



SCIENCES DU NUMÉRIQUE

SEMESTRE 8

Deep learning - Rapport final

Projet Classification d'Image : reconnaissance de panneaux de signalisation

Auteurs :

Aicha BENNAGHMOUCH
Yvan KIEGAIN DJOKO
Chloé HOSÉ WENDLING
Maëlis MARCHAND

1 Première partie : constitution de la base de données

Dans ce rapport, nous présentons notre démarche pour réaliser le projet Deep Learning de classification d'image.

1.1 Description du sujet choisi

Pour le sujet de notre projet, nous avons choisi une reconnaissance de panneaux de signalisation. Pour cela, nous avons choisi d'avoir 4 classes, correspondant aux 4 panneaux suivants :

- le panneau indiquant une voie réservée aux véhicules de transport en commun (1) ;
- le panneau zone piétonne, indiquant une voie réservée à la circulation des piétons (2) ;
- le panneau obligation de tourner à droite (3) ;
- le panneau obligation de tourner à droite à la prochaine intersection (4) ;

Voici les panneaux correspondants :

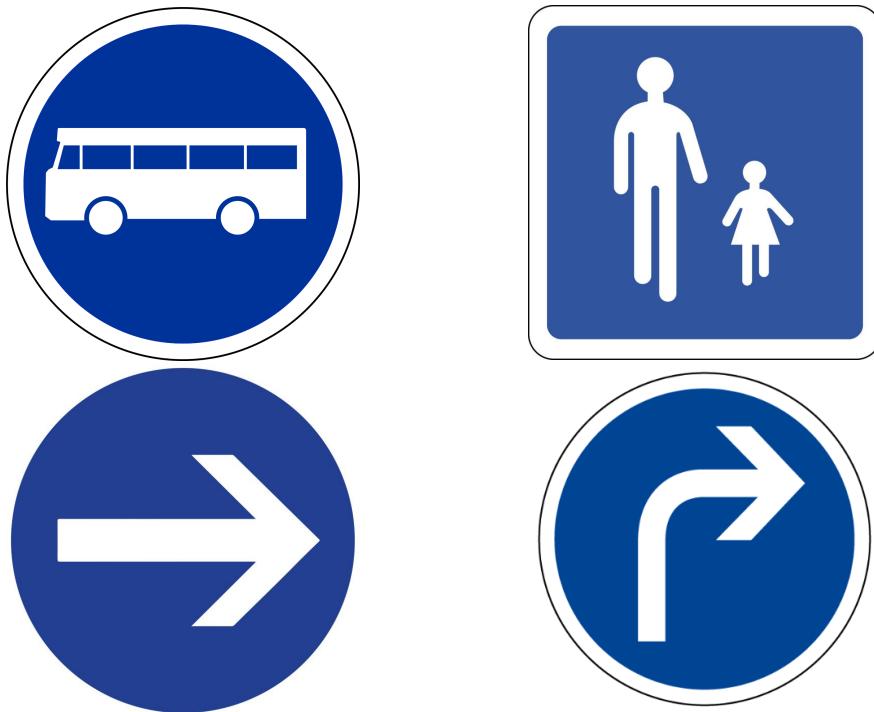


FIGURE 1 – Panneaux choisis pour nos classes (respectivement 1, 2, 3 et 4)

Il nous a semblé pertinent de choisir des panneaux qui se ressemblent, notamment dans les coloris et les formes. En effet, avec des panneaux singulièrement différents il aurait été très facile de les classer. Le but du projet étant entre autres d'entraîner un réseau de neurones, choisir des classes semblables nous permettra d'accentuer ce point.

1.2 Construction de la base de données

Voici le **lien Git** vers notre base de données et notre code (fichier **Projet.ipynb**).

1.2.1 Acquisition des données

Nous avons pris nous-mêmes la majorité des photos en nous promenant dans Toulouse. Pour chaque panneau, nous avons pris des photos sous différents angles et différentes luminosités afin de couvrir un

maximum de situations. Nous avons pris environ 350 photos par classe.

1.2.2 Annotation des données

Nous avons choisi de nommer nos classes ainsi :

- 1 : classe **bus** ;
- 2 : classe **pieton** ;
- 3 : classe **fleche** ;
- 4 : classe **virage** ;

Pour renommer nos classes, nous avons utilisé un script Python qui se trouve dans le fichier **renomme.py**. Chaque photo porte le nom de sa classe et est associée à un unique numéro.

Voici, par exemple, quelques photos que nous avons prises pour la classe **BUS** :



FIGURE 2 – Exemples de photos pour la classe **bus**

D'autre part, voici des exemples de photos que nous avons prises pour la classe **PIETON** :



FIGURE 3 – Exemples de photos pour la classe **piéton**

1.2.3 Partitionnement des images en ensembles d'apprentissage, de validation et de test

Pour partitionner nos images dans les différents ensembles d'apprentissage, de validation et de test, nous avons décidé de respecter les proportions suivantes :

- **80%** des photos dans l'ensemble d'**apprentissage** ;
- **10%** des photos dans l'ensemble de **validation** ;
- **10%** des photos dans l'ensemble de **test** ;

Pour répartir les photos dans les différents ensembles, nous avons procédé par classe : chacun s'est occupé de sélectionner des images de sa classe à mettre dans les ensembles en respectant les proportions ci-dessus et en couvrant un maximum de situations pour chaque ensemble (angle, luminosité, distance). Nous avons donc créé 3 dossiers **train**, **validation** et **test** dans chaque classe.

1.3 Notre pronostic

Nous pensons que notre algorithme pourra obtenir de bons résultats, étant donné que les panneaux de signalisation sont, par définition, faits pour être différenciés par l'oeil humain. Par ailleurs, ayant pris les photos nous-mêmes nous avons essayé d'obtenir une base de données fournie pour chaque panneau, tant par la diversité des angles que par celle de la luminosité, ce qui nous conforte dans l'idée que notre algorithme sera capable de différencier les classes. Cependant, nous avons choisi quatre panneaux bleus, dont deux qui se ressemblent beaucoup (flèches) afin de rajouter de la difficulté.

1.4 Script de chargement des données

Voici notre **script de chargement des données**.

Dans notre projet , nous avons choisi de classifier des images associées à 4 panneaux de signalisation différents : Bus, Fleche, Pieton et Virage.

Nous avons réparti les images dans 4 ensembles (et donc 4 dossiers, chaque dossier représentant une classe). Chacun de ces dossiers comporte 3 sous-dossiers : train (260-280 images par classe), validation (32-35 images par classe) et test (32-35 images par classe).

Nous avons choisi d'héberger notre base de données sur Github : l'intérêt est qu'un git clone depuis Google Colab est très rapide, ce qui garantit une certaine simplicité.

2 Deuxième partie : élaboration de notre solution et analyse de nos résultats

2.1 Construction du réseau de neurones

Nous avons commencé par construire ainsi notre réseau de neurones, en nous inspirant des TP :

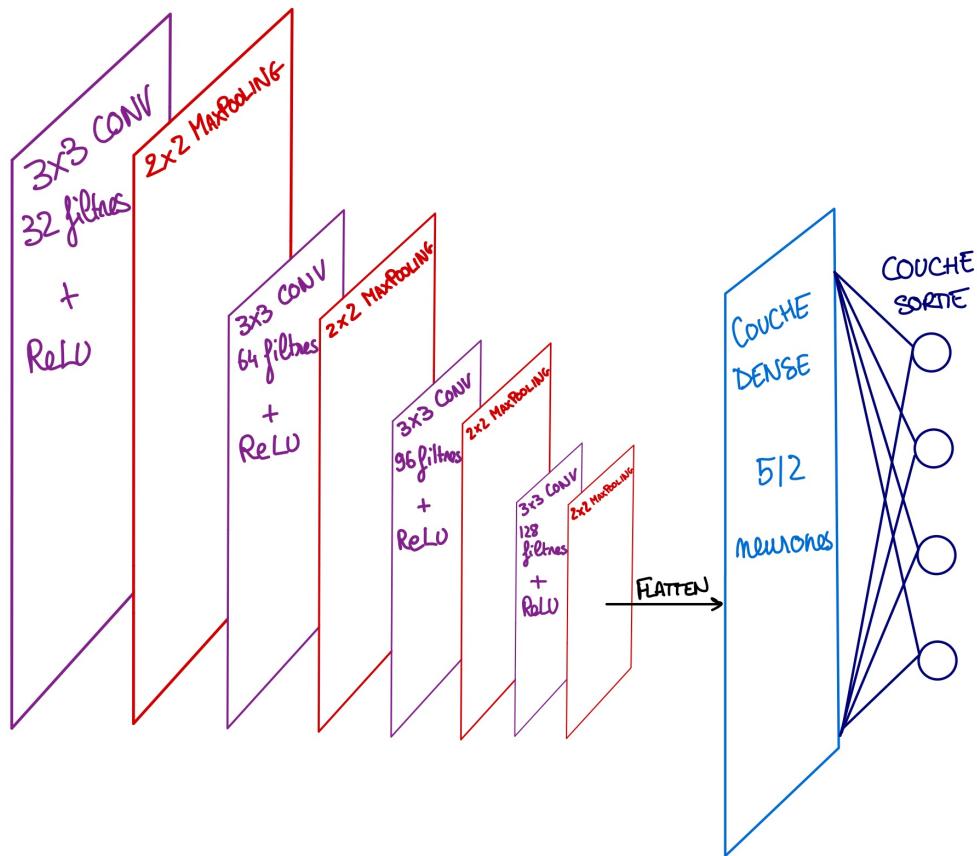


FIGURE 4 – Schéma de notre réseau de neurones

Ce que nous avons ensuite traduit en code de la façon suivante :

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), activation='relu'))

model.add(MaxPooling2D(2, 2))
# 2ème couche
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
# 3ème couche
model.add(Conv2D(96, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
# 4ème couche
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))

model.add(Flatten())    # "Mise à plat" (vectorisation) du tenseur pour permettre de la connecter à une couche dense
model.add(Dense(512, activation='relu'))  # Couche dense, à 512 neurones
model.add(Dense(4, activation='softmax'))  # Couche de sortie
```

FIGURE 5 – Modèle de notre réseau de neurones

On y voit notamment les **hyperparamètres** choisis : pour les couches de convolution, la fonction d'activation est **ReLU** et pour la couche de sortie, il s'agit de **softmax**.

2.2 Entrainement de notre réseau

Nous avons commencé par lancer un entraînement sur 10 epochs avec un taux d'apprentissage de 3e-4. Nous avons ensuite affiché :

- un graphe représentant la **Training and validation accuracy** en fonction des epochs ;
- un graphe représentant la **Training and validation loss** en fonction des epochs ;
- les **matrices de confusion** obtenues sur chacun des ensembles (entraînement, validation et test).

Voici un exemple de résultats obtenus suite à un entraînement standard de notre réseau :

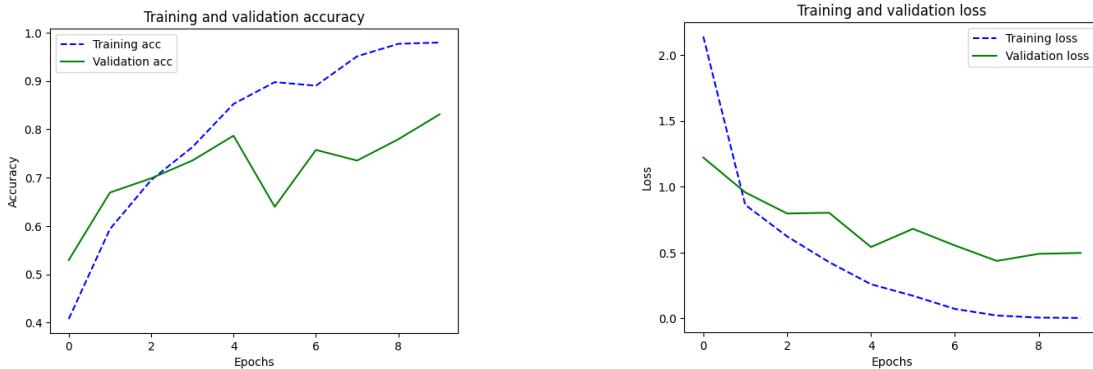


FIGURE 6 – Training and validation accuracy and loss

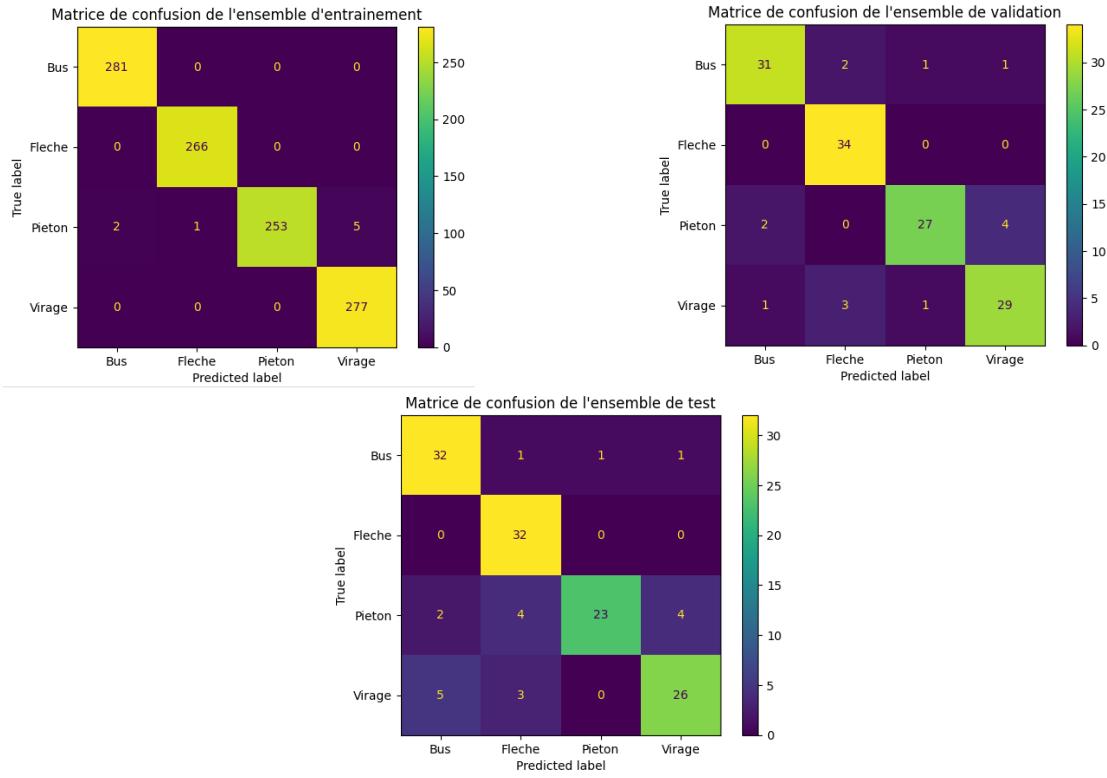


FIGURE 7 – Matrices de confusion obtenues sur les ensembles d’entraînement, de validation et de test

On observe que les résultats obtenus sont relativement bons sur l’ensemble d’apprentissage : on obtient une **training accuracy** qui tend vers 1 et une **training loss** qui tend vers 0.

En lançant plusieurs fois l’apprentissage, nous avons même pu observer une précision de 100% de notre modèle sur l’ensemble d’entraînement, comme le montre la matrice de confusion suivante :

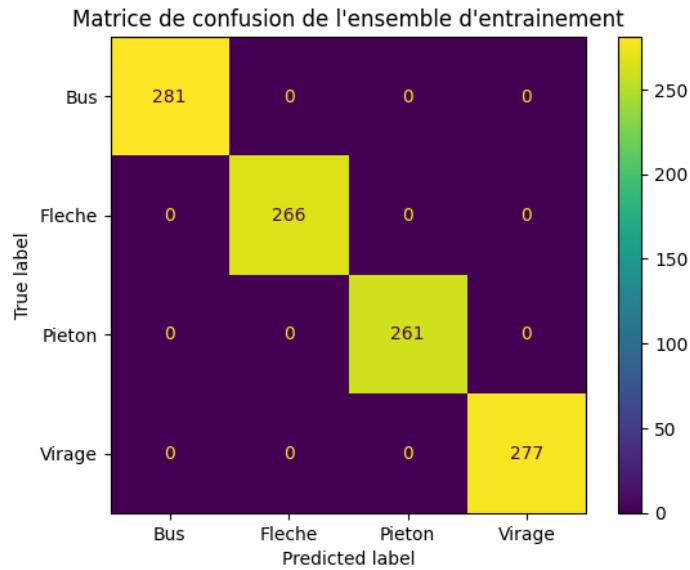


FIGURE 8 – Matrice de confusion parfaite obtenue sur l’ensemble d’entraînement

En revanche, on remarque que la **validation accuracy** est un peu moins bonne : on observe un **surapprentissage**. Cela signifie que le réseau a bien appris sur l'ensemble d'entraînement et s'est trop adapté à cet ensemble, ainsi il n'arrive pas à généraliser ce qu'il a appris sur l'ensemble de validation. Pour corriger cela, nous avons choisi de mettre en place une **augmentation de données**, afin d'améliorer nos résultats sur les ensembles de test et de validation.

2.3 Augmentation des données

2.3.1 Transformations appliquées

Afin d'élargir notre base de données, nous avons appliqué quatre transformations aux images de base, qui sont :

- une rotation de **40 degrés** ;
- un déplacement en largeur de **30%** de la largeur ;
- un déplacement en hauteur de **30%** de la largeur ;
- un zoom de **20%** ;

Avec cette augmentation de données, nous avons donc multiplié la taille de notre base de données par 5. Pour effectuer ces transformations, nous avons créé et utilisé un objet `ImageDataGenerator`.

2.3.2 Courbes

Nous avons ensuite établi, comme précédemment, deux courbes : la première montrant la précision de l'entraînement et de la validation (voir la figure **Training and validation accuracy**) et la seconde montrant la perte (figure **Training and validation loss**).

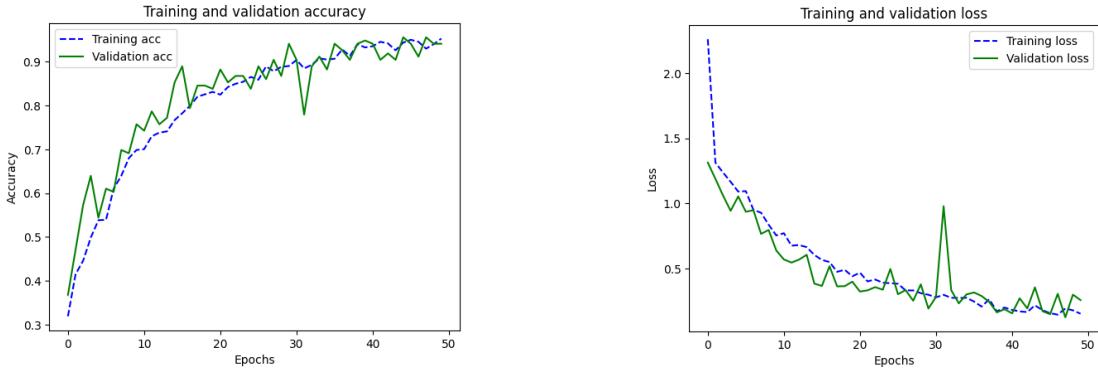


FIGURE 9 – Training and validation accuracy and loss

L'augmentation de données ralentissant l'apprentissage, avec 10 epochs notre modèle n'était pas assez entraîné, nous donnant ainsi de moins bons résultats qu'avant l'augmentation. Afin d'observer de meilleurs résultats avec cette augmentation de données, nous avons donc augmenté le nombre d'epochs (de 10 à 50) afin d'entraîner davantage notre modèle.

On remarque que les résultats obtenus sont meilleurs : la courbe d'efficacité pour l'ensemble de validation tend à se confondre avec celle de l'ensemble d'apprentissage, et il en va de même pour les courbes montrant l'évolution de la perte en fonction des epochs. On voit donc sur ces graphes qu'on a réussi à limiter le surapprentissage.

2.3.3 Matrices de confusion

L'augmentation de données et le fait de travailler sur plus d'epochs induit une certaine instabilité dans la précision, en effet on observe sur les courbes qu'il y a beaucoup plus de variations. Ainsi,

afin d'observer les meilleurs résultats possibles sur nos matrices de confusion, nous avons utilisé la technique du ModelCheckPoint : pendant l'entraînement on ne conserve les poids que si la validation loss a diminué, puis on charge les meilleurs poids obtenus. Cela nous permet donc d'augmenter la précision de nos résultats.

Voici les matrices de confusion que nous avons obtenues sur les différents ensembles, avec et sans le ModelCheckPoint :

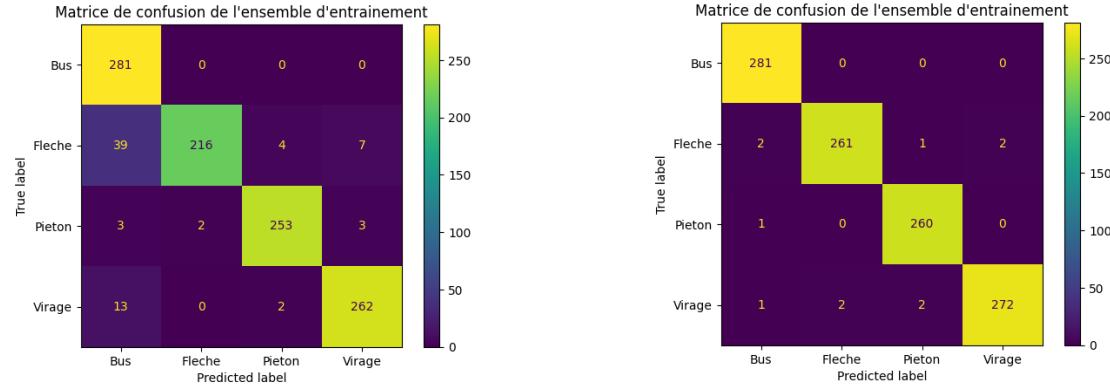


FIGURE 10 – Matrices de confusion obtenues sur l'**ensemble d'entraînement**, sans puis avec ModelCheckPoint

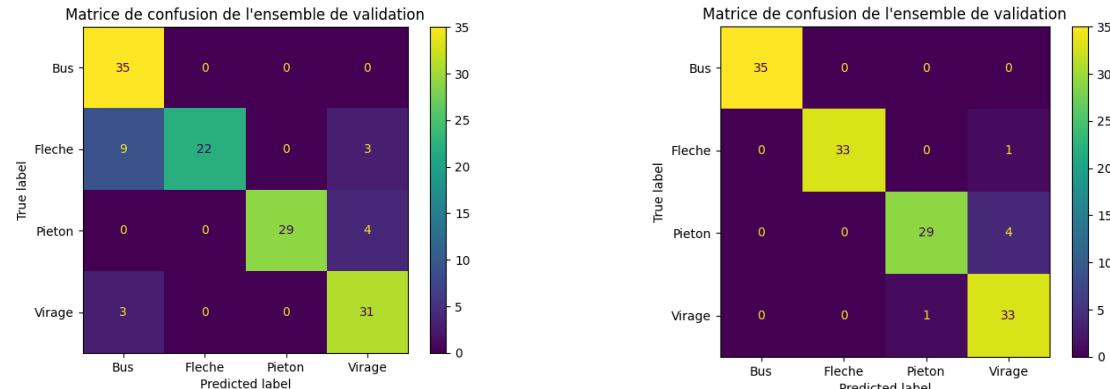


FIGURE 11 – Matrices de confusion obtenues sur l'**ensemble de validation**, sans puis avec ModelCheckPoint

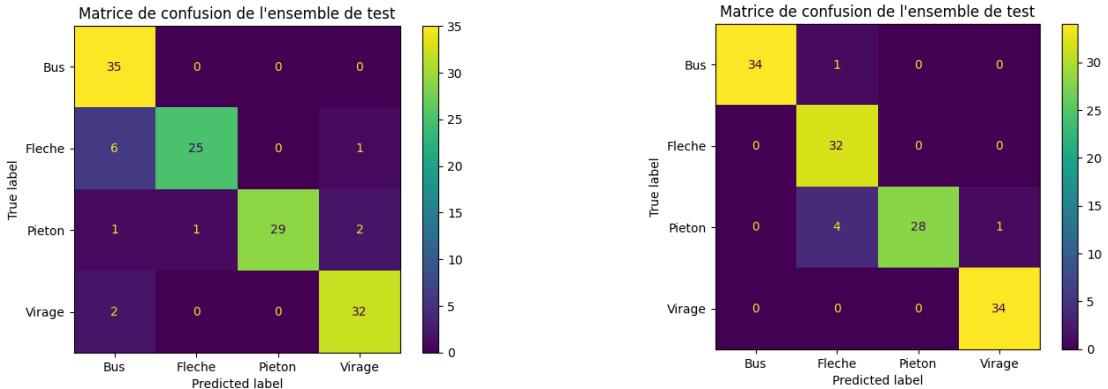


FIGURE 12 – Matrices de confusion obtenues sur l'**ensemble de test**, sans puis avec ModelCheckPoint

On observe bien qu'en conservant les meilleurs poids grâce au ModelCheckPoint, on améliore nos résultats sur tous les ensembles.

2.3.4 Analyse des images mal classées

Après avoir analysé les différentes matrices de confusion, nous avons voulu voir les images qui étaient mal prédites, afin d'analyser si l'erreur commise par notre réseau pourrait aussi l'être par l'oeil humain. Il en est ressorti plusieurs cas : dans celui des images de la figure 13 où un bus est confondu avec un piéton, au vu de la qualité et de la taille réduite des images (64x64), il est effectivement difficile de clairement différencier ces deux classes. Une piste d'amélioration dans ce cas là serait de moins réduire nos images et de passer à une taille 128x128.

Ensuite, il y a un cas où l'erreur est plus due à la similarité des classes comme virage et flèche (voir figure 14). Dès le début du projet, nous avions intuité que ce cas-là se présenterait et c'est d'ailleurs ce qui faisait la difficulté du projet. Cependant, en analysant les matrices de confusion, on remarque que notre réseau n'inverse pas si souvent ces deux classes, ce qui nous rassure quant à son implantation.

Enfin, notre réseau n'est pas parfait dans l'état actuel de notre projet et peut être sûrement encore être amélioré. Ainsi, on retrouve des cas où il est plus surprenant de se tromper sur la classe de l'image comme sur les éléments de la figure 15 car celles-ci sont assez facilement distinguables, l'image étant de meilleure qualité.

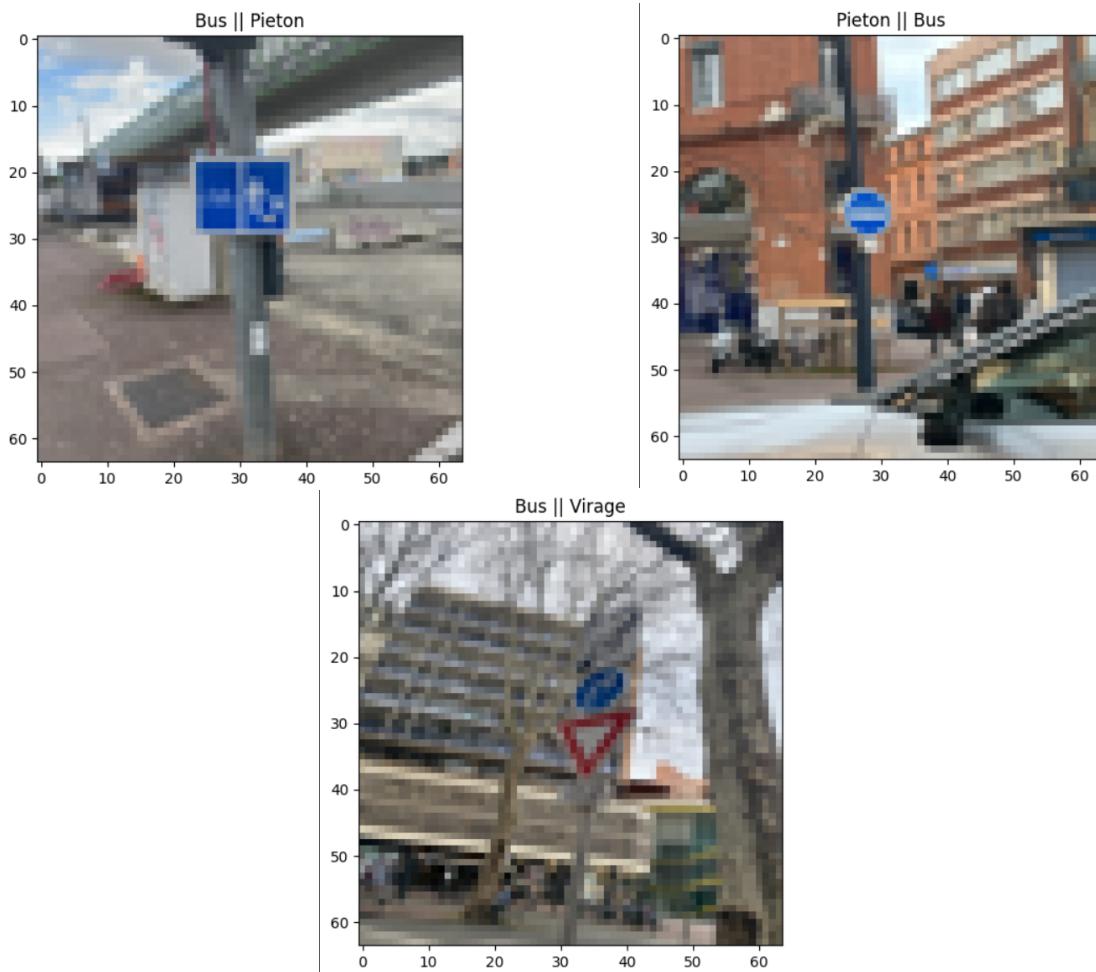


FIGURE 13 – Exemples d’images de mauvaise qualité difficilement classables (classe prédictive || classe réelle)

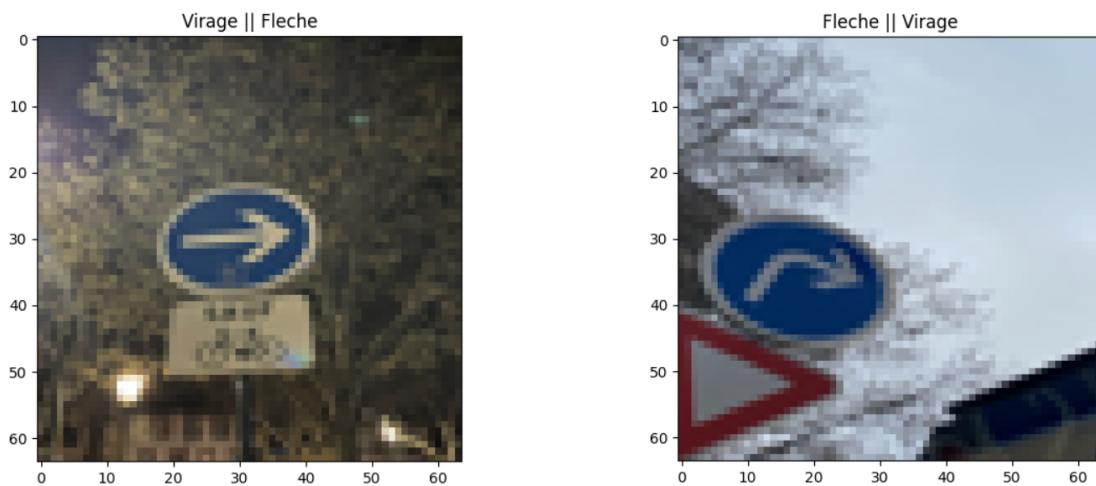


FIGURE 14 – Exemples d’images facilement interchangeables car similaires (classe prédictive || classe réelle)

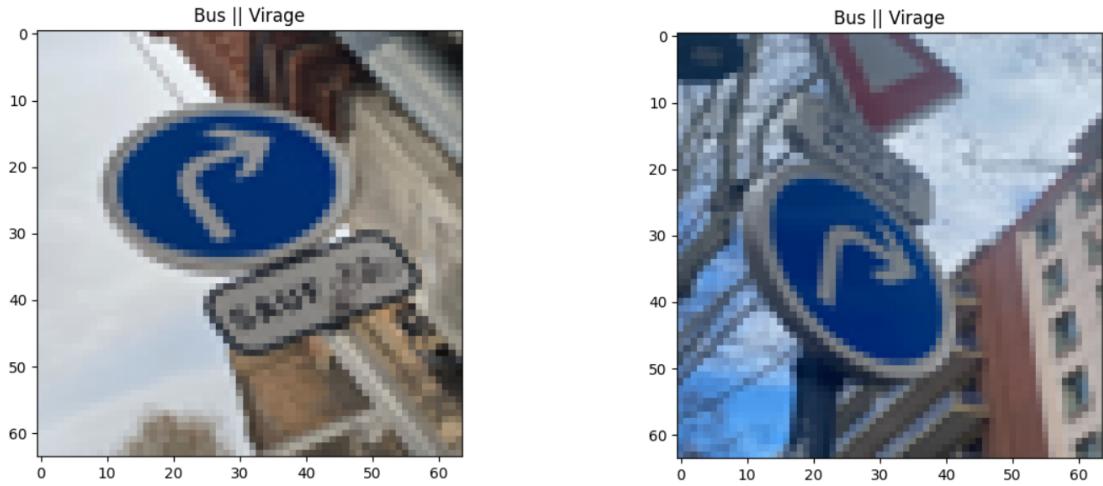


FIGURE 15 – Exemples d’images facilement reconnaissables mais non reconnues (classe prédictive || classe réelle)

3 Conclusion

En conclusion, au cours de ce projet de classification d’images de panneaux de signalisation, nous avons obtenu des résultats relativement bons. En effet, la reconnaissance des panneaux fonctionnait bien dès les premiers entraînements, et les résultats obtenus étaient ensuite très satisfaisants après l’augmentation de données.

Les résultats que nous avons obtenus sont relativement conformes à notre pronostic initial : nous pensions en effet que la reconnaissance allait bien fonctionner, puisque les panneaux sont reconnaissables par définition par les humains. De plus, ayant constitué nous-mêmes notre base de données, nous avons pris soin de faire de nombreuses photos différentes, en variant l’angle et la luminosité, afin que notre base de données soit la plus fournie possible.

Ce projet nous a permis de valoriser les compétences acquises lors des cours d’apprentissage profond, et de les appliquer sur un exemple concret.