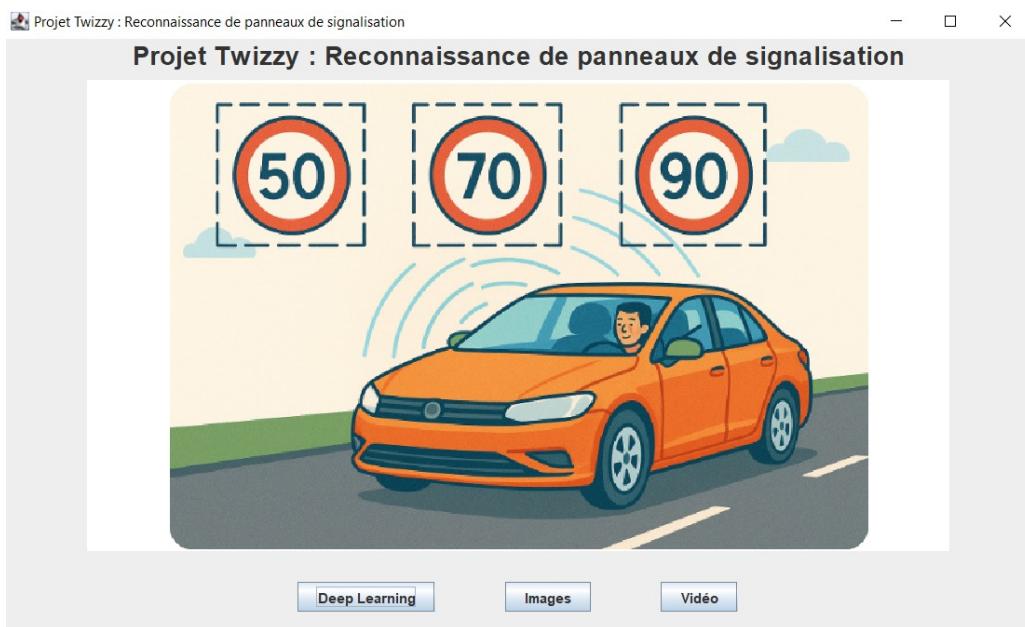


Projet Twizzy : Détection de panneaux de signalisation

Rapport Final



Réalisé par :

Emna DEBBECH – Imane BOUGHELEM
Haytam EL OUAGARI – Mohamed EL ARCHAoui

Groupe : 2A ISN, groupe SA

Année universitaire : 2024–2025

Table des matières

1	Introduction	2
1.1	Problématique	2
1.2	Github	2
2	Présentation du projet	3
2.1	Objectifs	3
2.2	Diagramme de cas d'utilisation	3
2.3	Diagramme d'activité	4
3	Traitement d'Image	6
3.1	Acquisition et Prétraitement de l'Image	6
3.2	Seuillage	6
3.3	Détection des Contours et des Formes	7
3.4	Reconnaissance des Panneaux	7
3.5	Affichage des Résultats et Analyse	7
3.6	Tests	8
4	Interface	9
4.1	Interface graphique Java (Swing)	9
5	Deep Learning	11
5.1	Base de données : German Traffic Sign Recognition Benchmark (GTSRB)	11
5.2	Description de l'architecture CNN (ResNet) utilisée	11
5.3	Résultats de l'apprentissage du modèle CNN	13
5.4	Interface	14
5.5	Tentative d'une application Web	15
6	Perspectives et problèmes rencontrés	17
7	Annexe	18

1 Introduction

La détection des panneaux de signalisation routière est un élément essentiel pour le déplacement sécurisé des véhicules autonomes. Cette technologie permet d'intégrer des informations en temps réel sur les limitations de vitesse et autres signalisations, optimisant ainsi la conduite automatique et assistée. Dans ce projet, nous avons pour objectif de concevoir une solution basée sur OpenCV et un noyau logiciel en Java afin d'identifier et de reconnaître automatiquement les panneaux de signalisation. Ce rapport détaille l'avancement du projet en mettant en avant les choix techniques, les méthodes de traitement d'image et les aspects liés au développement logiciel.

1.1 Problématique :

Les systèmes d'aide à la conduite des véhicules modernes s'appuient sur des capteurs et des algorithmes avancés pour détecter les panneaux routiers et ajuster les comportements de conduite en conséquence. L'un des défis majeurs est de garantir la fiabilité de la reconnaissance, malgré les variations de luminosité, les conditions météorologiques et les obstructions partielles des panneaux. Une mauvaise détection pourrait entraîner des erreurs d'interprétation critiques, compromettant ainsi la sécurité des usagers de la route.

Ainsi, notre problématique est la suivante : *Comment concevoir un système de reconnaissance de panneaux de signalisation fiable et rapide en utilisant OpenCV et Java, tout en assurant une précision élevée face aux différentes conditions environnementales ?*

1.2 GitHub

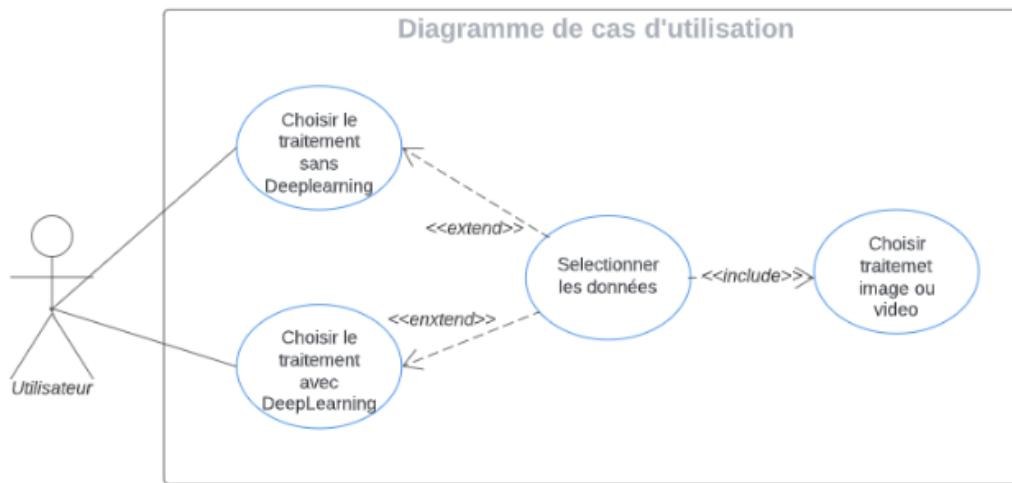
Voici le lien du projet Twizzy sur GitHub : <git@github.com:deb18e/Twizzy.git>

2 Présentation du projet

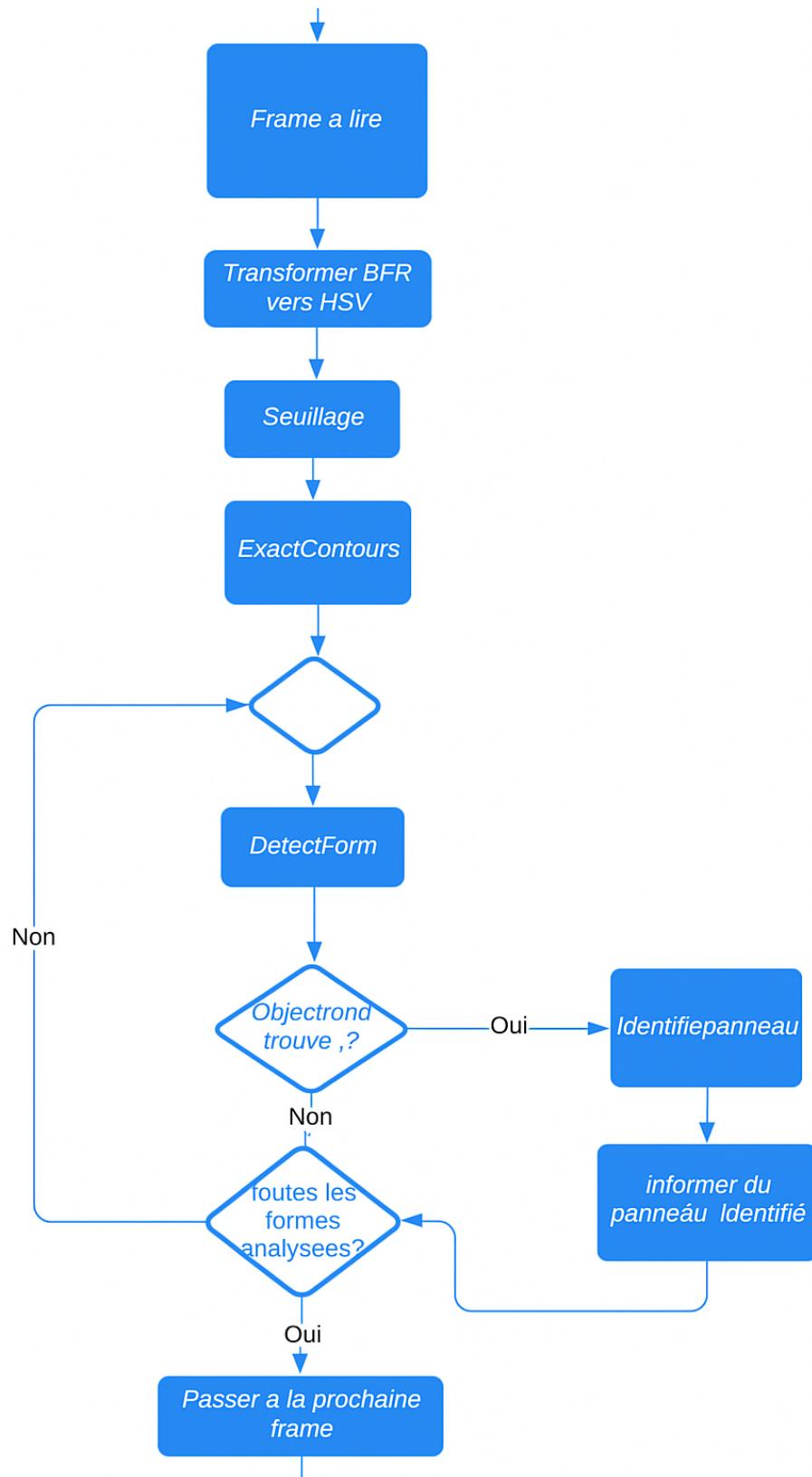
2.1 Objectifs

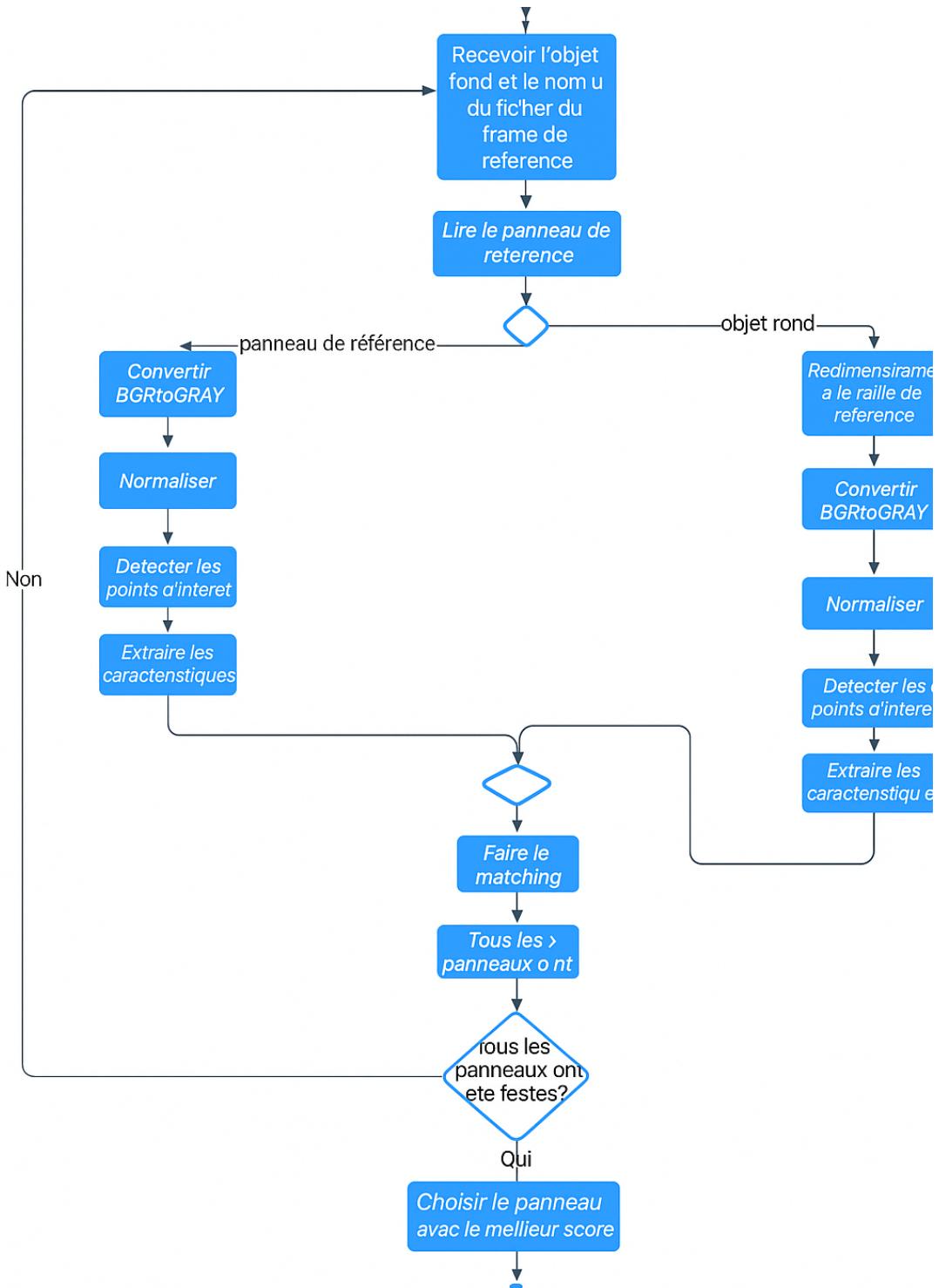
- développer des algorithmes avec OpenCV qui servent à la détection et la reconnaissance des panneaux de signalisation
- réaliser la détection sur des images et un flux vidéo
- développer un modèle de réseau de neurones convolutifs

2.2 Diagramme de cas d'utilisation



2.3 Diagramme d'activité





3 Traitement d'Image

Le traitement d'image repose sur OpenCV et comprend plusieurs étapes essentielles pour détecter et reconnaître les panneaux de signalisation :

3.1 Acquisition et Prétraitement de l'Image

- Conversion des images en espace de couleur HSV pour améliorer la segmentation des couleurs.
- Application d'un filtre de seuillage pour isoler les couleurs des panneaux.
- Lissage des images à l'aide de filtres gaussiens pour réduire le bruit.



(a) Image originale



(b) Prétraitement de l'image

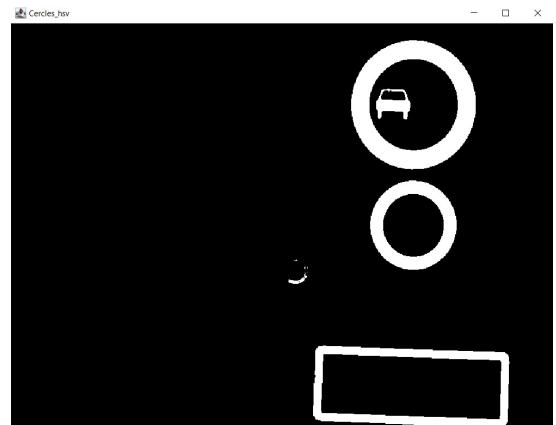
FIGURE 1 – Deux images côté à côté

3.2 Seuillage

Ceci consiste en la binarisation de l'image en attribuant la couleur blanche aux pixels qui sont rouges, et la couleur noire pour tous les autres pixels. Cette opération est effectuée en comparant chaque pixel à des valeurs seuils de référence pour la couleur rouge, qui sont fournies en entrée de la fonction de seuillage.



(a) Image originale



(b) Image après Seuillage

FIGURE 2 – Seuillage

3.3 Détection des Contours et des Formes

- Utilisation de l'algorithme de Canny pour détecter les contours des objets dans l'image.
- Extraction des formes géométriques, cercle dans ce cas.
- Validation des formes détectées en fonction de leur correspondance avec les panneaux de signalisation.



(a) Image originale



(b) anneaux détectés

FIGURE 3 – Reconnaissance de panneaux

3.4 Reconnaissance des Panneaux

- Extraction des descripteurs des panneaux en utilisant l'algorithme ORB.
- Comparaison des descripteurs avec une base de données de panneaux de référence.
- Attribution d'un score de similarité pour confirmer la reconnaissance du panneau.



(a) Image 1



(b) Image 2

FIGURE 4 – Deux images côte à côté

3.5 Affichage des Résultats et Analyse

- Le panneau correspondant à l'image original est celui avec le plus de similitude avec les panneaux qu'on a mis.

- Affichage des informations relatives aux panneaux identifiés (limitation de vitesse, interdictions, obligations, etc.).
- Stockage des résultats dans une base de données pour analyse ultérieure.

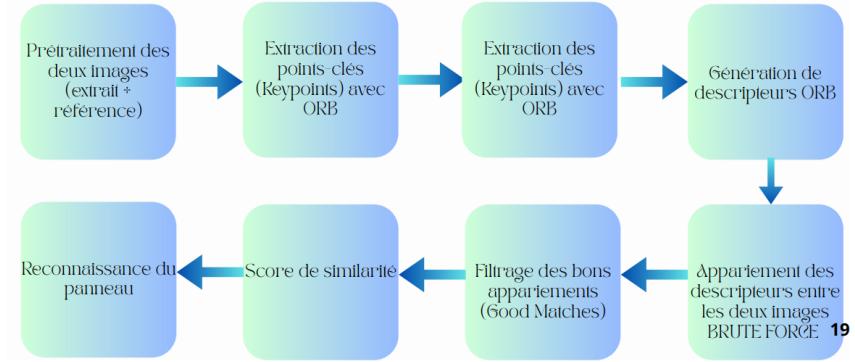


FIGURE 5 – Configuration finale

3.6 Tests

Des tests ont été réalisés sur plusieurs images pour évaluer l’efficacité des algorithmes de détection et de reconnaissance. Les résultats montrent une bonne précision pour l’extraction des panneaux rouges et circulaires, bien que des améliorations soient nécessaires pour détecter les panneaux partiellement occultés ou dans des conditions d’éclairage difficiles.

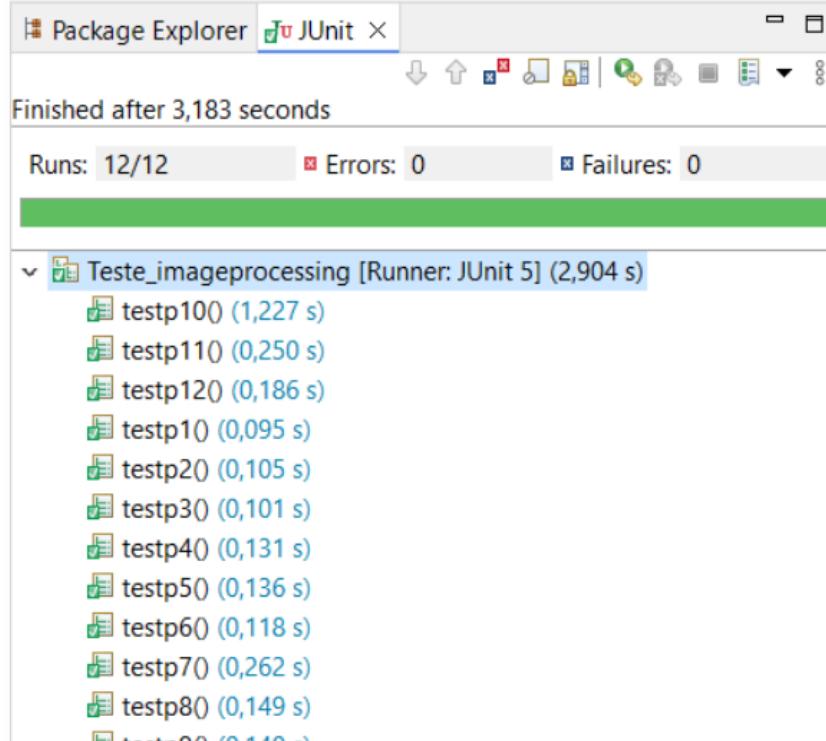


FIGURE 6 – Tests Unitaires

4 Interface

4.1 Interface graphique Java (Swing)

L'interface utilisateur du projet a été réalisée en **Java** à l'aide de la bibliothèque **Swing**. Elle vise à rendre l'utilisation de l'application accessible et ergonomique, avec une navigation simple entre les différents modes de traitement.

Page d'accueil La page d'accueil comporte trois boutons principaux :

- **Images** : permet à l'utilisateur de sélectionner une image depuis le répertoire du projet. Celle-ci est ensuite traitée par le noyau de détection afin d'identifier un éventuel panneau de signalisation.
- **Vidéo** : exécute directement la détection de panneaux sur une vidéo prédéfinie (par exemple `video2.avi`). Chaque image extraite est traitée en temps réel et le panneau détecté est affiché dynamiquement.
- **Deep Learning** : bouton prévu pour lancer une interface Python dans le cas d'une extension future par apprentissage profond (non implémentée dans la version actuelle).

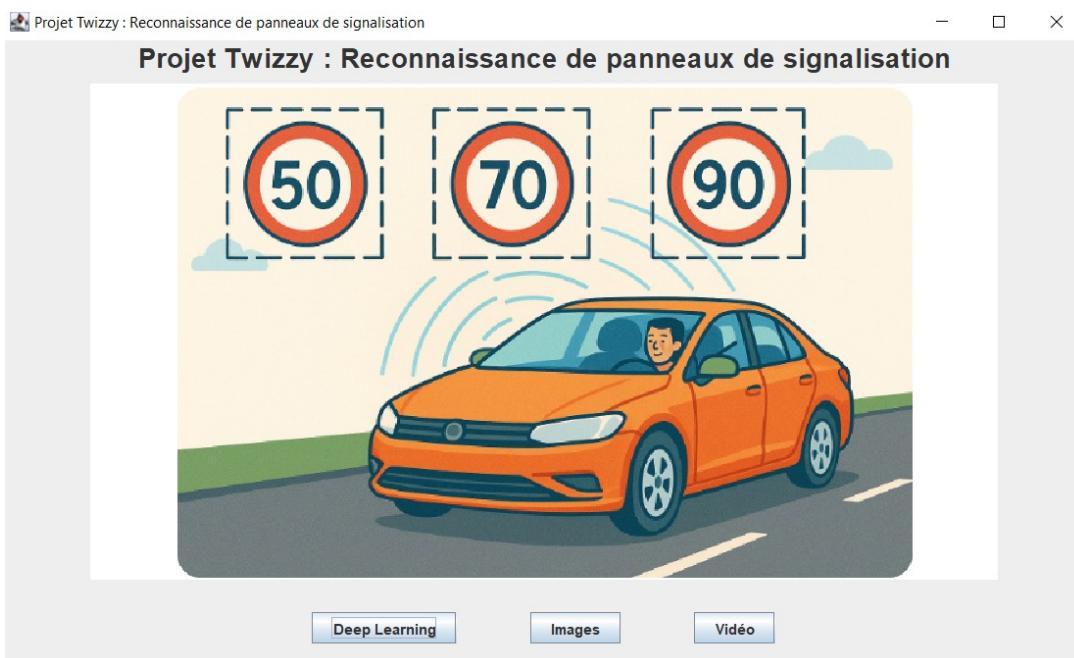


FIGURE 7 – Interface principale

Interface image Lorsqu'une image est sélectionnée, une nouvelle fenêtre s'ouvre et affiche :

- à gauche : le panneau détecté, redimensionné en 200×200 pixels ;
- à droite : l'image d'origine analysée, redimensionnée pour un affichage optimal.



FIGURE 8 – Interface Images : Exemple

Interface vidéo En mode vidéo, chaque image (frame) est analysée :

- le panneau détecté est mis à jour à gauche ;
- la vidéo défile à droite à un rythme d'environ 30 images par seconde.

La classe `VideoResultFrame.java` implémente cette logique en s'inspirant de `LectureVideo.java`, avec intégration directe du traitement d'image via `MainTraitementImage.processImage()`.

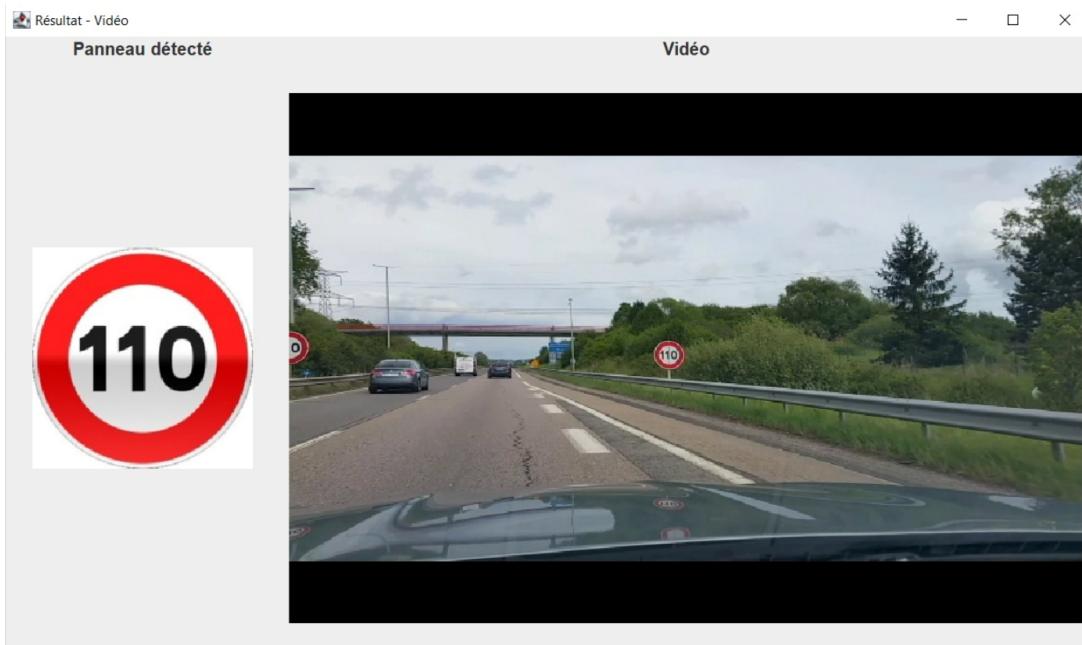


FIGURE 9 – Interface principale

Utilisation L'utilisateur n'a qu'à lancer le programme principal via la classe `HomePage`. Depuis cette interface, il peut directement interagir avec les différentes fonctionnalités du projet.

5 Deep Learning

5.1 Base de données : German Traffic Sign Recognition Benchmark (GTSRB)

Pour la partie Deep Learning du projet Twizzy, nous avons choisi d'utiliser la base de données **GTSRB (German Traffic Sign Recognition Benchmark)**. Il s'agit d'un benchmark public largement reconnu, introduit à l'occasion de la conférence *IJCNN 2011* dans le cadre d'un challenge de classification de panneaux routiers.

Présentation La base GTSRB est spécialement conçue pour la reconnaissance de panneaux de signalisation. Elle présente les caractéristiques suivantes :

- Plus de **50 000 images** annotées ;
- Plus de **40 classes** de panneaux différents (limitation de vitesse, interdictions, obligations, etc.) ;
- Une structure orientée **classification multi-classes sur image unique** ;
- Des images **réalistes** capturées dans des conditions de circulation variées (lumière, angle, distance).

Pourquoi ce choix ? Le choix de GTSRB s'explique par sa pertinence directe avec notre objectif de reconnaissance automatique de panneaux à partir d'images issues d'un flux vidéo. C'est une base riche, variée, bien annotée, et compatible avec des architectures de **réseaux de neurones convolutifs (CNN)**. Elle permet ainsi d'entraîner un modèle robuste sans nécessiter d'annotations manuelles supplémentaires.

5.2 Description de l'architecture CNN (ResNet) utilisée

L'architecture utilisée repose sur le principe du réseau neuronal convolutif résiduel, dit ResNet (Residual Network). Ce type d'architecture a été proposé pour pallier le problème de dégradation de performance rencontré lors de l'apprentissage de réseaux très profonds. L'idée fondamentale de ResNet est d'introduire des connexions directes appelées *skip connections* ou *connexions résiduelles*. Voici la structure détaillée de ce réseau :

1. **Entrée** : Le réseau prend en entrée une image RGB de taille $3 \times 224 \times 224$ pixels.
2. **Première couche convolutionnelle** :
 - Une convolution 7×7 avec 64 filtres, un stride de 2 et un padding adapté.
 - Cette couche permet de capturer des motifs de bas niveau (bords, textures) tout en réduisant la taille spatiale de l'image.
3. **MaxPooling** :
 - Une opération de max-pooling 3×3 avec un stride de 2.
 - Elle diminue encore la résolution spatiale tout en gardant les caractéristiques les plus saillantes.
4. **Blocs résiduels (Residual Blocks)** : Le réseau est composé de quatre grands blocs successifs, chacun contenant plusieurs *BasicBlocks*.
 - (a) **Bloc 1** :
 - Contient N_1 blocs résiduels avec des convolutions 3×3 .
 - Le nombre de canaux est 64.

(b) **Bloc 2 :**

- Contient N_2 blocs avec 128 canaux.
- Le premier bloc effectue un downsampling avec un stride de 2.

(c) **Bloc 3 :**

- Contient N_3 blocs avec 256 canaux.
- Le premier bloc réduit la taille spatiale avec un stride de 2.

(d) **Bloc 4 :**

- Contient N_4 blocs avec 512 canaux.
- Comme les précédents, le premier bloc applique un downsampling.

5. **Fonctionnement des blocs résiduels :**

- Chaque bloc contient deux couches convolutionnelles 3×3 , suivies de Batch Normalization et ReLU.
- Une connexion directe additionne l'entrée du bloc à sa sortie (*skip connection*), permettant une meilleure transmission du gradient lors de l'apprentissage.
- Si les dimensions ne correspondent pas (ex. à cause d'un changement de stride), une convolution 1×1 est utilisée dans le *shortcut*.

6. **Pooling global :**

- Un *adaptive average pooling* est utilisé pour transformer chaque carte de caractéristiques en une seule valeur par canal, produisant un vecteur de taille 512.

7. **Couche entièrement connectée (FC) :**

- Une couche linéaire (*fully connected*) qui mappe le vecteur de taille 512 vers un vecteur de sortie de taille 8.
- Cette couche donne les logits pour les 8 classes de classification.

8. **Sortie :**

- Une fonction *softmax* est appliquée pour convertir les logits en probabilités.
- L'indice avec la probabilité maximale correspond à la classe prédictive.

Cette structure permet un apprentissage efficace, même avec une grande profondeur, grâce aux connexions résiduelles qui facilitent la propagation des gradients. Elle est particulièrement robuste face aux problèmes de vanishing gradients et offre de très bonnes performances en classification d'images complexes.

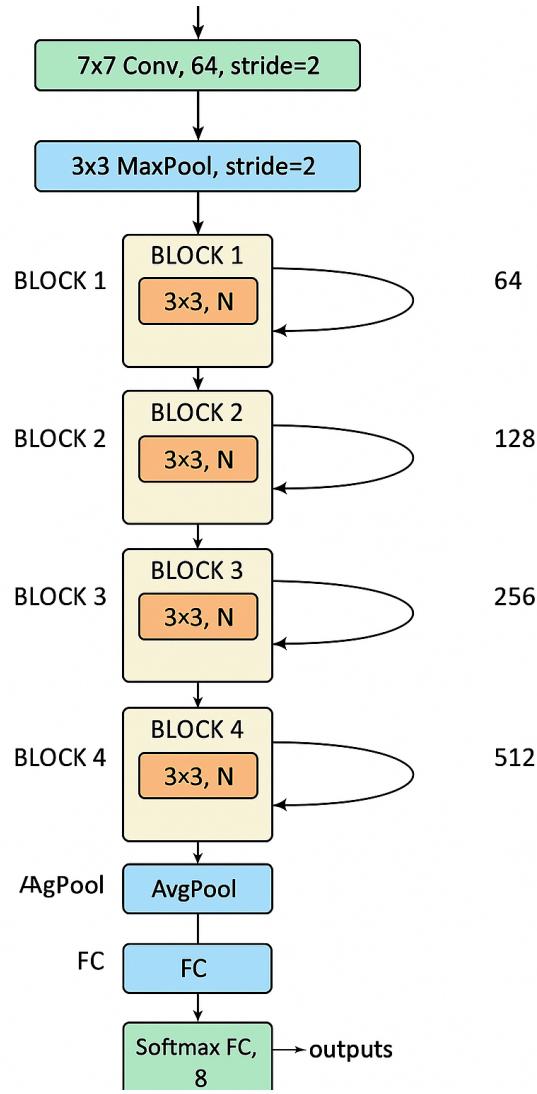


FIGURE 10 – Schème de l'architecture du CNN

5.3 Résultats de l'apprentissage du modèle CNN

L'architecture de réseau de neurones convolutifs (CNN) implémentée a été entraînée sur la base de données GTSRB sur 20 époques. Les métriques d'entraînement (train accuracy) et de validation (validation accuracy) ont été enregistrées et sont représentées dans la figure ci-dessous.

On observe une augmentation progressive et continue de la précision à la fois sur les données d'entraînement et de validation. Le modèle atteint :

- **90.65%** de précision sur le jeu d'entraînement ;
- **86.68%** de précision sur le jeu de validation.

```
Epoch [18/20], Train Accuracy: 86.68%, Valid Accuracy: 87.59%
Epoch [19/20], Train Accuracy: 88.82%, Valid Accuracy: 85.61%
Epoch [20/20], Train Accuracy: 90.65%, Valid Accuracy: 86.68%
```

FIGURE 11 – Schème de l'architecture du CNN

Ces résultats montrent que le modèle a bien appris à généraliser sans surapprentissage

excessif, puisque les courbes restent proches tout au long de l’entraînement. Un léger écart est visible à la fin (3–4 %), ce qui reste raisonnable pour une tâche de classification multi-classe avec plus de 40 catégories.

Stabilité Les courbes montrent également que l’apprentissage est stable : aucune chute brutale de précision ou oscillation n’est observée. À partir de la 15e époque, la précision de validation dépasse 85%, ce qui montre que le réseau a atteint une phase de convergence.

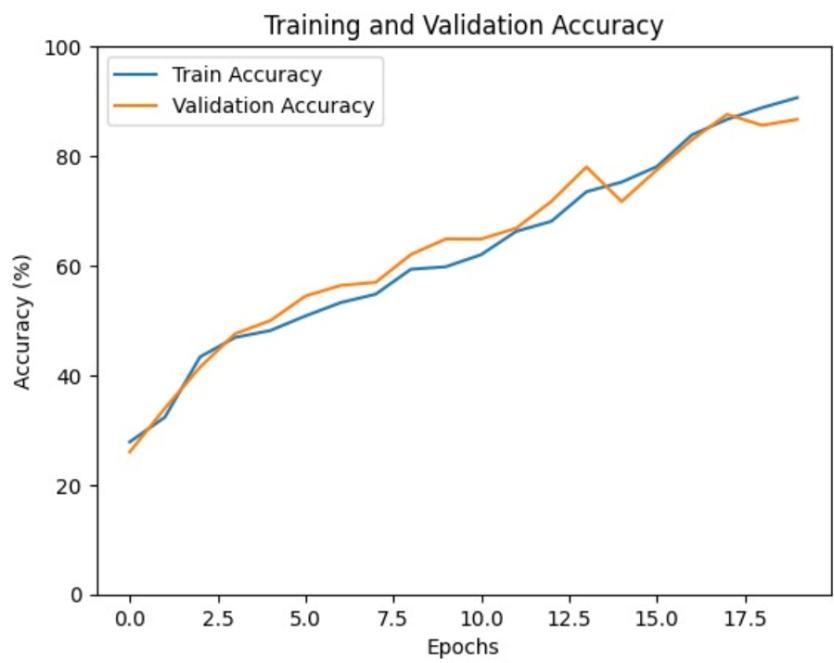


FIGURE 12 – Évolution de la précision d’entraînement et de validation en fonction des époques

Conclusion Le réseau entraîné peut donc être utilisé comme un modèle fiable de reconnaissance de panneaux dans un système embarqué ou une application temps réel. Ces performances valident également le choix de la base GTSRB pour cette tâche.

5.4 Interface

Il faut lancer le programme Interface de la branche Main et le tour est joué. l’utilisation de l’interface est simple, et il suffit de choisir une image.

Reconnaissance de Panneaux Routiers



FIGURE 13 – Interface Deep Learning

5.5 Tentative d'une application Web

L'un des obstacles que nous avons anticipés concernait la complexité d'utilisation de notre application par des utilisateurs externes. En effet, sans solution de déploiement adaptée, un utilisateur aurait été contraint de cloner le dépôt du projet, de configurer manuellement son environnement de travail, d'installer toutes les dépendances nécessaires, puis de lancer l'application localement. Ce processus, long et parfois source d'erreurs, aurait considérablement réduit l'accessibilité et la portabilité de notre solution.

Pour éviter cela, nous avons opté pour une solution plus élégante et professionnelle : l'hébergement de l'application via un site Web. Cela permet aux utilisateurs d'accéder directement à notre interface, sans se soucier des aspects techniques liés à l'installation ou à la configuration de l'environnement.

Les étapes de réalisation de cette solution ont inclus la création d'une image Docker contenant l'ensemble de notre application et de ses dépendances. Cette image a ensuite été utilisée pour lancer un conteneur localement durant la phase de développement et de test. Docker nous a offert un environnement reproductible et isolé, facilitant à la fois le déploiement et la maintenance de l'application.

Vous pouvez accéder au site web du projet ici : <https://cnn-3.onrender.com>

Sauf que le lancement du conteneur n'était que local car on s'est rendu compte que pour le rendre public, il fallait payer une certaine somme.

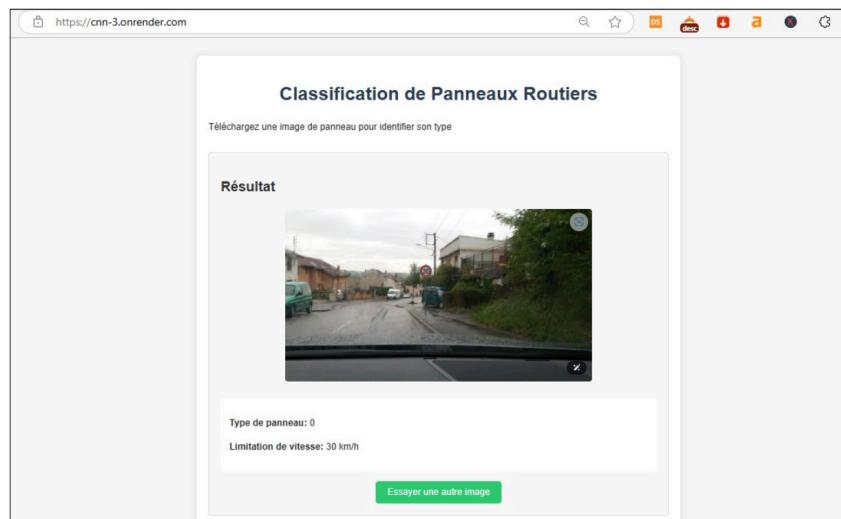


FIGURE 14 – Interface première de la partie Deep Learning

6 Perspectives et problèmes rencontrés

Ce projet nous a permis d'approfondir notre compréhension des différentes étapes du traitement d'images, depuis la prétraitement jusqu'à la reconnaissance d'objets à l'aide de modèles d'apprentissage profond. Nous avons acquis une bonne maîtrise des outils techniques tels que OpenCV pour la détection d'objets, ainsi que les réseaux de neurones convolutifs (CNN) pour la reconnaissance de panneaux de signalisation.

Malgré les compétences acquises, plusieurs contraintes ont freiné le développement d'une solution plus aboutie. L'utilisation d'un site Web payant pour accéder à certains outils ou bases de données a limité nos possibilités de mise en œuvre de méthodes plus avancées.

Nous avons également rencontré des difficultés techniques, notamment pour assurer la liaison entre le modèle CNN, développé en Python, et l'interface principale du projet, réalisée en Java. Cette intégration inter-langages a nécessité un temps important de recherche et d'adaptation.

Enfin, le principal obstacle à la bonne progression du projet a été lié à la gestion du travail en équipe. Une répartition des tâches initialement planifiée n'a pas été respectée, en raison du manque d'implication d'un membre du groupe. Cette passivité a entraîné une surcharge de travail pour les autres membres et a affecté la dynamique collaborative du projet.

Malgré ces difficultés, ce projet reste une expérience formatrice, tant sur le plan technique que sur le plan humain.

7 Annexe

```
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        out += self.shortcut(x)
        out = self.relu(out)
        return out
```

```
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes=8):
        super(ResNet, self).__init__()
        self.in_channels = 64
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2)
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2)
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)
    def _make_layer(self, block, out_channels, num_blocks, stride=1):
        layers = []
        layers.append(block(self.in_channels, out_channels, stride))
        self.in_channels = out_channels
        for _ in range(1, num_blocks):
            layers.append(block(self.in_channels, out_channels))
        return nn.Sequential(*layers)
    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1[x]
        x = self.relu(x)
        x = self.maxpool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avgpool(x)
```