



PROJET FINAL - TRAVAIL DE FIN DE SESSION :
CLASSIFICATION SUPERVISÉE D'IMAGES
DE PANNEAUX ROUTIERS ALLEMANDS

PAR

ALONSO CAROLINE, BAUDOUIN ROMAIN RÉGIS & VAN HYLCKAMA VLIEG
PEDRO
(CODE PERMANENT : ALOC18620100, BAUR25120100 & VANP15110101)

COMPTE RENDU RÉALISÉ DANS LE CADRE DU COURS :
FONDAMENTAUX DE L'APPRENTISSAGE AUTOMATIQUE
(8INF867)

QUÉBEC, CANADA

JUIN, 2024

Table des matières

Introduction.....	3
1 - Préparation des données.....	5
2 - Réduction de dimension.....	9
3 - Modèles choisis.....	10
4 - Evaluation des modèles via les métriques de validation adéquates.....	13
5 - Interface utilisateur.....	20
Conclusion.....	23
Bibliographie.....	24

Introduction

Notre projet consiste à faire de la classification supervisée d'images à savoir des panneaux de signalisations, plus précisément de panneaux allemands.

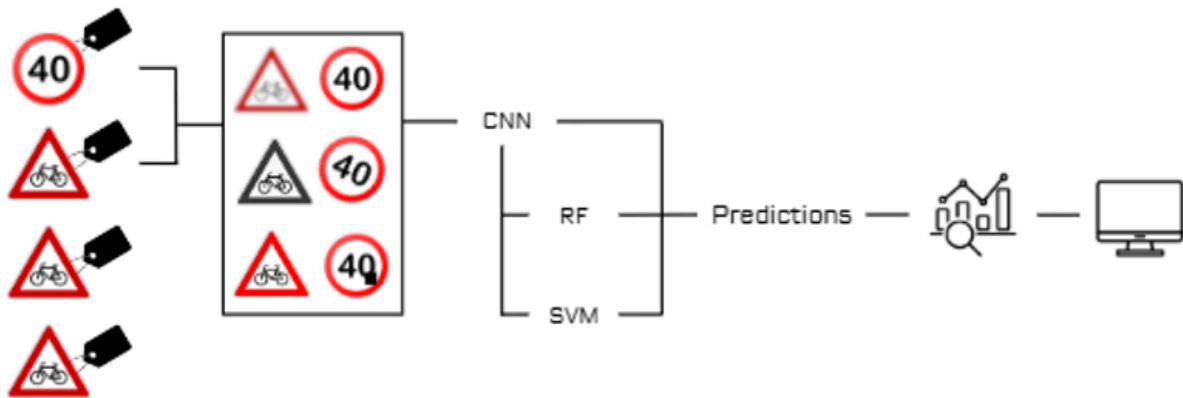
La classification des panneaux routiers est une composante essentielle pour améliorer la sécurité routière. En permettant aux systèmes de conduite autonome et d'aide à la conduite de reconnaître et de réagir adéquatement aux différents panneaux de signalisation, cela pourrait aider à la sécurité routière, réduire le nombre d'accidents et améliorer la fluidité du trafic.

En effet, les véhicules autonomes doivent être capables d'interpréter leur environnement, en l'occurrence l'environnement routier, en temps réel, de façon précise, afin de prendre des décisions appropriées, comme ralentir selon un panneau de limitation de vitesse ou s'arrêter à un panneau stop. Les systèmes de navigation GPS intégrés dans les véhicules modernes pourraient également bénéficier de la reconnaissance des panneaux routiers. En affichant les informations des panneaux directement sur l'écran de navigation ou en alertant le conducteur des changements de vitesse ou des zones de danger, ces systèmes peuvent améliorer la vigilance et la sécurité des conducteurs surtout étrangers.

Pour développer nos modèles de classification des panneaux routiers, nous utiliserons le German Traffic Sign Recognition Benchmark (GTSRB). Ce jeu de données, utilisé lors d'un challenge tenu à l'International Joint Conference on Neural Networks (IJCNN), est une référence dans le domaine de la reconnaissance des panneaux de signalisation et contient plus de 50000 images de panneaux, divisées en 43 classes distinctes. Ces images, prises dans des conditions réelles, variant notamment en luminosité et visibilité, fournissent un ensemble de données riche et diversifié pour l'entraînement et la validation de modèles de machine learning.

L'objectif principal de cette étude est de développer et d'évaluer des modèles de classification supervisée de panneaux routiers, nous n'aborderons donc pas la détection des panneaux (nous considérerons des images avec des panneaux centrées). Nous explorerons plusieurs approches, incluant des modèles de réseaux de neurones convolutifs (CNN), des forêts aléatoires (Random Forest), et des machines à vecteurs de support (SVM). Les performances de ces modèles seront comparées en termes notamment de précision, de rappel, ...

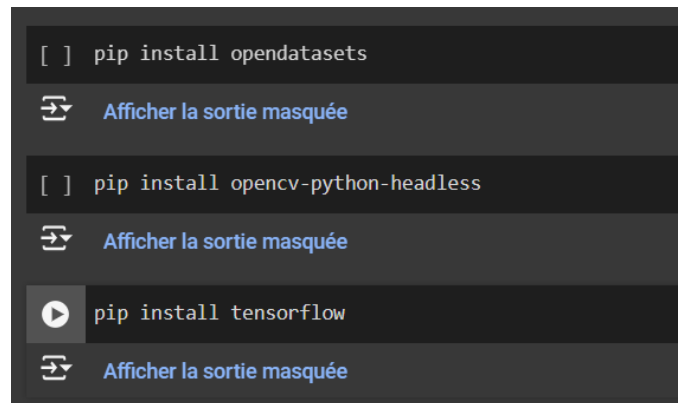
La structure de notre projet sera comme sur le schéma suivant :



Nous allons tout d'abord filtrer une partie des images (le dataset d'origine contenant plusieurs dizaines de milliers, cela rendrait nos modèles beaucoup trop gourmands en ressources et en temps). Le dataset réduit va ensuite être augmenté au travers de différentes transformations (qui seront détaillées dans la suite du rapport). Nous définirons ensuite nos 3 modèles, et les feront tourner afin d'avoir des prédictions. Nous évaluerons ensuite nos modèles via des métriques, une fois ces dernières jugées satisfaisantes nous exportons les modèles vers une application d'interface utilisant Tkinter.

1 - Préparation des données

Les packages opendatasets, opencv-python-headless et tensorflow ne sont pas inclus dans l'environnement d'exécution de Python sur Colab, il a donc fallu les ajouter :



```
[ ] pip install opendatasets
```

Afficher la sortie masquée

```
[ ] pip install opencv-python-headless
```

Afficher la sortie masquée

```
[ ] pip install tensorflow
```

Afficher la sortie masquée

Nous chargeons d'abord nos données à l'aide de la librairie pandas, nos données d'entraînement (Train) et de test (Test) se trouvent déjà dans des dossiers séparés.

```
# Dossiers racine contenant les images
rootImageDirTrain = 'gtsrb-german-traffic-sign/Train'
rootImageDirTest = 'gtsrb-german-traffic-sign/Test'

# Lecture des fichiers CSV
data_train = pd.read_csv("gtsrb-german-traffic-sign/Train.csv")
data_test = pd.read_csv("gtsrb-german-traffic-sign/Test.csv")
data_meta = pd.read_csv("gtsrb-german-traffic-sign/Meta.csv")
```

Par la suite, nous avons traité les images de la façon suivante :

- Nettoyage des données :
 - S'assurer qu'il n'y ait pas de données/étiquettes manquantes
 - S'assurer que toutes les étiquettes sont correctes
- Formatage des données :
 - Taille des images avec un resize de toutes les images en 50x50, effectué lors de la séparation des données entre entraînement et validation, ainsi que pour les données test.
- Équilibrage des classes (réduction et augmentation) :
 - Transformation photométrique (comme la saturation, couleur noir/blanc)
 - Transformation par occlusion (masquage aléatoire de certaines parties de l'image pour simuler une obstruction)
 - Transformation géométrique (rotation)

Nous vérifions qu'il n'y ait pas de données manquantes. Nous observons que le fichier Meta.csv contient une image non étiquetée. Nous choisissons de l'ignorer car nous n'utilisons pas le fichier Meta.csv dans la suite.

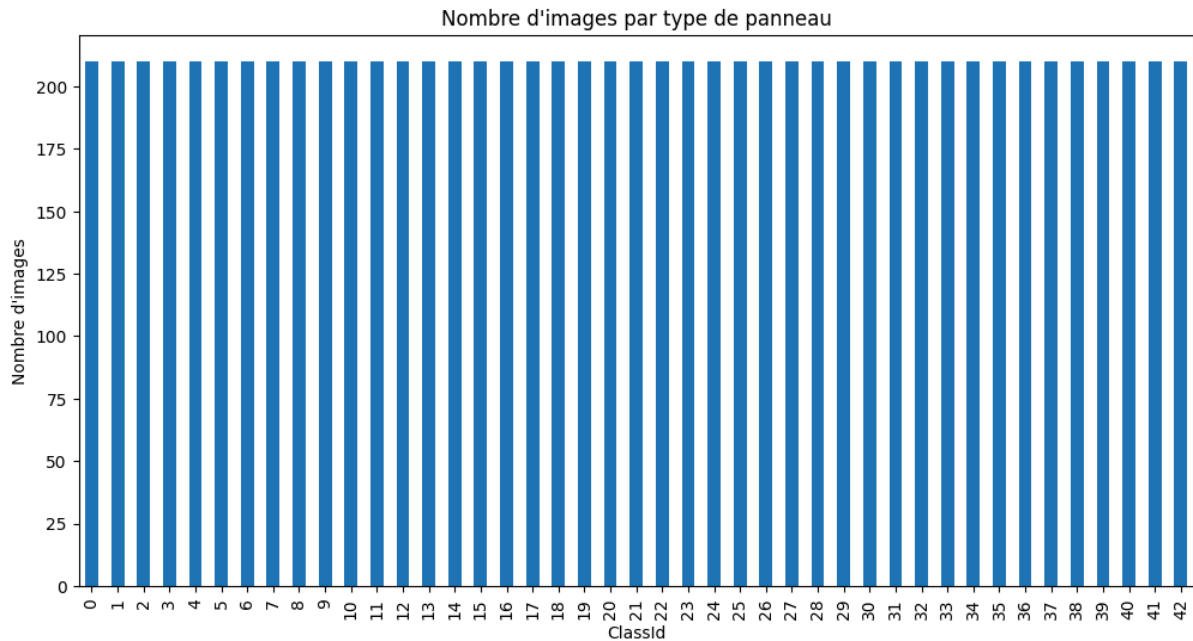
```
Width      False
Height     False
Roi.X1     False
Roi.Y1     False
Roi.X2     False
Roi.Y2     False
ClassId    False
Path       False
dtype: bool
Width      False
Height     False
Roi.X1     False
Roi.Y1     False
Roi.X2     False
Roi.Y2     False
ClassId    False
Path       False
dtype: bool
Path       False
ClassId    False
ShapeId    False
ColorId    False
SignId     True
dtype: bool
      Path  ClassId  ShapeId  ColorId  SignId
24  Meta/30.png      30        0        0      NaN
```

Par ailleurs, nous vérifions que toutes les images sont correctement étiquetées dans leur CSV respectif. Pour cela, nous vérifions que l'étiquette de l'image, qui contient ses informations de taille, de classe ou encore de chemin d'accès, correspond effectivement à l'emplacement de l'image et donc à sa classe.

```
Train :
Toutes les images dans les dossiers sont bien étiquetées.
Test :
Toutes les images dans les dossiers sont bien étiquetées.
```

Ensuite, pour l'équilibrage des classes, nous voulons que chaque classe ait le même nombre d'images. Pour cela, une réduction de certaines classes est nécessaire car certaines possèdent bien plus d'images que d'autres. Le nombre d'images par classe est défini par le nombre d'images contenus dans la plus petite classe (210 images).

Sur la page suivante nous pouvons voir le résultat obtenu après l'équilibrage des classes :



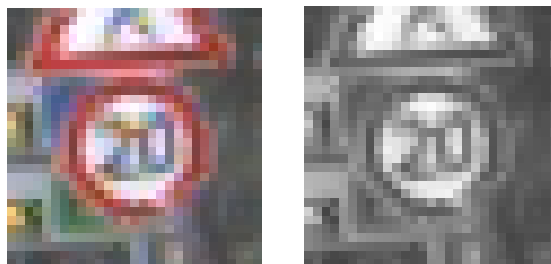
Nous avons souhaité entraîner nos modèles sur des images qui pourraient les mettre plus en difficulté dans des situations réelles (avec comme mentionné précédemment des transformations d'occlusion, de floutage, de rotation, ...).

Pour cela, nous avons ainsi augmenté notre dataset en appliquant au maximum une transformation par image.

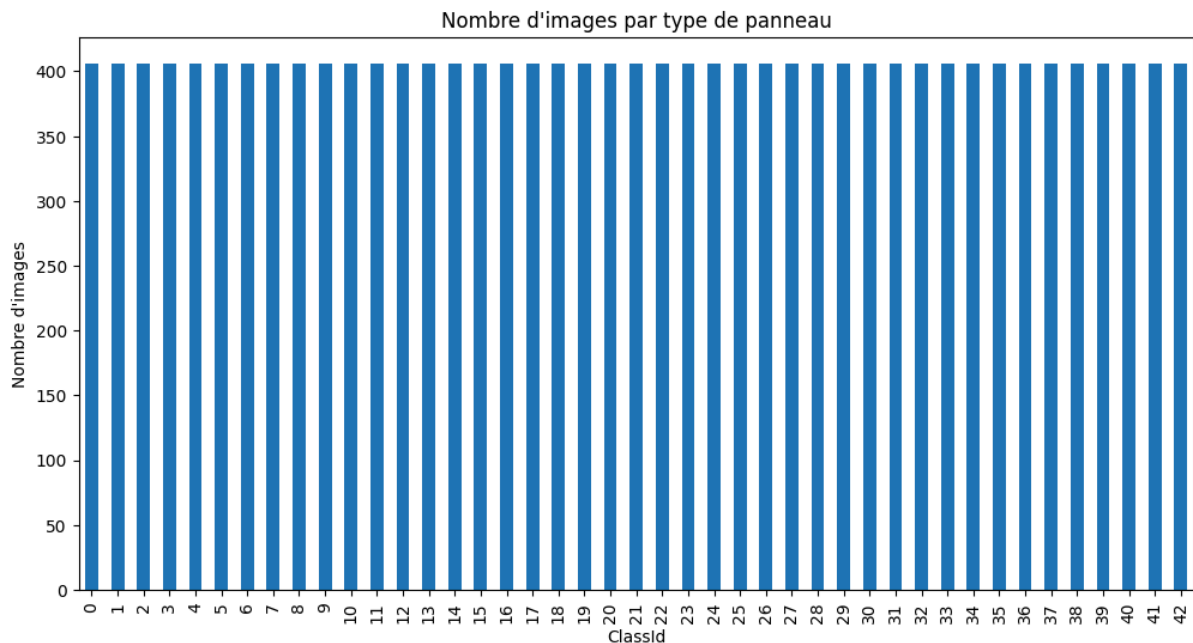
Chaque image subit une transformation parmi 15 options de transformations, certaines transformations étant des combinaisons de transformations abordées précédemment (par exemple, noir et blanc + flou forment une combinaison possible).

```
def transformClasses():
    taille = len(data_train)
    for i in range(0, taille):
        # Récupérer le chemin d'accès de l'image
        imagePath = 'gtsrb-german-traffic-sign/' + data_train.iloc[i]['Path']
        # Lui appliquer une transformation
        applytransform(imagePath, i, data_train.iloc[i]["ClassId"])
```

Voici un exemple de transformation d'une image en noir et blanc :



Nous vérifions à nouveau l'équilibrage des classes après transformations :



Nous avons bien ci-dessous 420 images pour chaque classe, ce qui correspond bien au fait d'ajouter une transformation pour chaque image déjà présente (soit faire doubler le nombre d'images que nous avions qui était de 210). De plus, cela nous permettra d'avoir suffisamment d'images pour nos modèles par la suite.

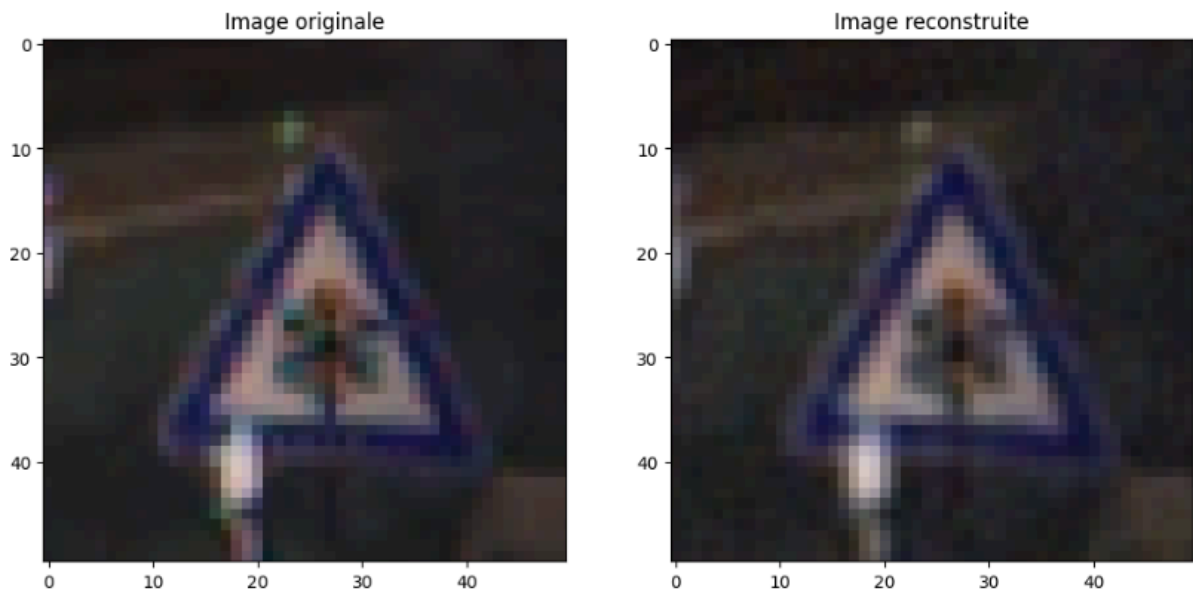
Enfin, nous créons nos données d'entraînement. Nous faisons un resize de toutes nos données d'entraînement, les images transformées et non transformées ont ainsi toutes la même taille. Nous divisons ensuite nos données entre données d'entraînement et données de validation avec un simple split. Voici un extrait du code de la fonction `create_train_data()`, qui montre l'étape où on prend l'image et où on la stocke dans un `numpy.array` :

```
# Récupération de l'image
image = cv2.imread("gtsrb-german-traffic-sign/" + path)
image_fromarray = Image.fromarray(image, 'RGB')
# Resize en 50x50
resize_image = image_fromarray.resize((50, 50))
image_data.append(np.array(resize_image))
image_labels.append(row["ClassId"])
```

Nos données d'entraînement et de tests étant déjà séparées dans des dossiers différents, nous avons donc séparé nos données d'entraînement et de validation tout aussi simplement, cela est rapide, adapté aux datasets volumineux ce qui correspond à notre cas et permettra de préserver nos ressources computationnelles (bien que d'autres choix soient possibles nous avons choisi de faire ce compromis).

2 - Réduction de dimension.

Pour la réduction de dimensions, nous avons choisi d'implémenter la PCA pour en observer les effets sur les performances de nos modèles. Notre classe PCAReductor (voir code source) implémente les différentes fonctionnalités liées à la PCA. Nous l'utilisons pour faire la réduction et elle permet aussi de reconstruire une image pour que l'on vérifie si la réduction s'est bien faite. Voici le résultat d'une reconstruction sur un panneau lorsque l'on réduit l'image à 1000 composantes :



Nous voyons que l'image de droite est bien reconstruite mais avec un certain "bruit".

Nous verrons dans la suite les effets que la PCA a sur les modèles.

Nous verrons aussi que pour avoir de meilleurs résultats nous utiliserons les features extractions réalisées à la suite de notre CNN, qui nous sert bien à réduire nos dimensions.

3 - Modèles choisis

Nous avons fait le choix d'implémenter les trois modèles suivants :

- CNN (Convolutional Neural Network)
- Random Forest
- SVM (Support Vector Machine)

Pour rappel :

- Les CNN capturent efficacement les relations non-linéaires complexes dans les images, ce qui est crucial pour distinguer nos 43 classes. Généralement, plusieurs couches de convolution sont utilisées, appliquant des filtres de convolution, suivies par des fonctions d'activation non linéaires (comme ReLU) et des opérations de pooling.
- Les Random Forests combinent plusieurs arbres décisionnels pour améliorer la robustesse et réduire le risque de surapprentissage, ce qui aide à gérer la diversité des panneaux routiers. Les Random Forests permettent d'identifier les caractéristiques importantes, ce qui peut être utile pour comprendre quelles caractéristiques des panneaux sont les plus discriminatives.
- Les SVM quant à eux trouvent l'hyperplan qui maximise la marge entre les différentes classes, ce qui est efficace même pour une classification multi-classes complexe comme celle des panneaux routiers.

Nous devons tout d'abord commencer par créer et mettre en forme nos données de tests (cf. fonction `createTestData` dans le colab). Celles-ci sont resize au moment de la création de l'array. Toutes les images de test du dataset de base sont utilisées. Cela veut dire que nos modèles auront des classes déséquilibrées sur lesquelles elles seront testées, mais c'est le choix que nous avons fait afin d'avoir le plus d'images de test possible sans avoir à effectuer de transformations dessus (hormis le redimensionnement).

Voici le code correspondant à notre CNN :

```
def create_simple_cnn(input_shape, num_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Nous pouvons voir que nous avons appliqué plusieurs couches, en effet cela permet au modèle d'être plus robuste aux variations dans les images d'entrée, c'est-à-dire qu'il permet au modèle de mieux généraliser aux nouvelles images qu'il n'a pas vues pendant l'entraînement. Ces couches permettent d'augmenter la capacité du modèle à capturer des relations complexes entre les pixels de l'image. Cela est crucial pour des tâches de classification plus difficiles où les différences entre les classes peuvent être subtiles.

Le CNN est le cœur de notre projet car nous utilisons par la suite une extraction de ses features en tant que réduction des dimensions afin d'entraîner les deux autres modèles, à savoir le SVM et la Random Forest. Voici comment se fait cette extraction :

```
layer_name = 'dense'
intermediate_layer_model = Model(inputs=cnn_model.input,
                                 outputs=cnn_model.get_layer(layer_name).output)
train_features = intermediate_layer_model.predict(X_train)
test_features = intermediate_layer_model.predict(X_test)
```

Nous n'utilisons pas la PCA pour la réduction des dimensions car nos résultats avec celle-ci étaient très mauvais. Voici les précisions obtenues sur SVM et la random Forest avec la PCA :

```
Random Forest Accuracy: 0.054077593032462394
```

```
SVM Accuracy: 0.0444972288202692
```

Voici maintenant les résultats obtenus après utilisation de la feature extraction sur le CNN :

```
Random Forest Accuracy: 0.9252573238321457
```

```
SVM Accuracy: 0.9201900237529691
```

Par ailleurs, voici les résultats obtenus sans réduction de dimension :

```
Random Forest Accuracy: 0.6971496437054632
```

```
SVM Accuracy: 0.7255789456124667
```

A noter, afin d'optimiser nos résultats avec le SVM et la Random Forest, nous avons utilisé une Grid Search pour Random Forest et un Random Search pour SVM. Voici les paramètres optimaux trouvés (pour les paramètres testés, se référer directement au code source) :

```
Optimized Random Forest Accuracy: 0.9252573238321457  
Best Parameters for Random Forest: {'max_depth': 20, 'max_features': 'log2', 'min_samples_split': 5, 'n_estimators': 300}
```

```
Optimized SVM Accuracy: 0.9247030878859858  
Best Parameters for SVM: {'kernel': 'linear', 'gamma': 1, 'C': 0.1}
```

Enfin, nous sauvegardons nos modèles pour pouvoir les réutiliser par la suite dans notre application de classification d'images.

4 - Evaluation des modèles via les métriques de validation adéquates.

Nous afficherons une matrice de confusion, qui est un outil visuel très répandu pour les classifications notamment lié au fait qu'elle est facilement lisible (le meilleur résultat correspondrait au fait d'avoir une matrice diagonale, avec tous les éléments, hors diagonale principale, étant des zéros).

Nous ferons aussi apparaître diverses métriques. Nous avons ainsi calculé et affiché la moyenne des métriques suivantes : accuracy, précision, recall/sensitivité, f1-score, spécificité, IoU et le ROC AUC (dont nous faisons aussi apparaître les courbes), et nous affichons également le nombre d'images de tests mal classifiées.

Pour rappel, chaque métrique donne une idée générale de la performance tel qui suit :

- L'accuracy mesure la proportion d'exemples correctement classifiés par rapport à l'ensemble des exemples (lorsque les classes sont équilibrées).
Accuracy = Nombre de prédictions correctes / Nombre total de prédictions
- La précision mesure la proportion des prédictions positives correctes parmi toutes les prédictions positives.
Précision = (Vrai positifs) / (Vrai positifs + Faux positifs)
- Le rappel, ou sensibilité, mesure la proportion des vrais positifs parmi tous les échantillons positifs.
Rappel = (Vrai positifs) / (Vrai positifs + Faux négatifs)
- Le F1-score est la moyenne harmonique de la précision et du rappel, et offre un équilibre entre les deux.
F1-score = (Précision × Rappel) / (Précision + Rappel)
- La spécificité mesure la proportion de vrais négatifs parmi tous les échantillons négatifs (vrai ou faux).
Spécificité = (Vrai négatifs) / (Vrai négatifs + Faux positifs)
- L'IoU mesure le chevauchement entre les cadres de délimitation prédits et de vérité terrain.
IoU = Intersection / Union
- Le ROC AUC (Receiver Operating Characteristic Area Under Curve) mesure la performance d'un modèle de classification à travers différents seuils de discrimination. La courbe ROC trace le taux de vrais positifs contre le taux de faux positifs.

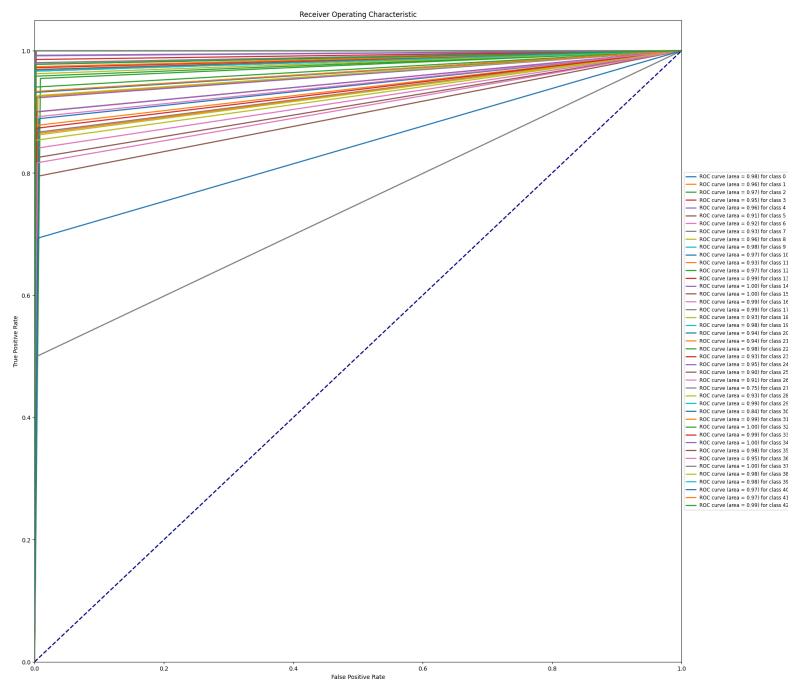
$$AUC = \int_0^1 ROC(t) dt$$

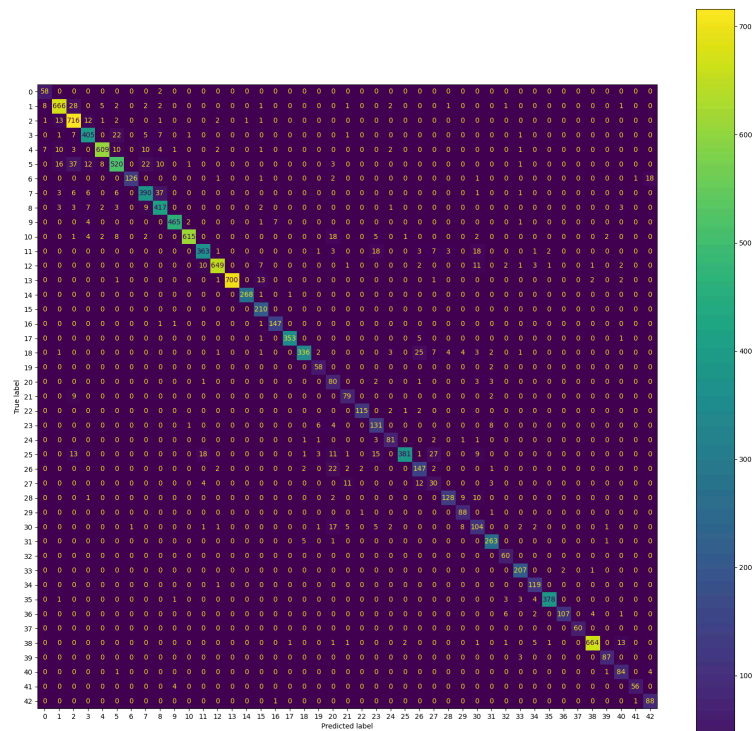
Pour toutes ces mesures, une valeur parfaite correspondrait à 1, pour le nombre d'images mal classifiées une valeur parfaite serait de 0.

Voici les résultats de nos différents modèles (pour une meilleure lisibilité des matrices de confusion et des courbes se référer au fichier 8INF867_Projet.ipynb dans le dossier zip, dans lequel est également le rapport fourni par la bibliothèque sklearn permettant d'avoir des détails de métriques pour chaque classe).

Performances du CNN :

```
Évaluation du modèle: CNN
Accuracy: 0.9191
Precision: 0.8818
Recall (Sensitivité): 0.9141
F1-score: 0.8940
Spécificité: 0.8818
IoU: 0.8216
ROC AUC: 0.9561
```

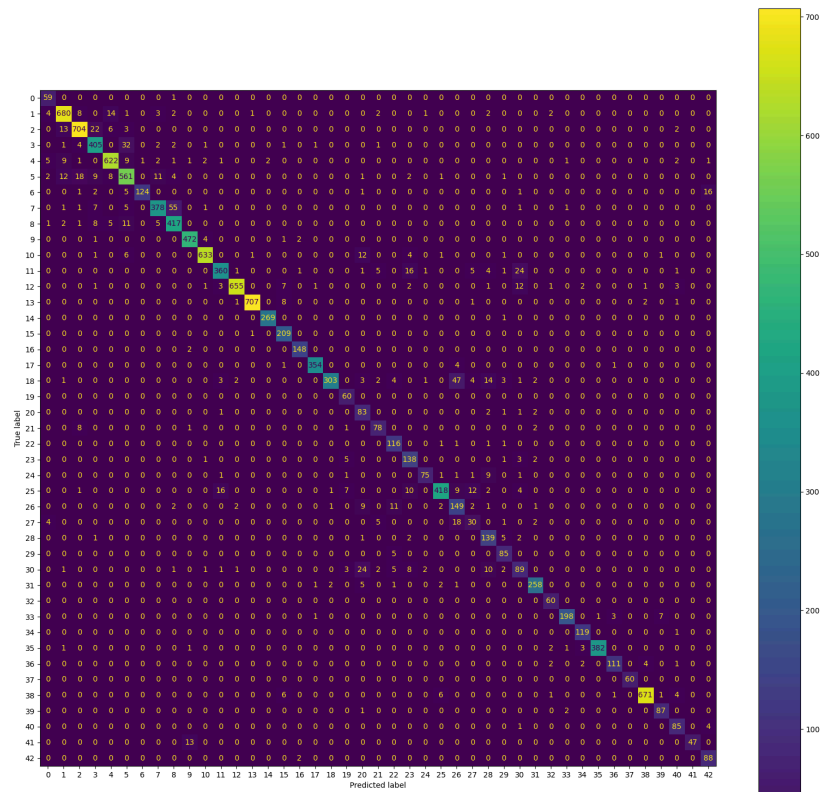
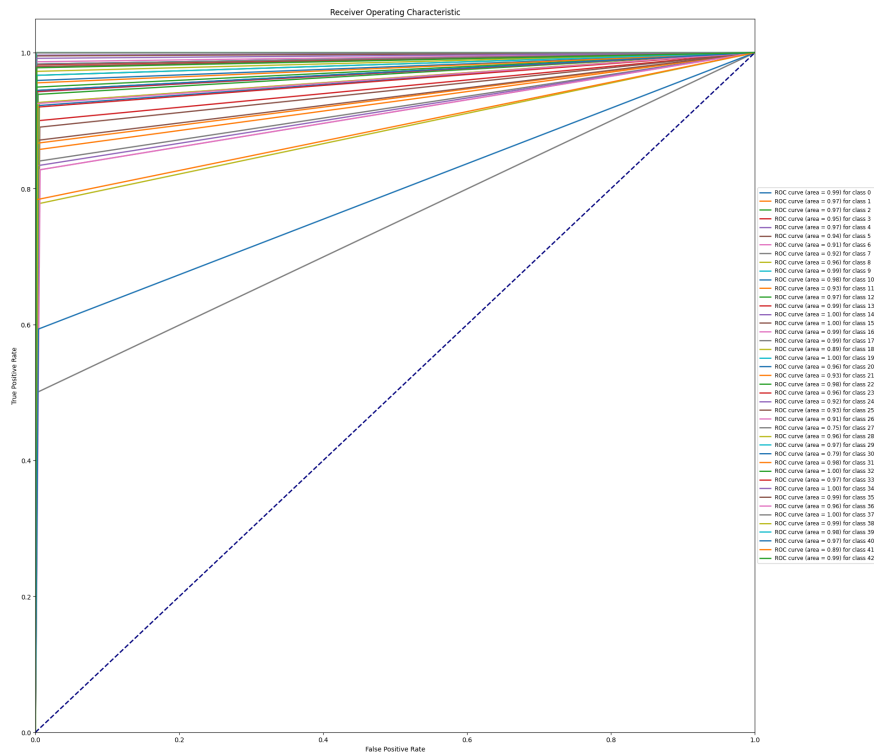




Misclassified images: 1022/12630

Performances du Random Forest :

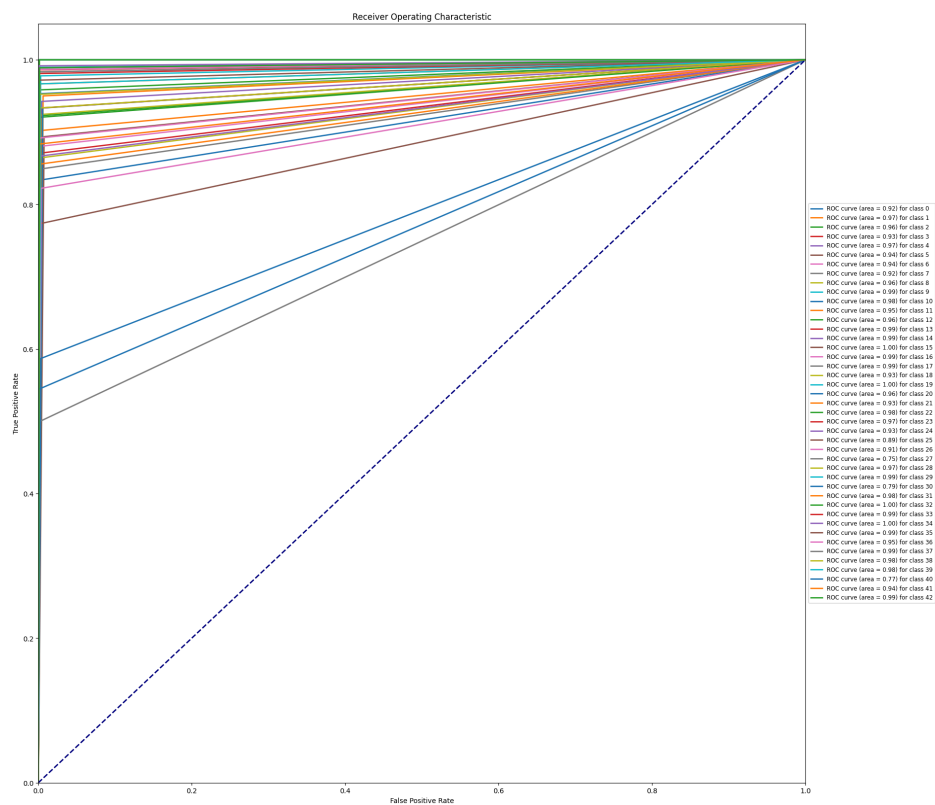
```
Évaluation du modèle: Random_Forest
Accuracy: 0.9253
Precision: 0.8890
Recall (Sensitivité): 0.9133
F1-score: 0.8973
Spécificité: 0.8890
IoU: 0.8257
ROC AUC: 0.9558
```

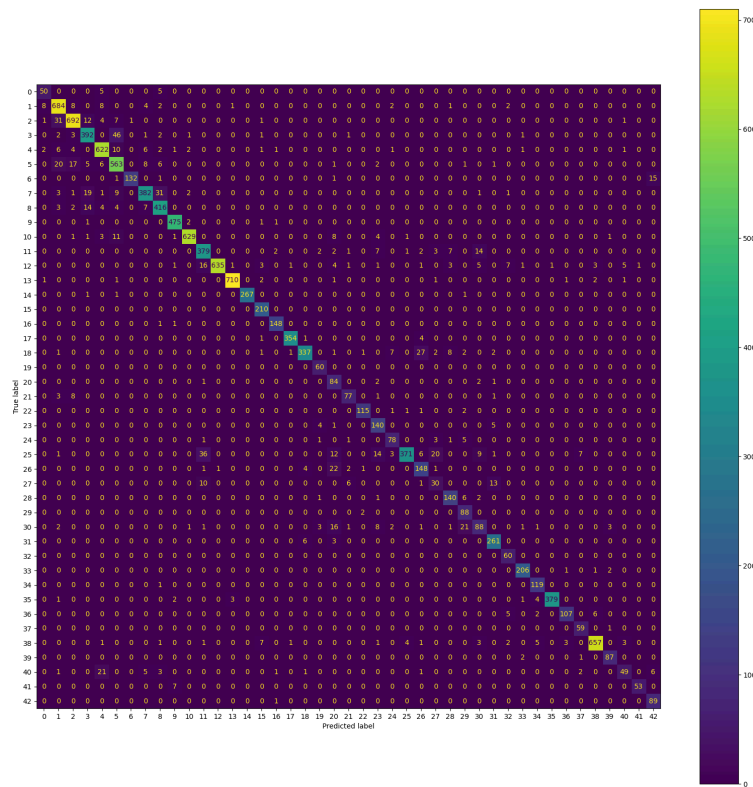


Misclassified images: 944/12630

Performances du SVM :

```
Évaluation du modèle: SVM
Accuracy: 0.9202
Precision: 0.8833
Recall (Sensitivité): 0.9047
F1-score: 0.8897
Spécificité: 0.8833
IoU: 0.8144
ROC AUC: 0.9514
```





Misclassified images: 1008/12630

Nous observons, à l'aide de la matrice de confusion, que les classes des panneaux indiquant la limite de vitesse sont ceux subissant le plus grand nombre d'erreurs de classement. En effet, ils sont souvent confondus entre eux. Ceci peut s'expliquer par leur ressemblance. Il est possible que la feature extraction n'arrive pas à extraire assez d'informations sur le contenu des panneaux, et se concentre davantage sur la forme de ceux-ci, ou encore que cela soit lié à la difficulté de reconnaître les chiffres spécifiquement sur les panneaux. Ce problème se retrouve sur les 3 modèles que nous avons utilisés et entraînés.

Pour ce qui est des courbes ROC, celles-ci se trouvent toutes au-dessus de la courbe $y=x$, ce qui montre un modèle plutôt précis (assez proche de 1), sauf pour certaines classes.

En effet :

- Pour la classe 27, plusieurs images contiennent une portion d'un autre panneau, ce qui peut entraîner une confusion pour les modèles
- Pour la classe 30, la plupart des images avec le flocon sont floutées par défaut, ou bien de mauvaise qualité.
- Pour la classe 40, plusieurs panneaux de rond-point sont tagués ou altérés sur les images ce qui explique que le modèle a plus de mal à s'entraîner, comprendre, et généraliser dessus.

Enfin, pour l'ensemble de nos modèles, les métriques obtenues peuvent être considérées comme très bonnes à chaque fois au-dessus de 80% (sachant que pour chaque métrique évaluée le meilleur résultat correspond à 100%). Le modèle Random_Forest avec features extraction semble néanmoins légèrement meilleur que le modèle CNN et le modèle SVM avec features extraction pour la détection de panneaux routiers sur des images. Il offre le meilleur équilibre entre précision, rappel, F1-score, spécificité et IoU, tout en ayant une courbe ROC AUC moyenne élevée, et le plus petit nombre d'images mal classifiées.

5 - Interface utilisateur

Notre interface a été réalisée sur Python, grâce à la librairie Tkinter.

Le code de l'interface est disponible dans le dossier zip, le fichier commenté correspondant s'intitule : interface.py.

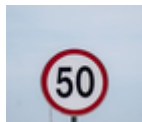
Il suffit de télécharger le dossier et de l'extraire, nous avons utilisé VSCode comme IDE (pour le lancer il faut veiller à avoir la bonne version des bibliothèques, notamment tensorflow, sinon cela ne fonctionnera pas avec nos modèles).

A noter que les modèles utilisés dans l'interface correspondent à ceux avec l'extraction de feature suite à la cnn, il s'agit des modèles téléchargés, récupérés ensuite via la librairie joblib pour les fichiers de format .pkl.

Il n'est possible que de tester une image à la fois, le résultat est affiché sous forme de probabilité.

Pour tester les modèles quelques règles vis-à-vis des images à mettre sont à respecter afin que le résultat soit plus ou moins correct :

- Mettre uniquement des panneaux de classes reconnues
- Mettre uniquement des panneaux de format Portable Network Graphics (.png) ou Joint Photographic Experts Group (.jpeg ou .jpg), à noter cependant que certains outils de conversion de format d'image peuvent compresser certaines données qui pourraient être essentielles à la reconnaissance du panneau, ce qui pourrait alors expliquer parfois de mauvais résultats.
- Centrer les panneaux dans les images (nos modèles ne servant que de classification et non pas de détection), par exemple pas :



- Ne mettre qu'un seul panneau à la fois, par exemple pas :



- Mettre des images réelles (les modèles ont été entraînés avec des photographies de panneau et non pas avec des logos), par exemple pas :



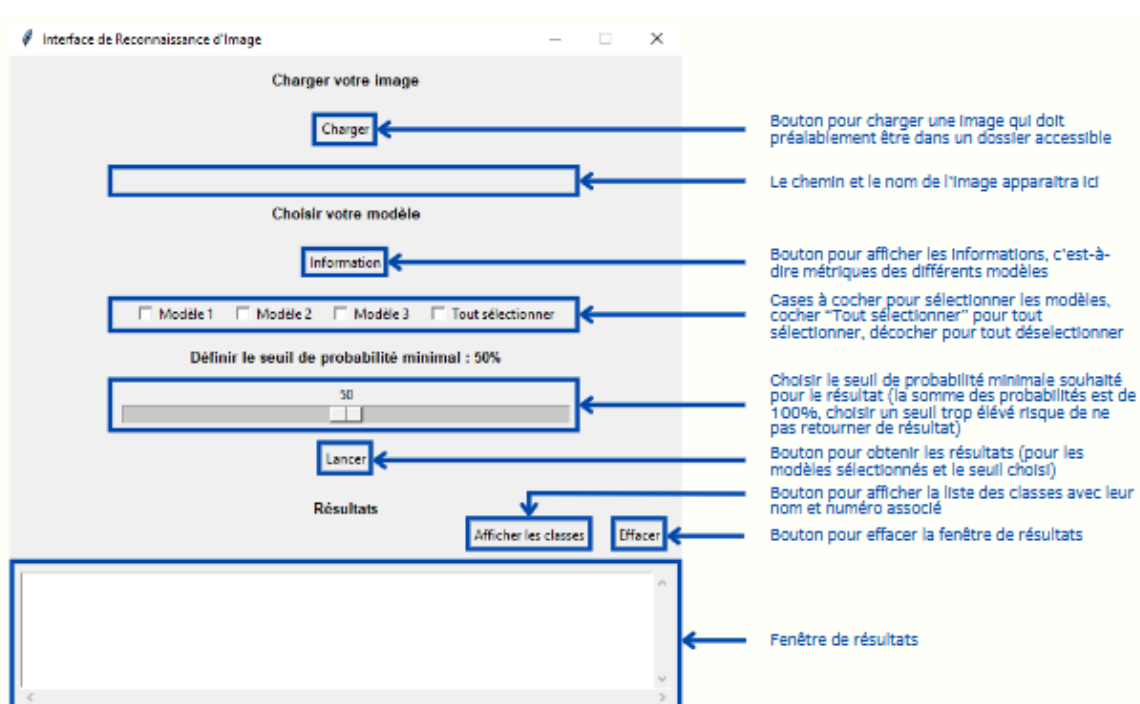
- Préférer mettre des panneaux de face (cela est une simple recommandation), par exemple pas :



- N'importe quelle taille est admissible du moment que l'image est plus ou moins carrée :



Voici les légendes de notre interface :



Ainsi, le téléversement de l'image à tester se fait via le bouton "Charger", les résultats pour les modèles sélectionnés sont affichés dans la fenêtre du bas avec l'affichage des probabilités, les informations des métriques de nos modèles sont accessibles via le bouton "Information", et les noms des classes sont rappelés via le bouton "Afficher les classes".

Voici un exemple d'utilisation, nous prenons l'image suivante intitulé 20_classe_0.png (étant donné qu'il s'agit d'un panneau de limitation de vitesse pour 20km/h, correspondant à la classe 0, cf. collab pour le détail des noms des classes) :



Interface de Reconnaissance d'Image

Charger votre image

Charger

Image chargée: C:/Users/pedro/Downloads/Images tests interfaces/20_classe_0.png

Choisir votre modèle

Information

☒ Modèle 1 ☒ Modèle 2 ☒ Modèle 3 ☒ Tout sélectionner

Définir le seuil de probabilité minimal : 25%

25

Lancer

Résultats

Afficher les classes Effacer

```
Résultats pour l'image : 20_classe_0.png
Modèle 1 : Résultat 0 - Détails : 0 (100.00%)
Modèle 2 : Résultat 0 - Détails : 0 (54.16%)
Modèle 3 : Résultat 0 - Détails : 0 (99.18%)
Conclusion : Le numéro le plus probable sur les différents modèles est
-----
%)
r les différents modèles est 0 avec une probabilité moyenne de 84.44%.
```

Pour cet exemple, chacun de nos modèles arrivent à trouver la classe 0, avec une probabilité de 100%, 54.16% et 99.18% respectivement pour le CNN, le Random Forest et le SVM. Nous affichons aussi une probabilité moyenne de 84.44%, ce que nous considérons comme très satisfaisant.

Conclusion

Ce projet nous a permis d'aborder la classification d'images avec un apprentissage supervisé. Nous avons appris à faire davantage de traitement de données, ce que nous n'avions pas énormément fait lors du TP hormis le fait de charger des données ou vérifier des données manquantes. Nous avons pu appliquer des concepts vus en cours, comme la PCA par exemple, et en observer les effets sur nos modèles.

Les trois modèles que nous avons entraîné ont des métriques que nous avons jugées acceptables et notre application d'interface permet donc à un utilisateur de tester les modèles sur des images de son choix (dans les limites que nous avons défini). En revanche, nous avons bien conscience que les capacités de nos modèles sont insuffisantes pour des applications réelles dans des domaines comme la conduite autonome, pour cela il faudrait notamment entraîner nos modèles sur davantage d'images (par exemple l'ensemble du dataset à disposition) en ayant les ressources adéquates.

Finalement, ce projet nous a beaucoup apporté que ce soit sur le plan professionnel que sur nos connaissances globales en intelligence artificielle. En effet, la nécessité de diviser nos tâches pour avancer ensemble sur ce projet nous a permis de gagner en expérience sur le travail d'équipe, et l'utilisation de l'outil Colab ainsi que des modèles de classification d'images nous ont apportés de nouvelles connaissances qui seront sans aucun doute utiles à notre futur travail en entreprise, surtout aux vu de l'intérêt grandissant pour l'IA dans de multiples domaines.

Bibliographie

- Mykola. (2018). GTSRB - German Traffic Sign Recognition Benchmark.
<https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>
- Sharma, S. (2021). GTSRB - CNN (98% Test Accuracy).
<https://www.kaggle.com/code/shivank856/gtsrb-cnn-98-test-accuracy>
- Gupta, Y. (2020). Traffic Sign Classification.
<https://www.kaggle.com/code/yashguptaab99/traffic-sign-classfication>
- Hamish. (2018). Preprocessing images with dimensionality reduction.
<https://www.kaggle.com/code/hamishdickson/preprocessing-images-with-dimensionality-reduction>
- Brownlee, J. (2020, 28 Août). Save and Load Machine Learning Models in Python with scikit-learn. Machine Learning Mastery.
<https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/>