Table des matières

[1 DETECTION DE LA PLAQUE D’IMMATRICULATION 2](#_Toc57477160)

[1.1 Ré-entrainer notre propre modèle 2](#_Toc57477161)

[1.1.1 Zoom sur le code 3](#_Toc57477162)

[1.1.2 Résultats 5](#_Toc57477163)

[1.2 Ré-entrainer notre propre modèle – V2 5](#_Toc57477164)

[1.2.1 ZOOM SUR LE CODE 6](#_Toc57477165)

[1.3 Utiliser un réseau de neurones pré-entrainé 9](#_Toc57477166)

[1.3.1 Zoom sur le code 10](#_Toc57477167)

[1.3.2 Résultats 11](#_Toc57477168)

# DETECTION DE LA PLAQUE D’IMMATRICULATION

La première étape dans la réalisation de cette application a été d’être capable d’identifier sur une image/photo de véhicule, la plaque d’immatriculation.

Objectif de cette partie :

* Entrée : une photo d’un véhicule pris par l’utilisateur de l’application
* Sortie : la photo de la plaque d’immatriculation du véhicule photographié

## Ré-entrainer notre propre modèle

Dans un premier temps, nous avons commencé par travailler avec supervisely et tensorflow, selon les indications du site internet fourni par Damien et Malik.

L’idée principale était d’entrainer notre propre réseau de neurones (dont l’architecture était fournie) de manière supervisée grâce au dataset founi par Supervisely.

Grâce à Supervisely, nous avons pu récupérer un dataset contenant 50 images comme celle-ci :

|  |  |
| --- | --- |
| *Image exemple 1* | *Image exemple 2* |
| *Une image contenant bâtiment, extérieur, photo, carrelé  Description générée automatiquement* | *Une image contenant extérieur, bâtiment, photo, bus  Description générée automatiquement* |

Il s’agit finalement de simple montage photo où ils ont collé des images de plaque d’immatriculation sur une image de paysage (buildings, routes, montagnes…).

Chaque image est associée à son « y » :

|  |
| --- |
| *Mêmes images avec un rectangle rouge pour identifier la plaque d’immatriculation* |
|  |

### Zoom sur le code

Ces images étaient donc séparées en train et test sets (44 images / 6 images), puis le modèle était entraîné.

|  |
| --- |
| *Création du modèle* |
| Une image contenant texte  Description générée automatiquement |
| Explications : le modèle est un CNN relativement classique avec des couches de Convolution alternées avec des Max\_Pooling, puis deux couches fully-connected (Dense) et une couche de prédiction (output layer). Le modèle mis à notre disposition était relativement « outdated ». |

Nous avons réalisé 100 epochs, mais le modèle ne s’améliore plus après une cinquantaine d’epochs et les résultats sont loin d’être aussi bons que ceux présentés sur le site internet.

Une image contenant texte, table

Description générée automatiquement

### Résultats

Les résultats obtenus sont loin d’être satisfaisants :

|  |
| --- |
| *Résultats obtenus sur le test set* |
|  |

Les résultats peu probants nous ont conduits à revoir notre modèle et notre approche.

## Ré-entrainer notre propre modèle – V2

Pour cette deuxième approche, nous avons commencé par aller chercher plus de données :

* Damien nous a fourni le dataset « Supervisely » entier (environ 10000 images)
* Malick nous a indiqué un dataset de voiture disponible sur Kaggle

|  |  |
| --- | --- |
| *Image supervisely* | *Exemple image kaggle 1* |
| *Une image contenant bâtiment, extérieur, bus, photo  Description générée automatiquement* |  |
| *Exemple image Kaggle 2* | *Exemple image kaggle 2* |
| *Une image contenant voiture, extérieur, route, herbe  Description générée automatiquement* |  |

### ZOOM SUR LE CODE

Nous retrouvons l’étape de chargement des données, mais cette fois-çi, en fonction du type de fichier (.json ou .xml) :

Une image contenant texte

Description générée automatiquement

Cependant, les données n’étaient pas utilisables telles quel. Avant d’utiliser nos données pour entraîner notre modèle, il a été essentiel d’opérer un nettoyage des données afin de fournir au modèle des entrées uniformisées :

1. Uniformiser la taille de l’image

|  |
| --- |
| *Uniformiser les données* |
| Une image contenant texte  Description générée automatiquement |
| Explications : le modèle prend en entrée des images de taille (128, 64) en nuances de gris. |

1. Charger les coordonnées de la plaque selon le format du fichier annotation (.json pour Supervisely et .xml pour Kaggle)

|  |
| --- |
| *Chargement des coordonnées dans le cas d’un .json* |
| *Une image contenant texte  Description générée automatiquement* |
| Exemple d’un contenu d’un fichier .json :  {"tags": [], "description": "", "objects": [{"description": "", "bitmap": null, "tags": ["E112YB13"], "classTitle": "Licence plate", "points": {"exterior": [[26.0, 43.0], [101.0, 43.0], [101.0, 61.0], [26.0, 61.0]], "interior": []}}], "size": {"height": 64, "width": 128}} |

|  |
| --- |
| *Chargement des coordonnées dans le cas d’un .xml* |
| Une image contenant texte  Description générée automatiquement |
| Exemple du contenu d’un fichier .xml :  <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> |

1. Modifier les coordonnées quand l’image a été « resized » pour avoir le rectangle positionné au bon endroit sur l’image qui a été elle-même « resized »

|  |
| --- |
| *Repositionner les coordonnées en fonction du « resized » opéré* |
| Une image contenant texte  Description générée automatiquement |
| Explications : *original* correspond à la taille originale de l’image (hauteur ou largeur selon si on étudie x ou y), *final* correspond à la taille de l’image finale (c’est-à-dire la taille « objectif ») et *to\_resize* correspond au point à modifier |

Finalement, nous avons « concatenate » les données pour avoir X\_train, X\_test, Y\_train et Y\_test.

Après traitement, voilà à quoi ressemble les images en entrée du modèle :

|  |  |
| --- | --- |
| *Exemples d’image en « input » du modèle* | |
|  |  |

AJOUTER LA PARTIE SUR LE MODÈLE + LES RÉSULTATS

Sachant que cette étape était essentielle pour la viabilité du projet de lecture de plaque d’immatriculation, nous avons choisi d’utiliser un modèle pré-trained, que nous avons adapté à notre problème, afin de mettre en valeur le travail de « lecture de plaque » dépendant de la performance de la « détection de plaque ».

## Utiliser un réseau de neurones pré-entrainé

Au cours de nos recherches, nous avons trouvé un modèle pré-entrainé, créé par Sergiomsilva[[1]](#footnote-1). Directement, ce modèle est apparu bien plus efficace et complet car :

* Le modèle peut détecter des plaques sur tout type de véhicule (voitures, motos…)
* Le modèle peut lire des plaques du monde entier
* Les résultats sont bien meilleurs – en termes de précision dans la détection et de régularité – que le modèle que nous avons entrainé nous-même (cf. 1.1)

Nous avons donc opté pour cette solution pour implémenter notre application.

### Zoom sur le code

|  |
| --- |
| *Fonction principale (main())* |
|  |
| Explications : Cette fonction principale prend en entrée une image (qui contient supposément un véhicule) et retourne une autre image contenant cette fois-ci uniquement la plaque d’immatriculation. |

|  |
| --- |
| *Fonction color2gray : transforme une image en image nuances de gris* |
| Une image contenant texte  Description générée automatiquement |
| Explications : Cette fonction prend en entrée l’image originale et retourne la même image en nuances de gris (divisé par 255). Cette étape est importante car le modèle utilisé a été entrainé sur des images en nuances de gris.  Nous avons essayé de faire des prédictions sans cette étape, le taux de réussite était très proche de 0%. C’est donc bien une étape essentielle. |

|  |
| --- |
| *Fonction get\_plate : trouve la plaque d’immatriculation sur l’image* |
| *Une image contenant texte  Description générée automatiquement* |
| Explications : Cette fonction prend en entrée l’image du véhicule puis retourne l’image de la plaque d’immatriculation. Pour être parfaitement précis, cette fonction ne trouve pas directement la plaque d’immatriculation, elle fait appel à la fonction detect\_lp() pour cela. |

|  |
| --- |
| *Fonction detect\_lp()* |
| *Une image contenant texte  Description générée automatiquement* |
| Explications : Cette fonction a directement été implémentée par Sergiomsilva pour nous permettre d’utiliser son modèle. Pour cette raison, nous ne l’avons pas modifié et utilisé tel quel.  Nous pouvons voir que c’est bien ici que l’on fait appel au modèle avec model.predict(T). |

### Résultats

Les résultats obtenus sont très satisfaisants.

Voici quelques résultats obtenus avec des photos trouvés sur internet :

|  |  |
| --- | --- |
| *Exemple 1* | |
| *Image originale* | *Image retournée par le programme* |
| *Une image contenant extérieur, route, voiture, rue  Description générée automatiquement* | *Une image contenant arts de la table, assiette, horloge, fermer  Description générée automatiquement* |

|  |  |
| --- | --- |
| *Exemple 1* | |
| *Image originale* | *Image retournée par le programme* |
| Une image contenant extérieur, voiture, bâtiment, route  Description générée automatiquement | Une image contenant extérieur, signe, vert, rue  Description générée automatiquement |

1. <http://sergiomsilva.com/pubs/alpr-unconstrained/?fbclid=IwAR0NTRUedQU3RiXDxzYv8PCvX3OyBsrAdaJSeVaGBZVFpn3CS_4OTPLXdug> [↑](#footnote-ref-1)