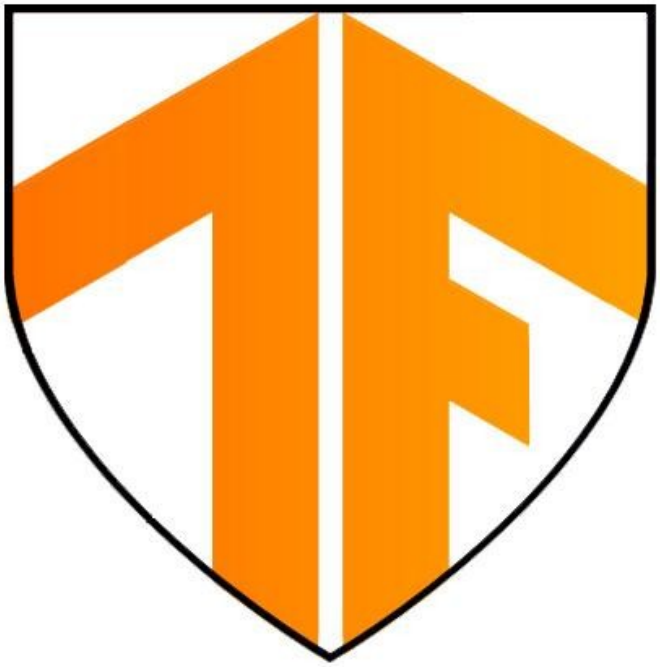
UNIVERSITÉ DE MONTPELLIER

I.U.T MONTPELLIER - SETE

RAPPORT DE PROJET

Reconnaissance de blasons simples par Machine Learning



Réalisé par
Paul VAN DER HOEVEN, Simon MOULIN, Sylvain REYNAUD, Tom ALEGRE

Sous la direction de
M. Victor POUPET

Pour l'obtention du DUT Informatique

Année universitaire 2019-2020

REMERCIEMENTS

Nous tenons à remercier toutes les personnes qui ont contribué à la réussite de ce projet ainsi que les personnes qui nous ont aidés à la rédaction de ce rapport.

Tout d'abord, nous adressons nos remerciements à notre tuteur de projet M. Poupet qui a accepté de nous prendre en charge pour ce projet et grâce à qui nous avons pu travailler sur un sujet passionnant. Ses conseils et son suivi nous ont permis de régler de nombreux problèmes rencontrés et d'avoir accès à des ressources de qualité.

Nous remercions également les auteurs de toutes les ressources que nous avons pu utiliser, notamment les propriétaires du site WorldSpinner [1], sans qui nous n'aurions pas pu traiter ce sujet. Mais aussi des rédacteurs de tutoriels que nous avons utilisés lors de notre travail, tels que Harrison Kinsley.

Enfin, nous tenons à remercier toutes les personnes qui nous ont conseillés et relus lors de la rédaction de ce rapport, notamment Mme Delebarre sans qui nous n'aurions pu apporter autant de rigueur dans le sérieux de ce rapport.

Sommaire

Table des illustrations

Glossaire

TensorFlow : outil open source d'apprentissage automatique (machine learning) développé par Google. [2]

Base64: codage utilisant 64 caractères différents, communément utilisé pour encoder les images. [3]

Précision(Accuracy) : statistique des modèles de classification. La précision correspond à la fréquence à laquelle le modèle prédit correctement la classe positive [4].
Par exemple :

$$\text{Précision} = \frac{\text{Vrais positifs}}{\text{Vrais positifs} + \text{Faux positifs}}$$

Itération (epoch) : cycle d'apprentissage complet sur l'intégralité de l'ensemble de données de manière à ce que chaque exemple ait été vu une fois. [4]

Blasonnement : Action de nommer un blason[5]

Poids et biais: variables internes des neurones

Introduction

L'intelligence artificielle s'invite dans de nombreux domaines dont une nombreuse partie ne vient pas forcément instinctivement à l'idée : la santé, les transports, la météorologie en sont des exemples. Une large majorité des personnes qui l'utilise de nos jours, le font en toute inconscience de leur acte. En effet, celle-ci est présente partout et est parfois cachée de la vue de tous, c'est par exemple le cas avec les réseaux sociaux ou avec YouTube, où les algorithmes de machine learning sont omniprésents. Le machine learning est tout particulièrement efficace et mis en avant dans le domaine de la reconnaissance d'images, un exemple commun est la différenciation d'animaux comme la différenciation des chats et des chiens. Les techniques de machine learning dans la reconnaissance d'images peuvent très bien différencier des images entre plusieurs catégories, ou alors reconnaître des formes, des couleurs, dans le but de reconnaître un objet par exemple. Ces techniques sont utilisées dans de nombreux domaines professionnels et à des fins commerciales, mais peuvent tout à fait être utilisées à plus petite échelle dans un but purement ludique ou de divertissement. Les limites résident seulement dans l'imagination et dans la manière d'obtenir les données destinées à entraîner les modèles d'intelligence artificielle. C'est dans une optique purement ludique que l'activité du blasonnement* se situe, étant une activité très spécifique, essayer d'obtenir la description d'un blason peut s'avérer une tâche ardue. En effet, il est nécessaire de connaître la légende des couleurs, formes, objets présents dans un blason par coeur, et de travailler pas à pas sur l'identification de chacun de ces éléments sur un blason donné.

Solutionner l'identification des blasons par un travail humain, ne permet donc pas de les identifier de manière rapide et efficace. Une perte de temps liée aux limites de l'humain, même très entraîné, ne permettra jamais de rivaliser avec une machine. De plus, le blasonnement étant une activité très spécifique, il pourrait être intéressant de la rendre plus accessible au grand public, qui pourrait plus facilement s'intéresser à ce domaine si des outils adaptés lui sont proposés.

Il paraît naturel, présentant ces deux sujets d'étude conjointement, de les lier afin de proposer une solution innovante. Cela renforce la présentation faite sur l'intelligence artificielle, outil très efficace même concernant des sujets très pointus et méconnus. Ce projet consistera donc à implémenter une solution d'identification automatique de blasons, à l'aide d'une intelligence artificielle. Permettant ainsi de proposer une approche simple, efficace mais aussi ludique du blasonnement.

Après avoir développé les différents besoins de la solution dans le cahier des charges, un rapport technique permettra d'expliciter les détails technologiques. Une présentation des résultats permettra ensuite de confirmer ou d'infirmer la viabilité de cette nouvelle solution ainsi proposée. Enfin, une mise au point des pratiques de travail utilisés, dégagera un aperçu des pratiques qui ont été mises en œuvre pour la réalisation de ce projet, et une critique permettra d'en tirer des points d'améliorations.

1. Cahier des charges

1.1 Analyse de l'existant

L'étude des blasons héraldiques est un sujet qui est peu connu du grand public. C'est pour cela qu'il existe assez peu d'outils en ligne pour la reconnaissance de blasons. La plupart des lexiques à utiliser pour reconnaître un blason, permettent bel et bien de les reconnaître ou d'en créer mais de manière longue et fastidieuse. Il existe de la documentation mais cela reste assez technique. En effet, les descriptions des blasons sont en héraldique, les couleurs et figures ne sont pas dans le langage courant. Ils possèdent plusieurs parties (abordées et détaillées ultérieurement) qui se ressemblent beaucoup et sont assez difficiles à distinguer. Il existe un site web: WorldSpinner, permettant d'en générer et de récupérer la phrase de description.

Après avoir analysé les outils et la documentation existante sur l'étude des blasons, il est nécessaire d'en faire une description et de présenter les besoins fonctionnels de ce projet.

1.2 Présentation du sujet et analyse du contexte

De par la problématique rencontrée pour identifier un blason de manière simple et efficace, il a été décidé d'utiliser des réseaux de neurones pour rapidement dissocier les différentes parties d'un blason. Un blason est composé de plusieurs sous-parties, celle-ci sont considérées comme des caractéristiques.
Elles comportent des termes techniques et il est assez difficile d'expliquer leur représentation graphique [cf figure 1] en expliquant avec des mots c'est pourquoi un schéma est beaucoup plus parlant.
En effet, les plusieurs parties sont: le field, la division et l'ordinary. La division et l'ordinary sont optionnelles. Chaque partie possède une couleur et peut avoir ou non une ou plusieurs formes représentées sous par des objets. La partie division a une couleur optionnelle qui sera appliquée sur une dissection du blason. L'ordinary représente une forme géométrique comportant sa propre couleur. Toute partie, forme ou couleur est traduite par des mots associés et des virgules ou des espaces séparent chacune de ses caractéristiques.

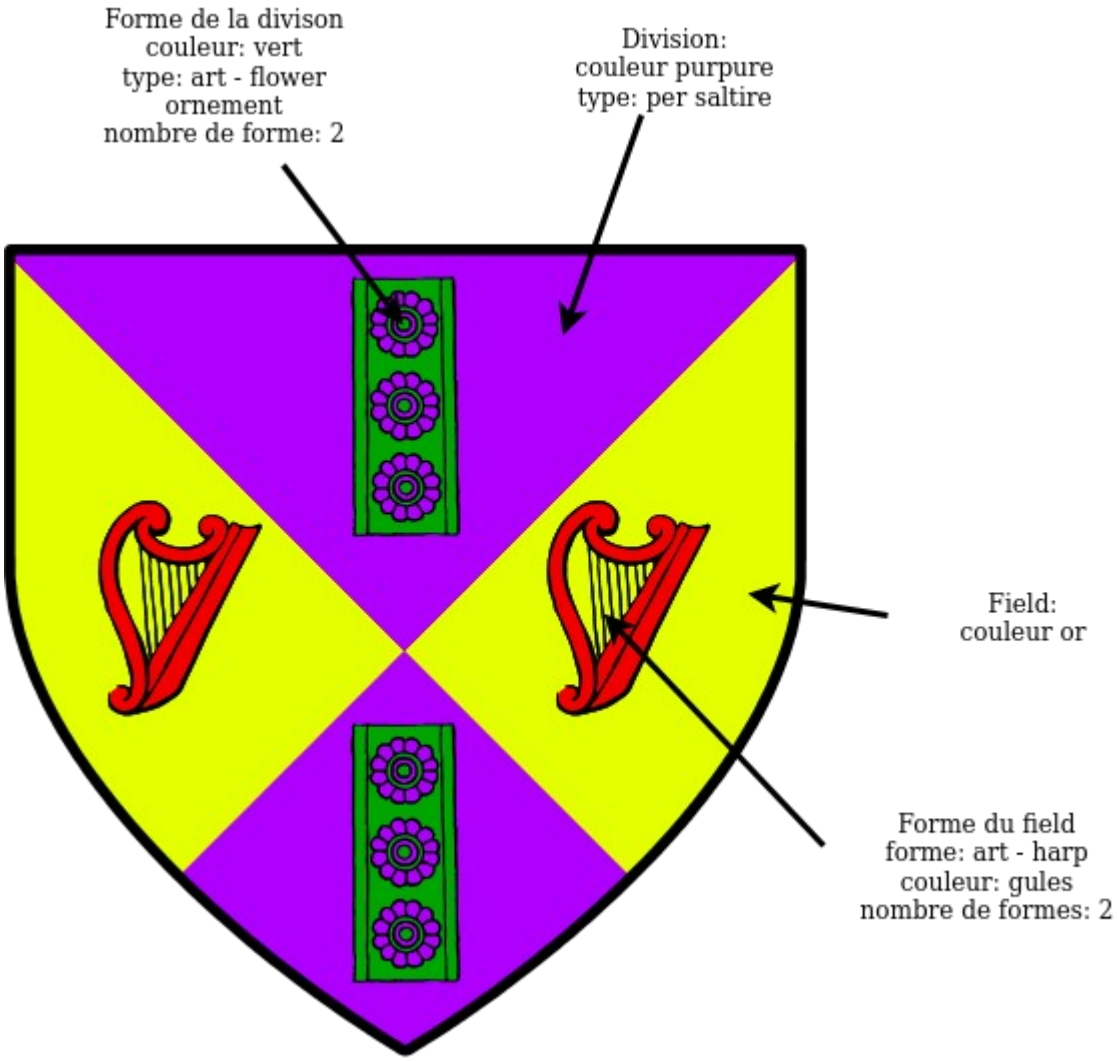


Illustration 1: Schéma de la disposition des différentes parties d'un blason
Description: *"or two harps gules, per saltire purpure two flower ornaments vert" [1]*

1.3 Analyse des besoins fonctionnels

L'objectif de ce projet est de réaliser un programme qui prendra une image de blason en entrée et générera sa description. Il sera en mesure de générer la description héraldique d'un blason lambda qu'il n'a jamais rencontré dans son entraînement. Le programme prendra donc en entrée une image d'un blason et permettra de récupérer en sortie ses caractéristiques une fois qu'elle aura été analysée par le réseau de neurones. Les caractéristiques qu'il a été décidé de traiter sont : la couleur de fond, le nombre de formes, la division, l'ordinary, la couleur de la division, la couleur de l'ordinary ainsi que le nombre d'objets associés pour chaque sous-partie. La description héraldique ne pourra donc pas être complète car cette analyse omet de détailler les noms des objets présents au sein du blason et se contente seulement de les dénombrer. Le programme sera capable de reconnaître des images de blasons déformées et dont les couleurs ne sont pas exactement les mêmes que les originales.

1.4 Analyse des besoins non fonctionnels

L'application devra être capable de fonctionner sur n'importe quel ordinateur compatible avec les langages et librairies utilisées à savoir Python et TensorFlow*. En effet, le tuteur de projet a imposé l'utilisation du langage Python, et du framework de machine learning TensorFlow qui est à privilégier par rapport à d'autres frameworks de machine learning tels que PyTorch, Apache Spark ou Caffe. Tensorflow est une librairie développée par Google qui permet de réaliser des applications de Machine Learning, elle est, de surcroît très utilisée par les professionnels. Pour mener à bien l'apprentissage de réseaux de neurones, il est nécessaire de mettre en place une base de données de blasons afin d'entraîner les modèles de machine learning. Enfin, des méthodes de gestion de projets devront être mises en place afin de pouvoir travailler conjointement et efficacement.

2. Rapport technique

2.1 Fonctionnement du machine learning

Le principe du machine learning peut être résumé à l'entraînement d'un modèle avec des données contenant des entrées et des sorties correspondantes. Son utilisation se divise en deux phases : la phase d'apprentissage et la phase de prédiction.

Ce modèle se présente sous la forme d'un réseau de neurones. Chacun de ces neurones possède ses propres variables qui sont modifiées pour correspondre aux sorties attendues. Tout neurone est interconnecté et permet de simuler une pseudo intelligence par apprentissage.
La phase d'apprentissage s'effectue en donnant à l'algorithme un grand nombre de données.

Cela permet aux neurones d'ajuster leur manière de deviner le résultat des données passées en entrée. Après le passage d'une très grande masse de données à travers l'algorithme, celui-ci sera capable de deviner les résultats d'une donnée qu'il ne connaît pas.
Les valeurs qui changent au sein des neurones s'appellent les poids et les biais*, ces valeurs sont représentées par des valeurs mathématiques, et changent selon les algorithmes dictés par le modèle de machine learning.
Dans le cas présent, il faut donc donner de nombreux blasons avec leur descriptions. Ainsi le programme pourra analyser un blason inconnu et il donnera sa description en héraldique.

Pour aller plus loin dans les explications, il est important d'insister sur le fait que les neurones sont présents sous forme de couches [cf figure 2].

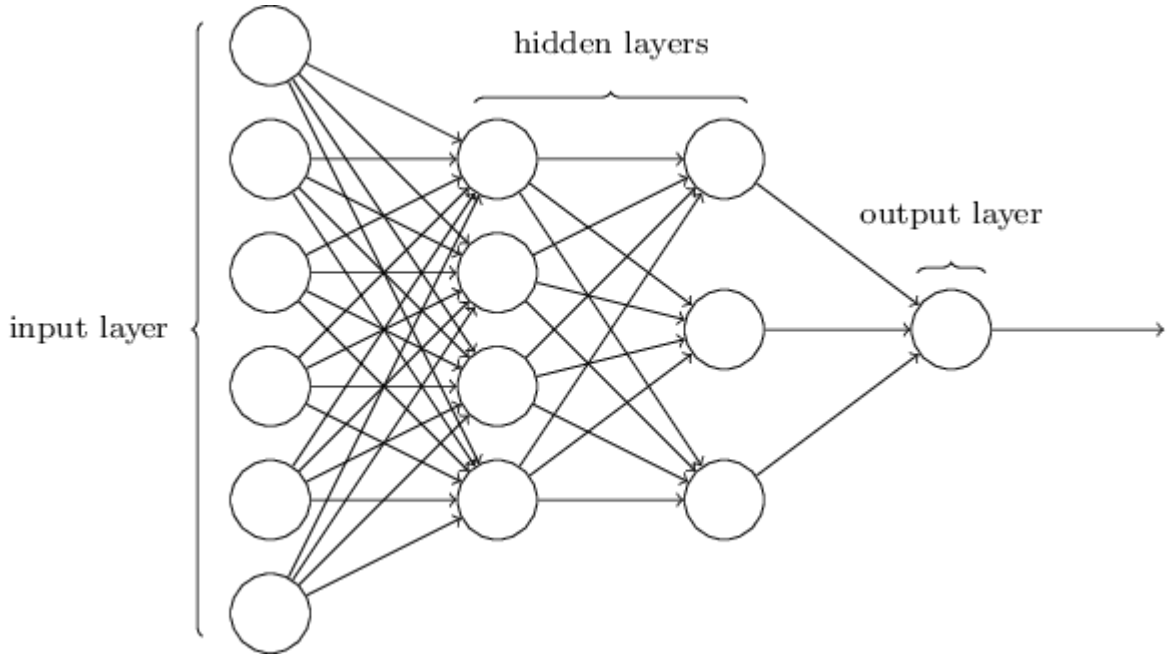


Illustration 2: Représentation d'un réseau de neurones [6]

Lors de l'utilisation d'un modèle de neurones, seul l'accès aux neurones d'entrée, et aux neurones de sorties est possible. Il est important de noter que dans le cas d'un modèle de machine learning traitant des images, les neurones d'entrées doivent représenter un pixel, ce qui implique que plus la taille de l'image sera grande, plus le nombre de neurones, et donc la complexité du modèle sera élevée, et ce de manière plus ou moins exponentielle. Il est tout à fait possible de décider le nombre de neurones et le nombre de couches qui seront disposées entre les couches d'entrées et de sortie. Jouer sur ce nombre de neurones et de couches peut permettre d'effectuer des actions de dégrossissement d'une image appelées convolutions, isolant certaines formes ou couleurs par exemple, cela sera évoqué par la suite. Ces caractéristiques représentant un modèle de neurones sont très importantes et ont un impact majeur sur l'apprentissage de celui-ci.
De plus, toujours concernant l'apprentissage, lors du lancement de celui-ci, il est nécessaire de donner des paramètres à la méthode qui s'occupera de ce processus pour qu'elle puisse suivre des règles. Celles-ci permettent d'appliquer un autre facteur modulable et ainsi de permettre encore une fois la modification de caractéristiques, qui pourront régler des problèmes que nous verrons par la suite. La principale donnée que nous pouvons faire varier est le nombre d'epochs*, ce terme technique indique le nombre d'itérations* d'apprentissage que l'algorithme doit effectuer, nous verrons par la suite son influence sur les résultats obtenus par notre modèle. Il faut savoir que le nombre de neurones par couche et le nombre de couches sont plus importants que les epochs, ce facteur à faire varier est donc très important lorsqu'il s'agit de régler l'efficacité d'un modèle de neurones.

La partie de prédiction du machine learning, consiste à vérifier que l'apprentissage a été effectué dans de bonnes conditions. En effet, deux cas de figure peuvent se présenter, le modèle peut tout simplement ne pas avoir réussi à apprendre des données que nous lui avons donné, dans ce cas il aura du mal à deviner le résultat d'un nouvel élément. Cela peut être dû à une trop grosse complexité du domaine traité, d'un trop grand nombre de variables à étudier, ou alors à une mauvaise conception du modèle de neurone. Pour pallier ce problème il est souvent nécessaire de modifier la structure du modèle. Les premiers problèmes de sous entraînement arrivent lorsqu'il est nécessaire de repérer des formes. Pour repérer des formes dans une image, des neurones simples ne suffisent pas toujours, et il peut être intéressant d'utiliser des convolutions. Les convolutions permettent d'ajouter une première couche de traitement de l'image avant d'entrer en contact avec les neurones, cette précouche supprime des informations de l'image en la réduisant au niveau de sa taille, tout en gardant les informations concernant les paternes de formes qu'elle pourrait avoir. Cette technique applique un masque de taille variable, communément 3x3, qui permet pour chaque pixel, d'obtenir son image dans une nouvelle matrice [cf figure 3]. Cette technique permet de réduire le nombre d'informations tout en conservant une part de données importantes, et ainsi de plus facilement reconnaître des formes.

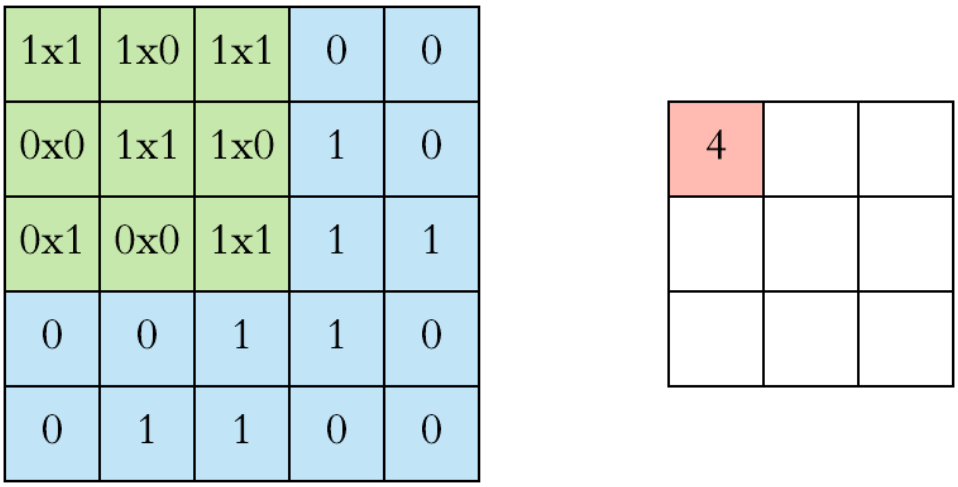


Illustration 3: Représentation de l'application d'une convolution [7]

L'ajout de cette couche de pré calcul, pourrait permettre de résoudre un problème de sous entraînement d'un réseau de neurones. Mais ce problème n'est pas le seul, en effet dans un deuxième cas de figure d'apprentissage, il se peut que le réseau ai trop appris des données fournies, cela peut s'expliquer de deux manières, soit le jeu de données fourni était trop petit, soit le réseau a appris de manière trop répétée sur ce jeu de données. Ce problème se nomme l'overfitting, il signifie le fait d'un apprentissage par cœur des données passées en entrée par le réseau de neurones, qui sera donc incapable de reconnaître de futures données jamais rencontrées, le pourcentage de réussite est donc en régression. Dans un premier temps il faut donc vérifier que le jeu de données était assez diversifié et grand, si ce n'est pas le cas il est nécessaire d'ajouter plus de données. Il est par exemple possible d'utiliser des techniques d'Image Augmentation*, consistant à appliquer des déformations sur les images du jeu de données initial, afin de virtuellement gonfler le nombre de données. Si le jeu de données utilisé est assez conséquent, il s'agit simplement de modifier le nombre d'epoch lors de l'entraînement, nombre qui influe, il est important de le rappeler, sur le nombre d'itérations d'apprentissage sur le jeu de données.

Ainsi, le machine learning est un algorithme entraîné avec des données qui peut ensuite faire des prédictions. Cette algorithme a été implémenté dans ce projet.

2.2 Conception

Comme expliqué dans la partie décrivant le fonctionnement du machine learning, la première étape pour se lancer dans le sujet est la conception d'un jeu de données assez imposant. Dans le cas des blasons, il est donc nécessaire d'obtenir des milliers d'images de blasons ainsi que leurs descriptions correspondantes. Puis, il est nécessaire d'organiser ces données afin d'avoir un format de données unifié et compatible avec l'algorithme de machine learning. Ces données seront ensuite divisées en deux parties, afin d'obtenir un jeu de données destiné à entraîner le modèle, et un autre, dit de test, destiné à être testé sur le modèle une fois entraîné. Ces étapes seront détaillées plus tard lors d'une partie dédiée.

Pour deviner la description d'un blason, il y a plusieurs modèles. Chaque modèle aura la responsabilité de traiter une information. Les différents réseaux de neurones seront donc chargés de la reconnaissance de l'Ordinary et de la Division, ainsi que de leurs couleurs, et du nombre de formes qu'ils présentent. A cela s'ajoute l'identification de la couleur du Field et de son nombre de formes. Il est donc nécessaire de réaliser huit réseaux de neurones différents, tous dédiés à la reconnaissance d'une information en particulier. Afin de réaliser chacun de ces modèles, il est nécessaire de modeler la répartition des neurones au sein de leur modèle. Le nombre de couches de neurones et de neurones par couche est défini arbitrairement en fonction de l'efficacité de l'algorithme. Ensuite, il faut entraîner l'algorithme avec les données qui ont été récupérées, organisées et formatées pour qu'il puisse apprendre. Mais avant cela, pour chacun de nos modèles, il a été nécessaire de formater la description de chacun des blasons. En effet, pour chaque programme, c'est-à-dire chaque modèle, la partie de la description représentant l'information à deviner n'est pas la même. Il est donc nécessaire d'écrire un script consistant à isoler la partie d'information importante. Pour la couleur du field par exemple, il est nécessaire d'isoler le premier mot de la description. Cette manipulation représente le plus gros changement entre chaque programme, et est la partie la plus délicate à réaliser, elle sera elle aussi détaillée lors d'une partie dédiée. La dernière phase consiste à tester le réseau sur des données qu'il n'a jamais rencontrées, c'est-à-dire sur le jeu de données de test. C'est ici que les résultats du travail effectué sont obtenus. Selon la nature des résultats, il est nécessaire de retourner à la conception du réseau de neurones afin de régler des problèmes de sous-entraînement ou d'overfitting. Si le résultat est assez convaincant, c'est-à-dire s'il dépasse les 95% de réussite environ, le modèle sera considéré comme bon et bien entraîné. Dans un premier temps les réseaux sont entraînés sur des données dites originales, c'est-à-dire les données récupérées depuis le site Worldspinner. Cette première phase permet de tester la robustesse du réseau. Il est ensuite nécessaire, pour la réussite du projet, d'entraîner les réseaux avec des données augmentées. Ces données sont générées aléatoirement en fonction de caractéristiques de déformation pré-établis. Cela permet de simuler des déformations qui pourraient être causées par une faible qualité de l'image ou d'une rotation due à une photographie pas très droite. Cette simulation est censée augmenter le taux de réussite des réseaux sur des images plus naturelles et ainsi donner un réel avantage au programme.

Enfin, il est nécessaire de rassembler tous les programmes en un seul et même script python afin d'obtenir tous les résultats d'un blason donné et ainsi de reconstituer sa description. Pour réaliser ceci, il doit être possible d'exporter et d'importer des réseaux de neurones. Par chance, c'est tout à fait possible avec TensorFlow ! Pour cela il est nécessaire d'enregistrer toutes les informations des réseaux entraînés, c'est-à-dire le poids et le biais de chaque neurone. Chaque modèle est donc sauvegardé dans un fichier contenant toutes ses informations, et qui, une fois importé, se comportera comme le modèle initial. Une fois ceux-ci exportés il est très facile de les importer dans n'importe quel programme. Ainsi, les huit caractéristiques récupérées permettent de composer la description du blason et ainsi d'effectuer une action de blasonnement.

2.2.1 Récupération des blasons et formatage des données

La première étape est la récupération et le formatage des données, sans cela il n'est pas possible de commencer à entraîner le réseau de neurones. L'objectif est de réunir au moins 50 000 blasons uniques afin de répondre aux besoins en termes de nombre de données d'un modèle de machine learning.

Il a d'abord fallu trouver une source nous permettant d'obtenir énormément de blasons. M. Poupet a donc conseillé l'utilisation du site Worldspinner qui permet de générer aléatoirement des blasons. Ce site représente donc une source de blason quasiment illimitée qui permet d'entraîner les différents réseaux avec énormément de données les rendant les plus performants possible.
La récupération des blasons se fait à l'aide d'un script automatisant la récupération de blasons sur le site WorldSpinner. Le script simule un clic sur le bouton *Respin* afin de générer un nouveau blason. L'image et le nom du blason sont récupérés dans le code HTML et écrits dans un fichier texte. Chaque ligne du fichier texte représentera la description d'une image, puis sa représentation en base64*. Ce script a dû être développé manuellement pour convenir aux attentes du projet.

Après plusieurs expérimentations et de par les nombreuses images, plus de 79 000, il a été nécessaire de formater les images en base64, associées à leurs descriptions. L'utilisation du format base64 a été nécessaire afin de limiter le nombre d'ouvertures et de lectures de fichiers, économisant ainsi les ressources disponibles, puisque toutes les informations sont contenues dans un seul et même fichier.

Dans le but d'avoir un ensemble de données des plus optimales pour l'entraînement du modèle, il est important de supprimer les blasons en double. Pour enlever les doublons, il suffit de supprimer les lignes identiques dans le fichier afin d'en garder une unique. L'extension TextFX[8] développée par Chris Severance sur l'éditeur Notepad++ dispose d'une option pour répondre à ce besoin.

Étant confronté à un fichier de données de plusieurs gigaoctets, il est nécessaire de réduire la taille des images afin de réduire le poids de l'ensemble des données. Un script Python utilisant la librairie PIL est utilisé pour répondre à ce problème. Les images ont été

réduites en 50 pixels par 50 pixels.
De plus, avec des images plus petites l'entraînement du réseau est beaucoup plus rapide car le nombre d'inputs est considérablement plus faible car chaque pixel représente un neurone d'entrée.

2.2.2 Décomposition de la description et des blasons

Chaque blason a une description correspondante. Il est nécessaire de la décomposer pour identifier chaque partie. Il y aura un réseau de neurones pour chaque partie. Il y a d'abord le field puis la division et enfin l'ordinary, chacune est séparée par une virgule. La division et l'ordinary sont optionnels. Ces deux derniers sont composés d'un mot-clé: "per" pour la division et "over" pour l'ordinary pour de leurs caractéristiques: le type, la couleur, le nombre de formes, le type de forme et la couleur de la forme. Cela a permis de pouvoir extraire ces différentes valeurs pour les donner aux réseaux de neurones.

2.3 Réalisation

2.3.1 Récupération des blasons et formatage des données

La récupération des données se fait à l'aide d'un script Python utilisant le framework Selenium. Le choix de Selenium a été fait puisque le générateur de blason de WorldSpinner fonctionne via Javascript. Il est donc difficile de gérer des blasons automatiquement avec les librairies de base telles qu'urllib ou bien requests. Selenium est de base un framework conçu pour les tests mais ici son utilisation sera détournée pour automatiser le processus.
Selenium charge la page du générateur dans le moteur de rendu de Google Chrome ce qui permet d'exécuter le code Javascript et donc de pouvoir récupérer les blasons. Afin de récupérer les données du blason, à savoir son nom (son blasonnement) et son image qui est au format base64, Selenium permet l'utilisation de sélecteur CSS avec find_element_by_css_selector().

```
img_b64 = WebDriverWait(driver, 20).until(
lambda driver:driver.find_element_by_css_selector('img.device-preview')
                .get_attribute('ng-src'))[22:]

nom = driver.find_element_by_css_selector('em.ng-binding')
          .get_attribute('innerHTML')[1:-1]
```

L'utilisation de WebDriverWait est obligatoire puisqu'il est nécessaire d'attendre le chargement de l'image avant de pouvoir la récupérer. Ainsi, afin de récupérer plusieurs blasons à la suite il faut comparer le nom du blason précédemment récupéré et le blason actuellement sur la page. Si cela n'est pas fait, le même blason sera enregistré plusieurs fois, ce test est fait après l'appel d'un clic sur le bouton "Respin" permettant de générer un nouveau blason.

```
driver.find_element_by_class_name('brass-button-small').click()
#  wait for the new image loaded by the click on the Respin button
WebDriverWait(driver, 20).until(
   lambda driver:
      driver.find_element_by_css_selector('em.ng-binding')
         .get_attribute('innerHTML')[1:-1] != nom)
```

Le problème principal de ce script est que les images récupérées sont lourdes, il faut donc réduire leur taille afin de réduire le temps de traitement et aussi résoudre un problème d'importation des images sur Google Drive pour les traiter sur Google Colab.
Cette réduction est faite avec la librairie PIL, puis, le fichier contenant tous les blasons est reconstruit.

```
bigImg = Image.open(BytesIO(base64.b64decode(img_b64)))
smallImg = bigImg.resize((50, 50), Image.NEAREST)
```

2.3.2 Réseaux neuronaux

Le programme principal du projet ne consiste qu'en l'importation et l'exécution des huit réseaux de neurones précédemment modélisés et entraînés.

Le code de chacun des réseaux est plus ou moins similaire aux autres, les différences résident surtout dans la présence de convolutions ou non, et dans le nombre de neurones par couche et de couches de neurones. Ces différences s'expliquent de manière arbitraire en fonction des résultats obtenus avec chaque réseau. En effet, certains réseaux sont plus propices à l'utilisation de convolution. La reconnaissance de la division par exemple, ne peut pas être réalisée sans convolution, puisque des résultats de l'ordre de 25% seraient observés. Les convolutions permettent donc d'obtenir de bien meilleurs résultats, avoisinant les 98%. Il est donc nécessaire de modéliser les réseaux une première fois, d'observer les résultats et de modifier la disposition des cellules ou la présence de convolutions afin d'obtenir des résultats corrects. Cependant, ceux-ci ralentissent le processus d'entraînement il ne faut donc pas en abuser. Le plus gros du travail de conception d'un réseau réside dans la répétition d'apprentissage et de tests en modifiant la structure du réseau jusqu'à obtenir des résultats potables.

Le code suivant représente la création d'un réseau de neurones comportant des convolutions. Le réseau est composé de huit couches de neurones, dont quatre couches destinées à effectuer deux opérations de convolution. Les couches suivantes sont des couches de neurones classiques, qui permettent la sortie de neuf valeurs, représentant les neuf valeurs possibles qui peuvent être prises par l'image dans cet exemple. C'est pourquoi la dernière couche est représentée par neuf neurones, c'est la couche de sortie.

```
model1 = tf.keras.Sequential([
```

```
    tf.keras.layers.Conv2D(32, (3,3), padding='same', activation=tf.nn.relu,
                           input_shape=(IMG_SIZE, IMG_SIZE, 4)),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Conv2D(64, (3,3), padding='same', activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D((2, 2), strides=2),
    tf.keras.layers.Flatten(),
    #tf.keras.layers.Flatten(input_shape=(IMG_SIZE, IMG_SIZE, 4)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(7, activation=tf.nn.softmax)
])

model1.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
```

Afin de diviser les données en un jeu d'apprentissage et de test, il est nécessaire de choisir une proportion de données. 75 % des données sont attribuées au jeu d'apprentissage et le reste au jeu de test. Il est nécessaire d'appliquer le petit bout de code suivant pour effectuer ceci.

```
nombre_donnees = round(len(imgs),-1)
nombre_donnees_apprentissage = int(round(nombre_donnees*0.75,-1))
# Pour l'entrainement
X = imgs[:nombre_donnees_apprentissage]
y = class_num[:nombre_donnees_apprentissage]
# Pour les test
X_test = imgs[nombre_donnees_apprentissage:nombre_donnees]
y_test = class_num[nombre_donnees_apprentissage:nombre_donnees]
```

Concernant la partie la plus critique de chacun des programmes, c'est-à-dire la classification des images par isolement d'information importante de leur description, il est nécessaire pour chaque programme d'écrire une fonction d'isolement. Cette fonction permet donc d'isoler l'information importante présente au sein de la description afin de classifier le blason et ainsi de construire un tableau de toutes les classifications des blasons, utile pour l'apprentissage du réseau de neurones, mais aussi pour les tests.

La fonction suivante en est un exemple. Elle représente la fonction permettant d'isoler la division de la description du blason. Pour chacune de ces fonctions il est nécessaire d'analyser les différentes formes que peuvent prendre les descriptions, en effet, une information donnée ne prend pas toujours la même forme dans la phrase de description du blason et il sera donc nécessaire de prendre en compte toutes ces formes pour bien isoler l'information importante.

```
desc = desc.replace(',','').replace('imperial','')
desc = desc.split(' ')

    if "gyronny" in desc:
        return "gyronny"

    if "quarterly" in desc:
        return "quarter"

    if "per" in desc:
        if "sinister" in desc:
            return "bend sinister"

        index_per = desc.index("per")
        return desc[index_per+1]
    else :
        return "aucun"
```

Ici, la fonction est assez simple. Certains mots-clés tels que “gironny” ou “quarterly” indiquent tout de suite la catégorie du blason. Autrement celle-ci est précédée par le mot “per”, il est donc judicieux de supprimer le mot “imperial” au début de la fonction, permettant ainsi d’éviter tout problème lors de la recherche de la séquence de lettres “per”. Il est assez simple de comprendre le programme, les seules subtilités résident dans les différents cas spéciaux, qu’il est nécessaire de gérer avec des conditions rébarbatives. Toutes les fonctions d’isolement se structurent de la même manière et suivent cette idée, certaines sont plus complexes de par le nombre de cas spéciaux, mais aucune ne se démarque réellement des autres.

Il est ensuite nécessaire de sauvegarder les réseaux de neurones afin de les exporter.

```
model.save('OrdinaryCNN.h5');
```

Le code précédent permet naturellement de sauvegarder le réseau au sein d’un fichier. Il est donc ensuite possible, lors du programme final, d’importer tous les fichiers des modèles précédemment entraînés. Le programme final est donc finalement assez simple, il consiste seulement à l’importation et à l'exécution successive de tous les réseaux de neurones.

Pour que notre programme de machine learning puisse fonctionner sur des blasons trouvés dans des livres par exemple, il est nécessaire de l'entraîner sur des blasons déformés.

2.3.3 Transformations sur les images

Pour faire des transformations sur les images, la classe ImageDataGenerator de Keras est utilisée. Cet objet permet de faire des transformations sur les images en choisissant les paramètres. Un exemple de paramètre:

```
datagen = ImageDataGenerator(
      rescale=1.3/255,
      rotation_range=10,
      zoom_range=0.2,
      brightness_range=(0.3, 1.0),
      width_shift_range=0.1,
      fill_mode='constant',
      cval=255)

#Entrainement
model.fit_generator(
        datagen.flow(X, y, batch_size=32),
        epochs=6,
        validation_data=(X, y))
```

Cet objet permet de produire des images transformées aléatoirement avec une rotation de + ou - 10°, un zoom de + ou - 20%, une luminosité de 30% à 100% ou un décalage de l'image jusqu'à 10% vers la gauche ou la droite.

La méthode fit_generator permet de faire ces transformations en même temps que l'entraînement du réseau et est plus efficace.

3. Résultats

3.1 Test/Validation

Pour vérifier que le réseau de neurones fonctionne, il faut vérifier sa capacité à effectuer des prédictions exactes. Après avoir entraîné le réseau de neurones, il faut le tester avec des blasons qu'il n'a jamais rencontrés lors de son entraînement. La moyenne de réussite obtenue lors de ce test permet d'inférer sur sa capacité de déduction.

Comme expliqué précédemment, les réseaux de neurones ont des attributs qu'il est possible de changer lors de leur création, et c'est donc sur ces paramètres qu'il est important de jouer afin de se rapprocher d'un résultat optimal. Il est important de noter que le résultat maximal est de 100 % de réussite, mais que lorsqu'un réseau complexe atteint plus de 95%, il est déjà très raisonnable de s'arrêter là. Il n'a pas été nécessaire, lors de ce projet, de beaucoup modifier les différents paramètres des réseaux car il a été assez naturel d'obtenir de très bons résultats.

L'utilisation de convolutions, était dans un premier temps le facteur le plus impactant sur les réseaux. Le seul réseau ne nécessitant pas de convolutions est celui qui consiste à repérer la couleur de fond, puisque la comparaison ayant été faite le réseau sans convolution atteint une précision* de 0.9997 contre 1 avec des convolutions. Pour tous les autres modèles, l'utilisation des convolutions a été privilégiée.

Les phases de tests ne peuvent pas réellement être illustrées ici, et des écritures de tests comme il est possible de faire dans certains langages ne s'appliquent pas ici. Seuls des tests au fur et à mesure de l'avancée du projet, permettant d'atteindre de bons résultats en modifiant des valeurs ou des agencements de réseaux de neurones, ont pu permettre l'avancée du projet et son aboutissement. La mise en place d'un réseau était en fait constituée d'une phase de tests et de répétitions jusqu'à obtenir un résultat correct.

Pour vérifier le fonctionnement de l'ensemble du projet, il faut regrouper tous les réseaux afin de tous les effectuer l'un après l'autre sur un même blason afin de construire sa description à partir des résultats de chaque réseau. Si le programme final ressorts les données attendues, c'est-à-dire les caractéristiques de la description du blason donné, alors tous les réseaux de neurones ont étés correctement entraînés.

3.2 Documentation technique

Le programme final peut être utilisé sur n'importe quel ordinateur possédant les dépendances de langages et de librairies. Pour un usage simplifié il est recommandé d'utiliser Google Colab. Pour blasonner une image personnelle, il est nécessaire d'importer celle-ci sur Google Drive, puis d'ouvrir le programme final sur Google Colab. L'utilisateur doit ensuite lancer le programme final pas à pas en veillant de modifier le chemin vers l'image à blasonner. Une fois cela fait, il est donc possible d'avoir accès à la description presque complète du blason.

3.2.1 Les blasons

Les blasons utilisés pour l'entraînement des réseaux sont contenus dans le fichier “blasons50_b64.txt”. Dans ce fichier, chaque ligne correspond à un couple de description de blason et d’image du blason correspondant en base64 et en résolution 50 pixels par 50 pixels. La description du blason et son image du blason sont séparées par un point-virgule.

3.2.2 Les réseaux de neurones

Chaque code contenant un réseau de neurones est stocké dans un fichier Google Colab, et chaque réseau pré-entraîné est sauvegardé dans un fichier contenu dans le dossier “modelSaved”. La plupart des réseaux de neurones utilisent des convolutions car cela est plus efficace. Des transformations sont appliquées aux blasons via un objet ImageDataGenerator du module Keras de TensorFlow.

3.2.3 Le regroupement des réseaux

Le regroupement des réseaux se fait au sein d’un nouveau fichier Google Colab qui permet l’importation de tous les réseaux et l’application de ceux-ci sur des images de notre base de données. Il est aussi possible de tester le programme sur des images importées dans Google Drive.

3.3 Résultats

Les résultats obtenus par les réseaux sont autour des 95% si les images passées en entrées sont des images déformées virtuellement ou originales. En revanche, les images prises par photographies sont très mal reconnues par les réseaux. Ces résultats sont plutôt décevants concernant les photographies mais sont tout à fait acceptables concernant les attentes du blason.

4. Gestion de projet

Dans cette partie, nous allons expliquer comment nous nous sommes organisés lors de la réalisation de ce projet. Nous aborderons donc la démarche personnelle, la planification des tâches, des outils utilisés et le bilan critique.

4.1. Démarche personnelle

Au début du projet, nous avons commencé par nous former au langage de programmation Python bien que certains membres de l'équipe y étaient déjà formés. Ensuite, nous avons appris à utiliser le framework d'intelligence artificielle Tensorflow afin de développer un programme de Machine Learning. Pour cela, nous avons suivi un cours présent sur la plateforme d'apprentissage Udacity [10]. L'utilisation de ce tutoriel et pas d'un autre a été faite sur les conseils de M. Poupet.

Cette partie du projet était plutôt personnelle et demandait du travail en autonomie. Cependant, nous nous sommes tout de même entraidés et avons posé des questions à notre responsable de projet afin de comprendre au maximum les notions nécessaires à l'aboutissement du projet.

Concernant notre organisation, nous avons travaillé sur ce projet en équipe, que ce soit à l'IUT tous ensemble mais également à distance depuis chez nous. Pour travailler plus efficacement sur une partie du code, nous avons réalisé des vidéoconférences à l'aide de Discord et avons utilisé sa fonctionnalité de partage d'écran, ainsi que la fonction Live Share de Visual Studio Code permettant de travailler à plusieurs sur un même projet à la manière d'un Google Doc.

Pour développer notre projet, nous avons utilisé un Google Drive et Google CoLab. Tous les membres du groupe ainsi que le tuteur avaient accès à ce drive. Google CoLab est un environnement de développement en ligne, le code est compilé et exécuté sur les serveurs de Google, ce qui est particulièrement pratique pour le machine learning car ce domaine requiert souvent de la puissance de calcul selon l'algorithme et la taille de l'ensemble de données. Il a été tout naturel de déposer les images de blasons sur le drive puisque Google CoLab permet aussi d'accéder aux fichiers stockés sur celui-ci. Ces outils nous ont permis d'avoir tous accès aux mêmes fichiers et de ne pas avoir de problèmes de version de nos programmes entre les membres du groupe.

Lors du développement du projet, nos principales sources d'informations ont été la documentation officielle de Tensorflow[9], le cours Udacity[10], Youtube ainsi que le site internet Stack Overflow [11].

Il a été nécessaire de rencontrer notre tuteur de projet à plusieurs reprises afin de l'informer des différentes avancées lors de notre projet, mais aussi afin de définir des objectifs à court terme. Des problèmes techniques ont aussi requis la mise en place d'une réunion afin d'obtenir l'aide de M. Poupet. Ces réunions étaient organisées ponctuellement

par notre tuteur ou par notre propre initiative.

4.2. Planification des tâches

Le but de ce projet était aussi d'apprendre le machine learning, il nous a donc été difficile de réaliser une planification des tâches car nous n'avions pas encore connaissance des différentes étapes à réaliser et du temps nécessaire requis pour chacune d'entre elles. Malgré cela, nous nous sommes défini des objectifs à court terme avec l'aide de M. Poupet et nous avons mis en place un Trello [cf figure 4] [12] . Les tâches sur le Trello ont été ajoutées au fur et à mesure de l'avancée du projet.

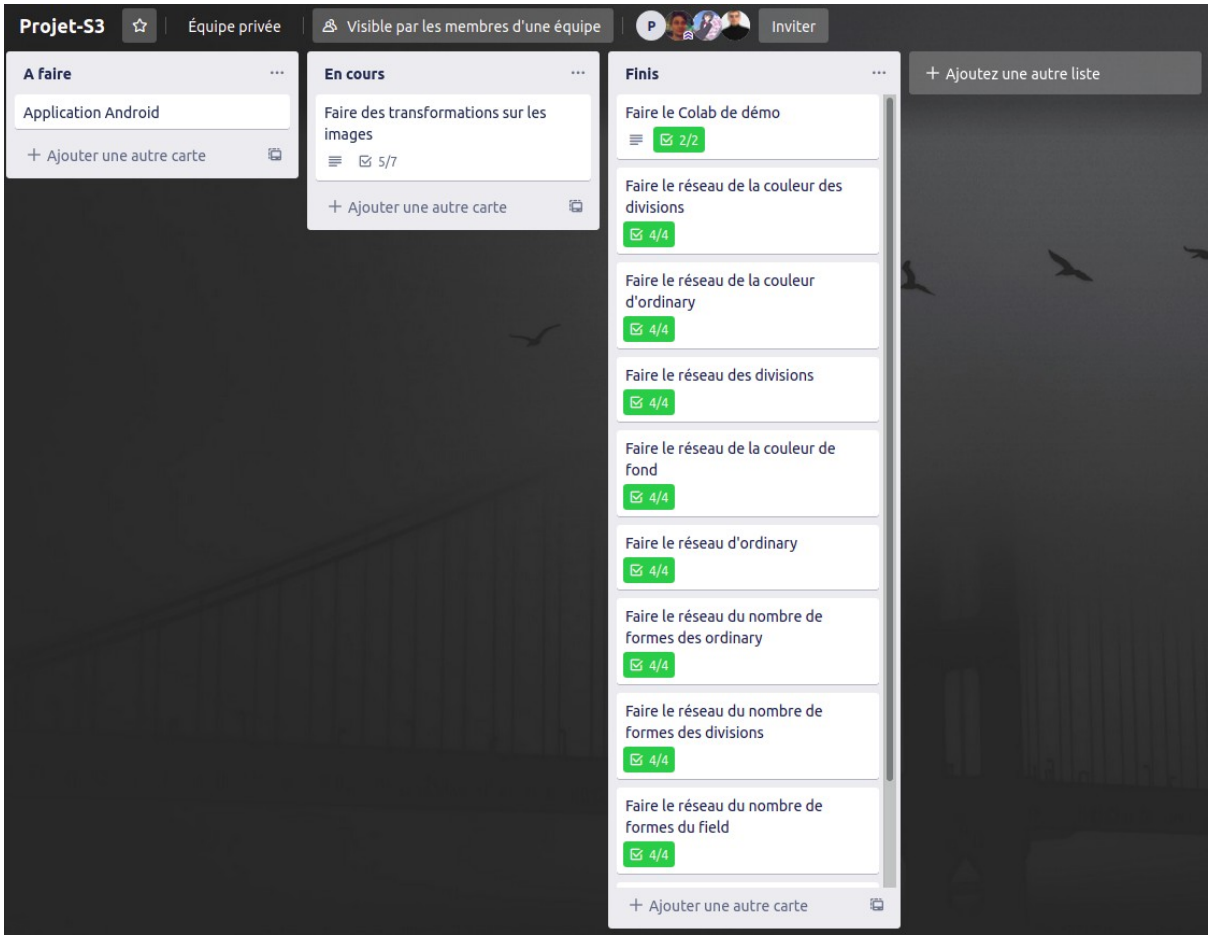


Illustration 4: Capture d'écran de notre Colab

Nous avons appris le machine learning en même temps que nous avons fait le projet. Nous avons perdu du temps sur des problèmes techniques ou à chercher comment réaliser une fonctionnalité. Par exemple, nous avons mis beaucoup de temps à trouver un bon format pour que l'importation des images soit suffisamment rapide. De même, nous avons mis du temps à trouver un moyen aisé de réaliser des transformations sur les images.

4.3. Bilan critique

Les méthodes de gestion de projets et le Trello ont assez peu étés utilisés lors du projet. Cela a eu un impact négatif sur l'avancée du projet. En effet, ayant pris du retard sur le début du projet car le machine learning était tout nouveau pour nous, l'organisation du projet étant mauvaise, nous ne savions pas par où commencer. L'organisation du projet pourrait donc être améliorée en utilisant une méthode de gestion de projet efficace. Une plus grosse utilisation de Trello pour suivre les avancements et la définition des tâches, serait aussi bénéfique dans le cadre d'un futur projet. Les résultats obtenus concernant la reconnaissance de blasons pris par photo par exemple ne sont pas à la hauteur de nos espérances. En revanche, la reconnaissance de blasons déformés et de blasons générés par le site WorldSpinner se fait sans aucun problème.

Conclusion

En conclusion, l'objectif de ce projet était de proposer un algorithme qui puisse nommer les blasons. Nous avons réalisé un programme qui détermine des blasons ayant des déformations. Nous avons surmonté les principales difficultés rencontrées qui ont été le formatage des données pour les réseaux de neurones. Nous avons beaucoup appris techniquement sur le langage python et le machine learning via la librairie TensorFlow. Ce projet ne marque pourtant pas la fin de ce travail puisque des ajouts pourraient être faits. En effet, l'efficacité des réseaux de neurones pourrait être améliorée afin de reconnaître des images photographiées depuis des environnements réels tels que des livres. Enfin, la portabilité du programme pourrait être développée afin de proposer celui-ci au grand public et ainsi démocratiser cette activité qu'est le blasonnement.

Bibliographie

[1] Worldspinner LLC, « Heraldry, Device, Coat of Arms editor and creator ». [En ligne]. Disponible sur: https://worldspinner.com/heraldry/device_editor/. [Consulté le: 15-oct-2019].

[2] Google, « TensorFlow », *TensorFlow*. [En ligne]. Disponible sur: https://www.tensorflow.org/?hl=fr. [Consulté le: 18-oct-2019].

[3] Base64 Guru, « What is Base64? | Learn | Base64 ». [En ligne]. Disponible sur: https://base64.guru/learn/what-is-base64. [Consulté le: 05-janv-2020].

[4] Google Developers, « Machine Learning Glossary », *Google Developers*. [En ligne]. Disponible sur: https://developers.google.com/machine-learning/glossary?hl=fr. [Consulté le: 02-déc-2019].

[5] É. Larousse, « Définitions : blasonnement - Dictionnaire de français Larousse ». [En ligne]. Disponible sur: https://www.larousse.fr/dictionnaires/francais/blasonnement/9770. [Consulté le: 02-déc-2020].

[6] M. Mouadil, « Introduction au Deep Learning », *Meritis*, 23-févr-2018. [En ligne]. Disponible sur: https://meritis.fr/ia/deep-learning/. [Consulté le: 05-janv-2020].

[7] D. Mishra, « Convolution Vs Correlation », *Medium*, 13-nov-2019. [En ligne]. Disponible sur: https://towardsdatascience.com/convolution-vs-correlation-af868b6b4fb5. [Consulté le: 05-janv-2020].

[8] C. Severance, « Notepad++ Plugins - Browse /TextFX at SourceForge.net ». [En ligne]. Disponible sur: https://sourceforge.net/projects/npp-plugins/files/TextFX/. [Consulté le: 05-janv-2020].

[9] Google Developers, « Module: tf | TensorFlow Core r2.0 », *TensorFlow*. [En ligne]. Disponible sur: https://www.tensorflow.org/api_docs/python/tf?hl=fr. [Consulté le: 05-janv-2020].

[10] Google and Udacity, « Intro to TensorFlow for Deep Learning | Udacity ». [En ligne]. Disponible sur: https://www.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187. [Consulté le: 05-janv-2020].

[11] Stack Overflow, « Stack Overflow - Where Developers Learn, Share, & Build Careers », *Stack Overflow*. [En ligne]. Disponible sur: https://stackoverflow.com/. [Consulté le: 05-janv-2020].

[12] Trello, « Trello ». [En ligne]. Disponible sur: https://trello.com. [Consulté le: 05-janv-2020].

Annexes

Table des annexes

Annexe 1 : Code du script générant les images et leurs descriptions

```
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium import webdriver
import base64
from PIL import Image
from io import BytesIO


url = 'https://worldspinner.com/heraldry/device_editor/'
chrome_options = Options()
chrome_options.add_argument("--headless")  # no window
driver = webdriver.Chrome(options=chrome_options)
driver.get(url)

f = open("blasons_png.txt", 'a')
i = 11638
while i <= 15000:
    i = i+1
    while True:
        try:
            img_b64 = WebDriverWait(driver, 20).until(
                lambda driver:
driver.find_element_by_css_selector('img.device-
preview').get_attribute('ng-src'))[22:]
            break
        except:
            pass
    nom = driver.find_element_by_css_selector('em.ng-
binding').get_attribute('innerHTML')[1:-1]

    im = Image.open(BytesIO(base64.b64decode(img_b64)))
    im.save("images/"+str(i)+".png", 'PNG')
    f.write(str(i) + ';' + nom + '\n')
    print(nom)
    print(str(i) + "\n")

    while True:
        driver.find_element_by_class_name('brass-button-small').click()
        #  wait end of js function trigged by the click
```

```
        try:
            WebDriverWait(driver, 20).until(
                lambda driver: driver.find_element_by_css_selector('em.ng-
binding').get_attribute('innerHTML')[1:-1] != nom)
            break
        except:
            pass
```

Annexe 2 : Lien vers les fichiers Google Colab

Annexe 3 : Script permettant de réduire la taille des images

```
from PIL import Image
import os

IMG_DIR = "./images"
RESIZED_IMG_DIR = "./images50"

WIDTH = 50
HEIGHT = 50

for root, dirs, files in os.walk(IMG_DIR):
  for filename in files:
    #print(filename)
    bigImg = Image.open(os.path.join(IMG_DIR, filename))
    im = bigImg.resize((WIDTH, HEIGHT), Image.NEAREST)
    im.save(os.path.join(RESIZED_IMG_DIR, filename))
```

Annexe 4 : Script permettant de joindre la description et la représentation base64 des images50

```
from PIL import Image
import os
import base64

IMG_DIR = "./images50"
INPUT_FILE = "./no dup blasons_png.txt"
OUTPUT_FILE = "./blasons50_b64.txt"

with open(INPUT_FILE, 'r') as f:
  with open(OUTPUT_FILE, 'w') as out:
    for line in f.readlines():
      num, desc = line.split(';')
      desc = desc.rstrip()
      encoded = base64.b64encode(open(os.path.join(IMG_DIR, num + '.png'
), "rb").read())
      out.write(desc + ';' + encoded.decode("utf-8") + '\n')
```

GRILLE DE RELECTURE DU RAPPORT

Cocher au moyen d'une croix chacun des items après vérification.

	Pour l'ensemble du rapport de projet	
	Mise en forme du propos	
X	**Police** de caractère et de taille uniforme pour le corps du texte (ex. : Times ou Calibri, 12 pt).	
X	**L'alignement** est justifié	
X	La présentation du texte est aérée : marges et interlignes raisonnables ; espace avant/après le paragraphe ; espace après les titres et sous-titres (Format > paragraphe)	
	Les titres	Les **titres des grandes parties** et différents avant-textes et post-textes **figurent en haut des pages** (insertion de sauts de page afin d'éviter les décalages à chaque modification)
		Ils ne sont pas suivis d'un signe de ponctuation (« : » ou « . »)
X	**En-têtes et pieds de pages**	L'**en-tête** comporte le nom du rapport et le nom des auteurs (sauf sur les avant-textes et les post-textes qui n'ont pas d'en-tête).
X		L'en-tête commence à l'introduction et disparaît après la conclusion
X		Les **pieds de page** comportent les numéros de pages sur celles devant être numérotées
	Formulation du propos (langue, expression)	
X	L'intégralité du texte est passée au **correcteur orthographique**, a été relue par une tierce personne et a été corrigée. Le texte ne comporte plus aucune faute.	
X	La **ponctuation** est utilisée de façon pertinente : articulation des idées, pauses, phrases de longueur raisonnable permettant une bonne compréhension des propos.	
X	Fréquemment utilisés, les **connecteurs logiques** permettent la compréhension des propos en reliant les idées logiquement (cause, conséquence, exemples, but, etc.)	
X	Les **prénoms et noms** des personnes sont indiqués dans l'ordre Prénom + NOM (en majuscules)	
X	Monsieur est abrégé « M. » et non « Mr » comme en anglais	
X	Les **chiffres** sont écrits en lettres : « dix heures », « vingt-cinq employés », sauf exception [chiffre suivi d'une unité de mesure par exemple : 10 Mo, 10 km/h…]	

X	**Relecture orthographique**	Indiquez les mots dont l'orthographe est à vérifier pour le rapport [ex. : base de données, programmation multi-agents…]
	Les avant-textes	
	Numérotation des pages en chiffres romains [sauf la page de couverture]	
X	**Page de couverture**	Le **titre du rapport** indiqué sur la page de couverture décrit précisément le projet.
X		Sans numérotation
X		Type de document : rapport de projet
X		Logos des institutions [université de Montpellier, IUT]
X		Indiquer le diplôme préparé
X		Auteurs, tuteurs/tutrices
X		Année universitaire
X	**Page de garde**	Page vide
X	**Remerciements**	Citez les personnes qui vous ont aidés et soutenus.
X	**Sommaire**	Fidèle au plan appliqué par la suite
X		Pas plus de 3 niveaux de titre
X		Il n'y a **pas de titres « orphelins »** dans le plan. Ex 1.1.1 [et pas de 1.1.2]
		Il est inséré automatiquement et a été mis à jour une dernière fois avant l'impression
X		Il peut être cliquable dans sa version numérique [facultatif]
X	**Glossaire**	Il comporte les termes découverts pendant le projet
X		Pas de mots anglais pour lesquels un terme français existe
X		Les mots du glossaire sont **identifiables dans le texte au moyen d'un astérisque** lors de la 1ere occurrence.
X		Elle est insérée automatiquement et a été mise à jour une dernière fois avant l'impression

X	**Table des figures**	Toutes les figures sont présentes [numérotation + légende]
	Corps du document	
X	Les pages sont numérotées en chiffres arabes et reprennent à 1 [avant-textes en chiffres romains non comptabilisés — utiliser des sections pour cela]	
X	Le plan est cohérent et permet d'illustrer l'ensemble du projet	
X	Insertion de notes de bas de page pour donner des informations complémentaires ou suggérer des lectures/sources d'informations supplémentaires.	
X	**Articulation, enchaînement :** chaque nouvelle partie est précédée d'une transition [essentielle !]	
X	**Les titres**	Les titres des parties sont numérotés sous la forme 1.1.1 et sont indentés
		Pas de titres orphelins [ex. : 1.1.1 mais pas de 1.1.2]
		Ils ne sont pas suivis d'un signe de ponctuation « : » ou « . »
	Les renvois dans le texte	
	Figures et diagrammes :	Ils sont présents car <u>nécessaires</u> à la compréhension des propos [et non pour remplir une page, illustrer des éléments non explicités, etc.]
		L'ensemble des figures et diagrammes sont numérotés et légendés [De quoi s'agit-il ? Qu'apporte cet élément supplémentaire ?] avec un titre précis, et apparaissent dans la table des figures.
		Chaque figure et/ou diagramme est commenté et il y a un renvoi dans le texte indiquant au lecteur quand il doit la/le regarder [« cf. figure 1 »]. C'est une illustration du propos et non l'inverse.
		Les extraits/illustrations de codes doivent être lisibles [attention aux fonds noirs, aux caractères trop petits, etc.]
	Les annexes	Un renvoi est effectué dans le texte, indiquant au lecteur quand il doit consulter chacune d'entre elles.
X	**Citations [attention au plagiat, même involontaire !]**	Toute partie inspirée d'un document consulté doit mentionner la source selon les normes IEEE (numéro de la référence entre crochets Ex. : [2]) et permettre de retrouver le document dans la bibliographie.

		Les citations insérées sans modification respectent les normes de présentation IEEE
Progression détaillée		
X	**Introduction**	Présentation du projet [contexte, objectifs] et du plan du rapport
X		L'annonce du plan correspond à celui effectivement suivi
X	**Analyse**	Analyse du contexte, des besoins fonctionnels et non fonctionnels
X		Toute personne extérieure au domaine de l'informatique est en mesure de comprendre ce qu'elle contient.
X	**Rapport technique**	Justification des choix de conception et de développement
X		Il faut écrire cette partie en s'adressant à des informaticiens.
X	**Résultat**	En fonction de votre projet vous pouvez présenter les tests, un manuel d'utilisation et/ou d'installation.
X	**Rapport d'activité**	Planification et organisation du travail ; recul sur le travail effectué
X	**Conclusion**	Synthèse et bilan
	Les post-textes	
	Les pages sont numérotées en chiffres romains qui reprend à I	
X	**Bibliographie**	Elle doit impérativement mentionner les informations demandées dans une bibliographie comme l'auteur.e, la date de parution, etc. Il ne peut en aucun cas s'agir d'une liste d'adresse URL.
X		Présentée selon les normes IEEE [utilisation de Zotero vivement conseillée !]
	Annexes techniques	Les annexes sont présentées dans une table
		Toutes les annexes portent un titre et sont numérotées
		Un renvoi est effectué dans le texte, indiquant au lecteur quand il doit regarder chacune des annexes.
X		Résumé rédigé en français ET en anglais, qui reprend de façon fiable le contenu du rapport. Il comporte environ une centaine de mots.

X	**La quatrième de couverture**	Mots-clefs relatifs au projet en français et en anglais

Résumé

Le projet consiste à nommer automatiquement des blasons suivant les règles de l'art héraldique. Cela permet donc aux personnes non initiées au blasonnement (nommage de blason) de pouvoir nommer les blasons qu'ils trouveraient dans un livre par exemple. Le programme prend en entrée un blason au format image et le programme reconnaît les couleurs et les différents éléments (division, ordinary) sur le blason afin de donner son nom en héraldique.

Le programme a été développé en Python à l'aide du Framework d'intelligence artificielle Tensorflow utilisant un algorithme de Machine Learning (apprentissage automatique).

Mots clés : *Apprentissage automatique*, *Tensorflow*, *Python*, *blason*, *héraldique*, *nommer un blason.*

Summary

The project consists in automatically naming blazons according to the rules of the art heraldic. This allows curious people who are not familiar with the blazon naming to be able to name the blazons they would find for instance in a book. The program takes as input a blazon in image format and the program recognizes the colors and the different elements (division, ordinary) on the blazon in order to give its name in heraldic.

The program was developed in Python with the Tensorflow AI Framework using a Machine Learning algorithm.

Keywords : *Machine Learning*, *Tensorflow*, *Python*, *blazon*, *heraldic*, *naming a blazon.*