Documentation Technique (Stage)

ES-SAIDI Naoufal

SOMMAIRE

Installation de l'environnement FIDLE (Windows)

1. Installer Python

Une version récente de **Python** est requise.
Les versions testées vont de **Python 3.9 à 3.11**.

Téléchargement :
https://www.python.org/downloads/

- Lors de l'installation, il faut **cocher impérativement l'option** :

Add Python to PATH

2. Créer un dossier de travail

- Tous les fichiers seront regroupés dans un même dossier, par exemple :

C:\fidle-tp

Vous pouvez créer ce dossier :

- via l'explorateur Windows
- ou via un terminal :

mkdir C:\fidle-tp

3. Créer un environnement Python virtuel

L'objectif est d'utiliser un **environnement virtuel** pour éviter de polluer l'installation Python du système.

3.1 Création et activation de l'environnement

- Utilisez **l'invite de commandes (cmd)**, **pas PowerShell**.

C:\> cd C:\fidle-tp

C:\fidle-tp> python -m venv --system-site-packages fidle-env

C:\fidle-tp> fidle-env\Scripts\activate

Si tout est correct, on devrais voir :

(fidle-env) C:\fidle-tp>

3.2 Installation de PyTorch avec la cmd suivante :

pip install torch torchvision torchaudio

3.3 Installation des autres dépendances

Toujours dans l'environnement virtuel :

pip install torch-geometric torchtext torchdata lightning tensorboard keras transformers datasets einops numpy scikit-image scikit-learn matplotlib plotly seaborn barviz pyarrow pandas pandoc pyyaml jupyterlab fidle

4. Installation des notebooks et des datasets

FIDLE fournit une commande dédiée pour installer automatiquement les notebooks et les jeux de données.

Depuis le dossier fidle-tp :

fid install –quiet

5. Vérification de l'installation

5.1 Structure attendue des dossiers

Après l'installation, on devrais obtenir la structure suivante :

fidle-tp

├── fidle # Notebooks

├── fidle-datasets # Datasets

└── fidle-env # Environnement virtuel

5.2 Vérification avec fid check

C:\> cd C:\fidle-tp

C:\fidle-tp> fidle-env\Scripts\activate

(fidle-env) C:\fidle-tp> fid check

Check environment :

Python	: Ok (3.10.x)

Fidle module : Ok
keras : Ok
numpy : Ok
sklearn : Ok
yaml : Ok
skimage : Ok
matplotlib : Ok
plotly : Ok
pandas : Ok
jupyterlab : Ok
torch : Ok
torchvision : Ok
lightning : Ok

6. Lancer JupyterLab

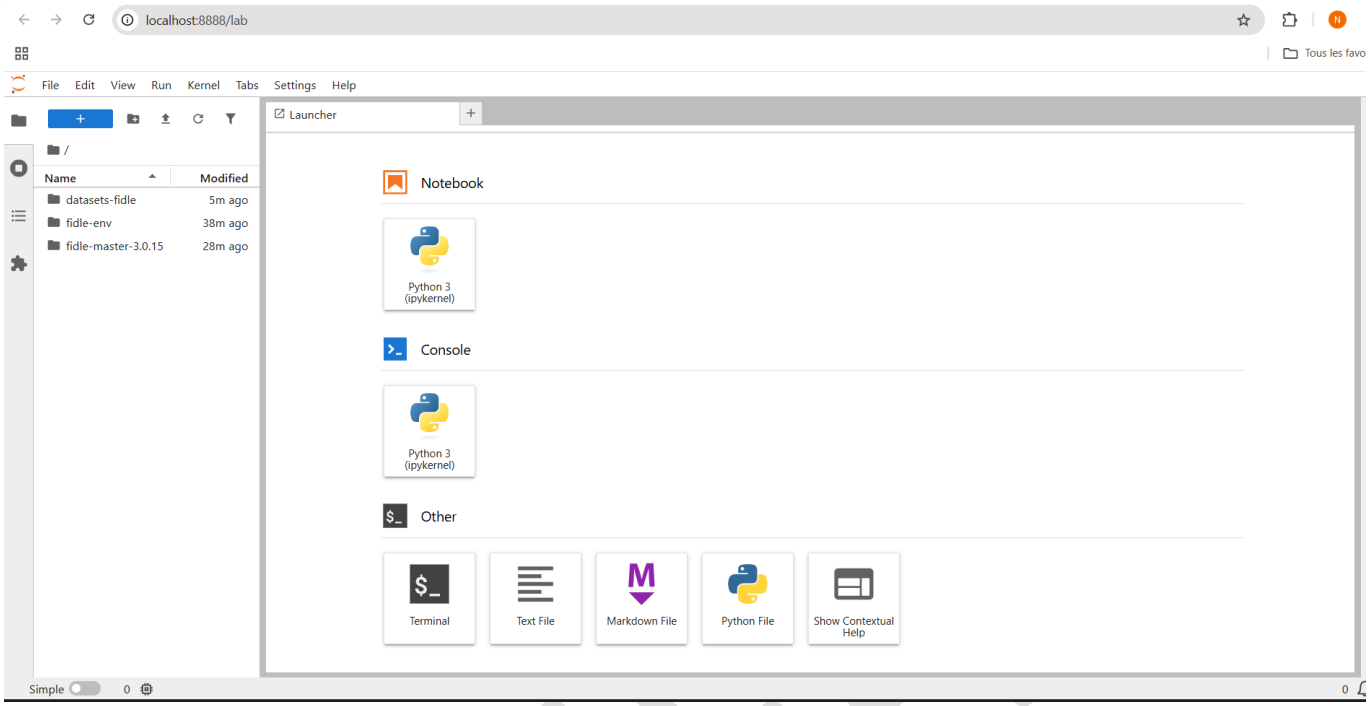
Tout simplement :

C:\> cd C:\fidle-tp

C:\fidle-tp> fidle-env\Scripts\activate

(fidle-env) C:\fidle-tp> jupyter lab

Le navigateur s'ouvrira **automatiquement** avec **JupyterLab.**



Projet : Reconnaissance automatique de chiffres à partir d'images

---

OBJECTIF GLOBAL DU PROGRAMME

Créer un programme capable de **reconnaître automatiquement un chiffre manuscrit** à partir d'une image, en utilisant un **réseau de neurones convolutif (CNN)** entraîné sur le dataset **MNIST**.

---

1) IMPORTS DES BIBLIOTHÈQUES

```
import numpy as np

import matplotlib.pyplot as plt

import cv2

from keras.datasets import mnist
```

from keras import models, layers

Ça sert à :

Librairie	**Rôle**
Numpy =>	calculs numériques (tableaux)
Matplotlib =>	afficher images et résultats
cv2 =>	lire et traiter les images
Keras =>	créer et entraîner le réseau de neurones

2) CHARGEMENT DU DATASET MNIST

(x_train, y_train), (x_test, y_test) = mnist.load_data()

Concept : dataset

- MNIST contient **60 000 images d'entraînement**
    - **10 000 images de test**
- Chaque image est un chiffre de **0 à 9**

Format :

x_train.shape = (60000, 28, 28)

y_train.shape = (60000,)

3) NORMALISATION DES IMAGES

x_train = x_train / 255.0

x_test = x_test / 255.0

Concept : normalisation

- Pixels ∈ [0, 255]
- On les ramène à [0, 1]

Cela permet de :

- Facilite l'apprentissage
- Accélère la convergence
- Évite des gradients instables

4) AJOUT DU CANAL

x_train = x_train[..., None]

x_test = x_test[..., None]

Concept : canal

- CNN attend des images au format :

(height, width, channels)

Ici :

- image en niveaux de gris → **1 canal**
- devient :

(28, 28, 1)

5) CRÉATION DU MODÈLE CNN

model = models.Sequential([

- **Concept : Sequential**

- empilement linéaire de couches
- simple et pédagogique

❖ **COUCHE D'ENTRÉE**

layers.Input((28,28,1)),

⇨ la taille des images en entrée

❖ **COUCHE CONVOLUTIVE 1**

layers.Conv2D(32, (3,3), activation='relu'),

Concept : convolution

- applique **32 filtres**
- chaque filtre détecte un motif :
  - bord
  - angle

- courbe

ReLU :

f(x) = max(0, x)

⇨ introduit de la non-linéarité

❖ MAX POOLING

layers.MaxPooling2D((2,2)),

Concept : pooling

- réduit la taille de l'image
- garde l'information importante
- rend le modèle plus robuste

❖ COUCHE CONVOLUTIVE 2

layers.Conv2D(64, (3,3), activation='relu'),

- Détecte des **formes plus complexes**
- Combinaison des motifs précédents

❖ FLATTEN

layers.Flatten(),

Concept :

- Transforme la carte 2D en **vecteur 1D**
- Transition vers les couches fully connected

❖ COUCHE DENSE

layers.Dense(128, activation='relu'),

⇨ apprentissage des **relations globales**

❖ COUCHE DE SORTIE

layers.Dense(10, activation='softmax')

Concept : Softmax

- transforme les scores en **probabilités**
- somme = 1

Ex :

[0.01, 0.02, ..., 0.95] → chiffre 9

6) COMPILATION DU MODÈLE

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

❖ **Optimiseur : Adam**

- ajuste automatiquement le learning rate
- très utilisé en pratique

❖ **Fonction de perte**

- compare prédiction / vérité
- pénalise les erreurs

7) ENTRAÎNEMENT DU MODÈLE

```
model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

Concept : apprentissage supervisé

- images + labels connus
- ajustement des poids par **backpropagation**

8) PRÉTRAITEMENT D'UNE IMAGE RÉELLE

```
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
img = cv2.resize(img, (28,28))
img = 255 - img
img = img / 255.0
img = img.reshape(1,28,28,1)
```

Pourquoi ces étapes ?

- même format que MNIST
- même échelle
- même nombre de canaux

Très important pour la cohérence

9) PRÉDICTION

```
prediction = model.predict(img)
digit = np.argmax(prediction)
```

Concept :

- le réseau sort un vecteur de probabilités
- argmax choisit la classe la plus probable

10) **AFFICHAGE DU RÉSULTAT**

```
plt.imshow(...)
plt.title(f"Chiffre reconnu : {digit}")
```

⇨ visualisation pour l'utilisateur

CODE PYTHON AU COMPLET :

Reconnaissance_Chiffre.ipynb

Code

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

from keras.datasets import mnist
from keras import models, layers
```

[2]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train / 255.0
x_test  = x_test  / 255.0

x_train = x_train[..., None]
x_test  = x_test[..., None]

print(x_train.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
(60000, 28, 28, 1)
```

[3]:
```
model = models.Sequential([
    layers.Input((28,28,1)),
    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

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

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dense_1 (Dense)	(None, 10)	1,290

Total params: 225,034 (879.04 KB)

Trainable params: 225,034 (879.04 KB)

Non-trainable params: 0 (0.00 B)

```
[4]: model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

```
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.9585 - loss: 0.1383 - val_accuracy: 0.9835 - val_loss: 0.0538
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - accuracy: 0.9857 - loss: 0.0450 - val_accuracy: 0.9885 - val_loss: 0.0360
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.9909 - loss: 0.0296 - val_accuracy: 0.9917 - val_loss: 0.0289
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 18s 11ms/step - accuracy: 0.9930 - loss: 0.0212 - val_accuracy: 0.9912 - val_loss: 0.0343
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - accuracy: 0.9949 - loss: 0.0155 - val_accuracy: 0.9897 - val_loss: 0.0360
[4]: <keras.src.callbacks.history.History at 0x16ae660b070>
```

```python
def predict_digit(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (28,28))
    img = 255 - img
    img = img / 255.0
    img = img.reshape(1,28,28,1)

    prediction = model.predict(img)
    digit = np.argmax(prediction)

    plt.imshow(img[0,:,:,0], cmap='gray')
    plt.title(f"Chiffre reconnu : {digit}")
    plt.axis('off')

    return digit
```

Phase de test :

```
[6]: predict_digit("mon_chiffre.png")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 143ms/step
[6]: np.int64(4)
```

```
[7]: predict_digit("mon_chiffre2.png")

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
[7]: np.int64(9)
```

❖ **CONCLUSION**

Ce programme met en œuvre un réseau de neurones convolutif afin de reconnaître automatiquement des chiffres manuscrits. Après une phase de normalisation et de mise en forme des images, un CNN est entraîné sur le dataset MNIST. Les performances obtenues permettent de reconnaître correctement des chiffres contenus dans des images réelles.

Documentation technique – Reconnaissance de panneaux de signalisation (GTSRB)

---

1. Objectif du projet

L'objectif du projet est de développer un modèle d'intelligence artificielle capable de reconnaître automatiquement des panneaux de signalisation routière à partir d'images.

Le système repose sur un réseau de neurones convolutif (CNN) entraîné à partir du dataset **GTSRB (German Traffic Sign Recognition Benchmark)** contenant 43 classes de panneaux.

---

2. Dataset utilisé

Le dataset GTSRB contient :

- plus de **50 000 images**
- **43 catégories** de panneaux

- différentes tailles, luminosités et angles

Les images d'entraînement sont organisées par dossiers correspondant aux identifiants de classes.

3. Prétraitement des données

Afin d'adapter les images au réseau de neurones :

- Redimensionnement en **64 × 64**
- Conversion BGR → RGB
- Normalisation des pixels entre **0 et 1**
- Encodage des labels en **one-hot**

Les données sont ensuite séparées :

- 80% entraînement
- 20% validation

4. Architecture du modèle

Le modèle utilisé est un **CNN simple et efficace** composé de :

- Convolutions pour extraire les caractéristiques visuelles
- MaxPooling pour réduire la dimension
- Couches denses pour la classification
- Dropout pour éviter l'overfitting

Structure :

- Conv2D (32 filtres)
- MaxPooling
- Conv2D (64 filtres)
- MaxPooling
- Flatten
- Dense 128
- Dropout 0.5
- Dense 43 (Softmax)

5. Compilation du modèle

Le modèle est compilé avec :

- Optimiseur : **Adam**
- Fonction de perte : **categorical_crossentropy**
- Métrique : **accuracy**

6. Entraînement

L'entraînement est effectué sur plusieurs epochs avec batch size de 32.

Durant l'apprentissage, on observe :

- diminution progressive de la loss
- augmentation de l'accuracy
- bonne généralisation sur le jeu de validation

7. Évaluation

Après entraînement, le modèle est testé sur le jeu de validation.

Les performances typiques obtenues sont élevées lorsque les images sont similaires au dataset.

8. Tests sur images externes

Lorsque le modèle est testé sur des photos réelles (téléphone, internet) :

on constate une baisse de précision.

Pourquoi ?

Parce que ces images contiennent :

- arrière-plans
- variations de lumière
- angles différents
- panneaux plus petits

Le réseau a principalement appris sur des panneaux **centrés et zoomés**.

9. Limites du système

Le modèle présente certaines limites :

- sensible au contexte autour du panneau
- difficulté lorsque le panneau est éloigné

- confusion possible entre formes similaires (ex : triangle)

10. Améliorations possibles

Plusieurs pistes permettraient d'augmenter les performances :

- ajout de data augmentation avancée
- détection automatique du panneau avant classification
- utilisation de réseaux plus profonds (ResNet, MobileNet)
- ajout d'images réelles dans l'entraînement

Programme Complet :

```
import os
import numpy as np
import cv2

from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

IMG_HEIGHT = 64
IMG_WIDTH = 64
NUM_CLASSES = 43

train_dir = r"C:\fidle-tp\datasets-fidle\GTSRB\Training\Images"

print("Train dir exists:", os.path.exists(train_dir))
```

```
WARNING:tensorflow:From C:\fidle-tp\fidle-env\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecate
d. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

Train dir exists: True
```

```
X = []
y = []

for class_id in range(NUM_CLASSES):
    class_path = os.path.join(train_dir, f"{class_id:05d}")

    for file in os.listdir(class_path):
        if file.endswith(".ppm"):
            img_path = os.path.join(class_path, file)

            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, (IMG_WIDTH, IMG_HEIGHT))

            X.append(img)
            y.append(class_id)

X = np.array(X, dtype="float32") / 255.0
y = np.array(y)

print("Images:", X.shape)
print("Labels:", y.shape)
```

```
Images: (39209, 64, 64, 3)
Labels: (39209,)
```

```
y = to_categorical(y, NUM_CLASSES)

print("y after one hot:", y.shape)
```

```
y after one hot: (39209, 43)
```

```
X_train, X_val, y_train, y_val = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=np.argmax(y, axis=1)
)

print("Train:", X_train.shape)
print("Val:", X_val.shape)
```

```
Train: (31367, 64, 64, 3)
Val: (7842, 64, 64, 3)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, (3,3), activation="relu", input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation="relu"),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation="relu"),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(256, activation="relu"),
    Dropout(0.5),
    Dense(NUM_CLASSES, activation="softmax")
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 29, 29, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 256)               1179904

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 1284203 (4.90 MB)
Trainable params: 1284203 (4.90 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

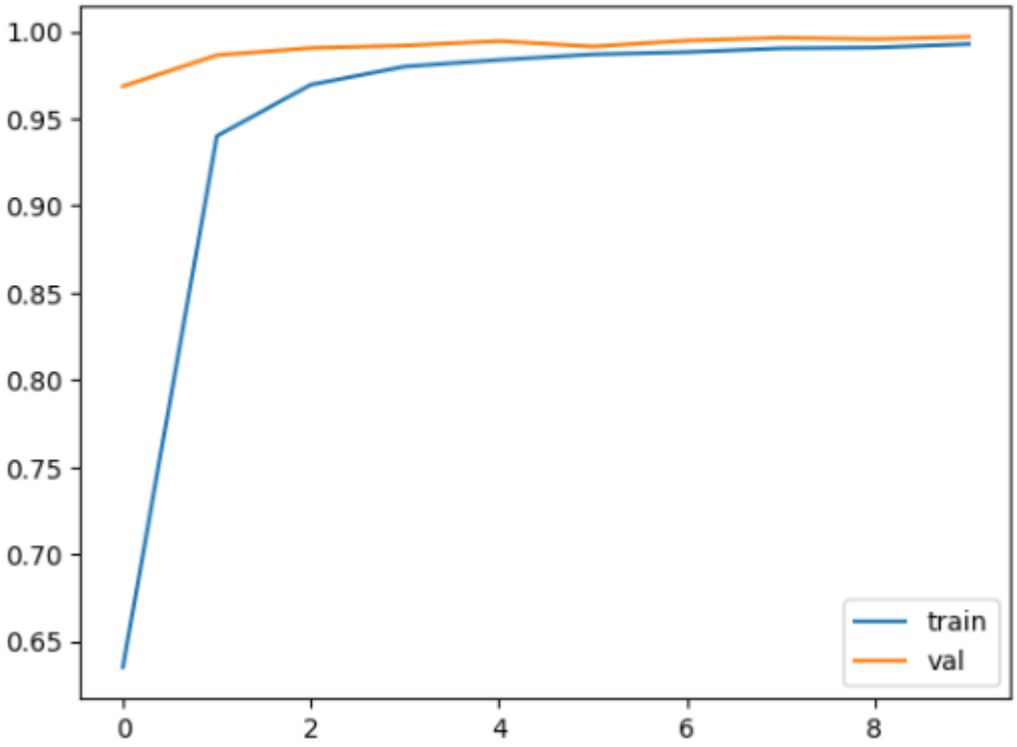
```python
model.compile(
    optimizer="adam",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

```python
history = model.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=10,
    batch_size=32
)
```

```
981/981 [==============================] - 66s 65ms/step - loss: 1.2514 - accuracy: 0.6351 - val_loss: 0.1285 - val_accuracy: 0.9685
Epoch 2/10
981/981 [==============================] - 62s 63ms/step - loss: 0.2004 - accuracy: 0.9399 - val_loss: 0.0648 - val_accuracy: 0.9864
Epoch 3/10
981/981 [==============================] - 61s 63ms/step - loss: 0.1028 - accuracy: 0.9694 - val_loss: 0.0422 - val_accuracy: 0.9904
Epoch 4/10
981/981 [==============================] - 62s 64ms/step - loss: 0.0683 - accuracy: 0.9798 - val_loss: 0.0381 - val_accuracy: 0.9918
Epoch 5/10
981/981 [==============================] - 64s 65ms/step - loss: 0.0551 - accuracy: 0.9836 - val_loss: 0.0239 - val_accuracy: 0.9944
Epoch 6/10
981/981 [==============================] - 63s 64ms/step - loss: 0.0439 - accuracy: 0.9868 - val_loss: 0.0356 - val_accuracy: 0.9913
Epoch 7/10
981/981 [==============================] - 62s 64ms/step - loss: 0.0397 - accuracy: 0.9881 - val_loss: 0.0273 - val_accuracy: 0.9946
Epoch 8/10
981/981 [==============================] - 63s 64ms/step - loss: 0.0308 - accuracy: 0.9902 - val_loss: 0.0238 - val_accuracy: 0.9963
Epoch 9/10
981/981 [==============================] - 63s 64ms/step - loss: 0.0306 - accuracy: 0.9907 - val_loss: 0.0204 - val_accuracy: 0.9955
Epoch 10/10
981/981 [==============================] - 64s 65ms/step - loss: 0.0234 - accuracy: 0.9927 - val_loss: 0.0198 - val_accuracy: 0.9968
```

```python
import matplotlib.pyplot as plt

plt.plot(history.history["accuracy"], label="train")
plt.plot(history.history["val_accuracy"], label="val")
plt.legend()
plt.show()
```




```python
loss, acc = model.evaluate(X_val, y_val)
print("Validation accuracy:", acc)
```

```
246/246 [==============================] - 4s 16ms/step - loss: 0.0198 - accuracy: 0.9968
Validation accuracy: 0.9968120455741882
```

Phase de TEST :

```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array

IMAGE_PATH = r"C:\fidle-tp\panneau1.png"

classes = {
    0: "Limitation de vitesse 20 km/h",
    1: "Limitation de vitesse 30 km/h",
    2: "Limitation de vitesse 50 km/h",
    3: "Limitation de vitesse 60 km/h",
    4: "Limitation de vitesse 70 km/h",
    5: "Limitation de vitesse 80 km/h",
    6: "Fin limitation 80 km/h",
    7: "Limitation de vitesse 100 km/h",
    8: "Limitation de vitesse 120 km/h",
    9: "Interdiction de dépasser",
    10: "Interdiction de dépasser pour poids lourds",
    11: "Priorité à la prochaine intersection",
    12: "Route prioritaire",
    13: "Cédez le passage",
    14: "Stop",
    15: "Circulation interdite à tous véhicules",
    16: "Interdit aux véhicules de plus de 3.5 tonnes",
    17: "Sens interdit",
    18: "Danger général",
    19: "Virage dangereux à gauche",
    20: "Virage dangereux à droite",
    21: "Double virage",
    22: "Chaussée déformée",
    23: "Chaussée glissante",
    24: "Rétrécissement de chaussée à droite",
    25: "Travaux",
    26: "Feux tricolores",
    27: "Passage piétons",
    28: "Passage d'enfants",
    29: "Passage de cyclistes",
    30: "Risque de verglas/neige",
    31: "Passage d'animaux sauvages",
    32: "Fin de toutes les interdictions",
    33: "Tourner à droite obligatoire",
    34: "Tourner à gauche obligatoire",
    35: "Tout droit obligatoire",
    36: "Tout droit ou à droite",
    37: "Tout droit ou à gauche",
    38: "Contournement obligatoire par la droite",
    39: "Contournement obligatoire par la gauche",
    40: "Giratoire obligatoire",
    41: "Fin interdiction de dépasser",
    42: "Fin interdiction de dépasser pour poids lourds"
}
img = load_img(IMAGE_PATH, target_size=(IMG_HEIGHT, IMG_WIDTH))
img_array = img_to_array(img) / 255.0
img_array = np.expand_dims(img_array, axis=0)

pred = model.predict(img_array)
classe = np.argmax(pred)
conf = np.max(pred)

plt.imshow(img)
plt.axis("off")
plt.title(f"{classes[classe]} ({conf:.2%})")
plt.show()

print("Classe:", classe)
print("Nom:", classes[classe])
print("Confiance:", conf)
```

Résultat :

```
1/1 [==============================] - 0s 115ms/step
```


```
Classe: 17
Nom: Sens interdit
Confiance: 1.0
```


```
Classe: 19
Nom: Virage dangereux à gauche
Confiance: 0.9999993
```

Conclusion

Le projet démontre la capacité d'un CNN à reconnaître efficacement des panneaux routiers.

Les résultats sont très bons sur les images proches du dataset et montrent les défis du passage vers des conditions réelles.