



UNIVERSITÉ DU QUÉBEC
À CHICOUTIMI

Mini-Projet
Détection et classification de panneaux
de signalisation

PAR VICTOR LEFEVRE, GABRIEL COURT ET BRICE GYEBRE

Objectifs du rapport

L'objectif de ce projet est de concevoir, mettre en œuvre et comparer plusieurs approches d'apprentissage automatique pour la classification d'images de panneaux de signalisation routière, en s'appuyant sur la base de données GTSDB (German Traffic Sign Detection Benchmark).

Ce travail a pour but d'explorer différentes familles de modèles, allant des réseaux de neurones multicouches (MLP) aux réseaux de neurones convolutifs (CNN), jusqu'à des approches de transfert d'apprentissage fondées sur des réseaux préentraînés.

Présentation de la base de données GTSDB

La base de données va être utilisée pour des tâches de détection et de reconnaissance de panneaux de signalisation.

Elle est constituée d'images couleur issues de scènes de circulation réelles, accompagnées d'un fichier d'annotations (gt.txt) décrivant, pour chaque panneau détecté, sa position dans l'image ainsi que son identifiant de classe.

Dans le cadre de ce projet, les annotations de détection sont exploitées pour extraire des imagettes de panneaux, transformant ainsi la base de données initiale en un jeu de données de classification d'images.

Diversité des classes de panneaux

On a donc 43 classes différentes de panneaux de signalisation.

Ces classes peuvent être regroupées en grandes catégories fonctionnelles :

- Limitations de vitesse (classes 0 à 8)
- Interdictions (classes 9, 10, 15, 16)
- Panneaux de priorité et d'obligation (classes 11 à 14, 17)
- Panneaux de danger (classes 18 à 31)
- Panneaux directionnels et obligatoires (classes 33 à 40)
- Fins de restrictions (classes 32, 41, 42)

Partie 1 - Exploration et préparation de la base:

Contexte et choix:

Dans cette première partie, l'objectif est de mettre en place un code simple permettant de prétraiter les données et de les mettre en forme selon nos besoins. L'idée principale est de construire une base de travail réutilisable pour les différentes parties du rapport (MLP, CNN et transfert d'apprentissage).

Les données d'entrée sont fournies sous deux formes :

- un répertoire d'images issu de la base *TrainIJCNN2013* de GTSDB, préalablement nettoyé afin de ne conserver que les images nécessaires (désigné ici par *Images*) ;
- un fichier d'annotations *gt.txt*, dans lequel chaque ligne correspond au tuple *(image_name, x_min, y_min, x_max, y_max, class_id)*.

Afin de faciliter la manipulation de ces données, l'ensemble est regroupé dans un dossier *data*. Un fichier *methode_de_base.py* est également créé au même niveau afin de centraliser les fonctions communes et indispensables au projet.

Ce fichier contient notamment des fonctions de chargement et de sauvegarde d'images, des fonctions de visualisation, ainsi qu'une fonction *load_annotation* permettant de charger les annotations sous forme de dictionnaire. Plus généralement, il regroupe toutes les fonctionnalités communes utiles aux différentes étapes du projet.

a) Visualisation et inspection rapide:

La première étape consiste à vérifier l'état et la qualité des données. Pour cela, nous avons mis en place un système de sélection aléatoire d'images à partir du répertoire *Images*. À l'aide du dictionnaire d'annotations précédemment construit, nous vérifions si une image contient des panneaux annotés, puis nous traçons les boîtes englobantes autour des panneaux détectés.

Afin de rendre cette visualisation plus pertinente, on fait en sorte de pouvoir spécifier un nombre minimum d'annotations par image. Cela permet, par exemple, de ne sélectionner que des images contenant au moins un panneau, ou au contraire d'accepter celle où il n'y en a pas.

Les figures ci-dessous représentent trois images exemples sélectionnées aléatoirement, chacune contenant au minimum trois panneaux de signalisation.



Remarques et constatations:

Après inspection d'un nombre important d'images, on constate que la base GTSDB présente une grande diversité de situations. Les panneaux apparaissent à des tailles très variables, certains sont grands et bien visibles, tandis que d'autres sont beaucoup plus petits, car situés plus loin. Les conditions de prise de vue varient également fortement, avec des différences de luminosité (plein soleil, zones d'ombre) et des angles de vue variés.

Contrairement à ce que l'on pourrait observer dans d'autres bases de données, aucune occlusion n'a été constatée, du moins pas dans les lots que j'ai visualisés, les panneaux sont toujours entièrement visibles. En revanche, certaines images présentent un léger flou, lié au mouvement ou à la distance.

b) Extraction de imagettes de panneaux :

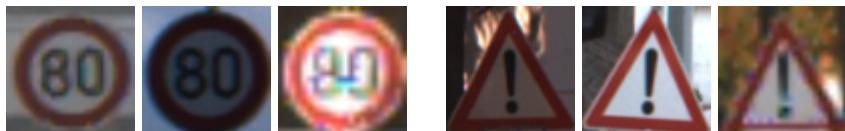
À partir du fichier d'annotations gt.txt, nous avons automatisé l'extraction des panneaux sous forme d'imagettes. Concrètement, une fonction parcourt le dictionnaire d'annotations, découpe chaque panneau dans l'image d'origine à l'aide des coordonnées, puis redimensionne l'imagette à une taille fixe (ici 64×64×3).

Les imagettes obtenues sont ensuite sauvegardées dans une arborescence simple qui conserve trié par class simplement.

Organisée sous la forme :

<taille>x<taille>/<class_id>/<image_idx>.png

Exemple trois panneaux de deux classes:



c) Filtrage des classes:

Une fois les imagettes extraites, nous avons calculé la répartition par classe afin d'identifier les classes les plus représentées et celles trop rares. De cette façon on met en évidence le déséquilibre des lots, ce qui fait que l'apprentissage risque d'être plus instable.

Pour simplifier le problème, nous avons choisi de ne conserver que les classes ayant plus de 20 imagettes. Avec ce filtrage nous travaillons sur un sous-ensemble plus homogène et suffisamment fourni pour l'entraînement. Après cette sélection, notre nombre de classes descend à 17.

Enfin, les données retenues sont divisées en trois sous-ensembles : entraînement (~70 %), validation (~15 %) et test (~15 %). Le découpage est effectué de manière stratifiée, c'est-à-dire en respectant la répartition des classes dans chaque sous-ensemble, afin de conserver une distribution similaire entre train/val/test. Les trois listes finales sont ensuite

sauvegardées (au format JSON) pour être réutilisées directement dans les phases d'entraînement et d'évaluation. Dans le cas où la stratification n'est pas appliquée, des situations extrêmes peuvent apparaître : certaines classes peuvent être absentes de certains sous-ensembles ou se retrouver en très forte minorité, ce qui pénaliserait drastiquement l'entraînement.

Résultat:

la liste sélectionnée de classes: [1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18, 25, 38]

TRAIN (total = 461)	VAL (total = 99)	TEST (total = 99)
Classe 1 : 7.38%	Classe 1 : 7.07%	Classe 1 : 7.07%
Classe 2 : 8.89%	Classe 2 : 9.09%	Classe 2 : 9.09%
Classe 3 : 3.25%	Classe 3 : 3.03%	Classe 3 : 3.03%
Classe 4 : 4.77%	Classe 4 : 4.04%	Classe 4 : 5.05%
Classe 5 : 5.64%	Classe 5 : 6.06%	Classe 5 : 5.05%
Classe 7 : 5.64%	Classe 7 : 5.05%	Classe 7 : 6.06%
Classe 8 : 7.16%	Classe 8 : 7.07%	Classe 8 : 7.07%
Classe 9 : 4.77%	Classe 9 : 5.05%	Classe 9 : 5.05%
Classe 10 : 9.54%	Classe 10 : 10.10%	Classe 10 : 9.09%
Classe 11 : 3.90%	Classe 11 : 4.04%	Classe 11 : 4.04%
Classe 12 : 8.24%	Classe 12 : 8.08%	Classe 12 : 8.08%
Classe 13 : 7.81%	Classe 13 : 8.08%	Classe 13 : 8.08%
Classe 14 : 3.25%	Classe 14 : 3.03%	Classe 14 : 4.04%
Classe 17 : 3.69%	Classe 17 : 4.04%	Classe 17 : 4.04%
Classe 18 : 4.12%	Classe 18 : 4.04%	Classe 18 : 4.04%
Classe 25 : 3.25%	Classe 25 : 3.03%	Classe 25 : 3.03%
Classe 38 : 8.68%	Classe 38 : 9.09%	Classe 38 : 8.08%

Partie 2 – Classification avec un MLP:

Contexte et objectif:

Dans cette partie, l'objectif est d'évaluer les performances de la base dans un cadre de classification à l'aide d'un réseau de neurones multicouches (MLP). Le MLP constitue un premier modèle de référence, volontairement simple, permettant d'analyser les limites d'une approche purement fully-connected sur des données d'images. Les résultats obtenus serviront ensuite de point de comparaison avec des architectures plus adaptées, notamment les CNN et le transfert de connaissances.

a) Préparation des données:

Les imagettes extraites lors de la partie précédente sont tout d'abord normalisées, en ramenant les valeurs des pixels de l'intervalle [0,255] vers [0,1]. Cette étape est indispensable pour stabiliser l'apprentissage du réseau.

Une data augmentation modérée est ensuite appliquée sur train et val lors du chargement des données. Elle comprend de légères rotations, translations ainsi que des variations de

luminosité. Cette augmentation est effectuée dynamiquement, uniquement pendant l'entraînement. Cela évite d'avoir à réellement créer et stocker des images supplémentaires. L'objectif est d'améliorer la capacité de généralisation du modèle.

Enfin, chaque image est aplatie afin de former un vecteur d'entrée compatible avec un MLP. Les labels de classes sont conservés sous forme d'entiers, directement exploitables par une fonction de coût de type cross-entropy.

b) Conception du MLP:

Le modèle retenu est un réseau multicouches simple, composé de plusieurs couches entièrement connectées. L'architecture suit une réduction progressive de la dimension des couches, jusqu'à une couche de sortie de taille égale au nombre de classes.

Les couches cachées utilisent une activation ReLU, tandis que la couche de sortie applique une fonction softmax, permettant d'interpréter les sorties comme des probabilités de classes.

Soit:

- **Entrée** : $64 \times 64 \times 364 \times 64 \times 364 \times 64 \times 3$
- **Couche cachée 1** : nombre de neurones égal à la moitié de la dimension d'entrée
- **Couche cachée 2** : 1024 neurones
- **Couche cachée 3** : 512 neurones
- **Sortie** : 17 neurones, correspondant au nombre de classes retenues

c) Entraînement:

L'entraînement du modèle est réalisé à l'aide de la fonction de coût Sparse Categorical Cross-Entropy, adaptée à une classification multi-classes avec des labels entiers.

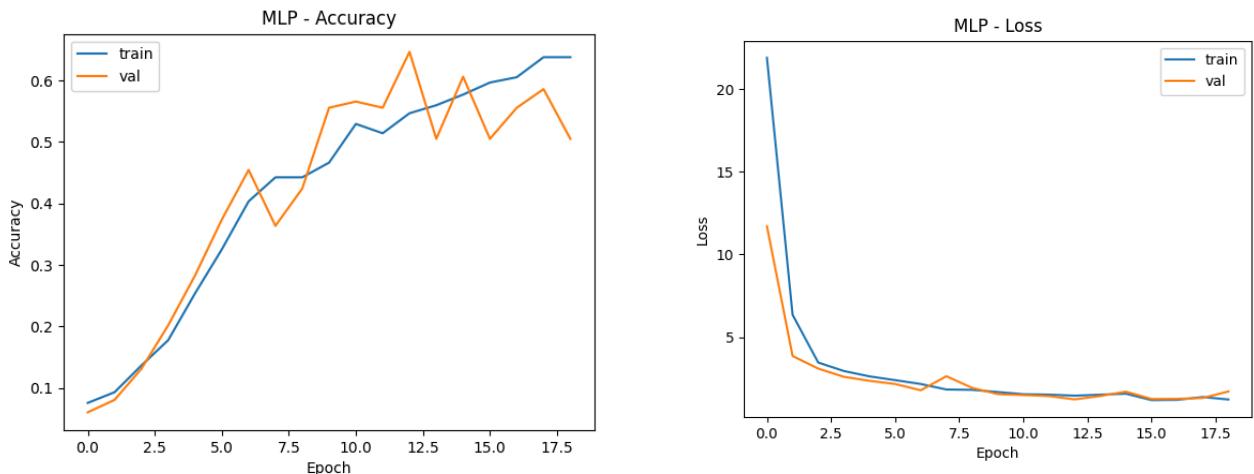
L'optimisation est assurée par l'algorithme Adam, avec un nombre d'époques fixé à 50.

Deux callbacks sont intégrés, l'un permet de sauvegarder automatiquement le meilleur modèle obtenu sur le jeu de validation, tandis que l'autre met en place un early stopping afin d'éviter le sur-apprentissage et d'améliorer la généralisation du modèle.

Enfin, le modèle final ainsi que l'historique d'entraînement sont sauvegardés.

À partir de l'historique d'entraînement, il est possible de tracer les courbes de loss et d'accuracy obtenues lors de l'entraînement et de la validation.

Nos courbes:



On observe un arrêt anticipé de l'entraînement autour de la 18^e époque. À partir de ce point, la fonction de coût (loss) ne diminue quasiment plus sur le jeu de validation.

La courbe d'accuracy montre un écart croissant entre l'entraînement et la validation : l'accuracy sur le jeu d'entraînement continue d'augmenter tandis que celle de validation a une tendance à diminuer. Ce comportement indique le début d'un sur-apprentissage, le modèle s'adaptant de plus en plus aux données d'entraînement sans améliorer sa capacité de généralisation.

d) Évaluation:

Les performances finales du MLP sont évaluées sur les jeux train et test, en termes de précision et de fonction de coût. Cette comparaison permet d'identifier un éventuel sur-apprentissage.

Résultat:

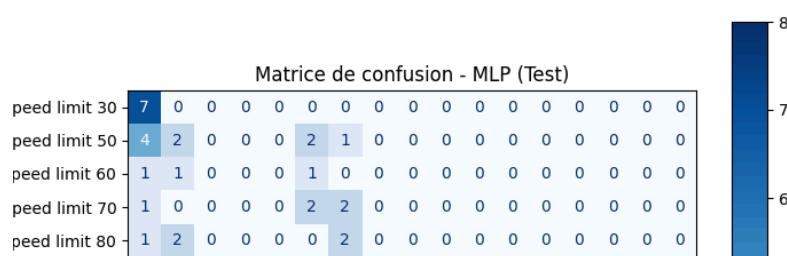
```
--- Performances TRAIN ---
2025-12-12 15:01:41.910652: I tens
rix multiplication. This will only
Loss : 1.0754 | Accuracy : 0.6616

--- Performances TEST ---
Loss : 1.2846 | Accuracy : 0.6364
```

On peut constater qu'il n'y a pas de sur-apprentissage marqué : les précisions obtenues sur les jeux d'entraînement et de test restent relativement proches. Malgré sa simplicité, le MLP parvient à atteindre une accuracy supérieure à 60 %, ce qui constitue un résultat satisfaisant compte tenu de son architecture basique et du fait qu'il ne prend pas en compte la structure spatiale des images.

Une matrice de confusion est ensuite calculée sur le jeu de test afin d'analyser plus finement les erreurs de classification.

Matrice de Confusions:



L'analyse de cette matrice montre que globalement, les performances sont correctes, mais que certaines classes sont plus fréquemment confondues. Ces confusions concernent principalement des panneaux visuellement proches, comme certaines limitations de vitesse ou des panneaux directionnels.

On le constate avec:

```
Classes les plus confondues :  
1 (speed limit 30) ==> 0 (speed limit 20) : 4 erreurs  
9 (no overtaking) ==> 14 (stop) : 4 erreurs  
14 (stop) ==> 15 (no traffic both ways) : 3 erreurs
```

Pour conclure sur cette partie, le MLP basique permet d'obtenir des résultats raisonnables, mais reste limité pour la classification d'images. Des tests supplémentaires avec des architectures plus profondes ou des couches plus larges ont permis d'atteindre environ 75 % de précision, au prix d'un coût computationnel nettement plus élevé, rendant ce compromis peu intéressant, (75% pour un modèle de 1.5Go).

Enfin, ces résultats confirment que le fait de ne pas considérer explicitement l'image comme une grille spatiale, contrairement aux réseaux convolutifs (CNN), limite fortement les performances du MLP. Cela justifie naturellement l'étude d'architectures convolutives dans la suite du projet.

Partie 3 – Classification avec couches

a) préparation des données

Les mêmes imagettes que pour le MLP ont été utilisées afin d'assurer une comparaison équitable.

Les images ont été redimensionnées en **64×64×3 et non aplatis**, afin de préserver la structure spatiale nécessaire aux convolutions.

Les tenseurs sont de forme :

$$(N, H, W, C) = (N, 64, 64, 3)$$

Les pixels ont été **normalisés** dans l'intervalle [0,1][0,1][0,1].

Aucune modification du découpage train / validation / test n'a été effectuée par rapport à la Partie 2.

b) Conception d'un réseau avec couches CNN

Structure du model :

Le réseau CNN proposé est une architecture convolutionnelle classique et raisonnable pour la classification d'images de petite taille.

Architecture utilisée

- **Entrée** : images $64 \times 64 \times 3$
- **Bloc 1**
 - Conv2D (32 filtres) + ReLU
 - MaxPooling2D
- **Bloc 2**
 - Conv2D (64 filtres) + ReLU
 - MaxPooling2D
- **Bloc 3**
 - Conv2D (128 filtres) + ReLU
- **Projection**
 - Flatten
- **Classification**
 - Dense (128) + ReLU
 - Dropout
 - Dense (nb_classes) + Softmax

Complexité du modèle

- **Nombre total de paramètres** : 4 372 691
- **Paramètres entraînables** : 100 %

Cette capacité permet au réseau d'apprendre des caractéristiques visuelles discriminantes (formes, contours, textures), contrairement au MLP

```

When using Sequential models, prefer using an `Input(shape)` object as the first layer in
the model instead.
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
→ Modèle compilé, 4372691 paramètres
Model: "sequential"



| Layer (type)                   | Output Shape        | Param #   |
|--------------------------------|---------------------|-----------|
| conv2d (Conv2D)                | (None, 64, 64, 32)  | 896       |
| max_pooling2d (MaxPooling2D)   | (None, 32, 32, 32)  | 0         |
| conv2d_1 (Conv2D)              | (None, 32, 32, 64)  | 18,496    |
| max_pooling2d_1 (MaxPooling2D) | (None, 16, 16, 64)  | 0         |
| conv2d_2 (Conv2D)              | (None, 16, 16, 128) | 73,856    |
| flatten (Flatten)              | (None, 32768)       | 0         |
| dense (Dense)                  | (None, 128)         | 4,194,432 |
| dropout (Dropout)              | (None, 128)         | 0         |
| dense_1 (Dense)                | (None, 659)         | 85,011    |



Total params: 4,372,691 (16.68 MB)
Trainable params: 4,372,691 (16.68 MB)
Non-trainable params: 0 (0.00 B)

```

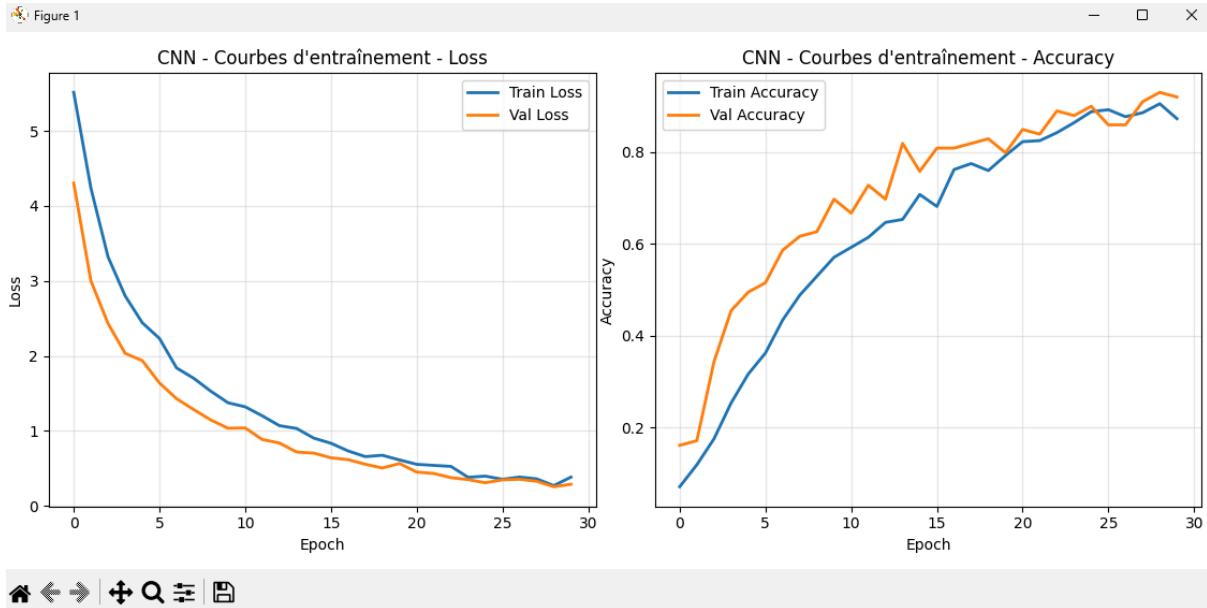
c) Entrainement

- Fonction de coût : Cross-Entropy catégorielle
- Optimiseur : Adam
- Nombre d'époques : ~30

Les courbes d'entraînement montrent :

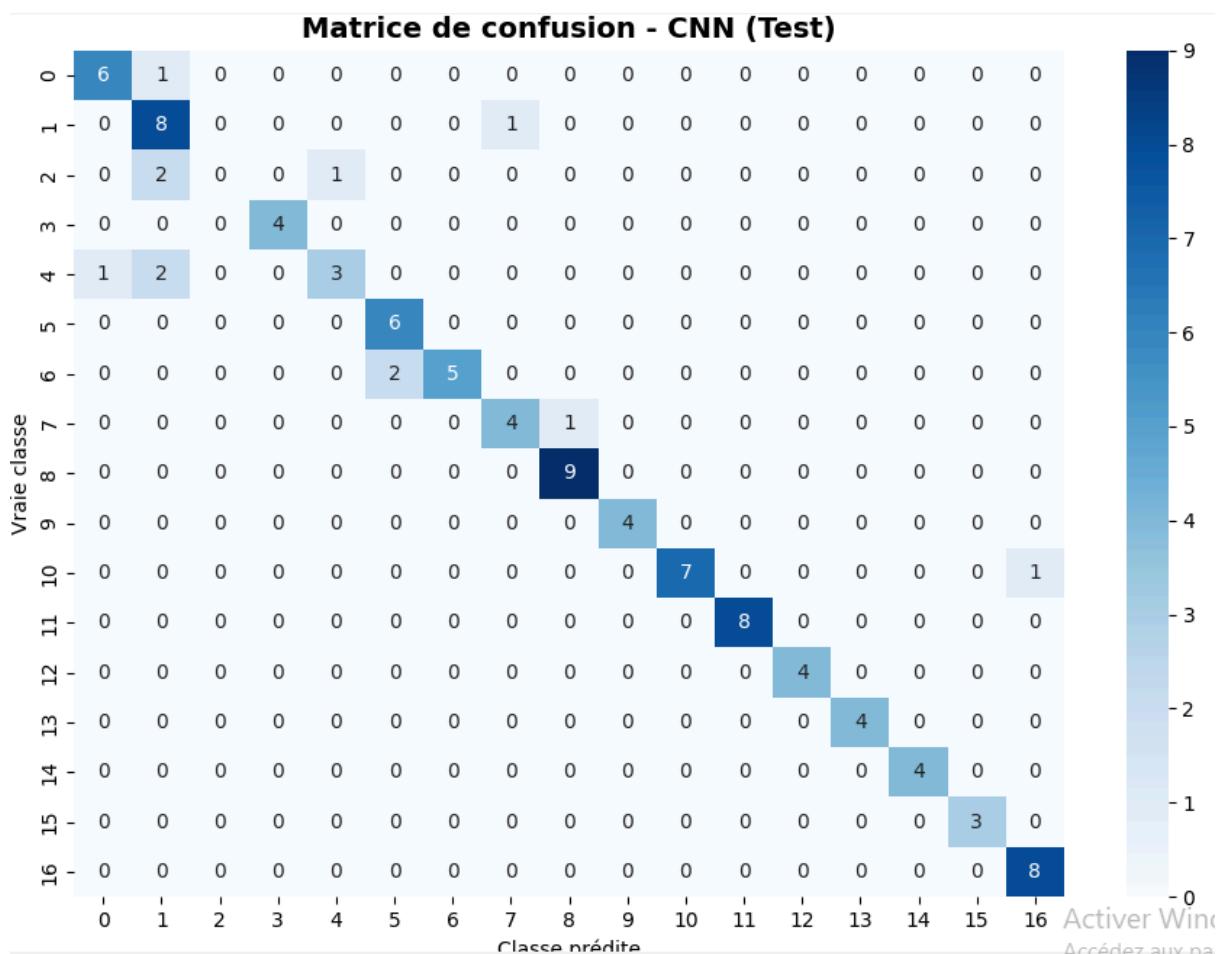
- une diminution régulière de la loss en entraînement et validation
- une augmentation progressive de l'accuracy, avec une validation stable.

tracage des courbes :



d) Evaluation

Matrice de confusion:



La matrice de confusion montre :

- une forte concentration sur la diagonale, indiquant de bonnes prédictions globales,
- des confusions résiduelles entre certaines classes visuellement proches (panneaux circulaires similaires, différences fines de symboles ou de chiffres)

Ces erreurs s'expliquent principalement par :

- la ressemblance visuelle entre certaines classes,
- la variabilité de taille, d'angle et de luminosité des panneaux

Résultats quantitatifs.

Jeu de données	Accuracy	Loss
Train	98.05 %	0.136
Test	87.88 %	0.374

La différence Train – Test = 10.17 % indique un léger sur-apprentissage, mais le modèle conserve une bonne capacité de généralisation.

Comparaison CNN vs MLP

Modèle	Accuracy Test	Capacité spatiale
MLP	~55–65 %	Aucune
CNN	87.88 %	Oui

Analyse comparative

- Le MLP, basé sur des images aplatis, perd toute information spatiale, ce qui limite fortement ses performances sur des images.
- Le CNN exploite explicitement la structure spatiale des panneaux (bords, formes, motifs), ce qui explique le gain de performance significatif.
- Le CNN converge plus rapidement et plus stablement que le MLP.

- Malgré un nombre de paramètres plus élevé, le CNN généralise nettement mieux.

Conclusion intermédiaire :

Le CNN est largement supérieur au MLP pour la classification d'images de panneaux routiers, tant en précision qu'en robustesse.

Partie 4 – Transfert de connaissances (Transfer Learning)

Le modèle utilisé dans cette partie est EfficientNet-B0, pré-entraîné sur ImageNet puis adapté à notre tâche de classification des imagettes de panneaux routiers. Seule la dernière couche fully-connected a été remplacée afin de correspondre au nombre de classes présentes dans notre dataset filtré. Les images, initialement en 64×64 , ont été redimensionnées en 224×224 et normalisées suivant les statistiques d'ImageNet afin d'assurer la compatibilité avec les poids pré-entraînés.

L'entraînement sur 12 époques montre une convergence régulière et stable : la loss d'entraînement diminue progressivement, passant de 2.73 à environ 0.12, tandis que la loss de validation suit une décroissance similaire jusqu'à atteindre 0.25. L'accuracy sur le jeu de validation progresse de manière continue pour atteindre plus de 90 %, ce qui indique que le modèle apprend efficacement à différencier les différentes classes du dataset.

L'évaluation finale sur le jeu de test confirme cette tendance avec une accuracy de 96.97 %, démontrant une très bonne capacité de généralisation. Le comportement du modèle sur l'ensemble des époques est cohérent, sans instabilité ni signe marqué de sur-apprentissage.

Dans l'ensemble, EfficientNet-B0 se révèle être un modèle performant, capable d'extraire rapidement des caractéristiques pertinentes sur un dataset relativement restreint, tout en conservant une excellente stabilité durant l'entraînement.

a) Mise en place du transfert de connaissances

(L'implémentation complète est fournie dans le code rendu.)

b) Entraînement du modèle

(L'implémentation complète est fournie dans le code rendu.)

c) Comparaison des performances avec le CNN entraîné sans transfert de connaissances

1) Comparaison des performances finales

Modèle	Accuracy Test
CNN entraîné from scratch	87.88 %
CNN pré-entraîné (EfficientNet-B0)	96.97 %

Le modèle pré-entraîné surpassé nettement le CNN entraîné sans transfert, avec un gain d'environ +9 % d'accuracy sur le jeu de test.

Cette amélioration s'explique par la capacité d'EfficientNet à exploiter des caractéristiques visuelles génériques déjà apprises (contours, textures, formes), réutilisées efficacement pour la classification des panneaux routiers.

2) Nombre d'époques nécessaires pour atteindre une bonne performance

Modèle	Nombre d'époques	Comportement
CNN from scratch	~25–30 époques	Apprentissage progressif
CNN pré-entraîné	~10–12 époques	Convergence rapide

Le CNN sans transfert nécessite près de 30 époques pour atteindre une accuracy validation élevée (>90 %), avec une montée en performance relativement lente lors des premières époques.

À l'inverse, EfficientNet-B0 atteint rapidement de bonnes performances, avec une accuracy validation supérieure à 90 % en moins de 12 époques, ce qui démontre l'efficacité du transfert de connaissances, en particulier sur un dataset de taille limitée.

3) Stabilité de l'apprentissage

- CNN entraîné sans transfert :
 - fluctuations visibles entre accuracy train et validation,

- présence d'un léger sur-apprentissage (écart train/test $\approx 10\%$),
- convergence plus lente et moins régulière.
- CNN pré-entraîné (EfficientNet-B0) :
 - courbes de loss et d'accuracy lisses et stables,
 - aucun signe marqué d'overfitting,
 - comportement cohérent tout au long de l'entraînement.

Le transfert de connaissances permet donc un apprentissage plus stable, mieux regularisé et plus robuste.

Partie 5 – Localisation

a) Génération des données

Un générateur automatique a été développé pour créer des images binaires contenant une seule forme géométrique (carré, disque ou triangle).

Les formes sont générées avec des variations aléatoires de position, rotation et échelle, et un masque cible représentant le contour de la forme est associé à chaque image.

(L'implémentation complète est fournie dans le code rendu.)

b) Apprentissage et test du réseau

Un réseau de neurones convolutionnel a été entraîné pour localiser le contour des formes à partir des images binaires.

Les tests réalisés avec les mêmes variations géométriques montrent que le réseau apprend correctement la tâche et généralise aux données non vues.

(L'implémentation complète est fournie dans le code rendu.)

c) Extension à plusieurs objets par image

Le générateur a ensuite été étendu afin de produire des images contenant plusieurs formes identiques dans une même image.

Le réseau parvient à détecter l'ensemble des contours présents, validant sa capacité à gérer plusieurs occurrences simultanément.

(L'implémentation complète est fournie dans le code rendu.)

Partie 6 –Synthèse et comparaison globale

