JEBBOUJ Hajar
RANDRUUT Boris
MAGAUD Marius

à l'attention de M. SKLAB Youcef

Détection automatique des plaques d'immatriculation

RAPPORT : Projet de Deep Learning

à Cergy pour le 25/01/2022

Table des matières

1 Introduction

À l'ère de la technologie, il est très difficile d'arrêter chaque véhicule sur la route et de vérifier sa plaque d'immatriculation lorsqu'une voiture commet une infraction. Avec l'augmentation de la fraude routière, les policiers deviennent également plus intelligents. Ils utilisent l'inteligence artificielle pour détecter les plaques d'immatriculation et en extraire le numéro d'immatriculation.

Notre but à travers ce projet était de pouvoir, à partir d'une image ou vidéo contenant des voitures, détecter automatiquement leurs plaques d'immatriculation, la lire, puis l'afficher en format textuel.

La première partie du projet va être la recherche d'un dataset contenant des images de voitures annotées selon la position de leur plaque d'immatriculation. Ensuite pour pouvoir repérer une plaque d'immatriculation au sein d'une image, il nous faut construire un modèle qui va prendre en entrée des images contenant des voitures, et extraire les coordonnées d'un rectangle (appelé bounding box) contenant leur plaque. Nous devons l'extraire par la suite de l'image pour le fournir à notre second réseau. Ce réseau va prendre en entrée la sortie du premier, à savoir une image sous forme de rectangle, contenant la plaque d'immatriculation de la voiture. Il donnera en sortie, du texte, représentant les caractères présents sur la plaque d'immatriculation de la voiture.

Pour ce qui est de l'organisation nous avons décidé de séparer le travail en deux parties, une première qui est la détection des plaques dans l'image, cette partie a été assurée par Hajar et Marius. Une fois la plaque extraite il faut en lire les caractères, cette partie-là a été gérée par Boris. La partie interface utilisateur quant à elle a été créée par Hajar.

2 Déroulement du projet

2.1 Lecture et préparation des images

Nous avons choisi d'entraîner et tester notre modèle de détection de plaques d'immatriculation sur un ensemble de 433 images de voitures, annotées sous le format PASCAL VOC. Ces images ont été récupérée sur Kaggle et sont de taille (267, 400).

(a) Image

```
<annotation>
    <folder>images</folder>
    <filename>Cars0.png</filename>
    <size>
        <width>500</width>
        <height>268</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>licence</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>226</xmin>
            <ymin>125</ymin>
            <xmax>419</xmax>
            <ymax>173</ymax>
        </bndbox>
    </object>
</annotation>
```

Bounding Box
Coordonnées de l'emplacement de la plaque d'immatriculation

(b) Annotation

Nous avons ensuite séparé ces images en 322 images pour entraîner notre modèle et le reste pour le tester. Puis, nous avons converti ces images en TF Records pour optimiser l'apprentissage du modèle.

2.2 Détection des plaques d'immatriculation

Pour pouvoir détecter les plaques d'immatriculation, nous avons adopté l'approche du Transfer Learning. A partir de Tensorflow Detection Model ZOO, nous avons choisi le modèle **SSD MobileNet V2 FPNLite 320x320** qui semble allier vitesse et précision.

2.2.1 Architecture du modèle

"SSD MobileNet V2 FPNLite 320x320" est un modèle de détection d'objets qui combine un modèle SSD (muni de Feature Pyramid Networks (FPN)) et un modèle MobileNet qui a été préentraîné sur le jeu de données COCO 2017.

a. SSD MobileNet

Un modèle SSD (Single Shot Detection) est un CNN qui apprend à prédire les positions des bounding boxes et à les classifier en un seul passage. Il est constitué d'une architecture de base (MobileNet dans notre cas) suivi par plusieurs couches de convolutions. Ce modèle contient aussi une phase de data augmentation (horizontal flip + random crop) et de redimmensionnement des images, nous n'avons donc pas besoin d'intégrer ces phases là dans le preprocessing de nos images.

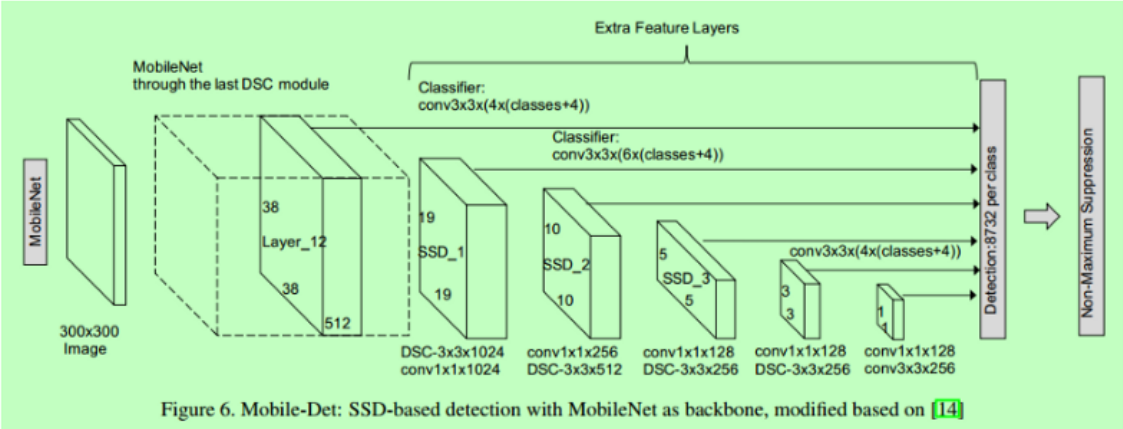


FIGURE 2 – Architecture de SSD MobileNet

L'avantage du modèle SSD est que nous pouvons détecté plusieurs objets dans une image en une seule fois. Contrairement au modèles de type RPN (regional proposal network) qui ont besoin de détecter les régions d'intêret présentes dans l'image avant de détecter les objets présents dans ces régions. Ainsi, le modèle SSD est beaucoup plus rapide.

L'architecture de base utilisée ici est un modèle MobileNet v2. Ce modèle est constitué de deux blocs, le premier pour un stride de 1 et le deuxième pour un stride de 2. Il y a trois couches dans chacun des deux blocs : une couche de convolution 1x1, une couche de Depthwise Convolution qui applique un filtre sur chaque canal, contrairement à la convolution classique qui applique un filtre sur l'ensemble des canaux, puis une autre couche de convolution 1x1. La différence entre MobileNet v1 et v2 est que la fonction d'activation de la dernière couche est linéaire au lieu de Relu6.

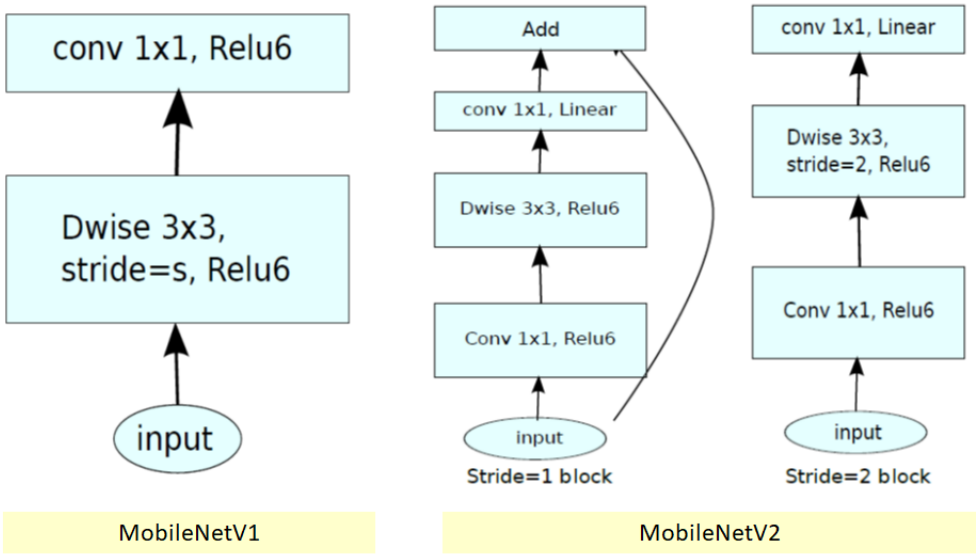


FIGURE 3 – Architecture de MobileNet v2

b. Feature Pyramid Network (FPN)

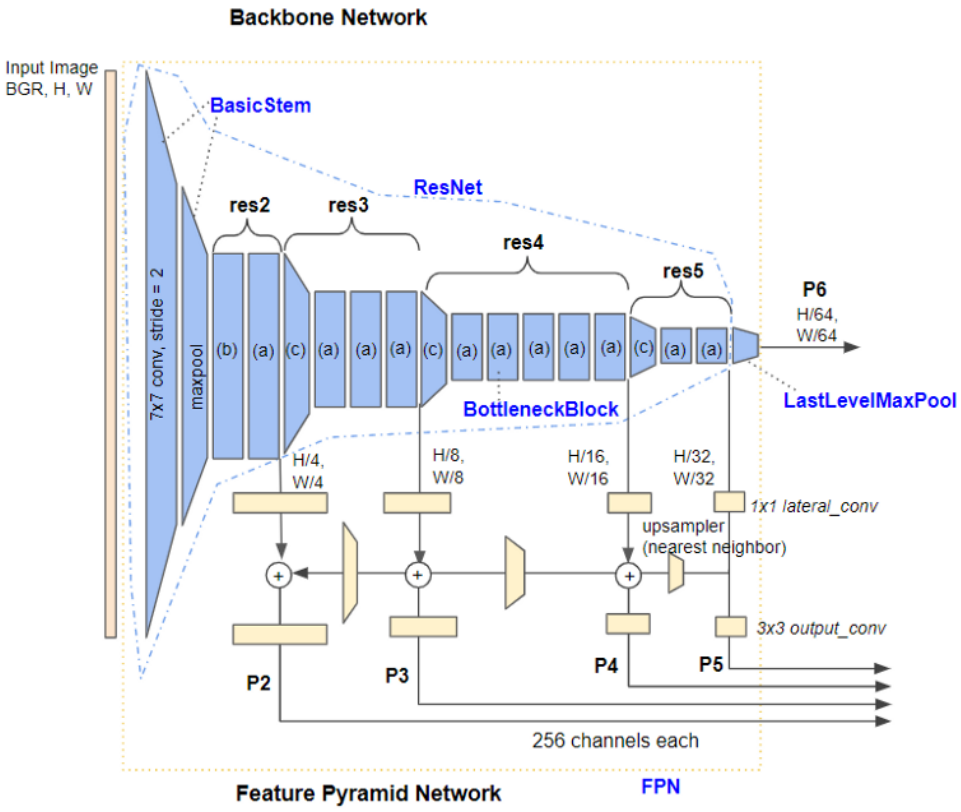


FIGURE 4 – Feature Pyramid Network (FPN)

Le rôle de FPN est d'extraire les feature maps depuis l'image en input.

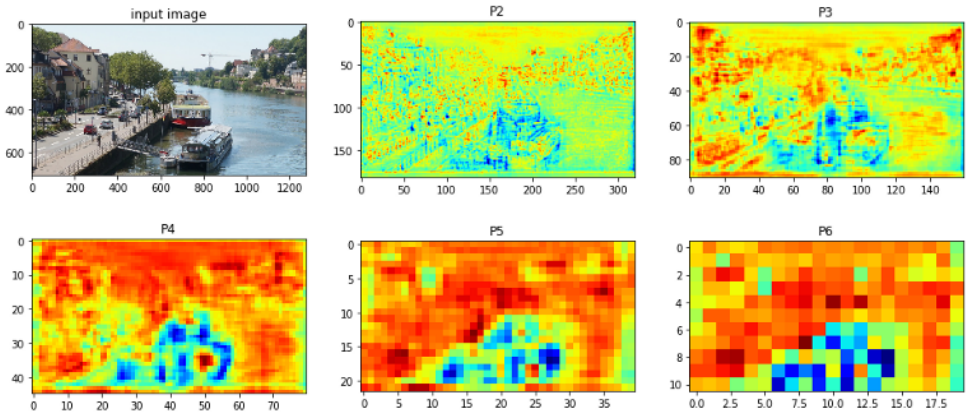


FIGURE 5 – Exemple d'input et output de FPN

c. Fonction de perte

La fonction de perte est la somme de la fonction de perte relative à la classification, celle relative à la localisation et celle relative à la régularisation :

— Classification loss : **Fonction de perte focal**
Durant l'apprentissage d'un modèle de détection d'objet, la fonction de perte Balanced Cross Entropy

classique ($BCE(p) = -\alpha \log(p)$ où α est un hyperparamètre et p la probabilité prédite de notre classe, ici nous n'avons qu'une seule classe) permet d'équilibrer l'importance des exemples positifs/négatifs mais peut être submergée par le déséquilibre du dataset puisqu'elle ne différencie pas les exemples faciles à prédire des exemples difficiles à prédire. Ainsi, pour remédier à cela, le modèle utilise une fonction de perte focale. En d'autres termes, la fonction de perte focale est une version améliorée de la fonction de perte d'entropie croisé (Cross Entropy), qui tente de résoudre le problème de déséquilibre de classe en attribuant plus de poids aux exemples facilement mal classés (ex : arrière-plan avec du bruit, seule une partie de l'objet est visible...) et d'alléger les poids lors de la prédiction des exemples faciles. La fonction de perte focale est définie comme suit :

$$FL(p) = (1p)^{\gamma} \log(p)$$

— Localization loss : **Smooth L1 Loss (Fonction de perte d'Huber)**
La fonction de perte relative à la localisation est une perte de type smooth L1 calculée entre la bounding box prédite y et les vrais valeurs x. La fonction de perte d'Huber qu'on emploie ici est définie comme suit :

$$LL(x, y) = \begin{cases} 0.5(x-y)^2, & \text{si } \mid x-y \mid < 1 \\ \mid x-y \mid -0.5 & \text{sinon} \end{cases} \quad (1)$$

La fonction de perte d'Huber n'utilise un terme au carré que si l'erreur absolue est inférieure à 1. Ainsi, elle est moins sensible aux outliers que la MSE (mean square error) et peut prévenir une explosion de gradient. En effet, dans la fonction MSE, nous mettons au carré la différence $\mid x-y \mid$ ce qui donne un nombre beaucoup plus grand que le nombre d'origine. Ces valeurs élevées se traduisent par des gradients explosifs. Ceci est évité ici car pour les nombres supérieurs à 1, les nombres ne sont pas au carré.

— Regularization loss : **Régularisation L2**

Le surapprentissage est un phénomène qui se produit lorsqu'un modèle de Machine Learning est limité au dataset d'apprentissage et n'est pas en mesure de bien fonctionner sur de nouvelles données, c'est à dire de généraliser. La régularisation est une technique utilisée pour réduire les erreurs en ajustant la fonction de perte de manière à pénaliser les poids trop larges et ainsi éviter le surapprentissage. Ainsi, en utilisant la régularisation, nous sommes essentiellement en train de diminuer la capacité de notre modèle à s'accorder à l'ensemble d'apprentissage pour lui faire gagner des capacité à prédire correctement de nouvelles images.

La fonction de perte relative à la régularisation L2 est définie comme suit :

$$RL = \frac{\lambda}{2m} \sum_{i=1}^{p} \mid\mid w_i \mid\mid_2^2$$

où λ est un hyperparamètre inférieur à 1, p est le nombre de couches, m le nombre d'inputs et w_i est la matrice de poids de la couche i).

d. Optimiseur

Pour minimiser la fonction de perte, nous utilisons l'algorithme de descente de gradient avec m Momentum.

La descente de gradient est un algorithme d'optimisation qui poursuit le gradient négatif d'une fonction objectif afin de localiser le minimum de la fonction.

Le problème avec la descente de gradient est que la valeur peut rebondir autour de l'espace de recherche sur des problèmes d'optimisation qui présentent une fonction très courbée, et ainsi l'algorithme peut rester bloqué dans des points (régions plates) de l'espace de recherche qui n'ont pas de gradient.

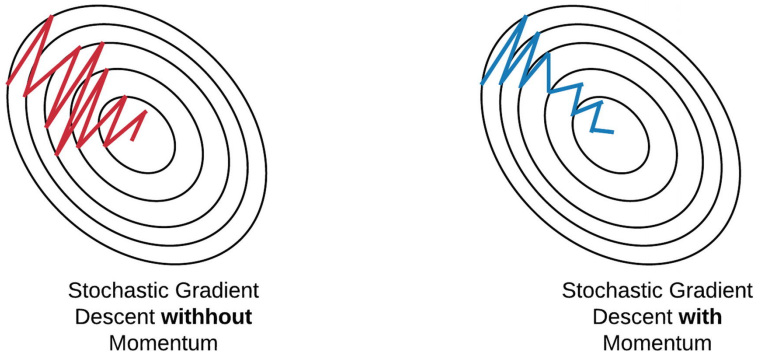


FIGURE 6 – Exemple : descente de gradient stochastique avec et sans momentum

La descente de gradient avec momentum est une variation de la descente de gradient classique qui permet à l'algorithme de créer de l'inertie dans une direction dans l'espace de recherche et de surmonter les oscillations de gradients et ainsi de traverser des régions plates de l'espace de recherche.

2.2.2 Apprentissage

a. Taux d'apprentissage

Puisque nous faisons du Transfer Learning et qu'il s'agit simplement de Fine-Tuning et non un apprentissage complet, nous avons opté pour un taux d'apprentissage qui va s'ajuster au fur et à mesure que nous avançons dans les training steps. Pour cela, nous avons combiné deux techniques : LLRD (Layer-wise Learning Rate Decay) et Linear Warmup Scheduler.

L'approche LLRD consiste à appliquer un taux d'apprentissage plus important pour les couches qui se trouve vers la sortie et un taux d'apprentissage plus bas pour les couches qui sont vers l'entrée. Ceci se fait en multipliant le taux d'apprentissage de base par un parametre appelé decay rate. Dans notre cas nous avons choisi le Cosine Learning Rate Decay :

$$\eta = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + \cos\frac{T^{cur}}{T_i}\pi) \tag{2}$$

Le but de cette approche est de modifier les couches les plus basses (qui apprennent des informations générales) moins que les couches les plus hautes qui sont plus spécifiques au problème.

La technique du Linear Warmup Schedular consiste à utiliser un taux d'apprentissage très bas au début pour un nombre de training steps appelés warmup steps puis de l'augmenter petit à petit.

En combinant ces deux techniques, nous obtenant le graphe d'évolution du taux d'apprentissage suivant :

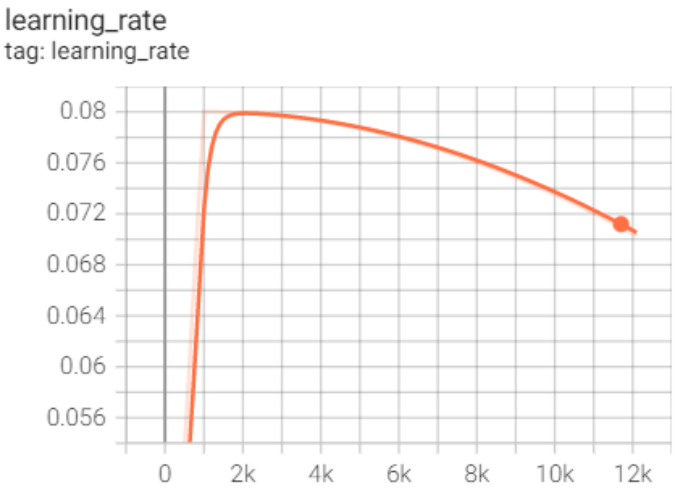


FIGURE 7 – Evolution du taux d'apprentissage

b. Training steps

Nous avons choisi d'entraîner notre modèle sur 322 images, 150 epochs et un batch size de 4 (nous ne pouvions pas augmenter cette valeur par soucis de GPU), ce qui nous donne :

$$num\_training\_steps = \frac{322}{4} \times 150 = 12075 \tag{3}$$

c. Précision

Le modèle affiche une précision de 78% sur l'ensemble de validation.

FIGURE 8 – Exemple de prédiction

d. Enregistrement du modèle

A la fin de l'apprentissage, nous avons gelé les paramètres, hyperparamètres et l'architecture de notre modèle (freezing graph) et nous l'avons enregistré sous format TFLite. Ceci permet de vous éviter de refaire la phase d'apprentissage pour pouvoir tester le modèle, ça va aussi nous permettre de l'importer sur notre interface graphique Streamlit (partie 2.4).

2.3 Lecture de la plaque d'immatriculation et affichage du texte

La reconnaissance de caractères est l'un des cas d'utilisation courants du Deep Learning. Par exemple, la conversion d'ordonnances manuscrites, la conversion de PDF ou d'images en texte, la vérification de signatures, etc. Dans notre cas nous allons donc l'utiliser dans l'identification de plaques d'immatriculation de véhicules.

Sur le marché, nous disposons de très bons services API payants pour l'OCR (Optical Character Recognition), comme Amazon Textract, Microsoft cognitive service, Google Cloud Vision, etc. La bonne nouvelle, c'est qu'il existe de bonnes APIs open-source qui sont disponibles gratuitement comme Pytesseract, easyocr, Keras-OCR. Elles ont également donné de bons résultats, similaires à ceux d'autres services d'API payants. Pour ce projet, nous allons tout d'abord essayer de créer notre propre modèle de reconnaissance de texte puis nous allons nous pencher sur l'une d'entre elles : EasyOCR.

2.3.1 Avec Keras

a. Segmentation de l'image de la plaque en images de caractères

Pour pouvoir séparer les caractères présents sur la plaque d'immatriculation, nous avons eu recours à un traitement d'image classique avec OpenCV. Nous avons tout d'abord agrandit l'image de la plaque détectée et extrait ces niveaux de gris, puis nous avons utilisé un flou gaussien avant de binariser l'image avec la méthode d'Otsu et de la dilater. Ces étapes permettent au final de pouvoir détecter les contours de l'image de façon plus efficace :

FIGURE 9 – Traitement de l'image de la plaque

Une fois les contours détectés, nous récupérons les coordonnées de chaque contour. Les contours qui nous intéressent (ceux des caractères) sont ceux dont la surface est inclus entre 2% et 9% de la surface totale de la plaque (ces valeurs ont été trouvés en testant plusieurs valeurs sur plusieurs plaques).

FIGURE 10 – Segmentation des caractères de la plaque

b. Reconnaissance des caractères

Pour reconnaitre les caractères que nous avons récupéré sur la plaque et les transformer en texte, nous avons pensé à créer un CNN avec Keras que nous avons entraîné sur un dataset contenant 1050 images de caractères (30 images par caractère, 35 caractères : 10 chiffres et 25 lettres de l'alphabet sans la lettre O). Nous avons utilisé 94% des ces images pour l'entraînement du modèle et le reste pour le tester.

L'architecture du CNN est la suivante :

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_57 (Conv2D)          (None, 58, 58, 32)        896

 max_pooling2d_38 (MaxPoolin  (None, 29, 29, 32)       0
 g2D)

 conv2d_58 (Conv2D)          (None, 27, 27, 64)        18496

 max_pooling2d_39 (MaxPoolin  (None, 13, 13, 64)       0
 g2D)

 conv2d_59 (Conv2D)          (None, 11, 11, 64)        36928

 flatten_19 (Flatten)        (None, 7744)              0

 dense_39 (Dense)            (None, 64)                495680

 dense_40 (Dense)            (None, 35)                2275

=================================================================
Total params: 554,275
Trainable params: 554,275
Non-trainable params: 0
```

FIGURE 11 – Modèle de reconnaissance de caractères

Pour la phase d'apprentissage, nous avons entraîné le modèle sur 8 epochs en utilisant un optimiseur Adam et une fonction de perte Cross Entropy.

Le modèle avait de bons résultats sur l'ensemble de test mais donnait des résultats très erronés sur les caractères récupérés à partir des plaques d'immatriculation. Ainsi, nous avons pensé à utiliser une autre alternative.

2.3.2 Avec EasyOCR

EasyOCR est un package OCR pour python. En l'utilisant, nous pouvons effectuer l'extraction de texte très facilement.

FIGURE 12 – Image Originale

D'abord nous devons rogner l'image pour en extraire ce qui nous interesse, c'est à dire la plaque. Nous allons donc faire appel à notre modèle de détection de plaques pour obtenir l'image contenant seulement la plaque et nous allons la redimensionner.

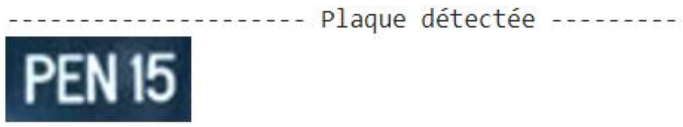


FIGURE 13 – Image de la plaque

Ensuite nous allons, à partir de celle-ci utiliser la méthode readtext() de easyOCR pour en extraire le texte.

```
---------- Format textuel ----------
['PEN 15']
```

FIGURE 14 – Conversion en format textuel

Pour optimiser le rendu textuel, nous avons utilisé le package python re pour pouvoir supprimer les caractères spéciaux de l'output de la fonction readtext(). Aussi, pour éviter de récupérer les textes en petits caractères qui peuvent être présents sur les plaques (le nom de la ville ou de l'état par exemple), nous avons défini un seuil à partir du quel le texte peut être détecté : par exemple, si le seuil est égal à 20% nous ne détecterons pas les lignes de texte dont la surface fait moint de 20% de l'image de la plaque.

2.4 Interface graphique

Pour résumer notre travail et pouvoir tester nos modèles, nous avons créé une interface graphique avec Streamlit sur laquelle on peut importer une image contenant une plaque d'immatriculation et obtenir le format textuel de cette plaque. Dans cette partie, nous allons vous montrer comment éxécuter et utiliser l'application, nous vous invitons à consulter notre notebook Colab pour consulter le code de création de l'application.

Execution de l'application

Pour éxécuter l'application (la partie Interface Graphique), il faut tout d'abord éxécuter les parties 'Setup' et 'Lecture et préparation des images' de notre Colab.

Ensuite, dans la partie 'Interface Graphique → Back-End', les variables **ipynb_path_folder_ssd** et **ipynb_path_labelmap** doivent être modifié de façon à ce qu'elles contiennent respectivement le chemin vers le dossier **export-license_plate_detection_model-ssd** qui contient notre modèle et le fichier **label_map.pbtxt** qui contient notre Label Map (tous les deux présents dans le drive partagé du projet).

```
[ ] ipynb_path_folder_ssd = os.path.join("/content",paths['LOCAL_EXPORT_FILE']) # Changez cette variable selon où se trouve le folder export-license_plate_detection_model-ssd
    !cp -rf {ipynb_path_folder_ssd} /content/

    ipynb_path_labelmap = os.path.join("/content",fichiers['LABELMAP']) # Changez cette variable selon où se trouve le fichier label_map.pbtxt
    !cp -rf {ipynb_path_labelmap} /content/
```

Il suffit ensuite de suivre les étapes du notebook pour créer les fichier backend.py et app.py. Au final, avant d'éxécuter l'application, votre répertoire /content/ doit ressembler à ça :

```
[ ] !ls

    app.py      export-license_plate_detection_model-ssd  sample_data
    backend.py  label_map.pbtxt                           Tensorflow
    drive       __pycache__
```

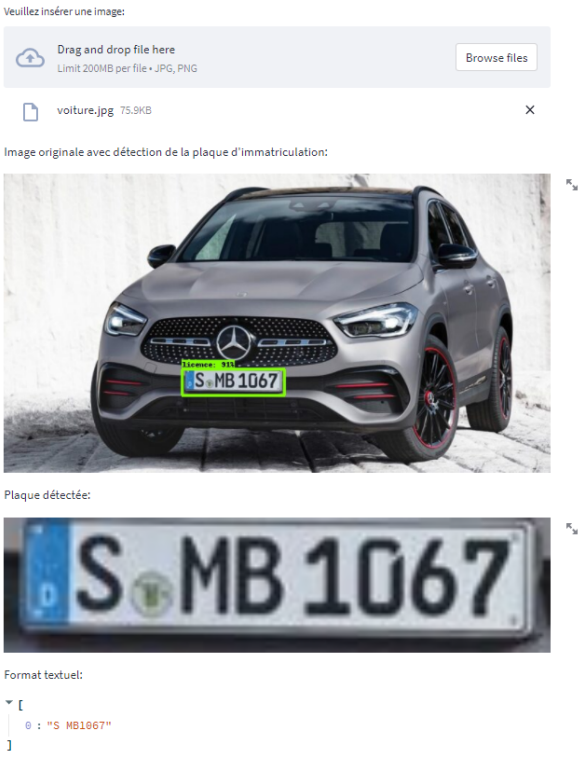
Il suffit ensuite d'éxecuter les cellules qui suivent pour récupérer le lien de l'application :

Ensuite, il faut s'authentifier sur ngrok, puis executer l'application. Finalement, on récupère l'`url` de la page où se trouve notre application.

N.B : Concernant la commande `ngrok authtoken XXXXX`, vous pouvez générer votre propre token d'authentification en créant un compte sur Ngrok. En vous connectant, vous trouverez ensuite votre token sur la page d'accueil dans la partie *"2. Connect your account"*. Ou vous pouvez tout simplement utiliser mon token d'authentification.

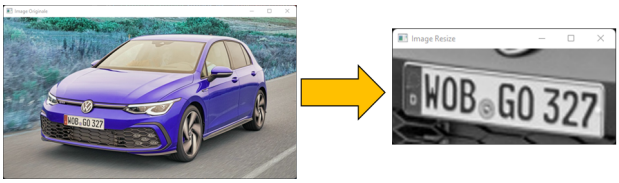
```
[135] !ngrok authtoken 21grpFLzc86FnEyFUlD0L3lw7nn_4o5z4jDVdcUg7DN1b36L2
      !streamlit run app.py&>/dev/null&

      Authtoken saved to configuration file: /root/.ngrok2/ngrok.yml

      from pyngrok import ngrok
      url = ngrok.connect(port='8501')
      url

      'http://d1f8-35-192-187-192.ngrok.io'
```

Une fois le lien affiché sur votre navigateur, vous pouvez cliquer sur 'browse files' pour importer une image sur l'application et ainsi voir la prédiction qu'en fait notre modèle.



2.5 Autres pistes explorées

Nous avons aussi exploré une autre piste, celle du modèle YOLO. En effet ce modèle de détection d'objet est un modèle pré-entrainé qui reconnaît énormément de classes d'objets. Nous pouvons donc imaginer utiliser celui-ci afin de résoudre notre problématique. Nous avons donc utilisé une version fine-tuné du modèle YOLO. Pour cela nous allons enlevé toutes les classes de sortie du modèle YOLO à l'exception des plaques d'immatriculation.



De cette manière nous arrivons facilement à extraire la plaque d'immatriculation de l'image, maintenant il faut lire celle-ci. Pour cela il existe un autre modèle pré-entrainé. On parle cette fois-ci plutôt d'une librairie qui permet de transformer une image en texte. Cette librairie est PyTesseract. De cette manière on a pu obtenir le résultat attendu en très peu de temps. Cependant cette exploration faisait très peu participer l'IA c'est pour cela que nous l'avons simplement explicité dans le rapport.

3 Conclusion

Dans ce projet, nous avons construit un système de détection et de reconnaissance automatique des plaques d'immatriculation en utilisant Tensorflow Object Detection (TFOD), OpenCV et EasyOCR.

Grâce à ce projet, nous avons appris à changer la configuration d'un modèle déjà existant, à faire du Fine-Tuning, à créer un système de reconnaissance de caractères, et à utiliser quelques techniques de base de traitement d'images.

Bien que notre modèle donne des résultats satisfaisants, une des pistes d'amélioration que nous pouvons citer réside dans la transformation d'images de caractères en format textuel. En effet, bien que la détection de la plaque d'immatriculation donne pour la plupart du temps des résultats corrects, la reconnaissance du texte n'est pas tout à fait optimal (le modèle peut confondre par exemple les caractères 6 et G). Cela pourrait être corrigé par exemple en entraînant le CNN de reconnaissances de caractères que nous avons créé sur des caractères issues de plaques d'immatriculation, mais aussi en utilisant des regex pour identifier le format textuel que devrait avoir une plaque d'immatriculation.

4 Cybergraphie

1. SSD: Single Shot MultiBox Detector

2. Feature Pyramid Networks for Object Detection

3. MobileNetV2: Inverted Residuals and Linear Bottlenecks

4. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision

5. Focal Loss for Dense Object Detection
6. STOCHASTIC GRADIENT DESCENT WITH WARM RESTARTS