Rapport de Projet Informatique

Collecte et classification de visualisation de données

Commanditaire

Romain VUILLEMOT

Élèves

Arnaud BRUGIÈRE
Alexis MARTIN-DELAHAYE
Luca GUÉRY
Louis KRAEMER
Marc ARNAL

Table des matières

Résumé

Dans le cadre de l'option Informatique à l'École Centrale de Lyon, les élèves doivent réaliser un projet technique en groupe sur une période de 3 mois. En ce qui nous concerne, le projet porte sur la collecte et la classification de visualisation de données. L'objectif est de construire un modèle, doté d'une intelligence artificielle, qui permet de classifier des images. Ces images sont des représentations graphiques de données comme le camembert, le nuage de points ou l'histogramme par exemple.

Au préalable, nous avons donc dû construire une base de données suffisamment grande contenant ce type d'images. Il nous a fallu définir des classes précises, afin que celles-ci soient identifiables les unes des autres par notre modèle de Machine Learning. Cette collecte de données s'est faite via des algorithmes de scrapping sur Google.
La seconde phase du projet consiste à définir quel modèle utiliser pour un besoin de reconnaissance d'images. Après des recherches, le choix des réseaux de neurones convolutionnels nous a paru évident puisqu'il est le plus adapté, il s'inspire en effet du modèle neuronal de reconnaissance de l'Homme.

L'utilisation de l'outil TensorFlow nous a permis de construire des modèles de réseaux de neurones en Python, que nous avons entraîné par la suite grâce à notre base de données. Une partie des résultats est consultable dans ce rapport. Finalement, notre projet, dont l'intégralité est consultable sur GitHub, permet l'implémentation automatique de toutes ces tâches. À partir d'un dossier d'images il est possible d'entraîner un modèle, et l'utilisateur n'a plus qu'à jouer sur les différents paramètres possibles pour ensuite tester la qualité du modèle sur un ensemble test.

Les résultats finaux sont plutôt satisfaisants, classant pour dix classes à plus de 70% de précision, et à plus de 90% de précision pour trois classes. Il faut cependant noter que les résultats ne peuvent être améliorés sans des ordinateurs plus puissants et des bases de données beaucoup plus conséquentes.

Remerciements

Tout d'abord, nous aimerions remercier chaleureusement Monsieur Romain VUILLEMOT, tuteur et commanditaire de notre projet, qui a été un soutien continu tout au long de l'année. Son aide et ses conseils avisés ont été très précieux. Il nous a guidés sur des pistes à suivre et des méthodes à adopter. Nous le remercions également pour sa disponibilité, notamment pour le travail de relecture et de conseil avant chaque présentation et le rendu final.

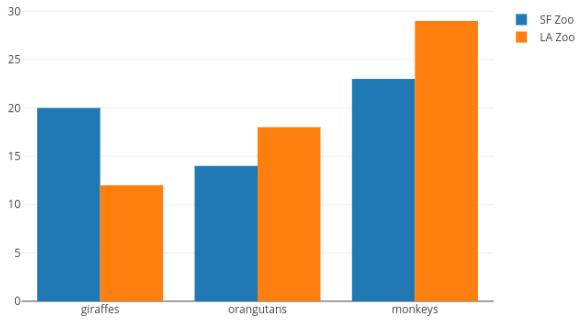
Puis nous aimerions remercier Théo JAUNET, actuellement en stage à l'université Lyon 1, qui a pu nous aiguiller en début de projet sur les différentes méthodes de Machine Learning adaptées à la classification. Sa connaissance du sujet a été d'une énorme aide pour nous projeter et avancer dans le projet.

Enfin, nous tenons à remercier Messieurs Mohsen ARDABILIAN et Daniel MULLER, responsables de l'option informatique qui ont fait en sorte qu'un tel projet puisse s'exécuter et pour leurs questions pertinentes lors des différentes séances de restitution.

Introduction

Avec l'arrivée d'internet et, plus récemment, du Big Data, la présence de données n'a fait que croître. Cependant, celles-ci sont stockées dans un état brut et il est alors difficile de les interpréter ou dans tirer un quelconque profit. Pour cela, la visualisation de données nous permet d'appréhender une masse considérable d'informations de façon simple et rapide.

Néanmoins, avec l'essor de ces visualisations s'est posé plusieurs problèmes dont un principal : la labellisation des différentes représentations.



Quel est le type de graphique ci-dessus ? Un diagramme à barre ? Un histogramme ?

Ce contexte a amené la création du projet présenté par ce rapport. Dans le cadre de notre dernière année à l'Ecole Centrale de Lyon en option informatique, nous nous sommes intéressés à la labellisation des différents diagrammes afin de simplifier leur classement.

Cependant, une fois que les différentes classes sont définies, il n'est pas imaginable que lorsque quelqu'un se demande la classe d'appartenance d'une certaine représentation, il doive demander à un des 5 membres de ce projet, laquelle est-elle. La solution naturelle a donc été de développer un réseau de neurone et de tirer parti de la puissance du "Machine Learning" pour prédire la classe d'un graphique.

Ce rapport présente donc la façon dont nous avons abordé le problème, les solutions adoptées pour la résolution de celui-ci et les résultats obtenus avec une vue critique de ces derniers.

L'intégralité de nos travaux est disponible sur Github en mode public. Il suffit de chercher le projet DatasetVis sur le profil d'AmigoCap : **https://github.com/AmigoCap/DatasetVis**.

1. Cadre et fonctionnement du projet

A. État de l'art de l'existant

a. État de l'art des visualisations de données

Deux projets se proposent de faire une ontologie de l'ensemble des data vizualisations existantes. Ils se nomment « dataviz catalogue » et «dataviz project » dont les liens sont consultables ci-dessous :

- https://datavizcatalogue.com/
- http://datavizproject.com/

Voici une proposition de compilation de ces 2 sources :

Arc Diagram
Area Graph
Bar Chart
Box & Whisker Plot
Brainstorm
Bubble Chart
Bubble Map
Bullet Graph
Calendar
Candlestick Chart
Chord Diagram
Choropleth Map
Circle Packing
Connection Map
Density Plot
Donut Chart
Dot Map
Dot Matrix Chart
Error Bars
Flow Chart
Flow Map
Gantt Chart
Heatmap
Histogram
Illustration Diagram
Kagi Chart
Line Graph
Marimekko Chart
Multi-set Bar Chart
Network Diagram
Nightingale Rose Chart

Semi Circle Donut Chart
Slope Chart
Sociogram
Solid Gauge Chart
Sorted Stream Graph
Span Chart
Sparkline
Spiral Heat Map
Spiral Plot
Spline Graph
Stacked Area Chart
Stacked Area Graph
Stacked Bar Chart
Stacked Bar Graph
Stacked Ordered Area Chart
Stem & Leaf Plot
Step by Step Illustration
Stepped Line Graph
Stream Graph
Sunburst Diagram
Swimlane Flow Chart
SWOT Analysis
Table Chart
Tally Chart
Target Diagram
Taylor diagram
Ternary Contour Plot
Ternary Plot
Three-dimensional Stream Graph
Timeline
Timetable

Non-ribbon Chord Diagram	Topographic Map
Open-high-low-close Chart	Transit Map
Parallel Coordinates Plot	Tree Diagram
Parallel Sets	Treemap
Pictogram Chart	Trendline
Pie Chart	Triangle Bar Chart
Point & Figure Chart	Venn Diagram
Population Pyramid	Violin Plot
Proportional Area Chart	Waffle Chart
Radar Chart	Waterfall Chart
Radial Bar Chart	Waterfall Plot
Radial Column Chart	Win-loss Sparkline
Sankey Diagram	Word Cloud
Scatterplot	

Bien sûr, toutes ces visualisations de données ne forment pas un ensemble homogène. Certaines se présente sous forme de graphe avec des axes (scatter plot, bar chart) d'autres non. Surtout, dans le cadre de notre projet, il s'agit d'identifier lesquels sont les plus utilisées pour pouvoir prioriser nos efforts de reconnaissance sur celles-ci. En effet, comme nous allons le voir maintenant, la partie algorithmique de classification des visualisations de données représente un véritable défi technique qui peut être résolu grâce au Machine Learning.

b. État de l'art des algorithmes de Machine Learning

Méthode de classification	Supervisée ou non supervisé ?	Classification binomiale ou multiclasses ?
CLASSIFICATION NAIVE BAYESIENNE	Supervisé	Multi classe
Principe : La méthode considère le vecteur x des valeurs des variables prédictives comme une variable aléatoire dont la distribution dépend de la classe. La classification est réalisée à partir d'un classifieur bayésien qui est soumis à l'apprentissage. Le but de l'apprentissage pour le classifieur bayésien est d'estimer la probabilité a priori des classes et d'estimer la densité de probabilités des classes.		
Avantages : L'efficacité et la simplicité de l'algorithme, et le peu de données nécessaires pour l'entrainer.		
Inconvénients : On est obligé de supposer que les variables prédictives ont des probabilités conditionnelles indépendantes.		
LES MACHINES A VECTEURS SUPPORTS	Non supervisé	Binomial
Principe : La méthode consiste à trouver un hyperplan optimal qui sépare les deux catégories. Cet hyperplan doit avoir la distance la plus faible possible avec chacune des catégories. Dans chaque catégorie, les points les plus proches de l'hyperplan sont appelés les vecteurs supports. L'espace entre les deux catégories est appelé la marge.		

Avantages : Ces algorithmes fonctionnent sur des problème complexes, ie non-linéaire et/ou avec beaucoup de dimension		
Inconvénients : L'algorithme est souvent moins performant que les forêts aléatoires et passe difficilement à l'échelle		
LES ARBRES DE DECISIONS	Supervisé	Multiclasse
Principe : La méthode consiste à classer une observation au moyen d'une succession de tests concernant les valeurs des variables prédictives. Chaque test est représenté par un nœud de l'arbre. Chaque branche correspond à une réponse possible à la question posée. La classe de la variable est déterminée par la feuille à laquelle parvient l'observation à l'issue de la suite de tests. La phase d'apprentissage consiste donc à trouver les bons tests pour classer correctement les observations par rapport à leur valeur pour la variable cible. L'objectif est le suivant : les feuilles doivent être homogènes en ne contenant que les observations appartenant à une seule et même classe		
Avantages : Fonctionne sur des problèmes complexe (non-linéaire, multiclasse). Peu de préparation de données nécessaires.		
Inconvénients : Risque important de surapprentissage. Le critère du premier nœud influe énormément l'ensemble du modèle de prédiction		
LES FORÊTS ALEATOIRES	Supervisé	Multiclasse
Principe : A partir d'un échantillon initial de N observations dont chacune est décrite par p variables prédictives, on crée artificiellement B nouveaux échantillons de même taille N par tirage avec remise. On entraine alors B arbres de décisions différents Parmi les p variables prédictives, on n'en utilise qu'un nombre m<p choisies au hasard. Elles sont alors utilisées pour faire la meilleure segmentation possible. L'algorithme combine plusieurs algorithmes faibles (les B arbres de décision) pour en constituer un plus puissant en procédant par vote : pour classer une nouvelle observation, on la fait passer par les B arbres et on sélectionne la classe majoritaire parmi les B prédictions.		
Avantages : Possède les avantages des arbres de décision		
Inconvénients : Peu intelligible, complexe à comprendre et à implémenter.		
LES RÉSEAUX DE NEURONES	Non supervisé	Multiclasse
Principe : Les réseaux de neurones consistent en un réseau orienté composé de neurones artificiels organisés en couches. Les neurones d'une couche donnée sont liés à tous les neurones de la couche précédente et de la couche suivante par des relations pondérées. De ces poids dépend le comportement du réseau, et leur adaptation au problème considéré et l'objectif de la phase d'apprentissage. Chaque neurone a une sortie qui est obtenue par l'application d'une fonction non linéaire de la somme pondérée des entrées, qui sont elles-mêmes les sorties des neurones de la couche précédente. Afin de calculer le vecteur poids pour chaque neurone, des algorithmes de rétropropagation ont été développés.		
Avantages : Les réseaux de neurones permettent de traiter des problèmes de classification non linéaires complexes.		
Inconvénients : Le choix de la structure du réseau de neurone est compliqué. Il existe un risque de tomber dans un minimum local lors de l'apprentissage.		

Ces algorithmes « standards » ne représentent qu'une portion des algorithmes de Machine Learning existants : ce sont les algorithmes de classifications (qui s'oppose à ceux de

régression). Il est intéressant d'en avoir fait un panorama car s'ils peuvent tous potentiellement réaliser notre travail de reconnaissance, certains s'appliquent mieux à notre problématique spécifique de classification de visualisation de données que d'autres et c'est le point que nous allons essayer d'explorer à présent.

B. Démarche globale

a. Organisation de l'équipe

Nous nous sommes organisés pour mener un travail en équipe cohérent et efficace. Le chef de projet Alexis Martin-Delahaye, a pour rôle de coordonner les efforts de l'équipe, de la communication avec le tuteur de stage, de la détermination du planning et des différents objectifs.

Le responsable fonctionnel, Marc Arnal est le responsable de la documentation fonctionnelle, de piloter les rédactions des présentations et des rapports, et d'aider le chef de projet dans ses tâches.

Les responsables techniques, Luca Guery, Louis Kraemer, et Arnaud Brugière forment l'équipe technique responsable de la construction de l'architecture du projet et l'a mis en application en produisant l'essentiel du code.

Figure 1 : Organisation de l'équipe

Cette organisation globale est en réalité plus souple : chacun contribue au code, aux phases de tests et d'amélioration de l'algorithme qui nécessite de nombreuses itérations sur lesquelles toute l'équipe est impliquée afin de paralléliser le travail.

Afin de mobiliser toute l'équipe il est important d'avoir un challenge technique mais aussi des perspectives d'applications qui nous permettent de nous rendre compte que notre projet a une véritable utilité. C'est ce que nous allons voir ci-dessous.

b. *Perspectives d'applications*

Notre étude est susceptible d'intéresser des domaines où le traitement de vaste volume de données et l'utilisation de data visualisation est un enjeu central. Notamment, notre démarche peut constituer une première étape dans l'extraction de données issues de data visualisation où il faudrait d'abord identifier le type de la visualisation avant de pouvoir en extraire les données clés : par exemple, la pente d'une line chart, les proportions de chaque aire d'un diagramme à camembert, le nombre de point d'un nuage de points.

Par exemple, on peut penser qu'un institut financier qui reçoit de nombreux graphiques représentants des parts de marchés, des tendances, des évolutions diverses sous forme de data visualisation peut vouloir en extraire des données brutes chiffrées afin d'être en mesure de mener des analyses quantitatives. Si ces visualisations représentent une source de données significatives pour lui, il a intérêt à automatiser ces tâches, et notamment la première étape de classification des différents types de visualisation. De même pour un laboratoire scientifique dont le champ de recherche demande de s'appuyer sur les analyses de nombreuses études existantes productrices de data visualisation et de données disponibles sous cette forme.

NB : Le point de vigilance juridique

Un point de vigilance juridique doit être abordé. En effet, pour entraîner l'algorithme, nous utilisons des images qui ne sont pas forcément libre de droit et la méthode de scrapping ainsi que le nombre massif de data visualisations extraites ne nous offre pas la possibilité de demander une quelconque autorisation à leur propriétaire. Certes, nous n'utilisons pas publiquement ces images, et nos fins ne sont pas commerciales ou lucratives, mais au contraire, académiques. Cependant, si la loi est claire sur l'utilisation – au sens d'exposition publique – des images, il existe un flou juridique sur l'utilisation – au sens d'entraînement pour un algorithme de machine learning – qui, en tant que matière première d'entraînement de l'algorithme, contribue très largement à la valeur de notre projet. Une option envisageable pourrait être de publier en crédit la liste des sites qui ont contribué au scrapping, mais ils ne sont pas nécessairement les auteurs de ces images dont cette solution trouve vite ses limites. Ce cadre étant posé, rentrons dans le concret avec un tour d'horizon de l'environnement technique du projet.

c. *Outils utilisés*

Les outils de travail que nous avons utilisé sont les suivants :

- Python avec les librairies suivantes :
  - Numpy pour la gestion des tableaux/matrice
  - OpenCV pour la gestion des images
  - Scipy pour la manipulation scientifique des données
- TensorFlow pour les réseaux de neurones
- D3.js pour la datavisualisation
- GitHub pour la gestion du travail collaboratif
- Une API en mode REST pour le site

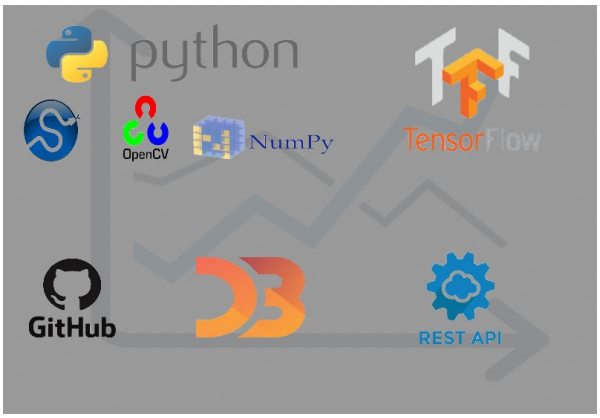


Figure 2 : Environnement technique du projet

Ces outils nous ont permis de réaliser le travail mais ce qui a compté en filigrane et la structuration du travail selon un planning précis détaillé ici.

d. Planning

Phase de cadrage

La phase de cadrage a consisté à :

- Déterminer le rôle de chacun et le mode de communication avec le tuteur
- Fixer les méthodes de travail et les outils adapté (GitHub)
- Faire l'état de l'art de l'existant et se documenter sur le cœur du sujet et toutes les problématiques connexes.

Détermination de l'algorithme à utiliser

Cette phase a consisté à sélectionner parmi tous les algorithmes de Machine Learning identifié celui qui serait le plus à même d'être adapté à notre cas. Pour ce faire, nous avons parcouru de la documentation scientifique, mais surtout nous avons échangé avec notre tuteur et avec un autre de ses élèves Théo Jaunet qui a déjà mené des recherches reliées à cette question de reconnaissance automatique de datavizualisation. Il nous a expliqué sa démarche pour sélectionner la meilleure méthode et a indiqué que les meilleurs résultats sont ceux que les réseaux de neurones convolutionnels peuvent apporter.

Détermination des classes de datavizualisation

Pour sélectionner les premières classes de datavizualisation, nous avons pris plusieurs critères en ligne de compte :

- Des types de graphiques très fréquents en datavizualisation (i.e. éviter des graphiques théoriques rarement utilisés)
- L'abondance des ressources disponibles dans le cadre du scrapping
- La propreté des ressources disponibles dans le cadre du scrapping, c'est-à-dire le fait qu'un faible nombre d'images parasites (qui ne sont pas de la classe considérée) polluent la base de données et soient chronophages à supprimer à la main.
- La différenciation entre les classes permettant dans un premier temps de ne pas introduire trop de confusion pour l'algorithme

Sur cette base, nous avons choisi les trois catégories suivantes :

- Aera chart (diagramme à aire)
- Bar chart (diagramme à barre)
- Pie chart (camembert)

Généralisation aux autres types visualisation de données

Nous avons enfin généralisé nos résultats à des jeux de données plus étendus. Cette extension s'est faite en deux étapes sur les mêmes critères que la sélection des trois classes initiales :

- D'abord, le passage de 3 classes à 6 classes. Ont été ajouté les classes suivantes : line chart, venn diagram, radar chart.
- Ensuite le passage de 6 classes à 10 classes. Ont été ajouté les classes suivantes : dot map, choropleth map et bubble chart
- A également été ajouté une catégorie correspondant à « aucune classe ».
-

Itération sur l'algorithme et sur les paramètres

Nous avons ensuite itéré sur les multiples possibilités d'organisation de notre réseau de neurones afin de déterminer quelle architecture qui articulent les différentes couches (pooling ? max-pooling ? fully-connected ?) donnerait les résultats les plus pertinents.

2. Spécification fonctionnelle et technique du projet

A. Constitution de la base de données

a. Scrapping

Une première partie de notre travail consiste en la collecte de visualisations de données, par exemple des nuages de points ou des histogrammes, afin de constituer une base de données d'images suffisamment grande pour entraîner un modèle de réseaux de neurones convolutionnels et le tester.

Grâce au script Python **google-scrapper_2.0.py** inspiré du git image-scrapers, il est possible de récupérer des centaines d'images issues d'une requête sur Google Image. En effet, ce script permet de lancer un driver de Google Chrome, de réaliser la requête souhaitée sur Google Image et de scroller automatiquement jusqu'en bas de la page afin de charger toutes les images possibles pour pouvoir ensuite les télécharger avec un JSON associé, contenant la provenance de l'image notamment. Ceci nous permet de garder une trace des sources utilisées. Nous avons apporté des modifications au script initial afin de renommer automatiquement les images téléchargées avec leur classe et un identifiant unique.

Nous avons essayé auparavant d'autres méthodes de scrapping qui n'utilisaient pas de driver, mais elle limitait toutes à 100 le nombre d'images maximum récupérables, limite de chargement des données fixée par Google.

Ainsi, si on souhaite récupérer des centaines d'images de "bar chart", on exécute le script Python comme suit, en prenant soin d'ajouter les "_" à la place des espaces :

```
python3 google-scrapper_2.0 bar_chart
```

On obtient alors des centaines d'images nommées suivant l'exemple suivant "bar_chat_1.jpg". Ceci nous sera très utile pour associer à chaque image sa classe. Nous avons utilisé ce script pour collecter des images des dix classes sélectionnées. Les résultats se trouvent dans un dossier situé dans le répertoire du script scrapper, dont le nom est **dataset**. Chaque sous-dossier de **dataset** correspond au résultat d'une requête.
Sur le git de notre projet DatasetVis, il est possible de suivre un tutoriel pour pouvoir utiliser ce script de scrapping (disponible en annexe).

Il a fallu ensuite nettoyer ce premier jeu de données pour obtenir une base de données propre et exploitable.

b. Nettoyage du jeu de données

Une fois les images des dix classes téléchargées, il faut nettoyer ce jeu car il comporte des images illisibles, des doublons, et des images qui ne correspondent pas à la requête réalisée, ou alors comportant trop de bruit. Voici les critères que nous avons utilisé pour

exclure les images non conformes. Ils sont issus de l'expérience. Cette liste n'est pas exhaustive mais correspond aux cas qui reviennent le plus souvent.

Pour les dix catégories, les images que nous avons retirées, en plus de celles qui n'étaient pas lisibles, valident un ou plusieurs critères suivants :

- Légende / ou bruit représentant une part trop importante de l'image
- Graphique regroupant 2 catégories (ex : à la fois bar et line chart)
- Graphique trop simpliste (type icône ou vecteur)
- Graphique tracé manuellement

Nous avons volontairement laissé du bruit dans nos données, introduit par du texte par exemple, des fonds hétérogènes ou encore des graphiques en trois dimensions. On obtient après nettoyage manuel au moins 300 images exploitables par classe. Notre dossier contient finalement plus de **3500 images.** Certaines classes regroupent plus de 500 photos, d'autres près de 400. Nous construisons alors un dossier comprenant un nombre semblable d'image pour chaque catégorie : trois catégories bien renseignées (**1350 images**), six catégories (**2100 images**) et 10 catégories (**3200 images**).

Il est nécessaire de noter que ce nombre reste faible pour obtenir de très bons résultats. En effet, si l'on prend l'exemple de la base de données MNIST faite pour essayer de classer les chiffres manuscrits, la base données comporte 70 000 images. On se rend bien compte qu'il est très compliqué de constituer une telle base de données dans le temps imparti, et avec les ressources à disposition. Les résultats obtenus seront donc à nuancer au regard de ce point.

c. Préparation des données pour les phases d'entraînement et de test

Une fois la collecte réalisée et le jeu de données propre, l'étape suivante consiste à la préparation de nos images pour l'entraînement du réseau de neurones.

Dans un premier temps, il est nécessaire de redimensionner toutes les images au même format. Le fichier **loadData.py** le permet. Il suffit de modifier dans **settings.py** le chemin vers le dossier contenant les images, ainsi que la largeur et la hauteur souhaitées.

Le script contient la fonction "resize_dataset" qui prend en entrée ces trois arguments, puis procède comme suit :

- Il charge chaque image de la base de données et compare son ratio au ratio désiré : si le ratio ne convient pas, on ajoute du blanc à droite ou en bas de l'image afin de ne pas déformer l'image originale lors du changement d'échelle
- On effectue le changement d'échelle de l'image pour obtenir les dimensions voulues
- L'image est enregistrée au format **".jpg**" dans le dossier **data**
- On obtient alors un nouveau dossier **/data** dans le même répertoire, il contient les images redimensionnées au format JPG.

Dans un second temps, il faut préparer le jeu de données pour l'entraînement et le test. Grâce au script **build_dataset.py,** on sépare l'ensemble d'images contenu dans le répertoire data en :

- Un ensemble test qui sera utilisé plus tard pour tester le modèle et calculer des métriques, créé dans le répertoire /test.
- Et un ensemble d'entraînement-validation qui va être sérialisé dans un fichier Pickle **dataset.pkl**

Le pourcentage est à modifier dans le fichier settings.py, il est de 10% par défaut pour l'ensemble test, donc 90% pour l'ensemble d'entrainement. Dans cet ensemble d'entrainement, une partie est allouée à l'apprentissage, l'autre à la validation.

Il est à noter que dans le dossier parent les images sont mélangées avant de procéder à la séparation et nous nous assurons que la quantité d'image pour chaque classe dans les deux dossiers créés reste proche afin d'éviter la surreprésentation d'une classe.

Le fichier Pickle en sortie du script contient :

- Le tableau des images d'entraînement ainsi que le tableau des labels respectifs pour chaque image
- Le tableau des images de validation croisée ainsi que le tableau des labels respectifs pour chaque image

Une image correspond à un tableau de pixels et son label respectif à un tableau de 0 contenant un seul 1 suivant l'indice correspondant à la classe de l'image. Par exemple, [0,0,1] si notre BDD contient 3 classes et que l'image de l'exemple appartient à la 3ème classe.

Ces quatre objets sont sérialisés pour pouvoir être lus en entrée du réseau de neurones. Il est possible de modifier le pourcentage pour chacun des deux ensembles d'images. Par défaut, 20 % des images sont pour la validation et 80% pour l'entraînement.

B. Les réseaux de neurones convolutionnels

a. Fonctionnement des réseaux de neurones

Le réseau de neurones est un système d'apprentissage automatique (Machine Learning) s'inspirant du fonctionnement des neurones biologiques.
Ces systèmes apprennent par l'expérience et sont particulièrement adaptés à la classification, à la reconnaissance de formes ou motifs ou encore dans des domaines statistiques (bourses, économie, etc.).

Architecture d'un réseau neuronal

Le réseau de neurones est en fait un système composé de plusieurs couches empilées, comportant des nœuds (neurones) auxquels sont attribués des poids. A chaque couche i, les valeurs provenant de la couche i-1 sont multipliées par ces poids et additionnés au niveau

d'un neurone de la zone i. Il peut à cet endroit également contenir une fonction d'activation, qui permet de rendre la sortie non linéaire. Ce réseau transforme donc une entrée (souvent un vecteur ou une matrice) en une sortie (vecteur de valeurs). Au niveau de la dernière couche, le nombre de sorties est égale au nombre de valeurs de décision voulues.

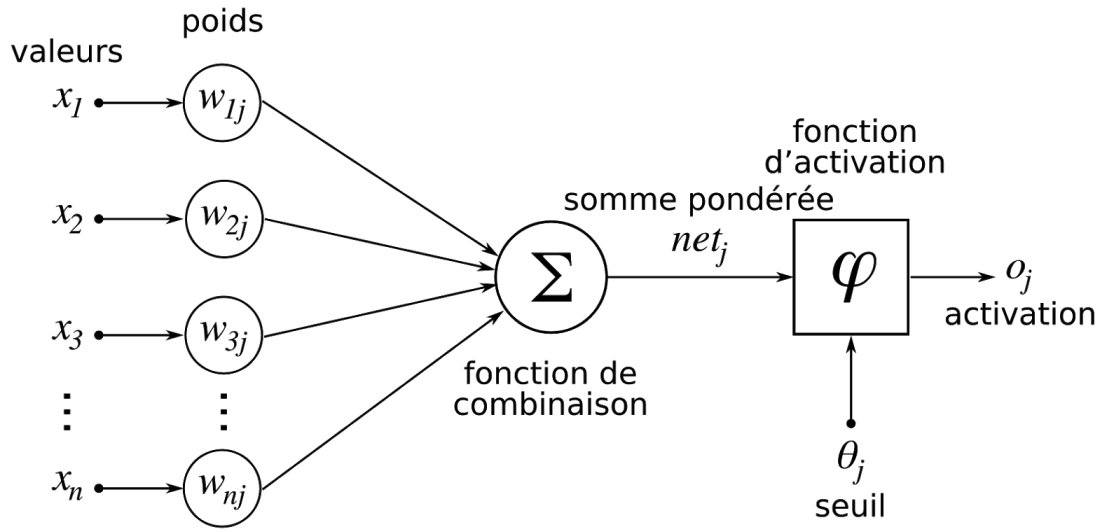


Figure 3 : Architecture d'une couche d'un réseau de neurones

Fonctionnement d'un réseau de neurone

Comme cité précédemment, le réseau de neurone se construit par expérience et itération. Il faut donc le faire apprendre, c'est la phase d'apprentissage.
Il faut alors distinguer deux cas distincts : le mode supervisé et le mode non supervisé. Dans le premier cas, le jeu de données en entrée contient des sorties connues et que l'on adresse au programme. Ainsi, il va essayer de trouver les correspondances et motifs récurrents au sein d'une même classe pour la généraliser par la suite. A l'inverse, dans le cas de l'apprentissage non supervisé, le réseau peut converger vers n'importe quelle solution finale. Il n'y a pas de classe prédéfinie.

Nous nous plaçons dans ce rapport dans le cadre de l'apprentissage supervisé, souhaitant *in fine* classer les visuels dans différentes catégories connues. Etant donné un jeu de données connu, dont on connaît les sorties, le système va alors essayer de déterminer les motifs et les caractéristiques de chaque classe. En sortie, nous obtenons un score, symbolisant la probabilité d'appartenance à une classe. Pour cela, il calcule l'erreur entre la prédiction (issue du réseau de neurones) et la valeur réelle. Une boucle de rétropropagation (backpropagation) permet alors de ramener l'erreur en sortie vers l'entrée, et en modifiant les poids à la prochaine itération essaie alors de minimiser cette erreur.

Dans le cadre de notre étude, les valeurs en entrée sont des images. Ces images sont en réalité une matrice d'entiers (pour chaque pixel), compris entre 0 et 255, et en 3 dimensions (image RGB, ou Red Blue Green, avec une dimension par couleur). C'est chacun de ces pixels qui va passer par les poids du réseau pour en donner une sortie finale.

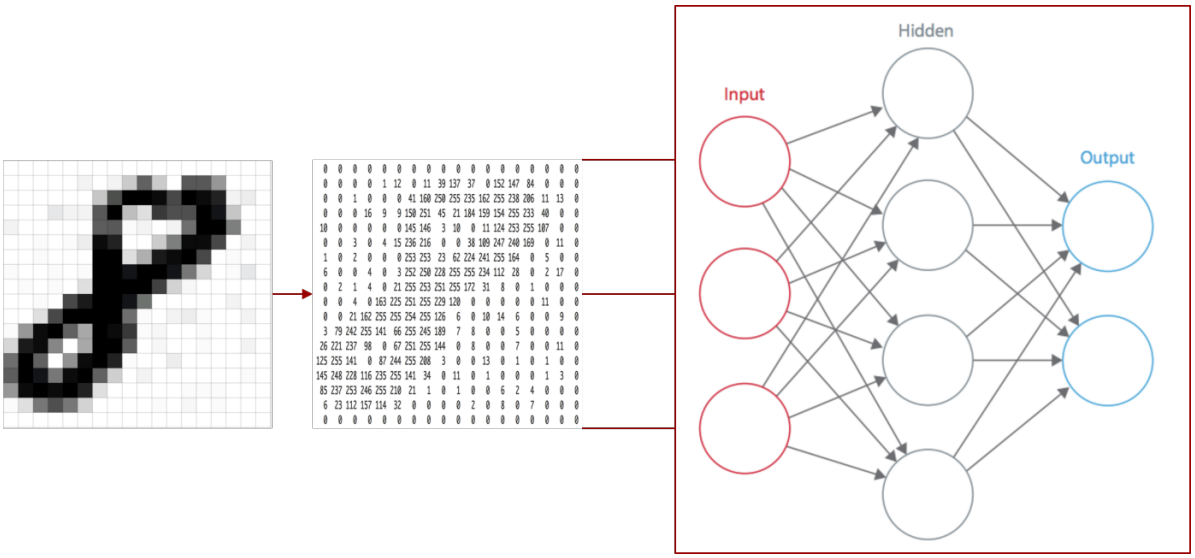


Figure 4 : Fonctionnement d'un réseau de neurones

Analogie avec les neurones biologiques :

Enfin, on parle de réseaux de neurones à cause de la ressemblance avec le système cognitif. Au niveau des synapses, les neurones reçoivent une information d'un neurone adjacent. Cette information est alors analysée puis modifier au sein du neurone et transmise au neurone suivant. C'est exactement ce qu'il se passe au sein d'une couche d'un réseau de neurones, avec l'addition des valeurs des couches antérieures pondérées de poids. La figure ci-dessus résume cette analogie.

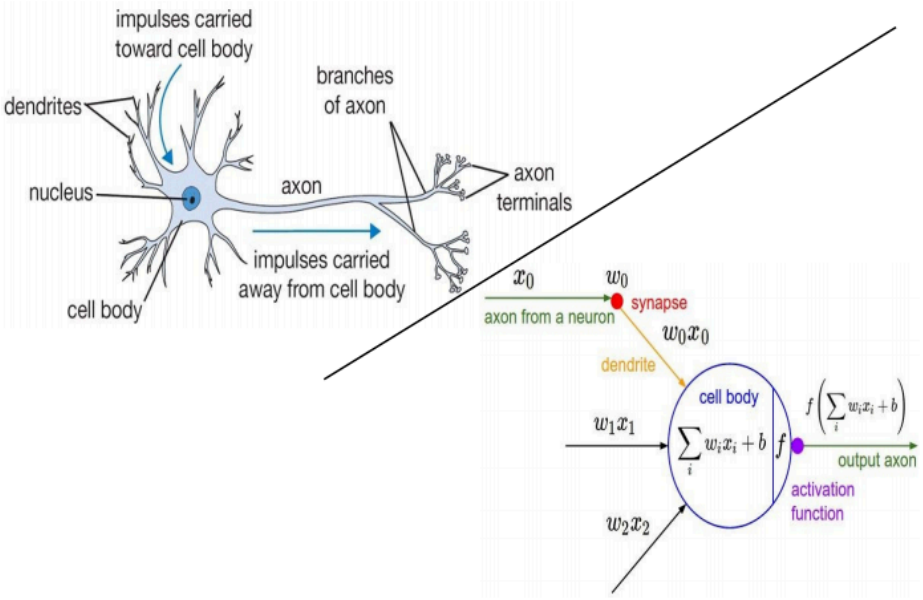


Figure 5 : Analogie avec les neurones biologiques

b. *Les réseaux de neurones convolutifs*

Au cours de notre projet, nous avons utilisé un réseau de neurones particulier, les réseaux de neurones convolutifs ou convolutionnels. Ces réseaux, inspirés du fonctionnement du cortex visuel, sont particulièrement adaptés à la reconnaissance d'images et de vidéos, et comportent une boucle de rétropropagation.

Contrairement au réseau général dans lequel chaque pixel est analysé un à un, ces derniers sont analysés par blocs (tuiles) via des filtres. Dans le cas d'un filtre de taille 5, les tuiles successives sont des carrées de 5x5. Ainsi, un pixel passera à travers un certain nombre de filtres (de taille égale à la taille de la tuile) avec ses plus proches voisins. Ce chevauchement permet de prendre en compte l'influence du voisinage. C'est ce qu'on appelle les couches de convolution. Ces couches ont une profondeur égale au nombre de filtres successifs utilisés. Voici un schéma permettant de mieux comprendre le rôle des filtres et le mode de lecture :

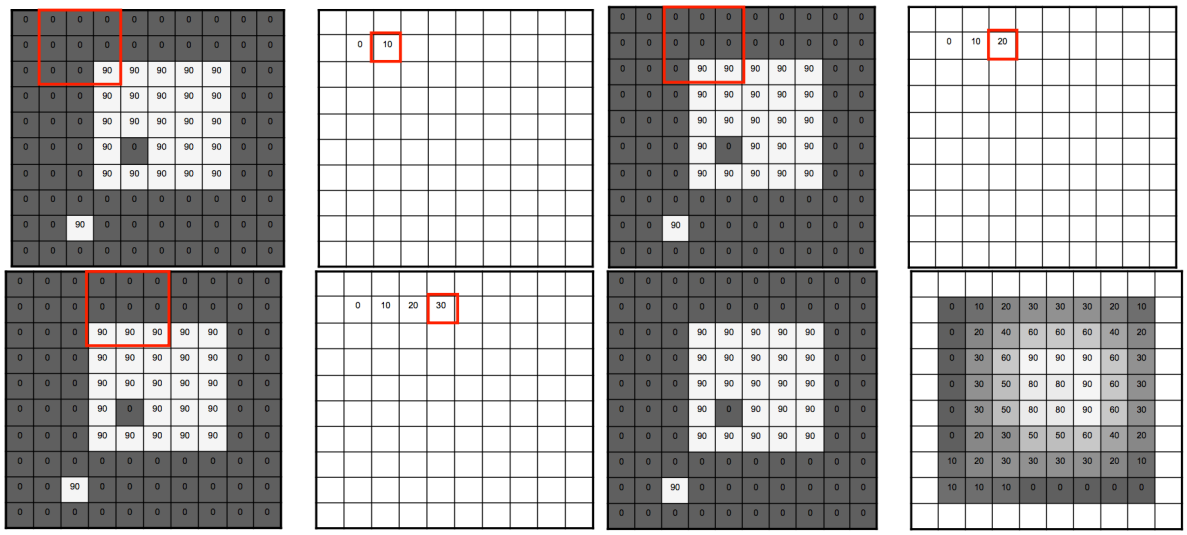


Figure 6 : Passage d'une image via un filtre au sein d'une couche de convolution

À la sortie d'une couche de convolution, peut suivre une couche de pooling (sampling, ou sous-échantillonnage), qui va permettre de réduire l'information en réduisant la taille de l'image intermédiaire, par exemple en ne prenant que la valeur maximale sur un bloc 3x3 (max-pooling). Ceci permet de réduire le sur-apprentissage (modèle qui colle trop aux données d'entrée) et de faire d'énormes gains de calculs.

Enfin, en sortie du réseau, se situent des couches entièrement connectées, semblables au réseau classique perceptron, dans laquelle tous les neurones sont connectés aux couches précédentes et suivantes. Il peut également y avoir un *dropout* à la sortie de ces couches. Cette dernière couche supprime aléatoirement une partie des valeurs (souvent la moitié) ce qui permet de réduire le sur-apprentissage.

Un des principaux avantages du réseau de neurones convolutifs est l'unicité des poids au sein d'une couche de convolution. En effet, en plus de réduire l'espace mémoire, ceci permet d'être invariant par translation.

Les deux figures ci-dessous permettent de comprendre le fonctionnement et l'architecture du réseau de neurones convolutionnels.

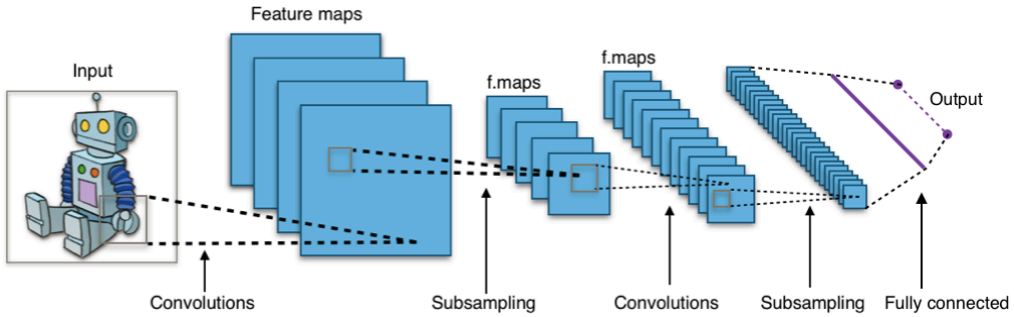


Figure 7 : Architecture générale d'un réseau neuronal convolutif

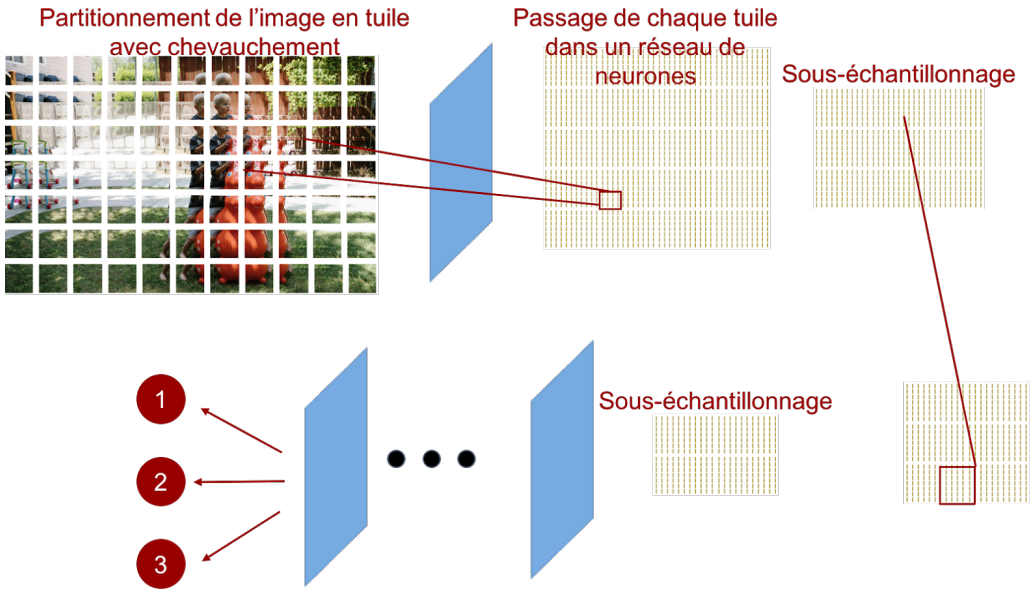


Figure 8 : Passage d'une image par un réseau de neurones convolutionnels

c. *Architecture de nos réseaux*

Si le seul moyen de déterminer l'efficacité d'un réseau de neurones convolutifs est d'entraîner le modèle associé, les expériences les plus connues nous permettent d'en déduire des préconisations pour la construction de nos réseaux. Nous avons donc opté pour des architectures respectant pour la plupart le schéma suivant :

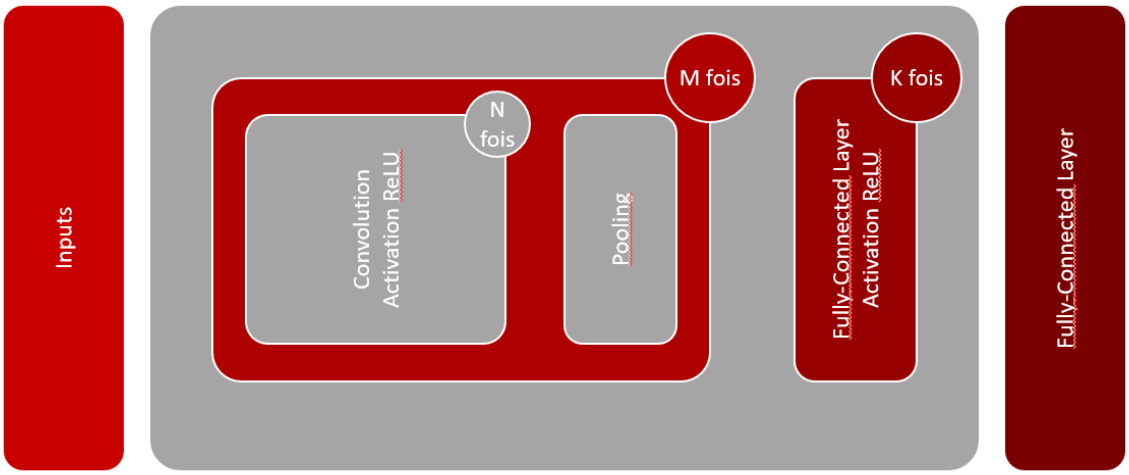


Figure 9 : Schéma classique d'un réseau de neurones convolutifs

Nous avons défini cinq architectures différentes aux caractéristiques diverses afin de tester différentes possibilités dans l'espoir de construire un modèle aussi performant que possible.

Un réseau plutôt classique

Dans ce réseau on va dans un premier temps utiliser à tours de rôles des couches de convolutions suivi d'une activation ReLU avec des couches de max-pooling (avant d'enchaîner avec des couches fully-connected et un dropout afin d'éviter le surapprentissage. A noter que la fonction ReLu ne garde que les valeurs positives (et remplace les valeurs négatives par 0). Le max-pooling quant à lui ne garde que la valeur maximale au sein d'une matrice 2*2

Architecture :

1. Convolution (32 neurones, taille du filtre 3*3)
2. Max-pooling (taille du filtre 2)
3. Convolution (64 neurones, taille du filtre 3*3)
4. Convolution (64 neurones, taille du filtre 3*3)
5. Max-pooling (taille du filtre 2*2)
6. Fully-connected (256 neurones)
7. Dropout (50% des neurones)
8. Fully-connected (Autant de neurones que de classes)

La double convolution

Pour ce second réseau, nous avons testé une architecture faisant se succéder des groupes de deux couches de convolutions suivies d'une couche de max-pooling avant des couches fully-connected.

Architecture :

1. Convolution (32 neurones, taille du filtre 3*3)
2. Convolution (32 neurones, taille du filtre 3*3)
3. Max-pooling (taille du filtre 2)
4. Convolution (32 neurones, taille du filtre 3*3)
5. Convolution (32 neurones, taille du filtre 3*3)
6. Max-pooling (taille du filtre 2)
7. Fully-connected (512 neurones)
8. Fully-connected (512 neurones)
9. Fully-connected (Autant de neurones que de classes)

L'average pooling

Ici, on teste l'average pooling: trois blocs composé d'une couche de convolution et d'une couche d'average-pooling se succèdent suivis de deux couches fully-connected et d'un dropout avant la couche fully-connected finale. L'average-pulling remplace chaque matrice 2*2 par la valeur moyenne des coefficients de cette matrice.

Architecture :

1. Convolution (32 neurones, taille du filtre 3*3)
2. Average-pooling (taille du filtre 2)
3. Convolution (32 neurones, taille du filtre 3*3)
4. Average-pooling (taille du filtre 2)
5. Convolution (32 neurones, taille du filtre 3*3)
6. Average-pooling (taille du filtre 2)
7. Fully-connected (512 neurones)
8. Fully-connected (512 neurones)
9. Dropout (50% des neurones)
10. Fully-connected (Autant de neurones que de classes)

Un réseau sans pooling

Le réseau suivant possède la particularité suivante, il ne possède pas de couches de pooling. Pour réduire les dimensions des données traitées, on utilise à la place des couches de convolutions avec un zero-padding.

Architecture :

1. Convolution (32 neurones, taille du filtre 3*3)
2. Convolution (32 neurones, taille du filtre 3*3)
3. Convolution (32 neurones, taille du filtre 5*5, zero-padding)
4. Convolution (32 neurones, taille du filtre 3*3)
5. Convolution (32 neurones, taille du filtre 3*3)
6. Convolution (32 neurones, taille du filtre 5*5, zero-padding)
7. Fully-connected (512 neurones)
8. Dropout (50% des neurones)
9. Fully-connected (Autant de neurones que de classes)

Un modèle mixte

Pour notre cinquième et dernier modèle, nous avons choisi de mêler max-pooling et average-pooling, convolution à 32 ou 64 neurones pour obtenir le réseau suivant :

Architecture :

1. Convolution (64 neurones, taille du filtre 3*3)
2. Convolution (64 neurones, taille du filtre 3*3)
3. Average-pooling (taille du filtre 2)
4. Convolution (32 neurones, taille du filtre 3*3)
5. Convolution (32 neurones, taille du filtre 3*3)
6. Max-pooling (taille du filtre 2)
7. Fully-connected (512 neurones)
8. Fully-connected (512 neurones)
9. Fully-connected (Autant de neurones que de classes)

d. TensorFlow : pourquoi ? Quelles particularités ?

TensorFlow est un outil d'apprentissage automatique développé par Google. Bien qu'il contienne un large éventail de fonctionnalités, TensorFlow est principalement conçu pour les modèles de réseaux neuronaux profonds.

Plusieurs solutions de ce type existent, on pourra notamment citer PyTorch ou Keras. Cependant, nous nous sommes tournés vers TensorFlow pour différentes raisons :

Google

La librairie ayant été développée par le géant américain, cela lui confère divers avantages :

- l'assurance d'une technologie efficace et à jour : Google étant l'une des entreprises les plus compétitives dans ce domaine, la qualité de l'outil est assurée
- une communauté importante : la communauté liée à un outil informatique importe presque autant que la qualité de l'outil lui-même, il est donc primordial d'en avoir une conséquente, une fois de plus, la réputation du géant de la Tech offre cet atout.

Différents niveaux de complexité des outils

TensorFlow offre différents outils aux complexités variables selon le niveau de détails avec lequel on souhaite modifier ou gérer son modèle. Dans notre cas, nous avons notamment utilisé les API de niveaux moyens, nous permettant de définir les différentes couches de nos réseaux nous-mêmes, gérer notre base ainsi que les indicateurs déterminés pour notre projet.

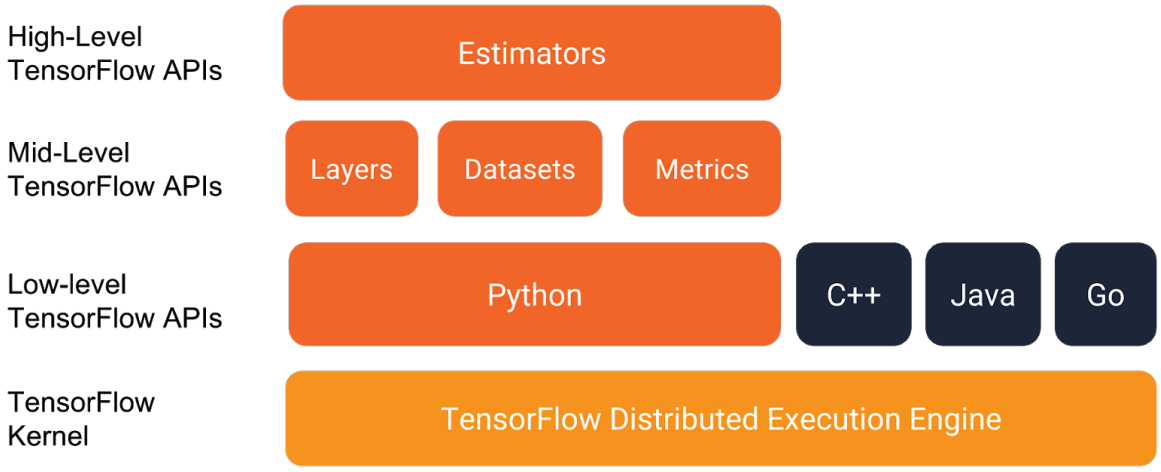


Figure 10 : Les différents de TensorFlow

3. Restitution, visualisation et interprétation des résultats

A. Structuration des résultats

Après entraînement de nos différents réseaux de neurones à l'aide des données contenue dans le fichier Pickle, nous avons pu analyser les résultats obtenus.

a. Détermination des métriques sur l'ensemble d'apprentissage

Loss et Accuracy :

Ces deux paramètres nous permettent de valider la pertinence du modèle entraîné. Une faible perte (loss), i.e. proche de 0, signifie qu'il y a peu de différence entre les prédictions et les résultats réels. Ici, on calcule une loss via la fonction softmax, qui prend en compte les scores (ici les probabilités d'appartenance à une classe).

L'accuracy (précision), à ne pas confondre avec la précision calculée pour l'ensemble de test, représente le nombre d'instances bien classées par le modèle. Une précision de 1 se traduit par un apprentissage parfait. Attention, cela ne signifie pas que le modèle aura des résultats parfaits sur l'ensemble de test.

b. Détermination des métriques sur l'ensemble de test

Valeurs de sortie :

En sortie du réseau, nous obtenons un vecteur de probabilité d'appartenance à chaque classe. La probabilité la plus grande détermine la classe à laquelle est attribuée l'image.
Il faut alors analyser ces attributions.

Matrice de confusion :

Cette matrice classe recoupent les prédictions avec les valeurs réelles. Sur la diagonale (True Positive) se situent les images bien classées. Cela permet d'avoir une vision de la précision du modèle :

		Classe prédite		
		A	B	C
Classe réelle	A	37	3	3
	B	6	37	0
	C	2	2	41

.

Figure 11 : Matrice de confusion

Accuracy :

Cette valeur calcule le nombre de d'instances bien classées. Elle permet de se rendre compte de la validité du modèle. Elle est égale à la somme des valeurs sur la diagonale divisée par la somme de tous les éléments de la table.

Recall ou rappel :

Cette valeur calcule pour chaque classe le nombre d'instances bien classées sur le nombre d'instances de la classe (True Positive / Total d'instances la classe). Cela permet de se rendre compte de la faculté du modèle à trouver ou non les éléments d'une classe.
Le rappel total d'un modèle et la somme des rappels de chaque classe divisée par le nombre de classes.

Précision :

Cette valeur calcule pour chaque classe le nombre d'instances bien classées sur le nombre d'instances attribuées à cette classe (True Positive / Total des prédictions de cette classe). Cela permet de se rendre compte de la pertinence du modèle dans la classification, i.e. à chaque fois qu'il prédit une classe, quelle est la chance pour que cette prédiction soit bonne. La précision totale d'un modèle et la somme des précisions de chaque classe divisée par le nombre de classes.

Sévérité du modèle (strictness class) :

Ce paramètre est un paramètre additionnel qui nous permet de classer mieux que le hasard. En effet, il n'attribue une instance à une classe que si cette dernière a une probabilité *x* fois supérieur au hasard (1/nombre de classes). Dans notre modèle nous avons pris 1,3. Ce choix est arbitraire mais marque la volonté de classer de plus sûrement que le hasard. Si tel n'est pas le cas, cette instance est attribuée à la classe 'uncategorized', qui regroupe toutes les instances pour lesquelles aucune classe ne se démarque.

c. Détermination des outils de visualisation adaptés

Lorsqu' un modèle entrainé est testé sur le dossier d'images test, l'ensemble des résultats et des paramètres de configuration du modèle sont sauvegardés automatiquement dans un fichier JSON, notamment les prédictions pour chaque image d'appartenir à telle ou telle classe. Ceci permet de garder une trace de nos expérimentations et de traiter ces fichiers JSON afin de représenter les résultats des expériences de manière plus intuitive.

Ainsi, à l'aide du framework D3.js, conçu pour faciliter la représentation de données sous des formats plus lisibles et exploitables analytiquement, nous avons opté pour la représentation Radar afin d'observer la distribution de probabilités pour chaque image test et voir si des tendances se dégageaient.

Voici un exemple d'image mal classé, le TreeMap a été prédit diagramme de Venn :

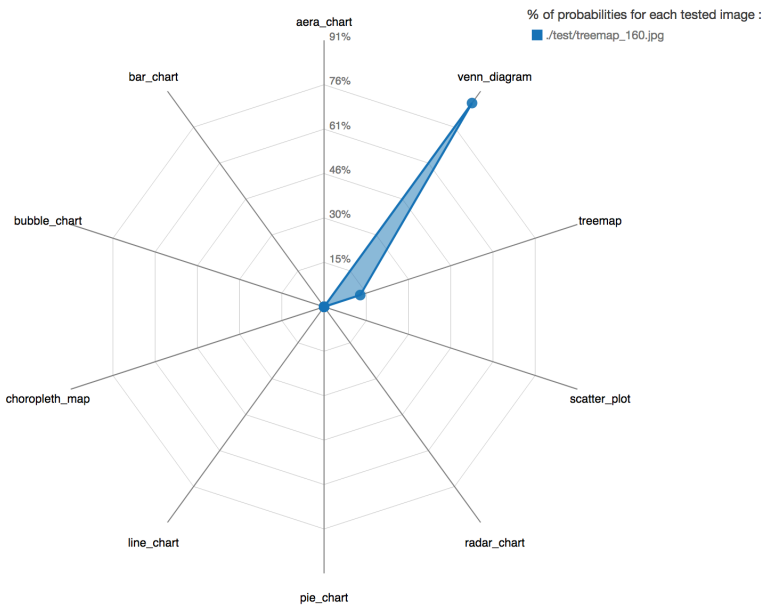


Figure 12 : Visualisation radar du vecteur des prédictions

d. Construction du site

Une partie importante de notre rendu est le site qui présente les résultats et utilise la visualisation D3.js cité précédemment.

Nous voulions non seulement montrer nos résultats pour les différents modèles mais aussi permettre à n'importe qui de pouvoir tester une image. Pour cela, nous avons décidé de développer une API (interface de programmation) REST ayant pour but de prédire pour une image donnée les probabilités pour chaque label.

Le langage utilisé ici est le même que durant tout le projet : Python. Afin de créer notre API nous utilisons le module Flask. Il permet la création d'une application Web rapidement et est parfaitement adapté à nos besoins.

L'API est constituée de deux points :

- /model : cette branche n'accepte que les méthode POST et requiert deux arguments "model" (numéro du réseau à utiliser) et "nb_class" (le nombre de labels sur lequel a été entraîné le réseau). A la suite d'une requête sur cette branche, l'API chargera le modèle voulu (qui est stocké sur le serveur) et renverra une réponse "success" lorsque celui-ci sera chargé.

- /upload : cette branche n'accepte, elle aussi, que les requêtes POST et ne requiert qu'un argument. Cet argument est un fichier image. Il sera alors uploadé sur le serveur, traité et la réponse sera un fichier JSON (objet Javascript) contenant les probabilités pour chaque classe.

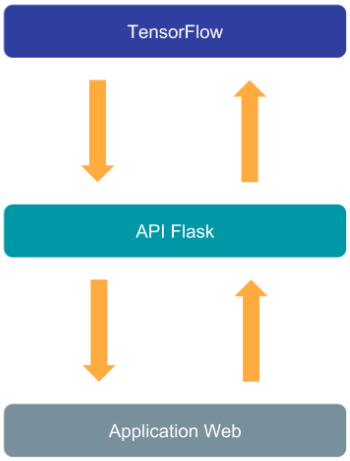


Figure 13 : Connections de l'API avec les différentes interfaces

Une fois l'API créée et testée, il faut la rendre accessible. Pour cela plusieurs possibilités s'offrent à nous :

- Un hébergement en ligne : les fichiers des réseaux sont très lourds (environ 600mb chacun) et vont donc demander un plan tarifaire élevé. Cependant ceux-ci sont souvent accompagnés de hautes performances ce qui, dans notre cas, n 'est pas nécessaire.
- Un hébergement local : le serveur tourne sur une machine chez un des développeurs et la machine est exposée de façon à être accessible via la box internet. La solution est moins à même de convenir pour une large utilisation mais se prête bien à un contexte de test.

Nous avons opté pour la deuxième solution. L'application Web permet donc de tester les différents réseaux via les images propres à l'utilisateur.

B. Analyse concrète de nos résultats

a. Influence des paramètres

Notre réseau de neurones construit, il est possible de modifier certains paramètres du réseau qui sont : le nombre d'itérations (epochs), la taille du filtre, la taille de l'image en entrée, la répartition des données d'entrée en apprentissage / test, ainsi que la vitesse d'apprentissage (learning rate). Cette dernière permet de pondérer la modification des poids avec le calcul du gradient. Avec une vitesse proche de 1, les poids seront modifiés de façon importante alors que plus on se rapproche de 0, plus la modification se fera doucement.

Itérations

Nous avons fixé le nombre d'itérations à 125 pour les différents réseaux. Nous pouvons justifier ce choix en visualisant l'évolution de la perte (loss) et de la précision (accuracy) lors de la phase d'apprentissage. Ci-dessous nous pouvons visualiser ces évolutions pour différents paramètres du réseau 1 (elles sont semblables pour les autres réseaux).

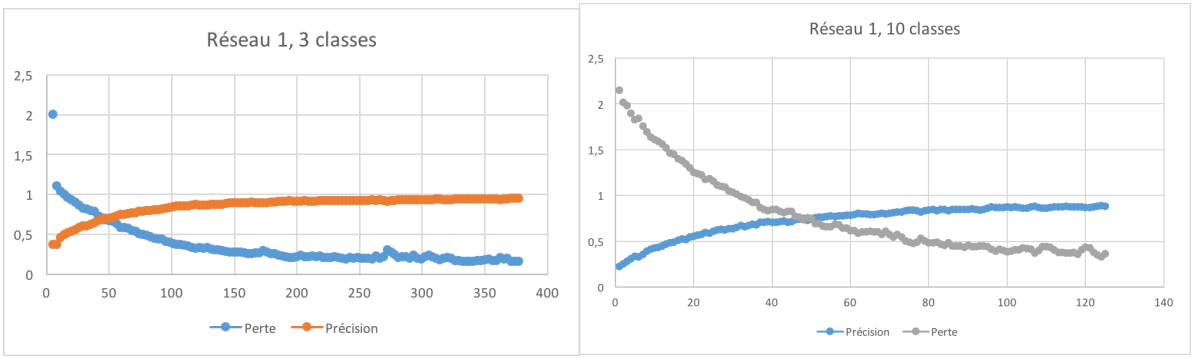


Figure 14 : Variation de la précision et de la perte en fonction du nombre d'itérations, pour différents nombres de classes

Ainsi, nous voyons bien que la perte et la précision ont atteint leurs asymptotes et donc on peut se satisfaire de ce nombre d'itération.

Taux d'apprentissage

Le taux d'apprentissage permet de définir la vitesse de convergence du modèle. Il est essentiel qu'il soit adapté puisque sinon le modèle ne peut converger, ou converge vers un résultat non optimal. Le graphe ci-dessous montre l'importance du choix d'un bon taux d'apprentissage :

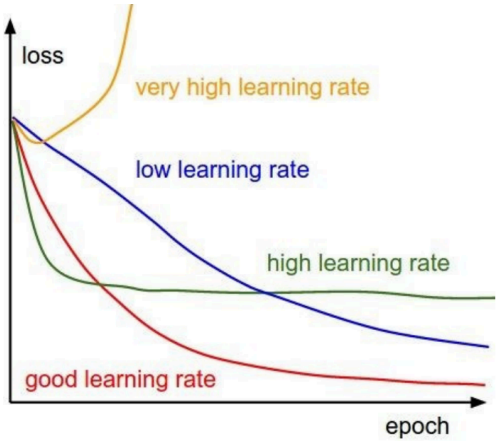


Figure 15 : Importance du choix d'un bon taux d'apprentissage

Les tests effectués nous permettent de nous arrêter sur un taux d'apprentissage de **0,001**. En effet, nous pouvons voir sur les graphiques ci-dessous la différence entre un taux trop rapide et un taux trop lent.

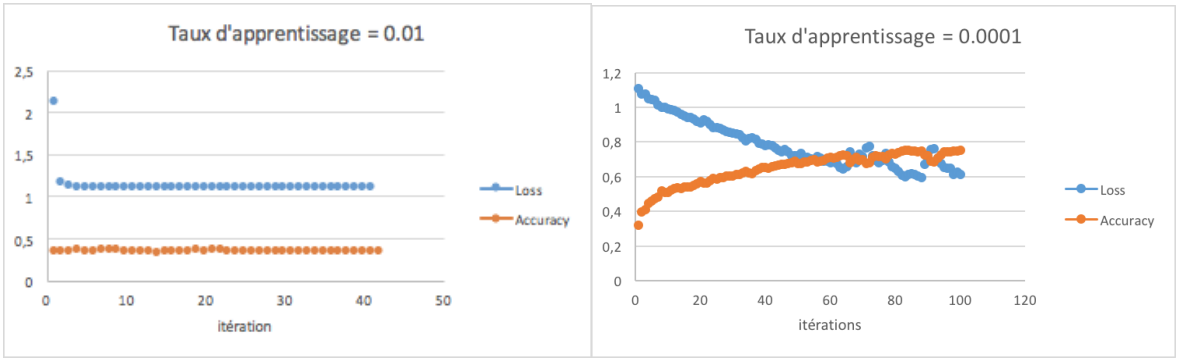


Figure 16 : Choix du taux d'apprentissage

Taille du filtre

Le filtre permet de déterminer la taille de la tuile. Via un filtre de taille 1, le réseau traitera tous les pixels indépendamment les uns des autres.

A partir du réseau 1, nous avons comparé les résultats pour différentes tailles de filtre. Il faut noter que la puissance de nos machines ne nous permet pas d'utiliser des filtres de tailles trop importantes. En effet, pour un filtre de taille 5x5, et un nombre de filtres égal à 32 au sein d'une unique couche de convolution, il y a 5x5x32 paramètres à calculer, soit 800 paramètres. Sur le réseau tout entier, cela fait plus de 10 000 paramètres à calculer à chaque itération. Il devient presque donc impossible pour nos machines de les faire tourner sur des bases de données comprenant plus de 3 catégories (donc plus de 2000 images). D'autant plus que si l'on augmente la résolution de l'image, les calculs seront d'autant plus longs.
Voici les résultats obtenus sur le réseau numéro 1 :

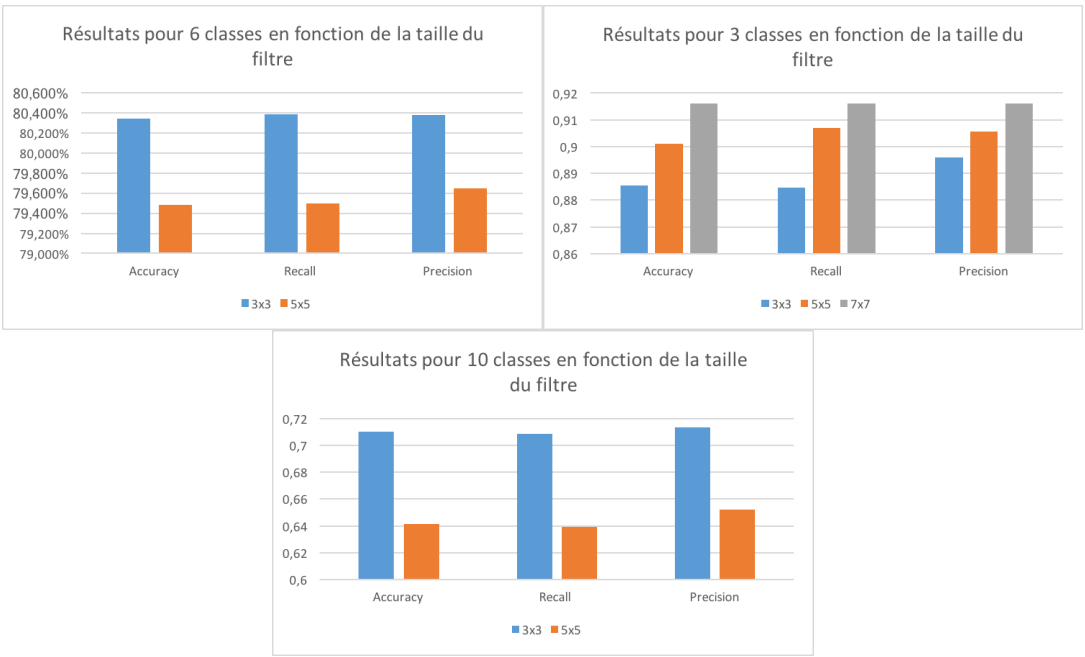


Figure 17 : Influence de la taille du filtre sur les résultats

Ainsi, nous pouvons constater que les résultats des différentes simulations ne suivent pas de logique claire : selon le nombre de classe, des filtres de taille 3x3 peuvent être plus ou moins pertinents.

Nous choisirons par la suite le meilleur modèle selon que l'on veut trouver sur trois classes (taille : 7x7), six classes (taille 3x3) ou 10 classes (taille 3x3).

Taille de l'image

La résolution de l'image va influencer les résultats attendus. On peut par exemple se convaincre qu'un modèle différenciera mieux des images de bonne qualité que des images de mauvaise qualité, où il un nuage de point et un diagramme en ligne peuvent être peu différenciables.

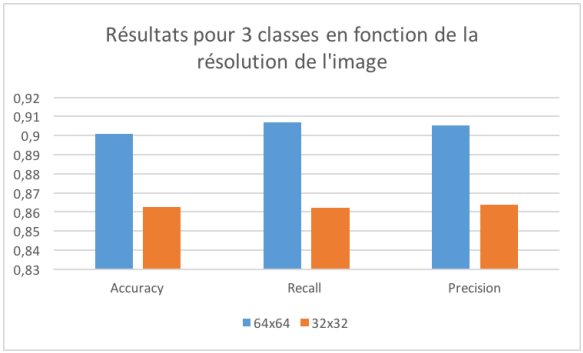


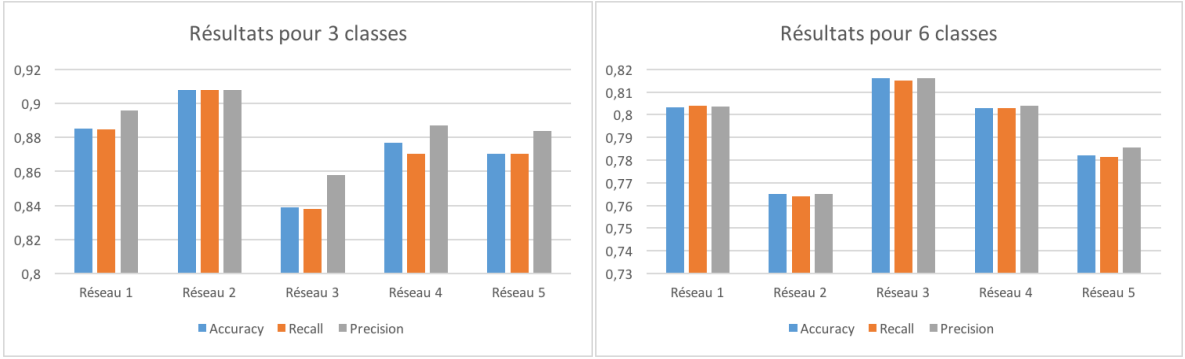
Figure 18 : Influence de la résolution des images sur les résultats

Ainsi, ce test effectué avec le premier réseau montre qu'une résolution de 64x64 donne de meilleurs résultats. Cela se retrouve pour tous les réseaux, et pour les filtres de taille 3x3 et 5x5. Cependant, il faut noter que l'on peut trouver pour certains réseaux de plus mauvais résultats avec une résolution de 128x128 qu'avec une résolution de 64x64.

b. *Influence du réseau*

Nous avons construit plusieurs réseaux pour essayer de comprendre l'influence du réseau sur les résultats. L'architecture des différents réseaux est détaillée dans la partie précédente.

Les résultats sont disponibles ci-dessous :



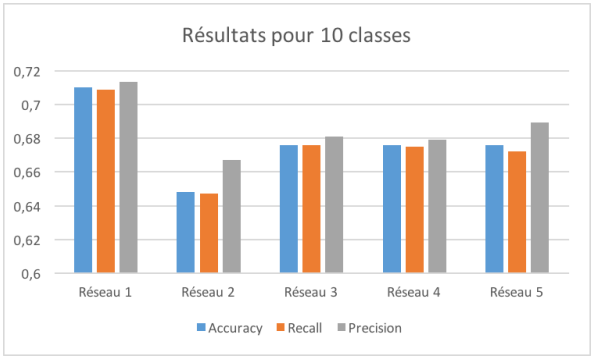


Figure 19 : Influence des réseaux sur les résultats

Ainsi, nous pouvons constater que les résultats dépendent encore une fois du nombre de classes.

Pour conclure, il est compliqué de prédire quel réseau et quels paramètres seront le mieux, avant de les tester sur les bases de données. De manière empirique, nous choisissons donc les modèles suivants :

	3 classes	6 classes	10 classes
Réseau	**1**	**3**	**1**
Taille des filtres	7x7	3x3	3x3
Taille des images	64x64	64x64	64x64
Taux d'apprentissage	0.001	0.001	0.001
Nombre d'itérations	125	125	125
Accuracy	91,60%	81,60%	71,03%

Figure 20 : Choix du meilleur modèle en fonction du nombre de classes à prédire

Bien entendu, pour généraliser le modèle, nous pouvons garder que le modèle à 10 classes, mais la précision sera plus faible. Si l'on est certain d'avoir un visuel de datavisualization dans les modèles à six classes ou trois classes, nous pourrons utiliser les modèles restreints qui apportent de meilleures performances.

Conclusion

Ainsi, après un premier travail d'investigation sur les méthodes employées pour la classification de visuels, nous avons fait le choix d'utiliser les réseaux de neurones et plus particulièrement les réseaux de neurones convolutionnels qui sont désignés comme les plus adaptés par les spécialistes du secteur. Pour nos différents modèles, nous avons appris à maîtriser des outils de dernière génération développés par des entreprises à la pointe.

Afin d'explorer les possibilités offertes par les réseaux de neurones convolutionnels, nous avons mis en place cinq réseaux aux structures différentes. Les règles d'architecture d'un réseau de neurones ne sont pas précisément définies mais ils existent des bonnes pratiques qui nous ont guidées lors de la mise en place de nos réseaux.
Si l'on peut penser que les performances d'un réseau par rapport à un autre sont difficiles à prévoir, ce projet nous l'a confirmé puisque nos résultats nous montrent qu'en fonction des caractéristiques des bases de données (taille des images, nombre de catégories…) le réseau le plus performant et les paramètres associés varient.

Cependant, nos résultats restent à nuancer pour deux raisons principales : notre base de données contenant moins de 500 images pour chacune des catégories, cela peut ne pas être suffisant pour permettre aux réseaux d'apprendre de façon suffisamment abstraite (la base de données MNIST comporte 7 000 images par catégorie !). De plus la puissance limitée de nos ordinateurs personnels nous a obligé à réduire la taille des filtres, ou la résolution des images et ainsi perdre des informations.

Finalement, nous avons pu mettre en place un réseau de neurones convolutionnels pouvant être réutilisés aisément via une plateforme dynamique qui permet de classifier une image en la faisant passer par nos réseaux, tout en étant conscients des limites de ce dernier liées aux moyens déployés pour ce projet.

4. Annexes

Annexe I : Guide d'installation et d'utilisation

Lancement

Afin d'installer l'ensemble des packages nécessaires au fonctionnement du script, installer le module pip (Page officielle) et exécuter la commande suivante :

```
pip3 install -r requirements.txt
```

Dans le terminal, placez vous dans le dossier que vous venez de clôner. Pour lancer le script, tapez :

```
python3 main.py
```

Après l'exécution du script, un fichier "results_<date_de_creation>.json" est créé. Celui-ci contient la date d'exécution, les paramètres utilisés ainsi, les différentes probabilités de label pour chaque image et des métriques sur chaque classe (précision et recall).

Ce fichier sera utilisé pour la visualisation des résultats à l'aide du framework d3.js.

Scrapping

Nous utiliserons le script **google-scrapper_2.0.py** issu du git image-scrapers afin de peupler notre base de données. Celui-ci permet de récupérer des images issues de google et fournit un JSON associé.

Afin d'installer l'ensemble des packages nécessaires au fonctionnement du script, installer le module pip (Page officielle) et exécuter la commande suivante :

```
pip3 install -r requirements-scraper.txt
```

Afin d'utiliser le script de scrapping, il faut:

- installer le driver chrome correpondant à la version du navigateur GoogleChrome présente sur l'ordi (Page officielle)
- modifier la ligne 24 du script et renseigner l'emplacement du driver

```
browser = webdriver.Chrome(executable_path=r'C:/Users/Desktop/chromedriver.exe')
```

Après modification du script, il faut remplacer les espaces de la requêtes par des "_", par exemple pour scrapper les images de "bar chart", on lancera la commande suivante:

```
python3 google-scrapper_2.0 bar_chart
```

Ce script nous permet de récupérer environ 400 images exploitables par classe et d'avoir des images correctement nommées pour faire tourner nos algorithmes.

Comment utiliser nos scripts ?

Le fichier <main.py> permet de lancer les scripts :

- **loadData.py** traite les images brutes et les redimensionnent à la taille voulue en JPG.

- **build_dataset.py** sérialise les images de train-validation et leurs labels correspondant dans un fichier Pickle.
- **reseau.py** définit les différents modèles de réseaux de neurones que l'on peut choisir dans le fichier **settings.py**
- **neuralnetwork.py** permet d'entraîner le modèle choisit à partir des données issues du fichier pkl et enregistre le modèle entraîné.
- **prediction.py** utilise le modèle entraîné sur un ensemble test créé au préalable et enregistre les métriques dans un fichier JSON.
- **result.py** permet d'initialiser la structure de notre fichier JSON, avec notamment tous les paramètres choisis pour le lancement des scripts.

Annexe II : Champs de publication : Sur quelles conditions / comment nous citer ?

Mots clés : Visualisation de données, Data vizualisation, classification, Machine Learning, CNN, Réseaux de neurones.

Champs de publication :
Notre étude peut être cité et les résultats peuvent être utilisé dans le champs de la recherche académique et/ou scientifique. Toute autre utilisation, y compris une reprise dans la presse, doit être soumise à l'accord du commanditaire ou de l'ensemble des élèves constituant l'équipe projet.

Formalisme de citation
Deux solutions pour citer au fil d'un texte écrit :

- note de bas de page reprenant la référence complète ci-dessous et précisant la ou les pages utilisées à l'appui de votre démonstration (+ bibliographie en fin de texte)
- Référence abrégée ci-dessous entre parenthèses à la suite du texte, faisant référence à une bibliographie en fin de texte

Toute référence à notre travail se doit donc de le citer selon la suggestion de formalisme définit ci-dessous :

Référence complète :
A. BRUGIÈRES, A. MARTIN-DELAHAYE, L. GUÉRY, L. KRAEMER, M. ARNAL, « Collecte et classification de visualisation de données », Rapport de projet informatique sous la direction de R. VUILLEMOT, Lyon, Ecole Centrale de Lyon, mars 2018.

Référence abrégée :
A. Martin-Delahaye et al. « Collecte et classification de visualisation de données » (2018), Ecole Centrale de Lyon.