0.8412 - accuracy: 0.0728 - val_loss: 0.9827 - val_accuracy: 0.0248
Epoch 2/5
193/193 [==============================] - 587s 3s/step - loss: 0.3537 - accuracy: 0.2269 - val_loss: 0.2575 - val_accuracy: 0.1901
Epoch 3/5
193/193 [==============================] - 585s 3s/step - loss: 0.2658 - accuracy: 0.3024 - val_loss: 0.2456 - val_accuracy: 0.1938
Epoch 4/5
193/193 [==============================] - 583s 3s/step - loss: 0.2552 - accuracy: 0.3180 - val_loss: 0.2390 - val_accuracy: 0.2094
Epoch 5/5
193/193 [==============================] - 580s 3s/step - loss: 0.2492 - accuracy: 0.3101 - val_loss: 0.2373 - val_accuracy: 0.2443
```

Figure 36 : Résultat d'entrainement

2.6.6 Analyse Des Courbes De La Fonction D'erreur Et De La Precision

Maintenant, nous allons visualiser les résultats.

```
def plot_learningCurve(history, epoch):
  # Plot training & validation accuracy values
  epoch_range = range(1, epoch+1)
  plt.plot(epoch_range, history.history['accuracy'])
  plt.plot(epoch_range, history.history['val_accuracy'])
  plt.title('Précision du modèle')
  plt.ylabel('Model')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

  # Plot training & validation loss values
  plt.plot(epoch_range, history.history['loss'])
  plt.plot(epoch_range, history.history['val_loss'])
  plt.title('Perte de modèle')
  plt.ylabel('Model')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

plot_learningCurve(history, 5)
```

Figure 37 : Code d'affiche les graphe de Perte et Précision

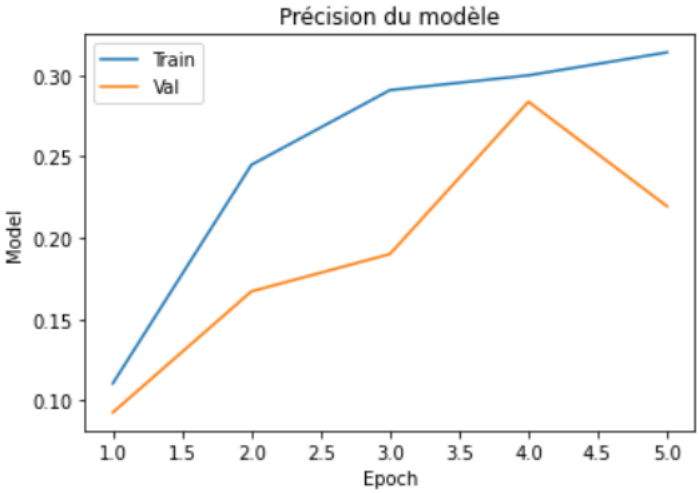


Figure 38 : Graphe de Précision

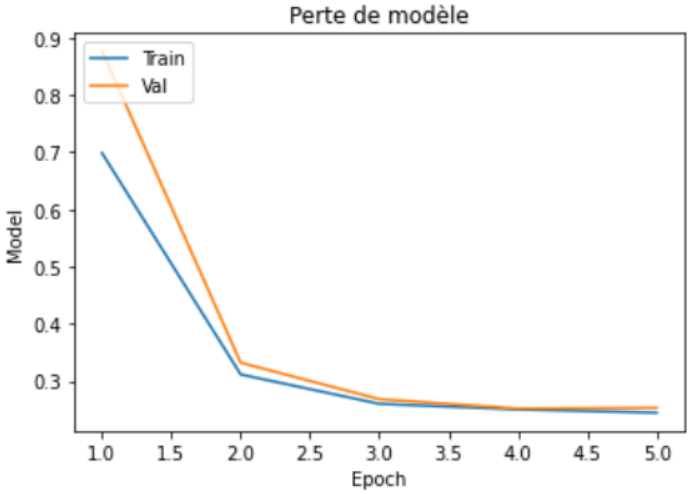


Figure 39 : Graphe de Perte

Nous pouvons constater que la précision de validation est inférieure à la précision d'apprentissage et que la perte de validation est inférieure à la perte d'apprentissage..

2.6.7 TEST DU MODEL

Nous allons tester notre modèle en lui donnant une nouvelle image. Nous allons normaliser l'image et la convertir en une taille de 350×350. La **classes** contient toutes les 25 classes que nous avons considérées. **model.predict()** nous donnera les probabilités pour les 25 classes.

Nous trierons les probabilités à l'aide de **np.argsort()** et sélectionnerons ensuite les classes ayant les 3 meilleures probabilités.

```
img = image.load_img('saaho.jpg', target_size=(img_width, img_height, 3))
plt.imshow(img)
img = image.img_to_array(img)
img = img/255.0

img = img.reshape(1, img_width, img_height, 3)

classes = data.columns[2:]
print(classes)
y_prob = model.predict(img)
top3 = np.argsort(y_prob[0])[:-4:-1]

for i in range(3):
  print(classes[top3[i]])
```

Figure 40 : Code de teste du modèle

```
Index(['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Family', 'Fantasy', 'History', 'Horror',
       'Music', 'Musical', 'Mystery', 'N/A', 'News', 'Reality-TV', 'Romance',
       'Sci-Fi', 'Short', 'Sport', 'Thriller', 'War', 'Western'],
      dtype='object')
```

Figure 41 : Résultat de teste

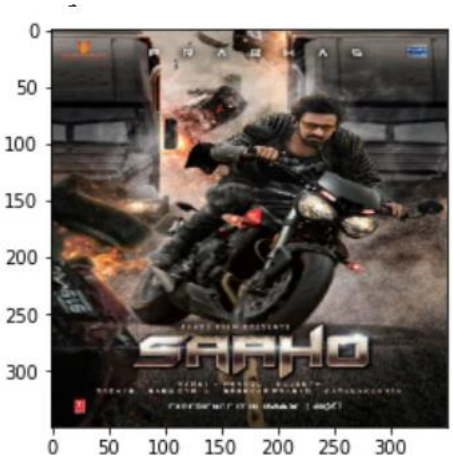


Figure 42 : Affiche de film

Comme on peut le voir pour l'affiche de film ci-dessus, notre modèle a sélectionné 3 genres :

```
Adventure
Drama
Action
```

Figure 43 : genre du film

Nous allons essayer avec une autre affiche de film :

```
img = image.load_img('/content/elti.jpg', target_size=(img_width, img_height, 3))
plt.imshow(img)
img = image.img_to_array(img)
img = img/255.0

img = img.reshape(1, img_width, img_height, 3)

classes = data.columns[2:]
print(classes)
y_prob = model.predict(img)
top3 = np.argsort(y_prob[0])[:-4:-1]

for i in range(3):
  print(classes[top3[i]])
```

Figure 44 : Code de teste du modèle

Pour l'affiche de film ci-dessous, notre modèle a sélectionné 3 genres

```
Index(['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime',
       'Documentary', 'Drama', 'Family', 'Fantasy', 'History', 'Horror',
       'Music', 'Musical', 'Mystery', 'N/A', 'News', 'Reality-TV', 'Romance
       'Sci-Fi', 'Short', 'Sport', 'Thriller', 'War', 'Western'],
      dtype='object')
Comedy
Drama
Romance
```

Figure 45 : Genre du film

Figure 46 : Affiche de film

2.7 EVALUATION DU MODELE SUR LE JEU DE DONNEES DE TEST

```
results = model.evaluate(X_test,y_test)

print('\nTest Loss: ', results[0])
print("Accuracy: %.2f%%" % (results[1]*100))
print('Test Accuracy: ', results[1])
```

```
35/35 [==============================] - 21s 597ms/step - loss: 0.2341 - accuracy: 0.2433

Test Loss:  0.2340725064277649
Accuracy: 24.33%
Test Accuracy:  0.24334251880645752
```

Figure 47 : Code et résultat d'évaluation du module

2.8 H5

Le paquet h5py est une interface Python pour le format de données binaire **HDF5.**

HDF5 nous permet de stocker d'énormes quantités de données numériques, et de manipuler facilement ces données à partir de NumPy.

2.8.1 SAUVEGARDE DU MODELE

Après la création de notre modèle on peut le sauvegarder pour pouvoir le réutilisé pour plus tard pour faire des prédictions. Il suffit utiliser **save()** pour que notre modèle serra enregistré avec l'extension h5.

```
#pip instal h5py
model.save('my_model.h5')
```

Figure 48 : Code de sauvegarde du modelé

Potentiellement sur notre ordinateur on devra installer **h5py** en faisant un **pip instal h5py**. Qui est une librairie utilisée pour cette sauvegarde. Pour pouvoir créer ce fichier avec l'extension **h5**, qui est un format de fichier spéciale pour enregistrer les modèles.

Donc une fois le modèle est sauvegardé, un fichier avec son nom est créé au niveau du répertoire.

2.8.2 CHARGEMENT DU MODELE

Pour réutiliser le modèle que nous avons entrainé précédemment, on va le charger par : **keras.models.load_model()** .

```
loaded_model = tf.keras.models.load_model("my_model.h5")
loss, acc = loaded_model.evaluate(X_test, y_test)
```

```
35/35 [==============================] - 23s 647ms/step - loss: 0.2344 - accuracy: 0.2195
```

Figure 49 : Code de chargement du modèle

2.9 JSON

JSON est l'acronyme de **JavaScript Object** Notation. JSON est utilisé pour stocker et transférer les données. **Python** supporte **JSON** grâce à un paquetage intégré appelé **json**.

Pour utiliser cette fonctionnalité, nous devons importer le paquet **json** dans le **script Python**

2.9.1 SAUVEGARDE DU MODELE

La sauvegarde du modele est constitué de deux parties principales:

- La première partie : sauvegarde de la structure du modèle
- La deuxième partie: sauvegarde des poids l'entrainement

- **Sauvegarde de la structure du modele** : Pour sauvegarder la structure de notre modèle avec une extension **.json**, il suffit d'utiliser **to_json()** pour transformer le modèle au format de chaine json, on suite avec la fonction **open()** on crée le fichier .json , et pour écrire la structure du modéle que nous avons obtenu au format json on utilise la fonction **json_file_write().**
- **Sauvegarde de l'entraineemt :** Pour enregistrer les poids d'entrainement dans un autre fichier, il suffit d'exécuter une méthode de sauvegarde des poids avec **save-weights().**

```
model_json = model.to_json()

with open("my_model.json", "w") as json_file:
    json_file.write(model_json)


model.save_weights("model_weights.h5")
```

Figure 50 : Code de sauvegarde du modelé

2.10 CONCLUSION

Dans ce chapitre nous avons passé en revue les différentes étapes de notre méthodes de classification. On a commencé par la phase de traitement des données, en passant par la phase de construction et entrainement de notre modèle pour arriver à la fin à la phase de test et évaluation des performances.

CONCLUSION ET PERSPECTIVES

Dès l'apparition d'Internet, le nombre de films visionnés en streaming n'a cessé d'augmenter. Actuellement, la quantité des films disponible sur Internet est tellement grande que les consommateurs sont généralement dans l'embarras du choix.

La problématique de ce travail consiste à catégoriser les films par genre en utilisant leurs affiches, cela dans le but d'offrir aux utilisateurs la possibilité de décider facilement sur le film à regarder sans perdre leur temps à choisir.

Pratiquement, un réseau neuronal profond est est construit afin d'estimer les probabilités d'appartenance d'une affiche aux différents genre. Les résultats de l'évaluation à montrer que le model proposer permet d'obtenir des performances prometteuses

A l'avenir nous prévoyons d'améliorer les performances de la classification en incorporant d'avantage d'informations et utiliser les riches métadonnées pour étudier d'autre propriétés des films. D'une autre part, nous envisageons, d'intégrer notre classifieur dans un système de recommandation afin de proposer aux clients des plates-formes de streaming du contenu qui correspond à leurs profils.

BIBLIOGRAPHY

[1] Aggarwal, C.C. (ed.): Data Classification: Algorithms and Applications. CRC Press (2014).
[2] Aha, D.W. (ed.): Lazy Learning. Springer (1997).

[3] Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F.: An overview of ensemble methods for binary classifiers in multi-class problems: experimental study on one-vs-one and one-vs-all schemes. Pattern Recogn. 44(8), 1761–1776 (2011).

[4] Gibaja, E., Ventura, S.: A tutorial on multi-label learning. ACM Comput. Surv. 47(3) (2015)

[5] Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining multi-label data. In: Data Mining and Knowledge Discovery Handbook, pp. 667–685. Springer (2010)

[6] Zhang, M.L., Zhou, Z.H.: A review on multi-label learning algorithms. IEEE Trans. Knowl. Data Eng. 26(8), 1819–1837 (2014)

[7] Bielza, C., Li, G., Larrañaga, P.: Multi-dimensional classification with Bayesian networks. Int. J. Approximate Reasoning 52(6), 705–727 (2011)

[8] Amores, J.: Multiple instance classification: review, taxonomy and comparative study. Artif. Intell. 201, 81–105 (2013)

[9] Charte, F., Rivera, A.J., del Jesus, M.J., Herrera, F.: QUINTA: a question tagging assistant to improve the answering ratio in electronic forums. In: Proceedings of IEEE International Conference on Computer as a Tool, EUROCON'15, pp. 1–6. IEEE (2015)

[10] Charte, F., Rivera, A.J., del Jesus, M.J., Herrera, F.: QUINTA: a question tagging assistant to improve the answering ratio in electronic forums. In: Proceedings of IEEE International Conference on Computer as a Tool, EUROCON'15, pp. 1–6. IEEE (2015)

[11] Katakis, I., Tsoumakas, G., Vlahavas, I.: Multilabel text classification for automated tag suggestion. In: Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD'08, pp. 75–83 (2008)

[12] Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: a new benchmark collection for text categorization research. J. Mach. Learn. Res. 5, 361–397

[13] Salton, G., Fox, E.A., Wu, H.: Extended Boolean information retrieval. Commun. ACM 26(11), 1022–1036 (1983)

[14] Boutell, M., Luo, J., Shen, X., Brown, C.: Learning multi-label scene classification. Pattern Recogn. 37(9), 1757–1771 (2004)

[15] Briggs, F., Lakshminarayanan, B., Neal, L., Fern, X.Z., Raich, R., Hadley, S.J.K., Hadley, A.S., Betts, M.G.: Acoustic classification of multiple simultaneous bird species: a multi-instance multi-label approach. J. Acoust. Soc. Am. 131(6), 4640–4650 (2012)
[16] Zhou, Z.H., Zhang, M.L., Huang, S.J., Li, Y.F.: Multi-instance multi-label learning. Artif. Intell. 176(1), 2291–2320 (2012)
[17] Duygulu, P., Barnard, K., de Freitas, J., Forsyth, D.: Object recognition as machine translation: learning a lexicon for a fixed image vocabulary. In: Proceedings of 7th European Conference on Computer Vision, ECCV'02, vol. 2353, pp. 97–112. Springer (2002)
[18] Snoek, C.G.M., Worring, M., van Gemert, J.C., Geusebroek, J.M., Smeulders, A.W.M.: The challenge problem for automated detection of 101 semantic concepts in multimedia. In: Proceedings of 14th ACM International Conference on Multimedia, MULTIMEDIA'06, pp. 421– 430 (2006)
[19] Turnbull, D., Barrington, L., Torres, D., Lanckriet, G.: Semantic annotation and retrieval of music and sound effects. IEEE Trans. Audio Speech Lang. Process. 16(2), 467–476 (2008)
[20] Wieczorkowska, A., Synak, P., Ra´s, Z.: Multi-label classification of emotions in music. In: Intelligent Information Processing and Web Mining. AISC, vol. 35, Chap. 30, pp. 307–315 (2006)
[21] Diplaris, S., Tsoumakas, G., Mitkas, P., Vlahavas, I.: Protein classification with multiple algorithms. In: Proc. 10th Panhellenic Conference on Informatics, PCI'05, vol. 3746, pp. 448–456. Springer (2005)
[22] Elisseeff, A., Weston, J.: A kernel method for multi-labelled classification. In: Advances in Neural Information Processing Systems, vol. 14, pp. 681–687. MIT Press (2001)
[23] Duda, R., Hart, P., Stork, D.: Pattern Classification, 2nd edn. John Wiley (2000)
[24] Cherkassky, V., Mulier, F.: Learning from Data: Concepts. Theory and Methods. Wiley-IEEE Press (2007)
[25] Aly, M.: Survey on multiclass classification methods. In: Technical Report, pp. 1–9. California Institute of Technology (2005)
[26] Galar, M., Fernández, A., Barrenechea, E., Bustince, H., Herrera, F.: An overview of ensemble methods for binary classifiers in multi-class problems: experimental study on one-vs-one and one-vs-all schemes. Pattern Recogn. 44(8), 1761–1776 (2011)

[27] Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory 13(1), 21–27 (1967)
[28] Quinlan, J.R.: C4.5: Programs for machine learning (1993)

[29] Clare, A., King, R.D.: Knowledge discovery in multi-label phenotype data. In: Proceedings of the 5th European Conference Principles on Data Mining and Knowledge Discovery, PKDD'01, vol. 2168, pp. 42–53. Springer (2001)
[30] Zhang, M.: Multilabel neural networks with applications to functional genomics and text categorization. IEEE Trans. Knowl. Data Eng. 18(10), 1338–1351 (2006)
[31] Zhang, M.: Ml-rbf : RBF neural networks for multi-label learning. Neural Process. Lett. 29, 61–74 (2009)
[32] Kongsorot, Y., Horata, P.: Multi-label classification with extreme learning machine. In: Proceedings of the 6th International Conference on Knowledge and Smart Technology, KST'14, pp. 81–86. IEEE (2014)
[33] Boutell, M., Luo, J., Shen, X., Brown, C.: Learning multi-label scene classification. Pattern Recogn. 37(9), 1757–1771 (2004)
[34] Xu, J.: Fast multi-label core vector machine. Pattern Recogn. 46(3), 885–898 (2013)
[35] Wang, J., Feng, J., Sun, X., Chen, S., Chen, B.: Simplified constraints Rank-SVM for multilabel classification. In: Proceedings of the 6th Chinese Conference on Pattern Recognition, CCPR'14, pp. 229–236. Springer (2014)
[36] J. P.-A. M. M. B. X. D. W.-F. S. O. A. C. Y. B. Ian J. Goodfellow, «Generative Adversarial Networks,» [1406.2661] Generative Adversarial Networks (arxiv.org).
[37] L. Yann, B. Yoshua et Geoffrey, «H. Deep Learning.,» May 2015. [En ligne]. Available: https://www.researchgate.net/publication/277411157_Deep_Learning.
[38] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent nervous activity. Bulletin of Mathematical Boiphysics, pages 115–133, 1943.
[39] M. Minsky and S. Papert. Perceptrons: an introduction to computational geometry. MIT Press, 1969.
[40] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 20
[41] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural net chips and automatic learning. IEEE Communication, pages 41–46, November 1989. invited paper
[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1106–1114, 2012.
[43] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. Journal of Physiology, 148:574–591, 1959.

Résumé

Dans ce projet de fin d'étude, nous proposons de classifier les films par genre en nous basant uniquement sur les images d'affiches de films. Un réseau neuronal profond est construit pour décrire conjointement l'apparence visuelle et les informations sur les objets, et classifier une image d'affiche de film donnée selon le genre. Comme un film peut appartenir à plusieurs catégories simultanément, il s'agit d'un problème de classification d'images multi-label. Pour ce faire, nous commençons par recueillir un ensemble de données à grande échelle sur les affiches de films, où chaque affiche est associée aux genres du film correspondant. Sur la base de cet ensemble de données, un réseau neuronal convolutif sera construit et entraîné, permettant par la suite de classifier de nouvelles affiches selon le genre. Dans la phase d'évaluation, nous montrons que la méthode proposée donne des performances prometteuses.

Mots clés : Classification par genre, affiche de film, classification multi-label, réseau de neurones profond, réseaux à convolution.

ملخص

في نهاية مشروع الدراسة هذا ، نقترح تصنيف الأفلام حسب النوع بناءً على صور ملصقات الأفلام فقط . يتم إنشاء شبكة عصبية عميقة لوصف المظهر المرئي ومعلومات الكائن بشكل مشترك ، ولتصنيف صورة ملصق فيلم معين إلى أنواع. نظرًا لأن الفيلم يمكن أن ينتمي إلى أكثر من نوع واحد ، فهذه مشكلة تتعلق بتصنيف الصور متعددة التصنيفات. لتسهيل الدراسات ذات الصلة ، نقوم بجمع مجموعة بيانات واسعة النطاق على ملصقات الأفلام. بناءً على مجموعة البيانات هذه ،

كلمات مرشدة. برمجيات، فصل الانشغالات