Deep Learning - Classification de Panneaux de Signalisation Routière

Nadia Medjdoub - Karim Ameur

1. Introduction au Projet

1.1 Introduction à la classification des panneaux de signalisation et son importance dans la conduite autonome

La reconnaissance des panneaux de signalisation, et plus généralement de l'environnement autour d'un véhicule, est un élément clé de la conduite autonome. Ce domaine repose sur deux aspects fondamentaux : la perception et la prise de décision. Un exemple marquant illustrant l'importance de la perception en conduite autonome est l'accident impliquant une Tesla Model S le 7 mai 2016 en Floride. Dans cet incident, la voiture, en mode Autopilot, a percuté un camion dont la remorque était blanche. Le système de vision embarqué a confondu cette remorque avec le ciel lumineux et n'a donc pas détecté l'obstacle provoquant un accident mortel. Cet événement a mis en lumière la nécessité d'améliorer la robustesse des algorithmes de perception afin de garantir une meilleure prise de décision pour le véhicule. En théorie, une conduite entièrement autonome pourrait, à terme, réduire à zéro le nombre de morts sur la route, mais cela dépend largement des progrès réalisés en matière de perception et d'intelligence artificielle.

1.2 Problématiques connexes à la reconnaissance des panneaux de signalisation

Notre travail se concentre sur la classification des panneaux de signalisation, mais plusieurs défis connexes sont à prendre en compte. La robustesse des systèmes de reconnaissance d'images peut être affectée par des conditions environnementales telles que la pluie, le brouillard ou un éclairage insuffisant. De plus, les différences de signalisation entre pays compliquent l'adaptabilité des modèles. Les attaques adversariales, comme les modifications visuelles de panneaux, représentent également un risque pour la sécurité des systèmes. Enfin, bien que la vision par ordinateur soit essentielle, elle doit être complétée par d'autres capteurs comme le LIDAR et le radar pour garantir une détection fiable. Ces défis, bien que cruciaux, ne seront pas abordés directement dans notre étude.

1.3 Présentation du jeu de données GTSRB

Dans le cadre de cette étude, nous allons nous appuyer sur le German Traffic Sign Recognition Benchmark (GTSRB), un jeu de données largement utilisé pour la classification des panneaux de signalisation. Disponible sur Kaggle (https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign), ce dataset a été conçu pour entraîner et évaluer des modèles de vision par ordinateur dans la reconnaissance des panneaux routiers.

Les principales caractéristiques du dataset GTSRB sont les suivantes :

- Nombre de classes : 43 catégories de panneaux de signalisation différents.
- Nombre d'images : Environ 50 000 images annotées, réparties en un ensemble d'entraînement et un ensemble de test.
- Variabilité des images : Les images présentent des variations en termes d'angles de vue, d'éclairage, de conditions météorologiques et de qualité, ce qui en fait un bon défi pour les modèles de classification.
- Labels : Chaque image est associée à un label indiquant la classe du panneau de signalisation correspondant.

1.4 Bibliothèques utilisées

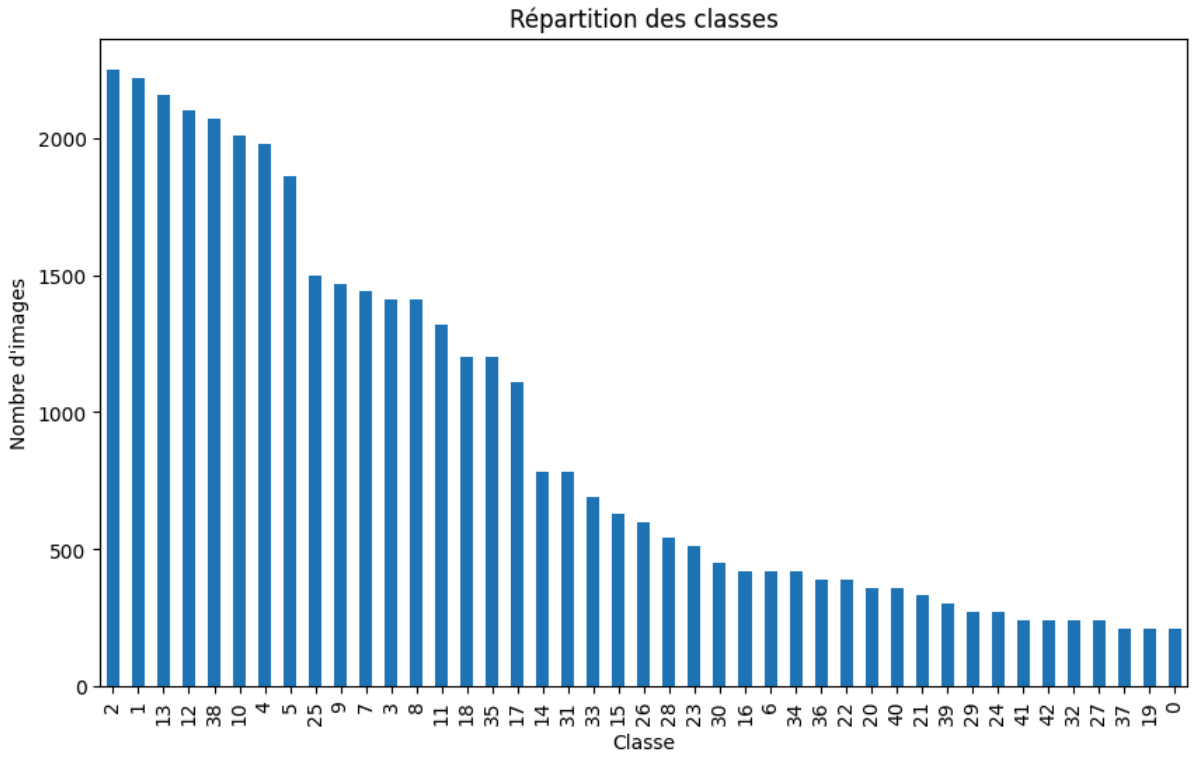
Pour mener à bien ce projet, plusieurs bibliothèques Python ont été utilisées. Ces bibliothèques couvrent un large éventail de fonctionnalités, allant de la manipulation des données à la visualisation en passant par la construction et l'évaluation des modèles. Parmi les bibliothèques principales, on trouve :
Pandas et NumPy pour faciliter la manipulation des données complexes, tandis que NumPy permet des opérations numériques efficaces; **OpenCV (cv2)** utilisée pour le prétraitement et le chargement des images, une bibliothèque clé pour la vision par ordinateur; **Matplotlib (**a visualisation de données) **et Seaborn** pour créer des graphiques statistiques attrayants; **TensorFlow:** une bibliothèque de calcul numérique pour le machine learning, **et Keras** : une API de haut niveau pour construire et entraîner des modèles de Deep Learning; **Scikit-learn** : qui

fournit des outils simples pour la classification, la régression et le clustering, et pour l'évaluation des modèles; enfin, **EarlyStopping de Keras** qui permet d'arrêter l'entraînement prématurément pour éviter le surapprentissage et gagner du temps.

2. Préparation des Données

Les images du dataset ont été chargées et prétraitées avec OpenCV, redimensionnées à 32x32 pixels et normalisées en les divisant par 255.0 pour améliorer la convergence du modèle. Les labels ont été extraits des noms des répertoires, chaque répertoire représentant une classe de panneau, puis convertis en valeurs numériques et en one-hot encoding pour être compatibles avec la fonction de perte du modèle.

Une exploration des labels a été réalisée pour vérifier la répartition des classes, avec une visualisation à l'aide de Seaborn afin d'identifier un éventuel déséquilibre et pouvoir le traiter le avec des modèles imbriqués. Dans notre cas l'équilibre des classes est à peu près respecté et les données ne demande pas de traitement particulier.



Les données ont été divisées en trois ensembles : 80% pour l'entraînement, 20% pour le test, et un ensemble de validation préparé à partir du dossier Test pour évaluer le modèle en dehors des jeux d'apprentissage. Afin d'améliorer la robustesse du modèle, des transformations aléatoires telles que la rotation, le décalage horizontal et vertical, le cisaillement, le zoom et le retournement horizontal ont été appliquées, générant des variations d'images pour aider le modèle à mieux généraliser et éviter le surapprentissage, en particulier lorsque le nombre d'images par classe est limité.

```
Forme de X (données images) : (39209, 32, 32, 3)
Exemple de labels encodés : [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Forme de X_train : (31367, 32, 32, 3)
Forme de X_test : (7842, 32, 32, 3)
Forme de Y_train : (31367, 43)
Forme de Y_test : (7842, 43)
```

Visualisation de la forme du dataframe

3. Conception et Implémentation du Modèle CNN

3.1 Le modèle Keras simple

Dans un premier temps, nous avons réalisé un modèle CNN avec Keras. L'objectif de cette étape était de poser les bases pour la création d'un modèle performant et précis. Voici la structure du modèle que nous avons conçu :

```
# Définition du modèle CNN
def create_model(optimizer):
    model = models.Sequential()
    model.add(layers.Input(shape=(32, 32, 3)))  # Entrée 32x32 avec 3 canaux (RGB)
    model.add(layers.Conv2D(32, (3, 3), activation='relu'))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(43, activation='softmax'))  # 43 classes
    model.compile(optimizer=optimizer,
                  loss='sparse_categorical_crossentropy',  # Pour la classification
multi-classes
                  metrics=['accuracy'])
    return model
```

Ce modèle est composé de trois couches convolutives suivies de couches denses. Les couches convolutives permettent d'extraire des caractéristiques spatiales des images, tandis que les couches denses combinent ces caractéristiques pour effectuer la classification. La fonction d'activation ReLU est utilisée pour introduire de la non-linéarité, et la fonction softmax en sortie permet d'obtenir une distribution de probabilité sur les 43 classes.

3.2 Optimisation des Hyperparamètres via GridSearch

Afin d'obtenir les meilleures performances possibles, nous avons implémenté ce modèle au travers d'une recherche par grille (GridSearch). L'objectif de cette étape était d'identifier les hyperparamètres optimaux pour le modèle, tels que la taille du lot (batch size), le nombre d'époques (epochs), l'optimiseur et le taux d'apprentissage (learning rate). Voici la grille des hyperparamètres testés :

```
param_grid = {
    'batch_size': [32, 64, 256],
    'epochs': [10, 20],
```

```
    'optimizer': ['adam', 'sgd'],
    'learning_rate': [0.001, 0.01]
}
```

Les performances de chaque combinaison d'hyperparamètres ont été évaluées et sauvegardées dans un fichier CSV pour une analyse ultérieure. Cette approche systématique nous a permis de comparer les résultats et de sélectionner la configuration la plus performante. Après la compilation et l'évaluation des différentes configurations, **les meilleures performances ont été obtenues avec les paramètres suivants** :

```
batch_size    epochs optimizer     learning_rate avg_val_score
256    20     adam   0.001  0.9643893639246622
```

Cette configuration a permis d'atteindre **une précision de validation de 96.44%,** ce qui démontre l'efficacité du modèle et des hyperparamètres choisis. L'optimiseur Adam avec un learning rate de 0.001 s'est avéré particulièrement performant, combinant une convergence rapide et une grande stabilité.

4. Modèle affiné

4.1 Optimisation et Évaluation du Modèle CNN avec Activation ReLU

Dans un deuxième temps, nous avons optimisé notre modèle CNN en utilisant la fonction d'activation ReLU et en ajoutant des techniques avancées pour améliorer ses performances.

Le modèle commence par plusieurs **couches convolutives** qui appliquent des filtres pour extraire des caractéristiques spatiales des images. Chaque couche convolutive est suivie d'une **Batch Normalization** pour normaliser les sorties et stabiliser l'entraînement.
Après chaque couche convolutive, nous appliquons un **MaxPooling2D** pour réduire la dimension spatiale des cartes de caractéristiques, ce qui permet de réduire le nombre de paramètres et de prévenir le surapprentissage.
Un **Dropout** est également ajouté après chaque couche de pooling pour désactiver aléatoirement une fraction des neurones pendant l'entraînement, ce qui aide à éviter le surapprentissage.

Au lieu d'utiliser des couches fully connected traditionnelles, nous avons opté pour une couche **GlobalAveragePooling2D**. Cette couche réduit chaque carte de caractéristiques à un seul nombre en prenant la moyenne de toutes les valeurs, ce qui réduit considérablement le nombre de paramètres et le risque de surapprentissage. Une couche dense avec 256 neurones et une activation ReLU est utilisée pour combiner les caractéristiques extraites. Cette couche est également suivie d'une Batch Normalization et d'un Dropout pour améliorer la généralisation. Enfin, la couche de sortie utilise une activation softmax pour produire une distribution de probabilité sur les 43 classes.

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 30, 30, 32)	896
batch_normalization_15 (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d_9 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_15 (Dropout)	(None, 15, 15, 32)	0
conv2d_13 (Conv2D)	(None, 13, 13, 64)	18,496
batch_normalization_16 (BatchNormalization)	(None, 13, 13, 64)	256
max_pooling2d_10 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_16 (Dropout)	(None, 6, 6, 64)	0
conv2d_14 (Conv2D)	(None, 4, 4, 128)	73,856
batch_normalization_17 (BatchNormalization)	(None, 4, 4, 128)	512
max_pooling2d_11 (MaxPooling2D)	(None, 2, 2, 128)	0
dropout_17 (Dropout)	(None, 2, 2, 128)	0
conv2d_15 (Conv2D)	(None, 2, 2, 256)	295,168
batch_normalization_18 (BatchNormalization)	(None, 2, 2, 256)	1,024
dropout_18 (Dropout)	(None, 2, 2, 256)	0
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 256)	0
dense_6 (Dense)	(None, 256)	65,792
batch_normalization_19 (BatchNormalization)	(None, 256)	1,024
dropout_19 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 43)	11,051

4.2 Validation Croisée

Pour évaluer la robustesse du modèle, nous avons utilisé une validation croisée en 3 folds. Cette technique consiste à diviser les données d'entraînement en 3 sous-ensembles et à entraîner le modèle 3 fois, en utilisant chaque sous-ensemble comme ensemble de validation une fois. Les résultats montrent une précision moyenne de 99.61% avec un écart-type très faible, ce qui indique que le modèle est stable et performant sur différents sous-ensembles de données.

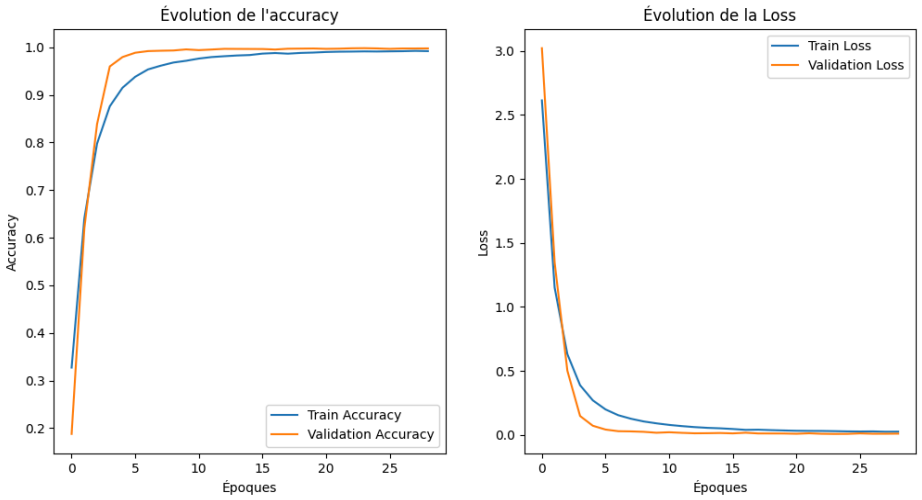
4.3 Entraînement Final et Early Stopping

Le modèle final a été entraîné sur l'ensemble des données d'entraînement avec un early stopping. Cette technique surveille la perte en validation et arrête l'entraînement si la performance ne s'améliore pas après un certain nombre d'époques (patience de 5 époques dans notre cas). Cela permet d'éviter le surapprentissage et de restaurer les meilleurs poids trouvés pendant l'entraînement.

4.4 Résultats et Évaluation

Les performances du modèle sur le jeu de test sont excellentes, avec une perte de 0.0059 et une précision de 99.81%. Ces résultats montrent que le modèle est très performant et généralise bien sur des données non vues. La faible valeur de la loss indique que le modèle fait des prédictions très proches des étiquettes réelles.

L'accuracy en entraînement et en validation augmente régulièrement, atteignant presque 100% après 20 époques, avec un faible écart entre les deux, indiquant un bon équilibre. La loss diminue rapidement, restant proche de 0, et la validation loss suit la train loss, confirmant l'absence de surapprentissage.



4.5 Déploiement sur le Jeu de Test GTSRB

Le modèle a été utilisé pour faire des prédictions sur le jeu de test issu du dossier Test de GTSRB. Les performances obtenues sont excellentes, a**vec une précision de 99.81%**, ce qui confirme que le modèle est prêt à être déployé dans un environnement réel.

5. Évaluation du Modèle

Les performances obtenues étaient satisfaisantes, nous l'avons donc utilisé sur notre troisième jeu de données, à savoir le jeu de test issu du dossier Test de GTSRB.

Exemple de prédiction

6. Améliorations et Expérimentations

Après plusieurs expérimentations, nous avons finalement opté pour un modèle de réseau de neurones convolutifs (CNN) en reprenant l'architecture du modèle ReLU précédent, mais en changeant la fonction d'activation par Swish.

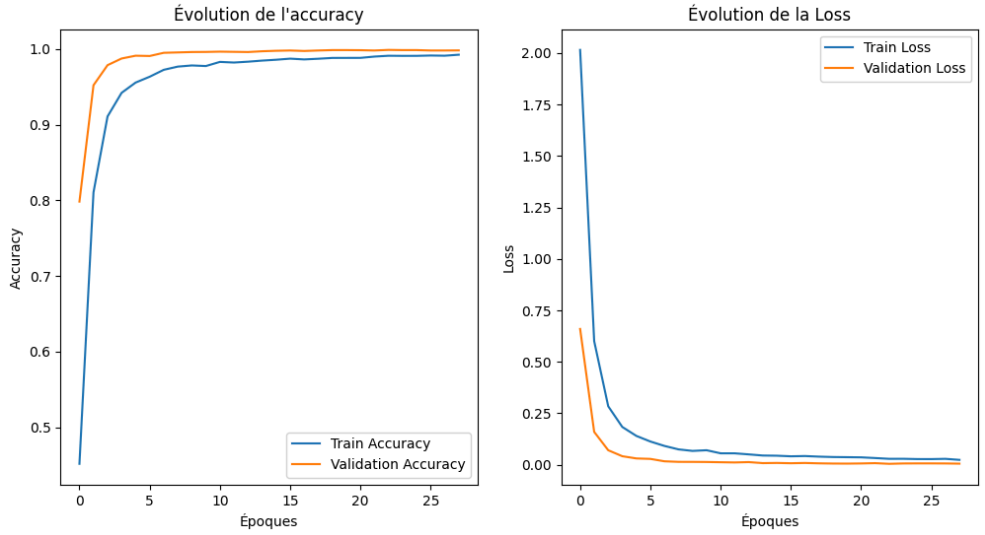
La fonction d'activation Swish, définie par la formule $\text{Swish}(x)=x \cdot \sigma(\beta x)$ où σ est la fonction sigmoïde et β est un paramètre (souvent fixé à 1), a montré des performances supérieures par rapport à l'activation ReLU traditionnelle. Swish permet une meilleure propagation des gradients, ce qui améliore la convergence du modèle et conduit à de meilleures performances sur des tâches complexes.
Pour évaluer les performances du modèle, nous avons utilisé une approche de validation croisée avec trois plis (StratifiedKFold). Cette méthode permet de diviser les données en plusieurs ensembles d'entraînement et de validation, assurant ainsi une évaluation plus robuste des performances du modèle. **Les résultats de la validation croisée ont montré une précision moyenne impressionnante de 99.61% avec un écart-type de 0.08%.**

Model: "sequential_16"

Layer (type)	Output Shape	Param #
conv2d_64 (Conv2D)	(None, 30, 30, 32)	896
batch_normalization_80 (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d_48 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_80 (Dropout)	(None, 15, 15, 32)	0
conv2d_65 (Conv2D)	(None, 13, 13, 64)	18,496
batch_normalization_81 (BatchNormalization)	(None, 13, 13, 64)	256
max_pooling2d_49 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_81 (Dropout)	(None, 6, 6, 64)	0
conv2d_66 (Conv2D)	(None, 4, 4, 128)	73,856
batch_normalization_82 (BatchNormalization)	(None, 4, 4, 128)	512
max_pooling2d_50 (MaxPooling2D)	(None, 2, 2, 128)	0
dropout_82 (Dropout)	(None, 2, 2, 128)	0
conv2d_67 (Conv2D)	(None, 2, 2, 256)	295,168
batch_normalization_83 (BatchNormalization)	(None, 2, 2, 256)	1,024
dropout_83 (Dropout)	(None, 2, 2, 256)	0
global_average_pooling2d_16 (GlobalAveragePooling2D)	(None, 256)	0
dense_32 (Dense)	(None, 256)	65,792
batch_normalization_84 (BatchNormalization)	(None, 256)	1,024
dropout_84 (Dropout)	(None, 256)	0
dense_33 (Dense)	(None, 43)	11,051

Total params: 1,401,667 (5.35 MB)
Trainable params: 466,731 (1.78 MB)

L'analyse des graphiques de la loss value et de l'accuracy montrent une convergence rapide du modèle. La courbe de l'accuracy indique que le modèle atteint rapidement une précision élevée, tant pour l'ensemble d'entraînement que pour l'ensemble de validation. De même, la courbe de la perte montre une diminution rapide, ce qui indique que le modèle apprend efficacement à minimiser l'erreur.

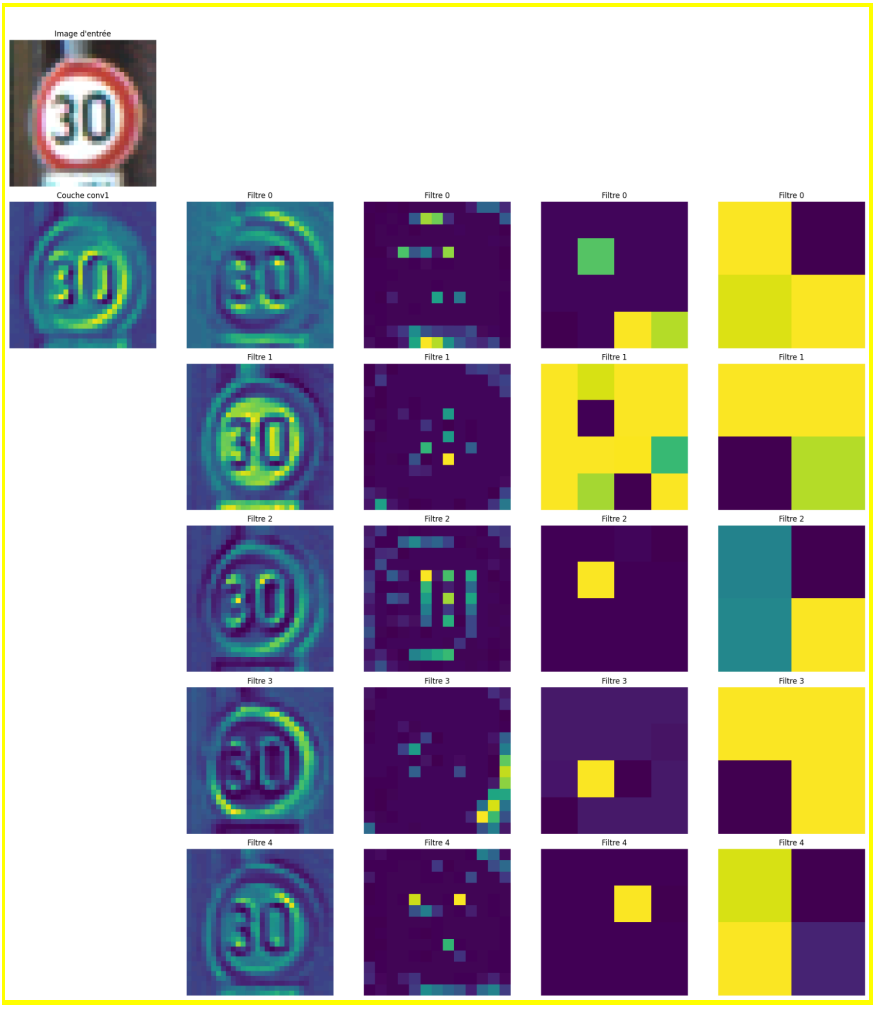


Après la validation croisée, nous avons entraîné le modèle final sur l'ensemble des données d'entraînement pendant 40 époques. Les courbes d'apprentissage montrent une augmentation rapide de la précision et une diminution de la perte, tant pour les ensembles d'entraînement que de validation. **Le modèle a atteint une précision de 99.87% sur l'ensemble de test, avec une perte de 0.0057. Cela représente un gain de 0,05 % de**

performance. Bien que cela puisse sembler minime, dans un contexte réel, cette amélioration est particulièrement significative. En effet, lors d'un trajet en voiture, il est possible de rencontrer plusieurs dizaines de panneaux, et comme mentionné en introduction, chaque erreur peut avoir des conséquences graves. Par conséquent, chaque gain en performance devient crucial.
La matrice de confusion montre que le modèle est capable de classer correctement la grande majorité des images de panneaux de signalisation, avec très peu de faux positifs ou de faux négatifs. Les courbes d'apprentissage indiquent que le modèle converge rapidement et généralise bien aux données de validation.
En conclusion, l'utilisation de la fonction d'activation Swish et l'architecture du modèle CNN ont permis d'obtenir des performances exceptionnelles sur le jeu de données GTSRB. Ces résultats démontrent l'efficacité de l'approche proposée pour la reconnaissance des panneaux de signalisation allemands.

7. Interprétation et Visualisation des Résultats

Exemple Activation à l'epoch 3:



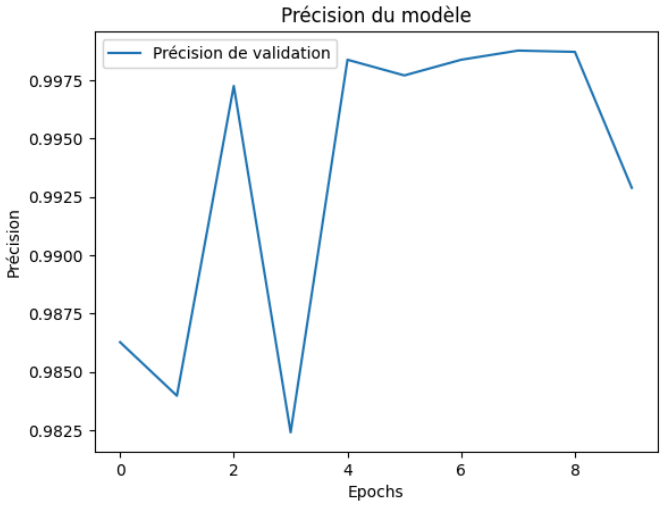
À travers les couches de notre modèle CNN, l'image d'entrée subit une transformation hiérarchique. Les premières couches détectent les caractéristiques de base comme les bords et les textures. Les couches intermédiaires combinent ces caractéristiques pour détecter des motifs plus complexes. Les couches profondes identifient des caractéristiques encore plus abstraites, permettant une reconnaissance robuste et généralisée des objets. Cette évolution permet au modèle de classer les images avec une grande précision, même en présence de variations.

8. Extension du Projet

Dans le cadre de l'extension du projet, un système de classification a été développé, combinant une classification binaire pour déterminer si une image est un panneau ou non, et une classification multi-classes pour identifier le type de panneau. Le système utilise deux jeux de données : CIFAR-10 pour les images non-panneaux et GTSRB pour les images de panneaux, qui sont concaténés, normalisés et divisés en ensembles d'entraînement et de test (80% - 20%).

Le modèle de classification binaire distingue les images de panneaux des non-panneaux à l'aide d'une architecture CNN avec une sortie sigmoïde, produisant une probabilité entre 0 et 1. Si cette probabilité est supérieure à 0.5, l'image est classée comme un panneau. Le modèle multi-classes, quant à lui, utilise la sortie du modèle binaire pour identifier la classe du panneau, avec une fonction d'activation softmax qui génère une distribution de probabilité sur 43 classes.

Lorsqu'une nouvelle image est soumise, elle est d'abord prétraitée et le modèle binaire détermine si c'est un panneau. Si c'est le cas, le modèle multi-classes prédit le type de panneau. Par exemple, une image de test avec une probabilité de 0.9975 d'être un panneau est classée dans la classe 22, et une image de référence correspondante est affichée pour comparaison.



9. Conclusion

Le modèle binaire que nous avons développé montre de bonnes performances, **avec une précision de validation atteignant 99.75%.** Cependant, il n'est pas parfait et peut encore commettre des erreurs, bien que celles-ci soient rares. Dans le contexte de la conduite autonome, où chaque erreur peut avoir des conséquences graves, même un taux d'erreur faible reste problématique. Une seule erreur de classification pourrait entraîner une mauvaise décision du véhicule, mettant potentiellement en danger la vie des usagers de la route.

Cette problématique renvoie à l'importance cruciale de la perception dans les systèmes de conduite autonome, comme l'a illustré l'accident de la Tesla Model S en 2016. Dans cet incident, une erreur de perception a conduit le véhicule à ne pas détecter un camion, confondant sa remorque blanche avec le ciel. Cet événement a mis en lumière les limites des systèmes de vision actuels et la nécessité de développer des modèles plus robustes. Bien que notre modèle soit performant, il reste vulnérable à des erreurs qui, bien que rares, pourraient avoir des conséquences dramatiques dans un contexte réel.

Les défis de la reconnaissance des panneaux de signalisation vont au-delà de la simple classification. Les conditions environnementales (pluie, brouillard, neige), les variations de signalisation entre pays, et les attaques adversariales (comme l'ajout de stickers sur les panneaux) constituent des obstacles majeurs. De plus, la vision par ordinateur doit être complétée par d'autres technologies, telles que le LIDAR et le radar, pour garantir une détection fiable dans toutes les situations.

En conclusion, notre modèle binaire et multi-classes offre de bonnes performances, mais il n'est pas infaillible. Dans un contexte où chaque erreur peut coûter des vies, il est essentiel de continuer à améliorer la robustesse et la précision des systèmes de perception. La conduite autonome promet de réduire à zéro le nombre de morts sur les routes, mais cet objectif ne pourra être atteint que si les modèles de reconnaissance visuelle atteignent un niveau de fiabilité absolue. Notre travail contribue à cette quête, mais il reste encore beaucoup à faire pour garantir la sécurité des systèmes autonomes dans des conditions réelles et variées.