# GTSRB - German Traffic Sign Recognition Benchmark

*Année universitaire  
2024 / 2025*

**TRAN Vincent  
BONANE Marius  
BOUMEZAOUED Hatim  
GRANT Harry**

Du 11/02/2025 au 09/03/2025

***GTSRB - German Traffic Sign Recognition Benchmark***

***Multi-class, single-image classification challenge***

Rapport du projet

# Introduction du projet

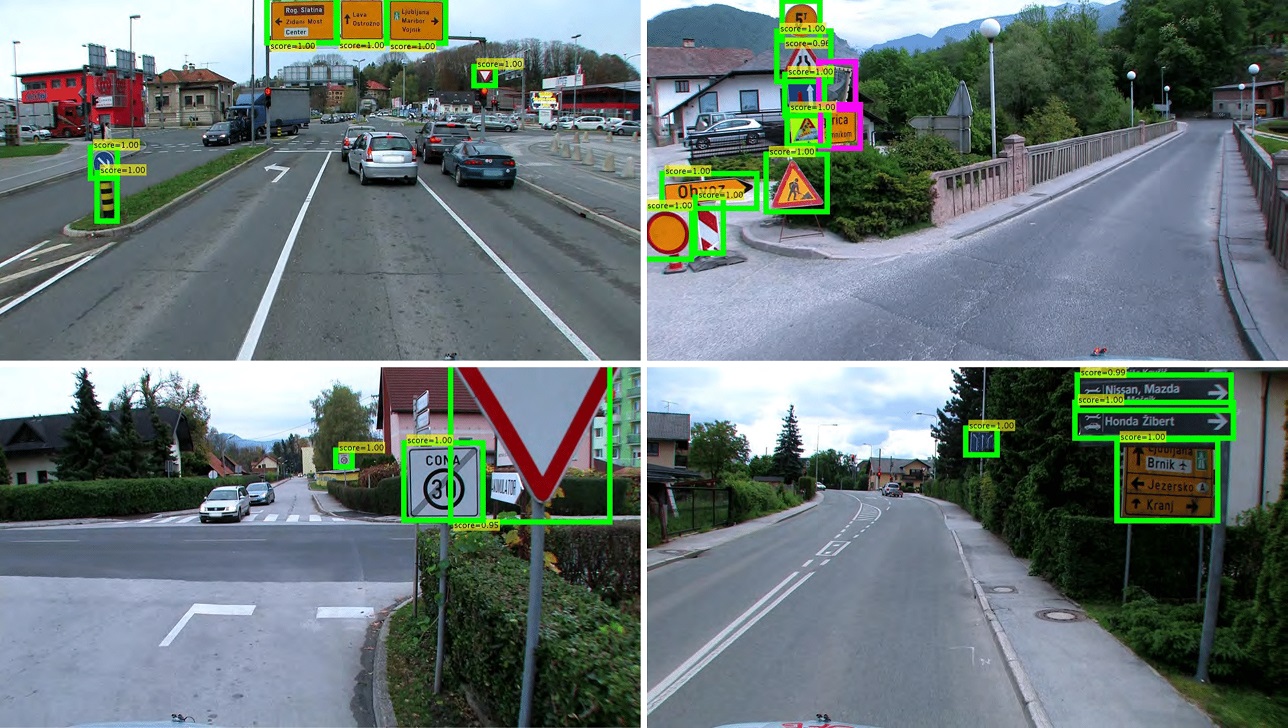
1. **Objectif au Projet**

Ce projet vise à concevoir un système de reconnaissance d'images permettant d'identifier avec précision différentes catégories de panneaux de signalisation routière. Il suit une méthodologie en neuf étapes, couvrant l'introduction et l'analyse des données, l'entraînement et l'évaluation du modèle, ainsi que des phases d'amélioration et d'expérimentation, jusqu'à l'interprétation des résultats. L'approche repose sur l'apprentissage automatique, en particulier les réseaux de neurones convolutionnels (CNN), et intègre diverses stratégies d'augmentation de données et de régularisation pour optimiser les performances du modèle.

1. **Origine du dataset**

Le GTSRB a été introduit dans le cadre de l’International Joint Conference **on Neural Networks (IJCNN) 2011** pour stimuler la recherche sur la reconnaissance de panneaux de signalisation. La reconnaissance automatique de ces panneaux est un défi en raison des variations d'**éclairage**, de **perspective**, de **dégradations**, ainsi que des différences dues aux **conditions météorologiques** et aux **occlusions**.

Le German Traffic Sign Recognition Benchmark (GTSRB) **est un projet conçu pour évaluer les performances des algorithmes de reconnaissance de panneaux de signalisation routière**. Il s'inscrit dans le domaine de la vision par ordinateur, en particulier pour les applications de conduite autonome et d'assistance à la conduite.



***Captures de panneaux de signalisation***

# Préparation de données

1. **Exploration des données**

Le dataset contient **plus de 50 000 images** de panneaux de signalisation allemand, réparties en **43 classes distinctes** (exemple : limitation de vitesse, interdiction de dépassement, signalisation de danger, etc.). Les images sont capturées dans des environnements réels, avec des variations de taille, de luminosité et d'angle de vue.

1. **Data Preprocessing**

Nous allons définir **une résolution de 64x64 pixels**, qui couvre la majorité des images. Ensuite, nous appliquerons une normalisation pour convertir les valeurs des pixels de l'image **en valeurs entre 0 et 1.**

Pour les labels, nous appliquerons **le one-hot encoding**, ce qui permettra **de transformer les catégories en vecteurs binaires**, facilitant ainsi l'apprentissage du modèle dans un contexte de **classification multi-classes.**

**Informations descriptives essentielles des panneaux :**

* **ClassId** : identifiant du pictogramme (43 classes)
* **ShapeId** : forme (triangle, cercle, etc.)
* **ColorId** : couleur (rouge, bleu, etc.)

**Caractéristiques des images annotées :**

* **Width / Height** : Dimensions
* **Roi.X1, Y1, X2, Y2** : coordonnées du **ROI** (zone du panneau dans l’image)
* **ClassId** : identifiant du panneau

Le **ROI** (**région d’intérêt)** permet d'extraire uniquement la zone utile d’une image, apportant :  
✅ **Réduction du bruit** (moins de données inutiles)  
✅ **Optimisation des performances** (traitement plus rapide)  
✅ **Meilleure précision** (focalisation sur les panneaux)

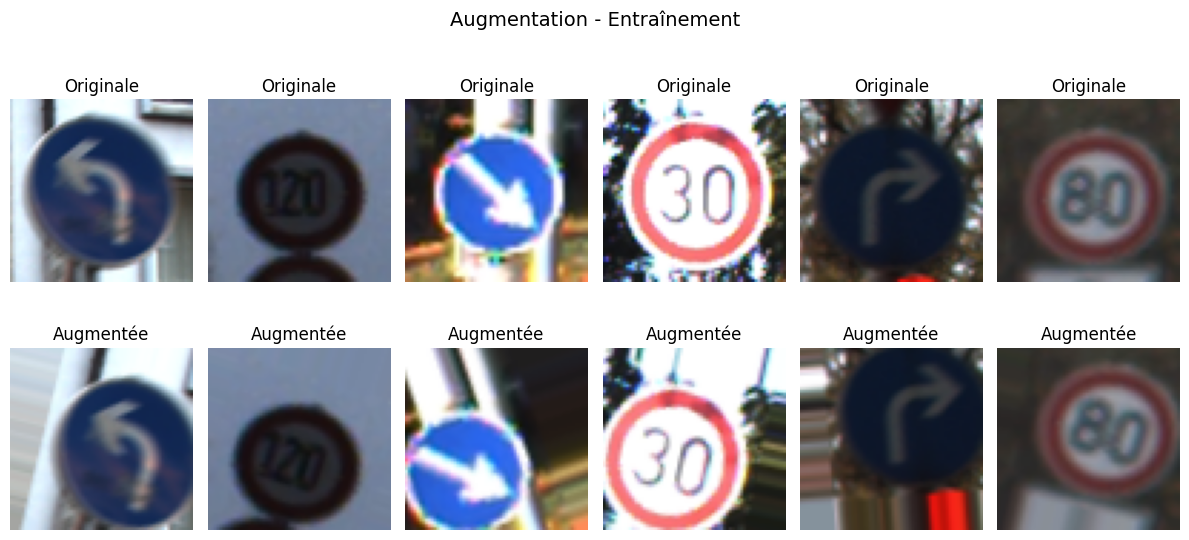


1. **Séparation du jeu de données**

On sépare d'abord les données **en train (70%) en test (15%) et en validation (15%).**

Cela permettra de garantir une répartition équilibrée des classes, évitant ainsi les biais lors de l'entraînement du modèle.

1. **Data Augmentation**

On applique la data augmentation pour faire plusieurs types de transformations (rotation et décalage) pour enrichir le dataset et améliorer la généralisation du modèle.

***Résultats obtenus pour la Data Augmentation sur le jeu d’entrainement***

# Architecture des modèles

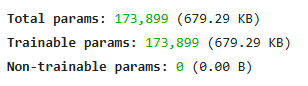
**Séparation des données :**

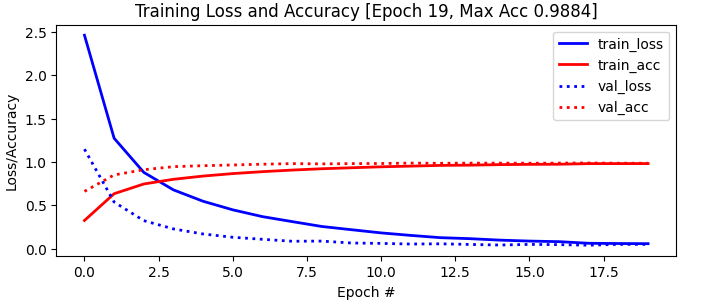
* **Avant augmentation des données :**
  + Entraînement : 28 327 images
* **Après augmentation des données (données doublés) :**
  + Entraînement : 56 654 images

# Architecture des modèles

Pour les trois modèles, nous avons configuré le même optimiseur Adam, qui sera appliqué à une entrée de dimension **64×64×3 (où 64×64 représente la résolution de l’image et 3 correspond aux canaux de couleur RVB) e**t produira une prédiction sur 43 classes en sortie.  
On utilise la fonction d’activation **ReLU** qui permet d'introduire de la non-linéarité dans le réseau, ce qui le rend capable d'apprendre des représentations complexes des données.

1. **Modèle Simple CNN**

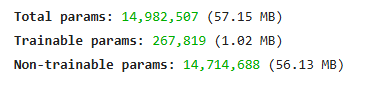


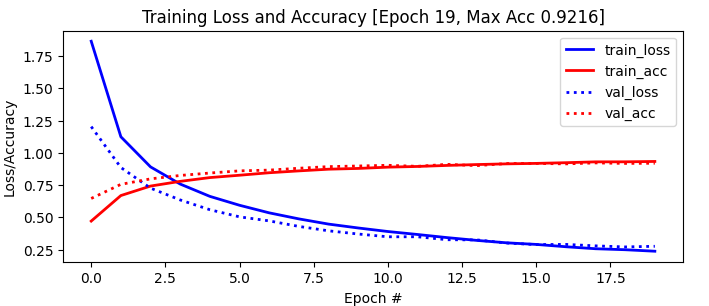


**Dernier relevé:** accuracy: 0.9813 - loss: 0.0584 - val\_accuracy: 0.9856 - val\_loss: 0.0503

Pour une première expérience on a paramétré pour la première couche dense du neurone  
**64 neurones avec 20 epochs**

1. **Modèle VGG16 pré entrainé ImageNet (Transfer Learning)**





**Dernier relevé:** accuracy: 0.9356 - loss: 0.2336 - val\_accuracy: 0.9190 - val\_loss: 0.2758

**Après l’extraction des features du VGG16, on va paramétrer plusieurs expérimentations** pour changer nombre de neurones dans la première couche dense ainsi que la taille des batchs pour voir la mise à jour des poids :

On relève deux types d’expériences pour le cas du VGG16 :  
Pour un temps exécution plus court moins de 2 minutes on trouve :

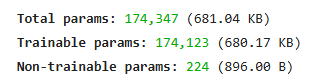
* 128 neurones = 90.4% val\_accurary avec 10 epochs
* 256 neurones = 91.2% val\_accurary avec 10 epochs

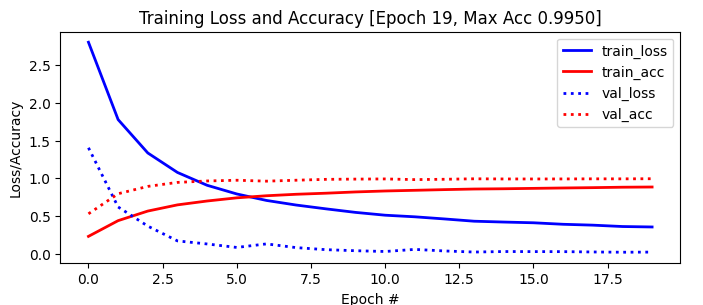
Pour un temps exécution plus long environ 5-10 min on trouve :

* **128 neurones = 92.1% val\_accurary pour 20 epochs**
* 256 neurones = 91.3% val\_accurary pour 20 epochs

**Le Transfer Learning avec VGG16** nous permet **d’obtenir un modèle performant** avec moins de données et un temps d'entraînement réduit, qui sera pré entrainé sur **ImageNet.**On bloque les premières couches du modèle (déjà bien entraînées) pour conserver les caractéristiques générales des images.

1. **Model Régularisation CNN**



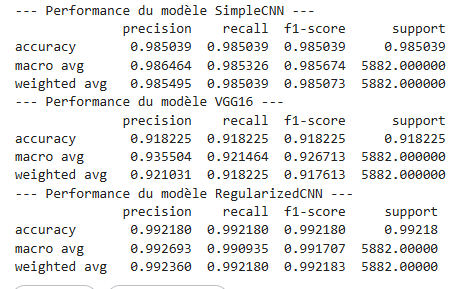


**Dernier relevé:** accuracy: 0.8857 - loss: 0.3516 - val\_accuracy: 0.9950 - val\_loss: 0.0220

Pour une première expérience on a paramétré pour la première couche dense du neurone  
**64 neurones avec 20 epochs. Et sur chaque couche, nous avons utilisé un Dropout à 0.25 et une BatchNormalization**

**Bilan :** Le dernier modèle de régularisation pour le CNN a permis **de limiter l'overfitting, comme en témoigne la courbe du val\_loss, qui reste en dessous de celle du train\_loss.**   
En comparaison, le modèle **SimpleCNN** présente un léger overfitting dès la 4ᵉ époque, tandis que le modèle **VGG16** commence à surajuster à partir de la 5ᵉ époque.

# Evaluation et comparaison des modèles



***Voici les derniers relevés des 3 performances de modèle***

**Différentes métriques utilisées :**

**Précision :** Mesure la qualité des prédictions positives. Utile si les faux positifs sont coûteux   
**Recall :** Mesure la capacité à détecter tous les cas positifs. Important si les faux négatifs sont critiques. **F1-Score :** Équilibre entre précision et rappel, utile quand les classes sont déséquilibrées.

**Accuracy :** Données équilibrées  
**Macro Average :** Données déséquilibrées (intérêt pour toutes les classes)  
**Weighted Average :** Données déséquilibrées (réflexion globale, mais sensible aux classes majoritaires) 

**Bilan de performance :**

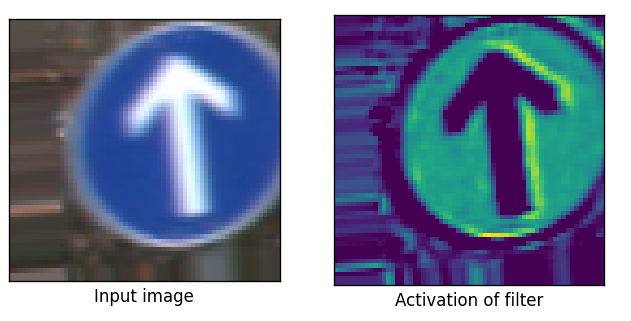
* **RegularizedCNN est clairement le meilleur choix pour ce jeu de données, avec des performances supérieures en termes de précision, rappel et F1-score.**
* **Simple CNN est une bonne alternative si l'on recherche un modèle plus simple mais performant.**
* **VGG16 semble moins adapté à ce problème spécifique en raison de sa complexité.**

# Visualisation de l’activation des couches CNN

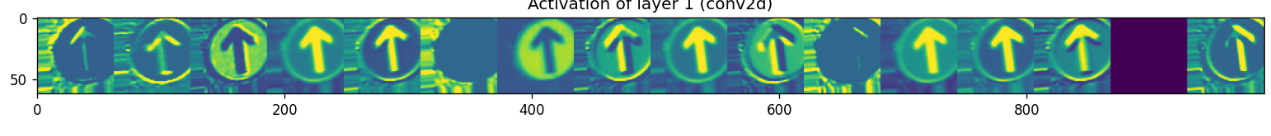
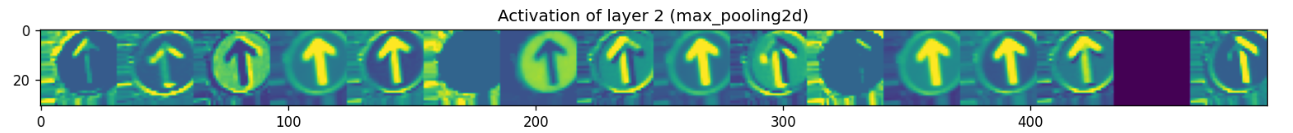
Nous allons déterminer les activations des couches convolutives pour notre modèle Simple CNN **d’un réseau de neurones convolutif (CNN)** qui mettent en évidence les caractéristiques spécifiques apprises à chaque étape de l’entraînement. **Les visualiser permet de comprendre quelles zones de l’image provoquent des réactions** dans les **différentes couches et d’analyser la manière dont le réseau interprète les motifs visuels.**

On remarque que **l'image de gauche représente l'image d'entrée, qui est une image d'une flèche blanche**, et l'image de droite montre **l'activation d'un filtre spécifique** dans une couche de convolution d'un réseau de neurones convolutionel (CNN). Cette activation **met en évidence les caractéristiques détectées par le filtre,** notamment les contours et les zones d'intérêt de la flèche.

On observe que **le filtre accentue les bordures de la flèche, indiquant qu'il est sensible aux contrastes** et aux formes présentes dans l'image d'entrée.

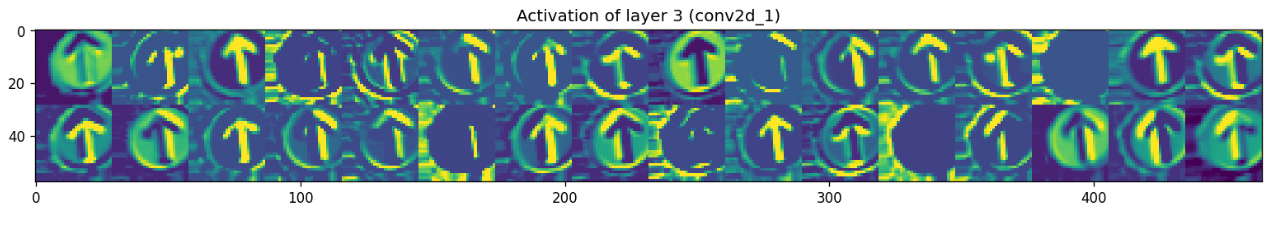


***Comparaison entre l’image d’entrée et l’activation du filtre***

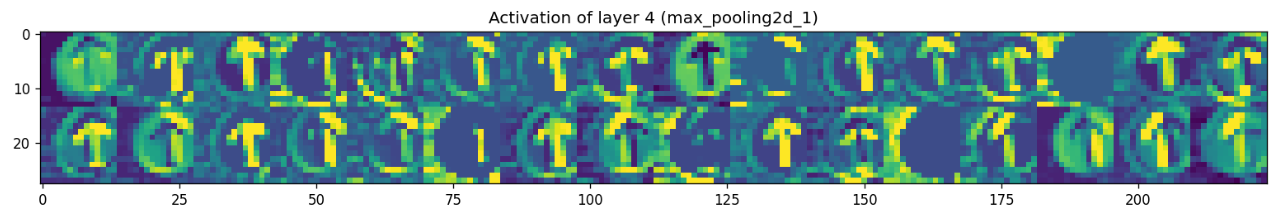


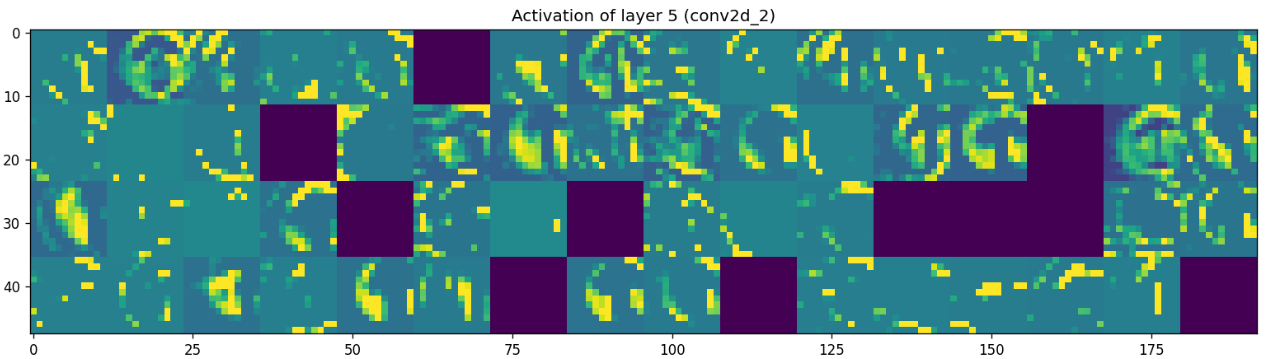
***Première et deuxième couche de convolution***

Les résultats montrent une progression dans le traitement des caractéristiques à travers les couches.   
**La première couche et la deuxième couche** détecte des caractéristiques avec des activations élevées**, La troisième couche** affine ces détections, et **la quatrième couche** de pooling réduit la dimensionnalité tout en conservant les informations essentielles.



***Troisième couche de convolution***



***Quatrième couche de convolution***

***Cinquième couche de convolution***

**Cette hiérarchie est typique dans les réseaux de neurones convolutifs et contribue à la capacité du modèle à extraire des caractéristiques pertinentes** pour la tâche de classification ou de détection.

**Cette structure permet de présenter les résultats de manière claire et de mettre en évidence l'évolution des activations à travers les différentes couches du réseau.**

# Conclusion

Le modèle **gagnant est celui avec la régularisation du CNN. On observe que la performance du modèle a un impact plus important sur sa qualité que sa complexité**.

Nous avons appris à analyser différents modèles tout en optimisant le temps d’exécution.  
Par exemple, il est possible d’optimiser la Data Augmentation afin de réduire la consommation de ressources.