

École Polytechnique de l'Université de Tours  
64, Avenue Jean  
Portalis 37200  
TOURS, FRANCE  
(33)2-47-36-14-14  
[www.polytech.univ-tours.fr](http://www.polytech.univ-tours.fr)

# Parcours des Écoles d'Ingénieurs Polytech

## 2021 / 2022

### Initiation à l'Intelligence Artificielle

Étudiant(e)

---

**Enzo CREUZET**  
**Maxime BROSSILLON**  
[enzo.creuzet@etu.univ-tours.fr](mailto:enzo.creuzet@etu.univ-tours.fr)  
[maxime.brossillon@etu.univ-tours.fr](mailto:maxime.brossillon@etu.univ-tours.fr)

Encadrant

---

**Jean-Yves RAMEL**  
[jean-yves.ramel@univ-tours.fr](mailto:jean-yves.ramel@univ-tours.fr)



# Avertissement

Ce document a été rédigé par **Enzo Creuzet** et **Maxime Brossillon** susnommés les auteurs. L'École Polytechnique de l'Université François Rabelais de Tours est représentée par **Jean-Yves Ramel** susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non-respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'appropriier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



# Table des matières

Introduction	4
Explication du sujet	4
Qu'est-ce que l'OCR et l'HTR	4
<hr/>	
1 – Recherches	5
1.1 – Qu'est-ce qu'une intelligence artificielle ?	5
1.2 – La base de données	7
<hr/>	
2 – Découverte de Tensorflow	8
2.1 – Notions élémentaires	8
2.1.1 – Les tenseurs	8
2.1.2 – CPU ou GPU ?	9
2.2 – Création d'un modèle d'apprentissage	10
2.3 – Classification d'images	14
<hr/>	
3 – HTR	16
3.1 – Outils utilisés	16
3.2 – Simple-OCR	17
3.3 – Changement de dataset	18
3.3.1 - A-Z Handwritten Alphabets dataset	18
3.3.2 – Adaptation de dataset	19
<hr/>	
Conclusion	20



# Introduction

## Explication du sujet

Le projet se dénomme "Initiation à l'Intelligence Artificielle en développant des scripts de classification d'images en python" : son objectif initial était de découvrir les techniques d'intelligence artificielle permettant de faire de l'apprentissage automatique au travers de recherches documentaires et du développement de plusieurs petits programmes réalisant de la reconnaissance automatique de contenus d'images.

Ce sujet étant vaste, nous avons préféré nous concentrer sur un seul type de reconnaissance automatique, la Reconnaissance Optique de Caractères (ROC), nous utiliserons pour la suite de ce rapport son équivalent anglais, OCR (Optical Character Recognition). Ici nos recherches se sont portées essentiellement sur l'HTR (Handwritten Text Recognition) c'est-à-dire la reconnaissance d'écriture manuscrite. Le but étant de travailler sur la technologie d'OCR, mais aussi sur celui de la manipulation d'image. Nous avons utilisé pour la totalité de ce projet le langage python.

## Qu'est-ce que l'OCR et l'HTR ?

L'OCR consiste à analyser et reconnaître le contenu d'un document contenant du texte. Cette technologie est utilisée dans de nombreux domaines, par exemple la reconnaissance d'une plaque d'immatriculation par un radar automatique, ou bien pour récupérer le contenu textuel d'un document scanné.

L'HTR, qui est une sous-catégorie de l'OCR, suit le même principe, mais s'applique à l'écriture manuscrite. Cela rend l'analyse bien plus complexe, car il existe une infinité de styles d'écriture, chacun possède une police qui lui est propre. Cette technologie moins fiable, car plus sujette à rencontrer des caractères qu'elle ne pourra pas reconnaître, est néanmoins utile pour certaines tâches, comme la classification du courrier dans le centre de tris, permettant un gain de temps considérable.

# 1

## Recherches

Lors des premières semaines, nous avons effectués un grand nombre de recherches, dans le but de mieux comprendre et nous familiariser avec la notion d'intelligence artificielle (IA).

### 1.1 – Qu'est-ce qu'une intelligence artificielle ?

Une IA est une technologie conçue dans le but de simuler l'intelligence humaine. Cette dernière est aujourd'hui présente dans de nombreux domaines, les principaux étant les technologies high-techs comme les moyens de communication et l'énorme traitement de données que leurs échanges impliquent, le domaine militaire, le domaine des transports.

Dans le cadre de notre projet, nous nous sommes principalement intéressés aux IA dédiés à la classification d'images, autrement dit, une IA capable, selon les caractéristiques propres – ou descripteurs – d'une image, de déterminer son appartenance, ou non à un groupe d'éléments similaires.



Pour donner un exemple concret, si l'on souhaite classer cette image de tomates selon sa variété, instinctivement nous allons observer sa couleur, la taille, la forme, etc... l'IA va reproduire ce raisonnement en analysant ces différents descripteurs permettant cette classification.

Figure 1 – exemple d'image pouvant être classifiée

Afin d'être capable d'effectuer cette classification, l'IA va devoir passer par une phase d'apprentissage. Il existe deux types d'apprentissages :

-L'apprentissage non supervisé, on commence avec éléments sans classification, et on cherche à les regrouper en créant des classes selon leurs points communs. L'IA va déterminer elle-même les points communs permettant la classification. Cette méthode est assez imprévisible, si l'on présente à l'IA une liste de photographies de chiens et de chats à classer, elle peut tout à fait faire une classification en fonction de si les clichés ont été pris à l'intérieur ou à l'extérieur.

-L'apprentissage supervisé, pour lequel on a déjà des classes d'éléments bien définies, et l'on entraîne l'IA à reconnaître la classe de chaque élément, afin d'être capable d'en faire de même avec de nouveaux éléments.

Un exemple de méthode connue pour la classification, la méthode des k plus proches voisins : pour la résumer, on a sur un espace de dimension n, avec n le nombre de descripteurs, des éléments déjà classifiés, placés en fonction du poids qu'ils représentent vis-à-vis de chacun de ces descripteurs. Puis l'on cherche à classer un nouvel élément. Pour cela on va le placer dans cet espace selon les mêmes critères que les précédents, puis observer dans un rayon de la taille voulue, quels éléments sont à l'intérieur de ce rayon d'observation. Si les voisins du nouvel élément appartiennent à plusieurs classes, on regarde pour quelle classe l'élément a le plus de voisins, et on en déduit que l'élément appartient à cette classe.

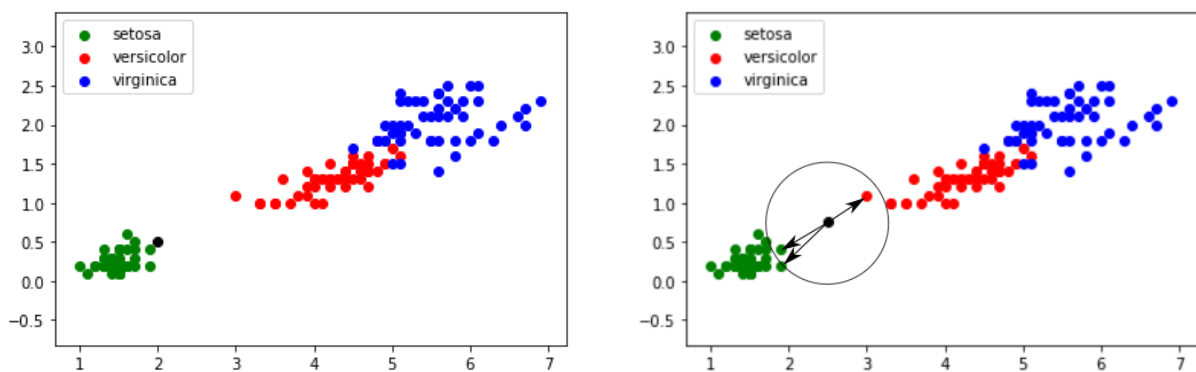


Figure 2 – deux exemples d'application des k plus proches voisins

Sur ces deux exemples, on classifie des iris, l'axe des abscisses correspond à la longueur des pétales, et celui des ordonnées à leur largeur, le point noir représente l'élément à classer. Pour le premier, il est évident que c'est une iris setosa. Pour le second en utilisant la méthode des k plus proches voisins, avec  $k=3$ , l'algorithme classera cet élément avec les setosas.

De plus en plus, on parle d'IA faible et forte. Une IA faible est une intelligence artificielle conçue pour la réalisation d'une tâche spécifique. Tandis que l'IA forte est une IA ayant pour but de se rapprocher au maximum de l'intelligence humaine, en étant une intelligence sensible. Ce type d'IA n'existe pas encore à proprement parler, ou seulement sous forme de prototype. Les intelligences artificielles dont bénéficient les assistants vocaux comme Alexa ou Siri, bien qu'évolutives car capable de s'adapter d'elles-mêmes à leur utilisateur, sont des IA faibles.

## **1.1 – La base de données**

Dans la suite de ce rapport, nous allons essentiellement traiter d'apprentissage supervisé. Comme expliqué précédemment, pour ce type d'apprentissage on détermine manuellement les classes à identifier. Puis pour chaque nouvel élément traité on cherche à déterminer sa classe.

Afin d'effectuer cet apprentissage, l'IA va s'entraîner en utilisant une base de données, ou dataset, cet entraînement s'effectue plusieurs fois afin d'affiner la précision de l'IA, on va parler d'étapes, ou epoch en anglais. Le choix du nombre d'epoch lors de la phase d'entraînement est important, car une IA pas assez entraînée aura une précision trop faible, mais une IA trop entraînée sera trop fidèle à son modèle, et sera incapable de classer de nouveaux éléments, c'est le sur-apprentissage. C'est un ensemble de données déjà classifiées, que l'on peut diviser en trois parties :

- la base d'apprentissage : cette partie est la plus conséquente en quantité de données, elle est utilisée pour la phase d'entraînement, pendant laquelle le programme va apprendre à reconnaître les caractéristiques de chacune des classes.

- la base de validation : elle permet d'ajuster la qualité du modèle après son entraînement, cette base est notamment utile pour effectuer un arrêt anticipé de l'entraînement, avant la réalisation du nombre d'epochs défini, afin d'éviter le sur-apprentissage.

- la base de test : cette dernière n'a pas été utilisée lors de l'apprentissage du programme, étant composée d'éléments inconnus de l'IA, elle permet ainsi l'évaluation de la qualité du modèle, en mesurant ses performances.

# 2

## Découverte de Tensorflow

### 2.1 – Notions élémentaires

#### 2.1.1 – Les tenseurs

En guise d'introduction, le tutoriel que nous avons suivi abordait quelques notions qu'il serait bon de qualifier d'essentielles. La première est la notion de tenseurs. En mathématiques, un tenseur est un objet à une ou plusieurs dimensions, à valeur dans un espace vectoriel. Avec Tensorflow, les tenseurs ont 4 dimensions, ou rangs ; de 0 à 3.

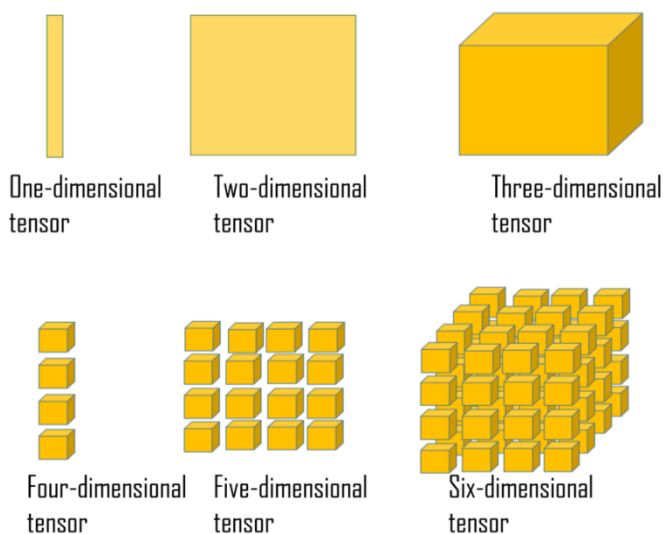


Figure 3 – différents tenseur, de dimension 1 à 6

Un tenseur de rang 0 correspond simplement à un nombre, un tenseur de rang 1 est représenté par une liste de nombres, un tenseur de rang 2 est une matrice, représentée généralement par une liste de listes de nombres, et enfin un tenseur de rang 3 est une « liste de matrices », qui représente un objet dont la forme est exprimée avec 3 nombres (ces trois nombres étant le nombre de matrices qui constituent la liste, puis la taille d'une matrice (en colonnes puis lignes)).

Afin de commencer de la partie dédiée à l'IA avec les bases du calcul entre tenseurs, nous avons suivi une série d'exemples d'opérations mathématiques, appliquées en python à des tenseurs :



Tout d'abord, on peut convertir des tenseurs en tableaux Numpy (une bibliothèque python très utile pour les calculs complexes), les ajouter ou les multiplier entre eux (pour l'addition d'un tenseur de rang 0 à un tenseur de rang 1, la valeur du tenseur de rang 0 était ajoutée à chaque élément du tenseur de rang 1), ou même faire un produit matriciel entre deux tenseurs de rang 2, qui agissaient alors comme des matrices (avec les restrictions usuelles du produit matriciel qui s'appliquaient, notamment à propos de la taille des matrices).

### **2.1.2 – CPU ou GPU ?**

Il est également intéressant de noter que les tenseurs sont chargés par le processeur (CPU) dans la mémoire de l'ordinateur par défaut, jusqu'à ce qu'une opération lui soit appliquée (addition ou multiplication), auquel cas le tenseur résultat sera chargé sur la mémoire dédiée au processeur graphique (GPU) de l'ordinateur.

En effet, l'architecture de ces deux processeurs est très différente, un CPU possède quelques cœurs conçus pour effectuer rapidement une liste de tâches spécifiques, tandis qu'un GPU possède beaucoup plus de cœur conçu pour effectuer simultanément beaucoup de calculs. Cette capacité rend ce dernier beaucoup plus rapide pour la plupart des calculs dédiées à l'intelligence artificielle. Ainsi, des bibliothèques spécifiques à ces calculs ont été développées, Tensorflow utilise l'une d'elles nommée CUDA® Deep Neural Network library (cuDNN).<sup>^</sup>

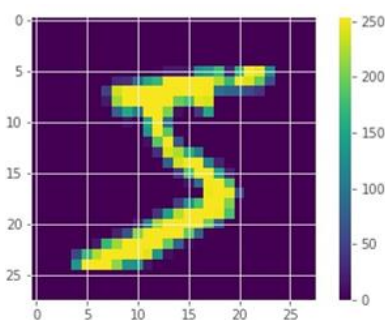
Pour comparer cette différence de performances, nous avons essayé d'entraîner le modèle abordé dans la partie suivante sur un CPU Ryzen 5 2600, puis sur un GPU RTX 2060. Une epoch de l'entraînement effectuée par le CPU prenait en moyenne 200 secondes, tandis qu'il n'en fallait que 10 secondes pour le GPU.

Les calculs peuvent être réalisés autant par l'un que l'autre, mais l'on peut, avec la fonction `tensorflow.device()`, spécifier sur quel processeur un tenseur doit être chargé, le CPU ou le GPU. Certaines opérations ne nécessitent pas forcément la puissance du GPU et, dans ce cas, le CPU suffira et le tenseur résultat sera chargé sur le CPU, même après une opération.

## 2.2 – Création d'un modèle d'apprentissage

Pour notre premier réseau de neurones, nous avons utilisé une base de données constituée de chiffres en écriture manuscrite, la « Digit MNIST Dataset ». Tout d'abord, à la différence de la plupart des bases de données qu'on peut utiliser, celle-ci est directement disponible via Tensorflow sans aucune installation nécessaire, ce qui est un avantage conséquent, certaines bases de données pouvant dépasser plusieurs Giga-octets.

La suite du tutoriel était relativement linéaire, avec des instructions étape par étape sur la marche à suivre et le code associé : tout d'abord, en observant une image correspondant à un des chiffres de la base d'apprentissage, on voit une variété de couleurs allant du bleu et le jaune formant un 5. Une valeur entre 0 et 255 déterminants la couleur du pixel.



Le problème de cette image était la valeur des pixels ; pour s'assurer que le réseau de neurones fonctionne comme convenu, l'image doit être en nuances de gris, avec des intensités allant de 0 à 1 : il fallait donc diviser la valeur des pixels par 255 pour obtenir la valeur souhaitée. Après cet arrangement effectué, on observe à nouveau certains chiffres de la base d'apprentissage, voici ce que l'on obtient:

Figure 4 – Une image avant traitement

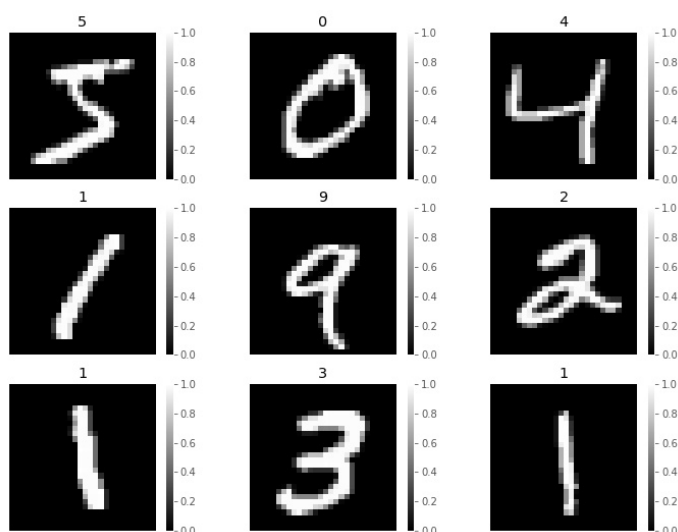


Figure 5 – Plusieurs images après traitement

Les images sont à présent en noir et blanc avec des pixels dont la valeur est située entre 0 et 1, et on peut à présent commencer la suite : la mise en place et l'apprentissage du réseau de neurones. La mise en place était relativement simple : il n'y avait que 2 couches. La première étape avant d'entrer dans la première couche était d'adapter les images pour qu'elles fassent 28 par 28 pixels. La raison pour cette modification était que le modèle créé pour ce tutoriel utilisait des couches de type « Dense » : dans un réseau de neurone, une couche dense est une couche ancrée à la couche précédente. Ce que cela signifie, c'est que les neurones d'une couche dense sont reliés à tous les neurones de la couche précédente. La première couche après cette modification était donc une couche dense avec 128 unités, ce qui signifie que l'objet en sortie de cette couche sera de forme (None, 128) lorsqu'on voudra analyser le modèle via Python. La seconde couche sert de sortie au modèle et est également une couche dense. Celle-ci n'a cependant que 10 unités étant donné que pour ce modèle en particulier nous n'avons que 10 classes : les 10 chiffres de 0 à 9. On peut voir ceci en affichant le résumé du modèle :

```

1.  Model: "sequential_2"
2.  -----
3.  Layer (type)                Output Shape          Param #
4.  -----
5.  flatten (Flatten)           (None, 784)           0
6.  -----
7.  dense_4 (Dense)              (None, 128)           100480
8.  -----
9.  dense_5 (Dense)              (None, 10)            1290
10. -----
11. Total params: 101,770
12. Trainable params: 101,770
13. Non-trainable params: 0
14. -----
15. None

```

Figure 6 – Résumé du modèle d'apprentissage

D'abord, on voit chaque couche et leur type, avec à côté la forme de l'objet qui sort de chaque couche et enfin le nombre de paramètres. On remarque que la première « couche » ne sert qu'à adapter les images et par conséquent n'a pas de paramètres.

La prochaine étape est donc de compiler le modèle. On fait cela avec la méthode `.compile()` en spécifiant certains paramètres : quel type d'optimisation on utilise, et on indique la fonction de perte et quelle unité de mesure on utilise.

```

1. model.compile(
2.     optimizer='adam',
3.     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
4.     metrics=['accuracy']
5. )

```

Figure 7 – Fonction de compilation d'un modèle

On a ici l'exemple du tutoriel, où l'on utilise l'optimisation d'Adam, où la fonction de perte est « Sparse Categorical Cross Entropy » et l'unité utilisée est accuracy, la précision. L'optimisation d'Adam est une méthode de descente de gradient stochastique basée sur l'estimation adaptative des moments de premier et de second ordre, qui sert donc à minimiser la « fonction objectif », à savoir donner le meilleur résultat possible. La fonction « Sparse Categorical Cross Entropy », quant à elle, est une fonction de calcul de perte utilisée dans des processus de classification à plusieurs classes. On note enfin que la « perte » représente la différence entre le résultat théorique donné par les classes des éléments et le résultat expérimental obtenu après entraînement du modèle.

Une fois le modèle compilé, on peut enfin commencer l'apprentissage. Cela se fait avec la méthode `.fit()`, qui sert à lancer l'entraînement du modèle sur un certain nombre d'epochs (à préciser dans les parenthèses quand on appelle la méthode) en spécifiant la base d'apprentissage d'où viendront les images, les classes auxquelles les images sont attachées, et enfin le nombre d'epochs. Ici, l'apprentissage se fait donc sur la base constituée de chiffres écrits à la main, donc les classes sont les chiffres de 0 à 9, avec 10 epochs. Après l'apprentissage, on a un dictionnaire dont les clés sont la perte et la précision de l'apprentissage et sont associées aux valeurs de perte et de précision pour chaque epoch. On peut donc extraire ces valeurs et en tracer les graphiques :

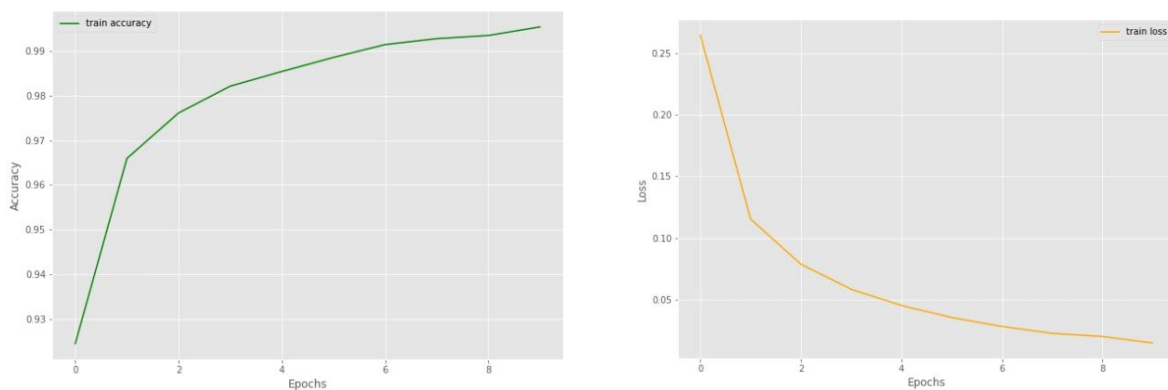


Figure 8 – Graphiques représentant l'évolution de la précision ainsi que de la perte d'un apprentissage

Le premier graphique représente la précision et le second représente la perte. On peut voir que la perte diminue au fur et à mesure tandis que la précision de l'apprentissage augmente avec le nombre d'époques. On peut comparer cette précision avec la précision obtenue sur la base de test, que nous n'avons pas encore utilisée, pour voir si le modèle est performant ou s'il ne fait qu'apprendre chaque chiffre par cœur. On va donc utiliser la méthode `.evaluate()` en spécifiant quelle liste de valeurs on utilise, ici la base de test, et quelles classes on a, ici les chiffres de 0 à 9. On a comme résultat après exécution du code une précision de 97,690 % et une perte de 0,084 sur la base de test. C'est un résultat très correct pour 10 époques, d'autant plus que le modèle utilisé est relativement simple.

Enfin, on peut utiliser, avec le module Scikit-Learn, la fonction `classification_report()` qui nous permet d'avoir la précision de chaque classe individuellement et la moyenne des précisions lors de l'application du modèle sur la base de validation. On peut ainsi observer pour chacun des chiffres la précision de l'entraînement effectué.

1.	Class 0: Digit 0				
2.	Class 1: Digit 1				
3.	Class 2: Digit 2				
4.	Class 3: Digit 3				
5.	Class 4: Digit 4				
6.	Class 5: Digit 5				
7.	Class 6: Digit 6				
8.	Class 7: Digit 7				
9.	Class 8: Digit 8				
10.	Class 9: Digit 9				
11.		precision	recall	f1-score	support
12.					
13.	0	0.99	0.99	0.99	980
14.	1	0.99	0.99	0.99	1135
15.	2	0.97	0.99	0.98	1032
16.	3	0.96	0.99	0.97	1010
17.	4	0.99	0.97	0.98	982
18.	5	0.98	0.97	0.97	892
19.	6	0.98	0.97	0.98	958
20.	7	0.97	0.98	0.98	1028
21.	8	0.98	0.96	0.97	974
22.	9	0.98	0.97	0.97	1009
23.					
24.	accuracy			0.98	10000
25.	macro avg	0.98	0.98	0.98	10000
26.	weighted avg	0.98	0.98	0.98	10000

Figure 9 – Tableau récapitulant les résultats de l'application du modèle sur la base de validation

On voit qu'on arrive finalement à une précision moyenne de 0,98, à savoir 98 %, avec des précisions par classe allant de 97 à 99 %.

## 2.3 – Classification d'images

La suite du tutoriel portait sur la classification d'images. Le départ est le même que pour notre premier réseau de neurones, il faut trouver une base de données contenant les images que nous allons utiliser. Ici, nous avons utilisé la base « 10 Monkey Species » disponible sur le site Kaggle. On a donc une base constituée de 10 classes, représentant différentes espèces de singes. La base contient les détails à propos des numéros et noms des classes, des noms scientifiques (latins) et usuels (en anglais) des singes, et le nombre d'images de chaque classe pour la base d'apprentissage et de validation.

On commence donc le code à partir de cette base, et la première étape après avoir importé les modules est de mettre en place les chemins d'accès aux données de la base d'apprentissage et de validation, qui sont dans des répertoires différents. Il faut aussi redimensionner les images pour avoir des données plus « uniformes ».

Comme précédemment, on met en place et on compile le modèle qu'on va utiliser. Cette fois-ci, on a 9 couches et les paramètres de compilation sont les mêmes que notre premier réseau de neurones, à la différence près que la fonction de perte est « Categorical Cross Entropy ». Cette fonction est la même que la précédente, « Sparse Categorical Cross Entropy », à la différence près que cette dernière a besoin de classes numérotées de manière « conventionnelle », alors que celle utilisée ici nécessite des classes numérotées sous forme « one\_hot », qui est un type d'encodage qui consiste à encoder une variable à n états sur un nombre binaire à n bits, dont un seul prend la valeur 1.

On lance ensuite l'entraînement comme précédemment avec cette fois 20 epochs, et on a comme résultat une précision de 74,58 % et une perte de 0,8789. Il faut remarquer que la base de données utilisée n'est pas simple : bien que les espèces de singes soient toutes différentes, certaines photos pourraient se ressembler et donc correspondre à plusieurs espèces (pour le programme). Ci-dessous, on a les graphiques de la précision et de la perte pour la base d'apprentissage et de validation :

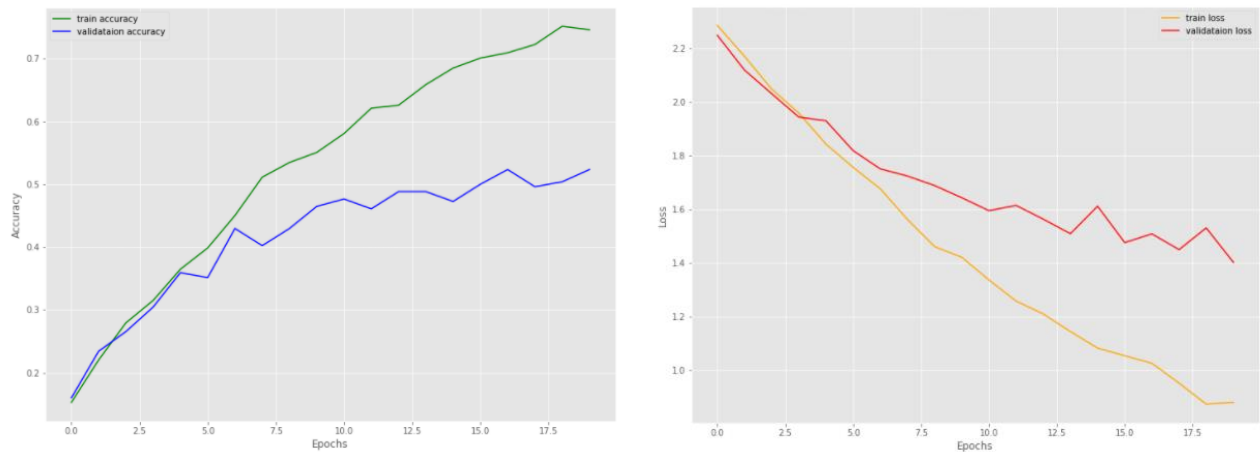


Figure 10 – Graphiques permettant la comparaison l'évolution de la précision ainsi que de la perte, sur la base d'entraînement, et celle de validation

On voit que la précision de la base d'apprentissage tourne effectivement autour des 75 % et que la perte est d'environ 0,9, mais dans le cas de la base de validation, on a une précision très basse d'à peine plus de 50 % et une perte de 1,4. Il est possible que continuer l'apprentissage sur plus d'epochs aurait contribué à améliorer le résultat, mais pour avoir un résultat plus satisfaisant, il semble pertinent d'également changer le modèle utilisé.

# 3

## HTR

### 3.1 – Outils utilisés

N'ayant pas ni les compétences pour créer de A à Z un programme d'HTR complet, ni le temps de les acquérir, nous avons utilisé un programme déjà fonctionnel. Ce programme a été publié sur la plateforme GitHub par Amadeus Suryo Winoto, il nous y explique comment mettre en place un modèle d'OCR simple, pour de l'écriture manuscrite à travers un Jupyter Notebook, un outil créé pour faciliter la présentation de travaux de programmation, permettant dans un même document, de rédiger du texte et du code sous forme de blocs.

Les principaux outils utilisés sont python 3, Tensorflow, ainsi qu'Open-Cv, une bibliothèque graphique développée par Intel, très utile pour le traitement d'image.

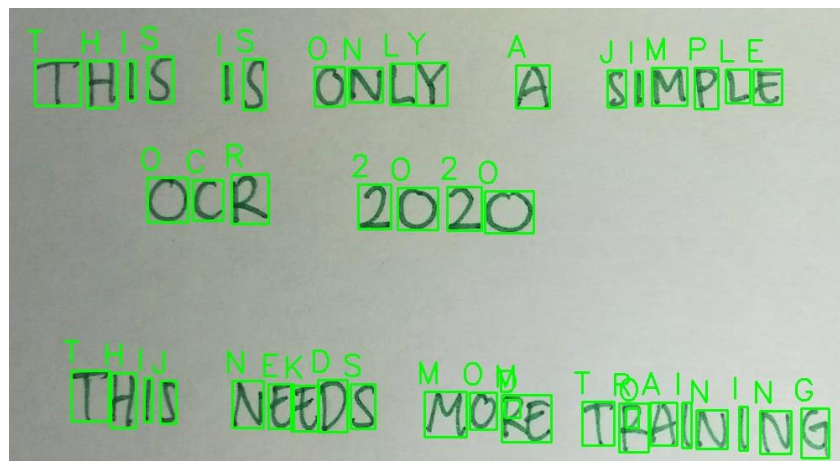


Figure 11 – un résultat du programme

Une des recherches autour de ce programme, fut de l'utiliser avec un autre dataset, toujours pour de l'HTR. En effet, celui utilisé à l'origine contenait 62 000 images pour 93 caractères différents. Avec seulement 200 images pour certains caractères, ce qui comme expliqué lors de l'introduction, peut être assez limité pour reconnaître chaque caractère quel que soit le style d'écriture du document.

Nous avons fait l'expérience avec un premier dataset, contenant près 370 000 lettres majuscules allant de A à Z. Puis nous avons adapter un dataset contenant 30 000 chiffres de 0 à 9 pour le fusionner avec le précédent, plus de détails sur ce sujet seront donnés ensuite.



Afin de travailler sur ces programmes, nous avons mis en place un environnement virtuel à l'aide du logiciel anaconda, en utilisant la version 3.9.10 de python. Les bibliothèques nécessaires à son fonctionnement sont les suivantes : tensorflow 2.8.0, imutils 0.5.4, matplotlib 3.5.1, pillow 9.1.0, numpy 1.22.3, opencv-python 4.5.5.64, scikit-learn 1.0.2, scikit-image 0.19.2.

## **3.2 – Simple-OCR**

Ce programme nous a permis de nous familiariser avec la notion d'OCR, en nous proposant après nos différentes recherches une application concrète. Il est construit en quatre étapes :

La première étape consiste à la préparation du dataset. On y identifie les différentes classes, et importer les données du dataset. La base d'apprentissage, ainsi que la base de test seront automatiquement créées grâce à scikit-learn.

La seconde étape est la création du modèle de deep-learning à l'aide de Tensorflow, la méthode « `model.summary()` » nous résume l'architecture du réseau.

La troisième étape est l'entraînement du modèle. La technique d'augmentation de données est utilisée. Cela consiste à appliquer des transformations sur les images déjà existantes du dataset, cette technique est très utile aux applications pour lesquelles il est compliqué d'obtenir beaucoup d'images, par exemple dans le cas d'analyse d'images pour aider à la détection de cancers, l'augmentation de donnée va permettre d'avoir plusieurs variations avec une seule image, et donc d'obtenir un meilleur résultat lors de l'entraînement du modèle.

Pour la quatrième et dernière étape, l'analyse d'une image contenant du texte manuscrit, OpenCV va être utilisé pour appliquer les transformations nécessaires à cette analyse. Parmi ces transformations, l'algorithme de la détection de Canny est utilisé, cet algorithme est très utile à la classification d'image, il permet de détecter les différents contours dans les images, et ainsi repérer plus facilement et efficacement les éléments qui nous intéressent.

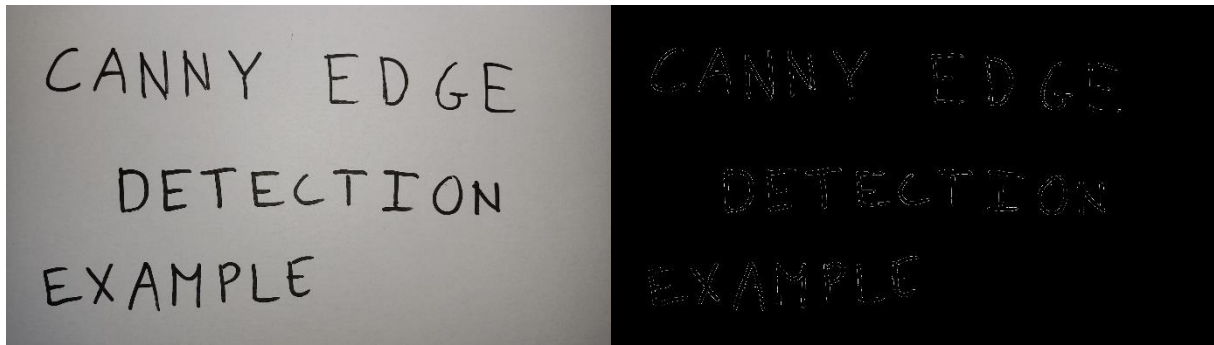


Figure 10- un exemple d'une image après application de l'algorithme de Canny avec OpenCV

### 3.3 – Changement de dataset

#### 3.3.1 - A-Z Handwritten Alphabets dataset

... Nous avons essayé d'utiliser un dataset différent, composé d'écriture manuscrite des 26 lettres de l'alphabet en majuscules. Quelques modifications mineures ont dû être apportés au programme, le dataset d'origine était constitué de dossiers, dont le nom correspondant au code ASCII de chaque caractère qu'il contenait, tandis que pour celui-ci, le nom des dossiers correspondait directement à son caractère.

Nous avons ainsi retiré les chiffres de la liste des classes. Et apporté des modifications aux lignes 12 et 13 du troisième bloc de code. La conversion du code ASCII vers son caractère a été supprimée.



A l'issue de l'exécution du programme avec ce dataset. Nous nous sommes rendu compte que tous les « I » du dataset étaient de cette forme : , il n'y en avait aucun sous celle-ci : . Le résultat était intéressant, car il nous a permis de nous rendre compte de l'importance d'un entraînement avec un dataset complet et varié.



Figure 10 - I ou I

Comme on peut l'observer sur cette image, le programme était donc incapable de reconnaître le « I » à gauche. Tandis que celui de droite est correctement reconnu.

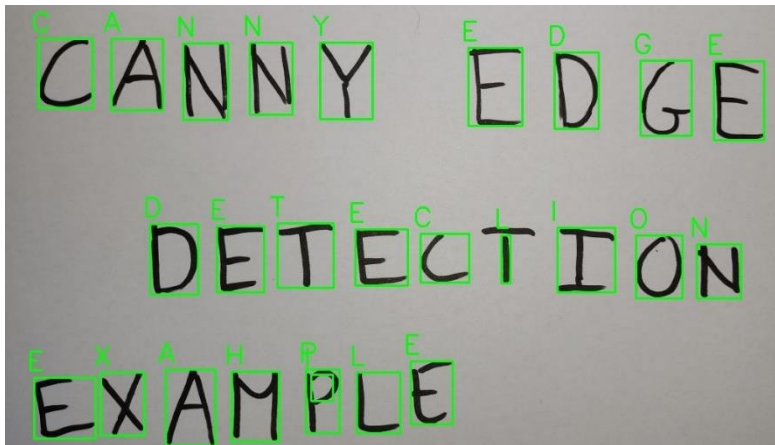


Figure 11 – Exemple d'HTR avec le niveau de certitude

[DETECTED CHAR]	H	-	77.40%
[DETECTED CHAR]	A	-	99.99%
[DETECTED CHAR]	P	-	100.00%
[DETECTED CHAR]	E	-	95.34%
[DETECTED CHAR]	N	-	97.69%
[DETECTED CHAR]	L	-	59.18%
[DETECTED CHAR]	O	-	95.95%
[DETECTED CHAR]	C	-	59.72%
[DETECTED CHAR]	E	-	99.74%
[DETECTED CHAR]	I	-	100.00%
[DETECTED CHAR]	E	-	99.57%
[DETECTED CHAR]	D	-	99.92%
[DETECTED CHAR]	T	-	97.07%
[DETECTED CHAR]	E	-	90.40%
[DETECTED CHAR]	G	-	99.90%
[DETECTED CHAR]	D	-	93.38%
[DETECTED CHAR]	E	-	57.08%
[DETECTED CHAR]	N	-	100.00%
[DETECTED CHAR]	N	-	100.00%

Aussi, nous pouvions voir en sortie la certitude de chacune des prédictions. Ce qui permettait de mieux comprendre les erreurs du programme. Par exemple, ici pour le second T, le haut et le bas de la lettre n'étant pas attachés, seule la partie basse a été détecté comme étant une lettre.

### 3.3.2 – Adaptation de dataset

Après cet essai, nous avons modifié ce dataset pour en créer un composé en plus des lettres majuscules de l'alphabet, des chiffres de 0 à 9. Pour avoir un dataset uniforme, nous avons modifié le second afin d'avoir des images exactement au même format que les précédentes.

Ce dataset contenait des images de 128x128 pixels, le chiffre était en noir sur fond blanc, et la profondeur de couleur des images était de 24bits. C'est-à-dire qu'il y avait pour chaque pixel, un octet dédié au rouge, un au vert et un au bleu.

Nous avons donc créé un programme en python, générant une copie de ce dataset en appliquant à chaque image les transformations suivantes à l'aide des bibliothèques OpenCV, et scikit-image :

```
for folder in os.listdir('data') :
    if not os.path.exists('resized_dataset/' + folder) :
        os.makedirs('resized_dataset/' + folder)
    n = 0
    for file in os.listdir('data/' + folder) :
        file_path = 'data/' + folder + '/' + file
        image = cv2.imread(file_path, 0)
```

Dans un premier temps, on parcourt les fichiers du dataset original pour les lire, tout en créant, lorsque l'on arrive sur une nouvelle classe son dossier respectif.

Le paramètre « 0 » pour la fonction imread() permet d'ouvrir l'image en nuances de gris. Ce qui nous évite d'appliquer cette transformation par la suite.

```
inverted_img = skimage.util.invert(image)
resized_image = cv2.resize(inverted_img, dsize=(28,28), interpolation=cv2.INTER_CUBIC)
future_file_path = 'resized_dataset/' + folder + '/' + folder + '-' + str(n) + '.png'
n += 1
cv2.imwrite(future_file_path, resized_image)
```

On utilise en premier la fonction resize pour avoir une image de 28x28 pixels, les calculs nécessaires pour inverser les couleurs de l'image - et obtenir un chiffre blanc sur fond noir - seront ainsi moins lourds. La variable « n » est utile à la génération du nom des fichiers. On finit par enregistrer l'image transformée.

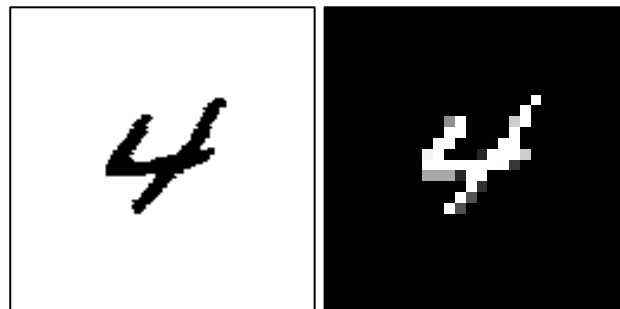


Figure 12 – Une image, avant, et après transformation.



# Conclusion

Ce projet nous a permis de réfléchir aux moyens de résoudre différentes problématiques, tout en prenant conscience de la complexité de la mise en œuvre d'algorithme d'intelligence artificielle.

Ce travail nous permettra dans un futur proche d'aborder d'autres projets plus ambitieux liés à l'informatique. Il nous permet d'avoir un regard très constructif sur l'enseignement que nous attendons en étant une sorte d'introduction aux différents projets qui seront réalisés plus tard dans nos études.

Le projet initial étant une initiation à l'intelligence artificielle, cela nous a permis de découvrir l'impressionnante diversité de possibilités d'applications de cette technologie, et de faire nous même le choix d'orienter le projet vers un type d'application qui nous intéressait.

La principale difficulté de ce projet fut tout de même la compréhension du principe de fonctionnement de ces intelligences artificielles. Même si aujourd'hui plusieurs modules en python, comme Tensorflow, ou Pytorch que nous n'avons pas utilisé, facilitent la mise en œuvre de ces programmes, l'acquisition des notions reposant derrière leur fonctionnement demandait un certain temps, de ce fait, nous n'avons pas pus comme prévu initialement, développer notre propre programme d'HTR.

Malgré cela, ce projet fut très enrichissant, car nous avons en plus de découvrir l'intelligence artificielle, appris à organiser un projet s'étalant sur une période de plusieurs mois.

De plus, nos enseignements en informatique se limitant jusqu'ici à des compétences basiques, ce projet nous permettra de justifier pour la suite de notre cursus, de connaissances spécifiques aux domaines de l'intelligence artificielle, dans le cadre d'éventuelles recherches d'entreprise.

Pour finir, nous souhaitons remercier notre encadrant Mr Jean-Yves Ramel. Ce dernier nous ayant apporté une aide appréciable dans la réalisation de ce projet.



# Bibliographie

Image de tomate :

<https://www.jardiner-malin.fr/fiche/tomate-brandywine.html>

Exemple d'application de la méthode des k plus proches voisins :

[https://pixees.fr/informatiquelycee/n\\_site/nsi\\_prem\\_knn.html](https://pixees.fr/informatiquelycee/n_site/nsi_prem_knn.html)

Image de tenseurs :

<https://forum.huawei.com/enterprise/fr/qu-est-ce-que-tensorflow-2-x/thread/801627-100383>

Tutoriel d'initiation à Tensorflow que nous avons suivi :

<https://debuggercafe.com/introduction-to-tensors-in-tensorflow/>

Programme d'HTR sur lequel nous avons travaillé :

<https://github.com/Xinihiko/Simple-OCR>

Dataset initialement utilisé pour le programme :

[https://github.com/sueiras/handwriting\\_characters\\_database](https://github.com/sueiras/handwriting_characters_database)

Dataset des lettres majuscules de A à Z au format CSV:

<https://www.kaggle.com/datasets/sachinpatel21/az-handwritten-alphabets-in-csv-format>

Script nécessaire à l'extraction du contenu du dataset :

<https://www.kaggle.com/code/sachinpatel21/csv-to-images/script>

Images tirées de ce dataset :

<https://www.nist.gov/srd/nist-special-database-19>

Dataset des chiffres de 0 à 9 :

<https://www.kaggle.com/datasets/pradheeprio/handwritten-digit-classification>

Vous trouverez ici un dépôt Github sur lequel se trouve nos différents essais sur le programme d'HTR, ainsi que sur les transformations de dataset :

[https://github.com/Enzo17101/PROJET\\_PEIP\\_S4](https://github.com/Enzo17101/PROJET_PEIP_S4)



# Comptes rendus hebdomadaires

## Compte rendu n°1

Lors de cette première séance, nous avons eu quelques pistes données par notre encadrant pour commencer le projet correctement, notamment à propos de termes à définir pour comprendre le principe derrière les bases de l'apprentissage.

## **Semaine 1 :**

A la suite des indications de notre encadrant, nous avons donc commencé nos recherches sur l'IA en général, et plus spécifiquement sur les types d'apprentissages et les concepts liés à ceux-ci : par exemple ce qu'est un dataset, un descripteur, la différence entre l'apprentissage supervisé et non supervisé, etc...

## Compte rendu n°2

Nous avons continué d'essayer de définir les termes essentiels, que nous avons regroupé dans un document pour les consulter au besoin. Nous avons réfléchi à un type d'IA pour la suite du projet, et avons finalement décidé de travailler sur l'OCR, et plus précisément l'HTR.

## **Semaines 2 et 3 :**

Durant la semaine, nous avons continué de remplir notre document de recherches, en y ajoutant des liens vers des vidéos, cours ou tutoriels utiles pour notre projet. Certains d'entre eux nous avaient déjà été donnés par notre encadrant, mais nous en avons trouvé certains autres aussi utiles qui nous ont aidé à finir de nous familiariser avec ce sujet.

## Compte rendu n°3

Nous avons continué les recherches, et avons commencé à réfléchir aux moyens de réaliser une solution d'HTR.

#### **Semaine 4 :**

Nous avons réussi à trouver quelques vidéos à propos des concepts les plus fondamentaux de l'IA et de ce projet en général : deep-learning, fonctionnement basique d'un réseau de neurones.

#### **Compte rendu n°4**

Un des tutoriels trouvés la semaine passée nous semblait particulièrement bien réalisé, et plus long que d'autres que nous avions trouvés auparavant, nous avons décidé de le suivre jusqu'au bout : il nous a permis d'apprendre à utiliser le module python Tensorflow, aussi, il contenait un exemple concret de construction d'un modèle d'apprentissage et de classification d'image, ce que nous jugions utile pour la suite du projet.

#### **Semaines 5 à 8**

Ce tutoriel était fait en 8 parties et était basé sur Tensorflow : les trois premières concernaient des éléments plus théoriques mais tout de même essentiels. La quatrième partie était le début de la pratique avec la création d'un réseau de neurones et l'apprentissage sur deux bases de données : la première constituée de chiffres écrits à la main, et la seconde basée sur des images en noir et blanc de vêtements. La partie suivante était également une partie assez théorique, avec une explication sur les réseaux de neurones convolutifs et un petit exemple pratique. Ensuite, la sixième partie parlait de classification d'images et utilisait pour cet exemple une base de données constituée d'images de singes. L'avant-dernière partie se basait sur la classification d'images avec des modèles inhérents à Tensorflow, avec une partie théorique sur ces modèles. Enfin, la dernière partie était sur le « transfer learning », qui consiste à réutiliser (en l'adaptant) un modèle existant pour changer de domaine d'expertise, par exemple, l'adaptation d'un modèle de classification de voiture, en un modèle de classification de camions.

#### **Compte rendu n°5**

Faute de temps et de compétences, nous avons fait le choix de trouver un modèle d'HTR déjà fonctionnel, afin de travailler dessus.

#### **Semaine 9 et 10 :**

Le programme étant très bien commenté, nous avons pris le temps de comprendre son fonctionnement dans sa quasi-totalité.



### Compte rendu n°6

Nous avons observé le comportement du programme en changeant de dataset. Et commencé la rédaction du rapport.

### **Semaine 11 et 12 :**

Nous avons principalement avancé sur la rédaction du rapport. Et travaillé sur les transformations de dataset pour le programme d'HTR.

# Initiation à l'intelligence artificielle

## Résumé

Le but de ce projet était de découvrir les technologies d'intelligence artificielle, notamment en développant des scripts de classification d'image. Nous avons ainsi appris qu'il existait plusieurs types d'intelligences artificielles et découvert leur fonctionnement. Ce projet étant libre sur l'objectif final, nous avons dans un premier temps fait le choix de développer une solution de reconnaissance d'écriture manuscrite. Pour finalement se tourner vers un programme de reconnaissance déjà fonctionnel car notre objectif initial était trop ambitieux. Autour de ce programme nous avons travaillé sur les transformations de base de données et observer son comportement selon la base de données utilisée.

## Mots-clés

Programmation, intelligence artificielle, base de données, apprentissage supervisé, modèle, reconnaissance d'écriture manuscrite, manipulation d'image, langage Python.

## Abstract

The aim of this project was to experiment with the artificial intelligence technology, mainly by creating image classification programs. We thus discovered that there are several AI types and found out about the way they work. The final task of this project was rather open, so we decided to try and make a handwritten text recognition program. However, it was a bit too difficult for us, so we ended up using an already existing one. We worked on modifying the dataset and observing how the program reacts based on the dataset used.

## Keywords

Programming, artificial intelligence, dataset, supervised learning, model, handwritten text recognition, image processing, Python language

Encadrant académique  
*Jean-Yves Ramel*

Étudiant(e)s  
*Enzo Creuzet*  
*Maxime Brossillon*