MUSSARD Wassim

MOULARD Hugo

UPEC
Connaissance - Action
UNIVERSITÉ
PARIS-EST CRÉTEIL
VAL DE MARNE

Intelligence Artificielle

Captcha Resolver

Master Conception des Systèmes et Cybersécurité

Année 2024-2025

Kurosh Madani

Table des matières

Introduction

Contexte

Les Captchas, conçus pour distinguer les humains des robots, posent des défis uniques à la vision par ordinateur. Ce projet vise à développer un modèle d'intelligence artificielle capable de résoudre ces Captchas en classifiant des images comme contenant ou non des voitures. Le modèle est implémenté avec un **réseau neuronal convolutif** (CNN), en utilisant des outils comme *Keras* et *TensorFlow*.

Objectif

L'objectif de ce projet est de reproduire une intelligence artificielle capable de résoudre des Captchas, en appliquant les concepts étudiés en cours, notamment ceux liés aux réseaux de neurones convolutifs (CNN) et au traitement d'images.

Organisation du projet

Nom du fichier/dossier	Description
data	Contient le jeu de données d'entraînement (train) et de test (valid)
IA_car_CNN.ipynb	Notebook de l'IA de reconnaissance
IA_car_CNN_export.py	Notebook convertit en python (.py)
car_not_car_model.h5	Modèle IA obtenu à l'issu de l'entraînement
ia_car.py	Permet l'importation du modèle (.h5) et de faire la prédiction.
game.py	Code dédié à l'interface utilisateur

Préparation des données

Organisation des données

Les données sont divisées en deux ensembles, répartis en deux classes :

- Entraînement (train) : Pour pouvoir entraîner notre IA ;
- Test (valid) : Pour pouvoir évaluer notre IA à l'issue de son entraînement.

Cette organisation facilite le chargement des données avec des outils comme **ImageDataGenerator**.

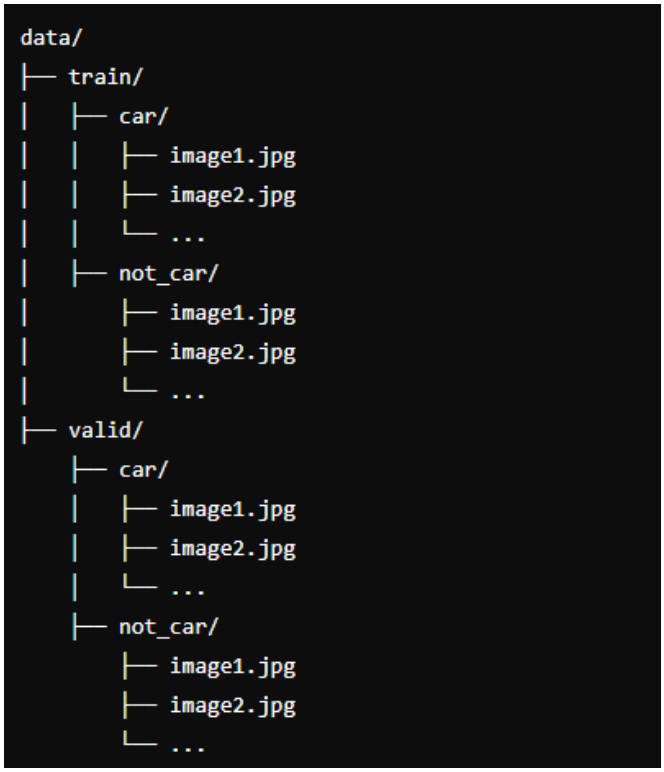

Figure 1 : Représentation de notre arborescence de fichiers

Choix de la cible

Le choix de la « *target* » (ou variable cible) dans notre modèle repose sur la nature de la tâche que nous cherchons à résoudre, à savoir une **classification binaire**. La cible représente ici la classe d'appartenance de chaque image, avec deux possibilités : 1 si l'image correspond à une voiture (*car*) et 0 si elle n'en est pas une (*not_car*). Ce format binaire est particulièrement adapté à notre problématique, car nous cherchons uniquement à différencier deux classes distinctes.

La conception de notre modèle a été pensée pour ce type de tâche : la couche de sortie utilise un unique neurone avec une activation sigmoïde, qui transforme les sorties du modèle en une probabilité comprise entre 0 et 1. Une valeur proche de 1 indique une forte probabilité que l'image appartienne à la classe « *car* », tandis qu'une valeur proche de 0 indique une probabilité élevée pour la classe « *not_car* ». Ce choix de target s'intègre également parfaitement avec la fonction de perte utilisée, à savoir la **binary cross-entropy**, qui mesure l'écart entre les prédictions probabilistes du modèle et les étiquettes réelles.

Les données étant organisées dans deux dossiers distincts (*car* et *not_car*), chaque image est automatiquement étiquetée selon son dossier d'appartenance. Cette approche permet une gestion efficace des données, tout en garantissant une correspondance explicite entre les images et leurs labels. Pendant l'entraînement, le modèle ajustera ses poids pour minimiser les erreurs entre les probabilités prédites et les étiquettes réelles. Par exemple, une image étiquetée comme « *car* » (1) entraînera une correction des poids si la probabilité prédite est trop proche de 0, et inversement pour une image « *not_car* » (0).

Préparation des données

La qualité et la structure des données jouent un rôle déterminant dans la performance des modèles de deep learning.

Taille et répartition : L'ensemble d'entraînement contient 418 images, tandis que l'ensemble de validation en contient 418. Les images sont équilibrées entre les deux classes « *car* » et « *not_car* ».

Augmentation des données : Les techniques suivantes ont été appliquées à l'ensemble d'entraînement pour améliorer la généralisation du modèle :

- Rotation aléatoire (jusqu'à 20°).
- Décalages horizontaux et verticaux (jusqu'à 20 %).
- Zoom aléatoire (jusqu'à 20 %).
- Inversion horizontale.

Enfin, une étape importante était de faire la normalisation. Cette étape permet de redimensionner les images pour avoir des valeurs comprises entre 0 et 1, ce qui accélère la convergence lors de l'entraînement.

```
rescale=1.0/255.0
```

Modèle utilisé

Pour le modèle, on a décidé de partir sur un modèle simple mais qui semble être efficace pour une tâche de classification binaire. Le modèle utilisé est un réseau neuronal convolutif (CNN) construit à l'aide de l'API Keras. Les CNN sont particulièrement adaptés pour les données visuelles grâce à leur capacité à capturer les relations spatiales.

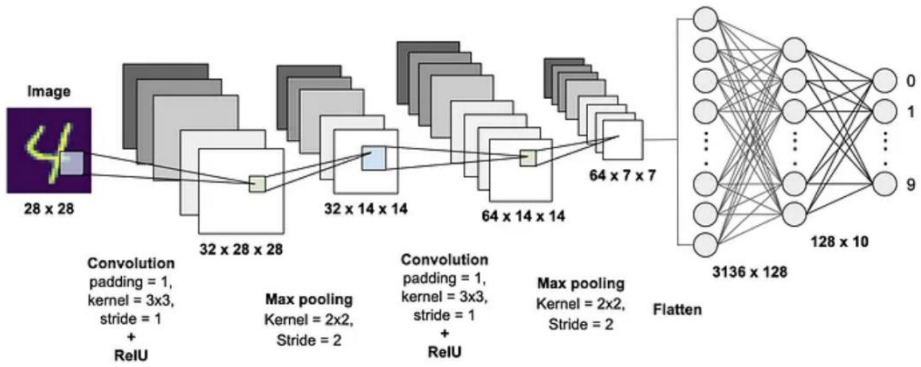

Figure 2 : Modèle CNN (*cf. link*)

Entrée :

- Les images sont redimensionnées à une taille fixe de 100x100 pixels pour uniformiser les dimensions. Cette taille d'entrée a été choisie pour standardiser les images et réduire leur complexité. Une image plus grande (par exemple 512x512) aurait nécessité beaucoup plus de puissance de calcul. Ainsi, en redimensionnant toutes les images à 100x100, on simplifie le traitement tout en gardant des informations visuelles suffisantes pour la classification.

Couches convolutives :

- Ces couches utilisent des filtres de taille 3x3 pour extraire des caractéristiques locales comme les bords et les textures. Ce qui est intéressant, c'est que la convolution permet de détecter des motifs à petite échelle, ce qui est essentiel pour identifier les contours des objets (par exemple, les bords d'une voiture).
- La fonction d'activation « *ReLU* » permet d'introduire de la non-linéarité dans le modèle.

Pooling (MaxPooling) :

- Cette fonction nous permet de réduire les dimensions en sélectionnant les valeurs maximales dans une fenêtre de 2x2. Réduire la taille des cartes de caractéristiques diminue le nombre de paramètres du modèle, ce qui le rend plus rapide et moins sujet au sur-apprentissage.

Connexion des couches *(flatten)* :

- La fonction Flatten permet d'aplatir les données multidimensionnelles (comme les cartes de caractéristiques 2D générées par les couches convolutives) en un vecteur 1D.
- On l'utilise ici pour connecter les couches convolutives (2D) aux couches entièrement connectées (1D).
- La couche suivante est une couche « *Dense* » et les couches denses nécessitent une entrée unidimensionnelle. Ainsi, la couche « *Flatten* » est donc essentielle pour effectuer cette transition.

Couches complètement connectées (Dense) :

- Après l'extraction des caractéristiques, ces couches effectuent la classification binaire (voiture ou non). Les couches denses permettent de **combiner toutes les caractéristiques extraites** par les couches convolutives pour prendre une décision.

Dropout :

- Pendant l'entraînement, cette **couche désactive aléatoirement 50 % des neurones de la couche précédente**. On utilise le « *Dropout* » pour éviter le sur-apprentissage (*overfitting*) en forçant le modèle à ne pas dépendre excessivement de certains neurones.

Sortie - *Dense (1 neurone, activation=Sigmoid)* :

- Une seule unité de sortie avec une fonction d'activation **sigmoïde**, renvoyant une probabilité entre 0 (not_car) et 1 (car). Un seul neurone est nécessaire pour produire une probabilité dans une tâche de classification binaire.

Ce qui nous fait un total de **8** couches principales :

- 2 couches convolutives ;
- 2 couches de pooling ;
- 1 couche de flatten ;
- 1 couche dropout ;
- 2 couches dense.

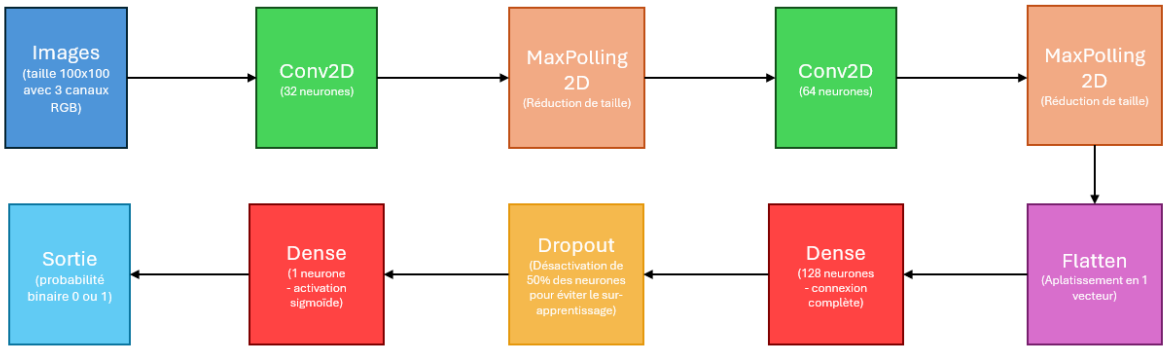

Figure 3 : Schéma simplifié du réseau de neurones

Entraînement et validation

Paramètres d'entraînement

On a utilisé l'optimiseur « *Adam* », qui est connu pour sa rapidité de convergence et son efficacité sur de nombreux problèmes avec une fonction de perte « *Binary Crossentropy* » qui adaptée aux tâches de classification binaire.

```
model.compile(optimizer=Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy'])
```

Résultats obtenus

Notre modèle a été entraîné sur plusieurs époques, et les performances ont été évaluées sur l'ensemble de validation. Notre IA a atteint une précision moyenne de **95 %** sur les données de validation. Cependant, quelques erreurs persistent, en particulier pour des images ambiguës ou présentant des objets qui peuvent visuellement être similaires à des voitures.

Analyse des performances

Points forts

Notre IA possède des **performances élevées**. En effet, le modèle présente une bonne précision sur l'ensemble de validation, démontrant la pertinence des techniques employées.

Limitations

Notre modèle est limité par la qualité et la diversité des images fournies. En effet une base de données plus importante améliorerait la capacité de généralisation.

D’autant plus que les CNN sont souvent perçus comme des "boîtes noires", rendant difficile l’analyse des erreurs. Et comme pour de nombreux modèles de *deep learning*, il est difficile de comprendre précisément pourquoi le modèle commet certaines erreurs.

Enfin, notre IA est uniquement orientée dans la reconnaissance d’images de voitures, mais elle n’est pas capable de reconnaître autre chose que des voitures.

Suggestions d’amélioration

Pour améliorer les performances de notre IA, on pourrait **augmenter la base de données** en ajoutant des images issues de sources variées ou des données synthétiques ou encore appliquer **des techniques d’explicabilité** comme « *Grad-CAM* » qui pourraient permettre d’identifier les zones d’intérêt sur lesquelles le modèle se concentre.

Résultats préliminaires

Voici les résultats que nous obtenons pour une IA entraînée avec **100 epochs**. Nous avons également ajusté le seuil de prédiction à **70%** permettant ainsi d’éviter que notre modèle puisse donner des résultats erronés :

Ainsi si le seuil est supérieur à **70%** alors l’image pourra être considérée comme étant une voiture.

Figure 4 : Résultats préliminaires

Résultats finaux

Voici les résultats que nous obtenons après avoir connecté notre IA à notre interface utilisateur :

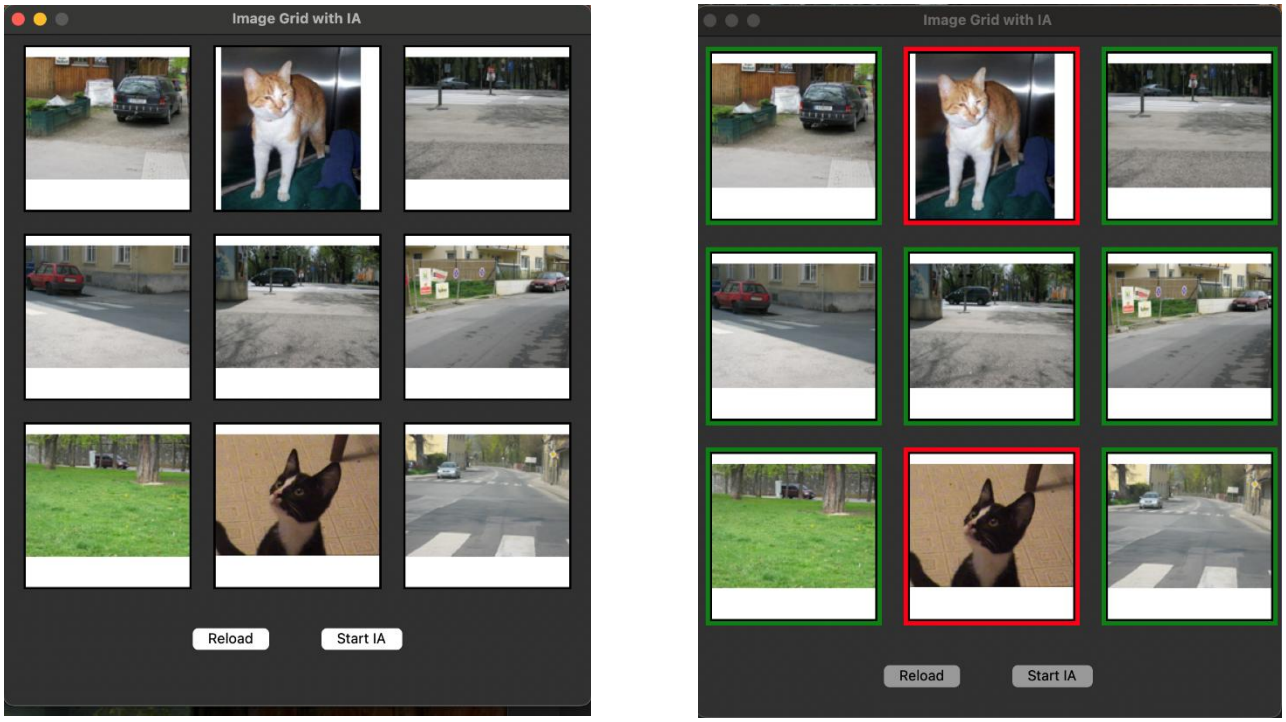

Figure 5 : Résultats finaux

Comme on peut le voir, notre IA a réussit à repérer les images « *car* » des images « *not_car* ».

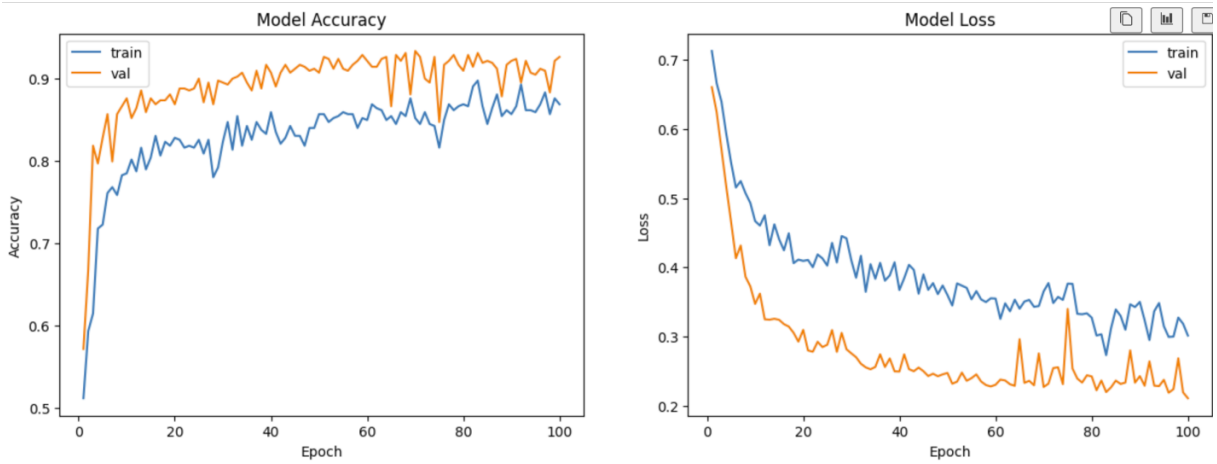

Figure 6 : Graphes exprimant la précision/perte en fonction du nombre d'epochs

Nous pouvons constater l'efficacité de notre modèle au fur et à mesure que le nombre d'epochs passe. En effet, pour le « *valid* » on atteint environ les **90%** de précision au bout de **50 epochs** avec un « **Loss** » de **0,24%**.

Axes d'améliorations

En prenant en considération les limitations de notre modèle, nous avons décidé de l'améliorer. En effet, toujours en se basant sur le modèle **CNN**, l'objectif *(notre cible)* de cette amélioration **permet de réaliser de la reconnaissance multiple** (*fleurs, chiens, chats et de voitures*).

Reconnaissance multiple

Cette nouvelle version de notre IA, repose sur les mêmes principes que la version précédente. Pour éviter la redondance du sujet, nous expliquerons les principales différences à prendre en compte.

Arborescence de fichiers

Pour ce faire, nous avons revu notre arborescence de fichiers, où dans le dossier de test *(valid)* et d'entraînement *(train)* n'avons plus de dossiers « car » et « not_car » mais « car », « flowers », « cats », « dogs » et « none ». Le dossier « none » contient un ensemble d'images ne présentant pas de « car », « flowers », « cats » ou « dogs ». Enfin les images du dossier de test sont différentes des images contenues dans le dossier d'entraînement.

De binaire à catégorique

Lors de la préparation des données, nous avions dû changer le mode de classification la passant de « **binary** » à « **categorical** ». En effet, car le mode « *binary* » ne permet pas de faire de la reconnaissance multiple.

Modification de la taille du dataset

Dans l'optique d'améliorer les performances de notre IA, nous avons profité de cette occasion pour augmenter la taille de notre dataset passant de **418** à **1084** images pour le dataset d'entraînement et de **418** à **704** images pour le dataset de test.

Modification du modèle

Nous avons modifié notre modèle CNN en ajoutant 2 couches supplémentaires :

- Conv2D(128, (3, 3), activation='relu') ;
- MaxPooling2D((2, 2)).

Ces ajouts de couches nous permettent d'avoir un modèle plus profond et d'obtenir des résultats qui nous rapproche de notre cible *(classification catégorique)*. En effet, avec l'ajout de la couche « Conv2D » et de ses **128 filtres**, cela permet à notre IA de capturer des informations plus complexes, notamment dans le cadre de classification précise.

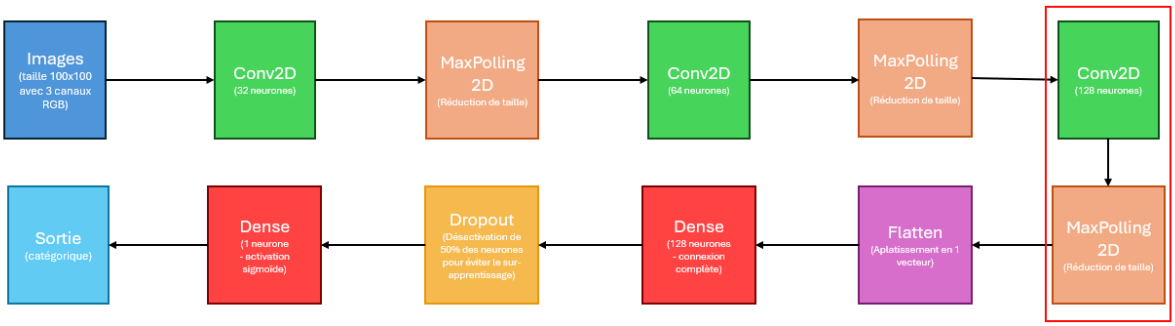

Figure 7 : Schéma simplifié du réseau de neurones

En sortie, nous obtenons une réponse catégorique. C'est-à-dire que si notre IA cherche une « *car* » alors :

- Soit elle nous dira **la probabilité que l'image soit une voiture**
- Soit elle nous dira que **l'image n'est pas une voiture** et ajoutera une probabilité que **l'image soit d'une autre catégorie** *(ex : dogs)*

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
L'image n'est pas un(e) car. Elle est probablement un(e) dogs avec une confiance de 0.59.
```

Résultats finaux

Voici les résultats que nous obtenons pour notre IA entraînée avec **125 epochs**. Nous avons également ajusté le seuil de prédiction à **50%**.

```
Model took 2960.13 seconds to train
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.9062 - loss: 0.2724
Test-set classification accuracy: 90.62%
```

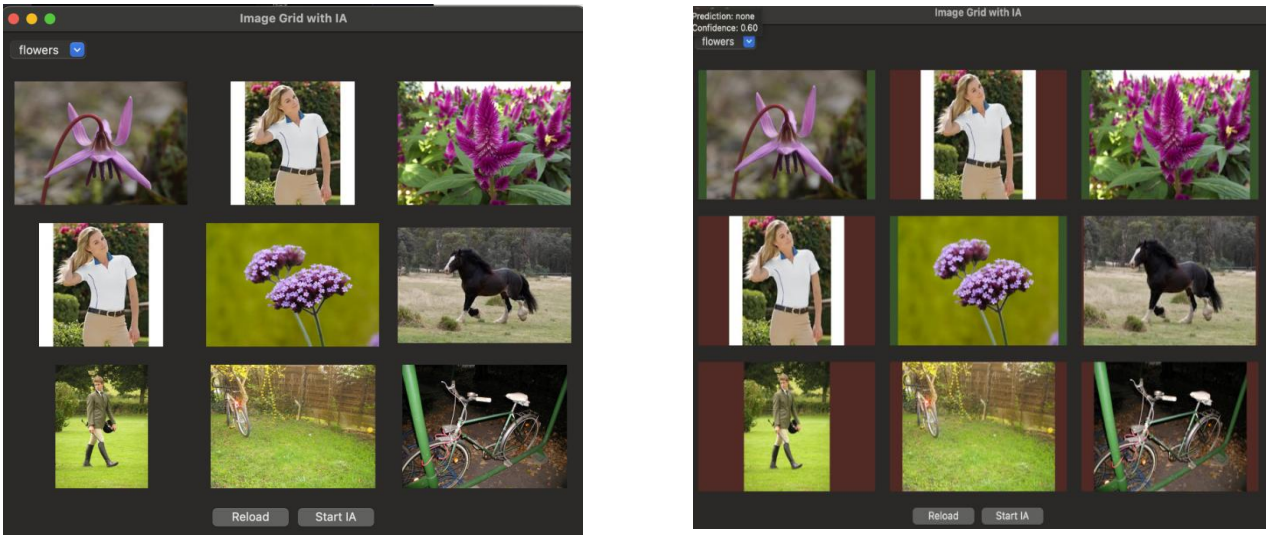

Figure 8 : Résultats finaux

Comme illustré dans la figure 8, nous avons un modèle qui a une précision de **90%** et notre modèle est capable d'effectué le test pour de la reconnaissance de fleurs (catégorie : *« flowers »*) et notre IA a réussit à différencier les fleurs des images « *none* ».

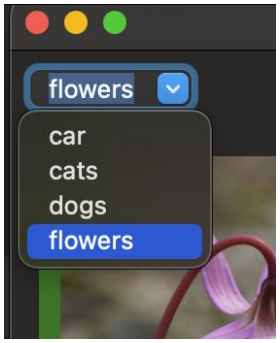

Figure 9 : Menu de sélection

Nous offrons également l'opportunité aux utilisateurs de choisir la catégorie de reconnaissance de notre IA à l'aide du menu de sélection situé en haut à gauche de la fenêtre.

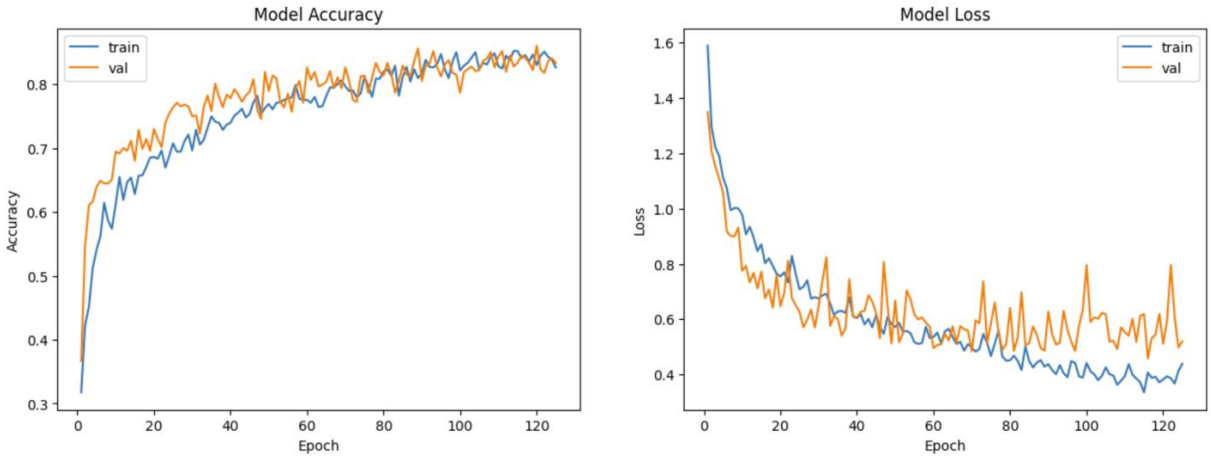

Figure 10 : Graphes exprimant la précision/perte en fonction du nombre d'epochs

Encore une fois, nous pouvons constater l'efficacité de notre nouveau modèle au fur et à mesure que le nombre d'epochs passe. En effet, pour le « *valid* » on atteint environ les **80%** de précision au bout de **60 epochs** avec un « **Loss** » de **0,6%**.

Conclusion

Dans un premier temps, ce projet a permis de reproduire une intelligence artificielle capable de résoudre des captchas en classifiant les images comme "car" ou "not_car". En appliquant les concepts théoriques appris, nous avons démontré l'efficacité des réseaux convolutifs. A l'issu de la création de la première version de notre modèle, nous avons obtenu des résultats prometteurs et identifié des axes d'amélioration comme l'augmentation de la base de données ou encore réaliser un modèle de classification catégorique.

Nous avons donc réalisé une deuxième version de notre modèle, toujours en se basant sur le modèle CNN, qui est capable de reconnaître différentes catégories (car, cats, dogs, flowers). Dès lors, on constate une capacité d'apprentissage plus complexe, un besoin d'apprentissage plus long. En effet, étant limité par la taille de notre base de données, le modèle semble faire plus d'erreurs que lors de la classification binaire. Cependant, il est tout de même capable de répondre à nos attentes, notre cible, c'est-à-dire de réaliser la classification catégorique.

Enfin, tout en s'inspirant du modèle CNN, nos modèles répondent donc bien à nos objectifs. Cependant, il reste encore des axes d'amélioration permettant de renforcer les performances et la robustesse de nos modèles, notamment avec l'utilisation de modèles plus avancés comme : Vision Transformers (ViT), Recurrent Neural Networks (RNN) ou encore Capsule Networks (CapsNet).