Annexes

# Classification des panneaux de signalisation par CNN :

## Importation des bibliothèques :



import os import pathlib

import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split from tensorflow.keras.preprocessing import image

from tensorflow.python.keras.preprocessing.image import ImageDataGenerator, img\_to\_array, array\_to\_img, load\_img

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.layers import Conv2D, maxPool2D, Dense, Flatten,

Dropout

from tensorflow.keras.models import Sequential from sklearn.metrics import accuracy\_score

* Définir les constantes (chemin et autres) :



# directories train\_dir = '..\Train' test\_dir = '../'

# fixing width and height of each image img\_height = 30

img\_width = 30

## Visualisation d’un échantillon de chaque classe :



# Visualizing all the categories num\_categories = len(os.listdir(train\_dir)) img\_dir = pathlib.Path(train\_dir) plt.figure(figsize = (14,14))

i = 0

for j in range(num\_categories): plt.subplot(7,7,j+1) plt.grid(False) plt.xticks([])

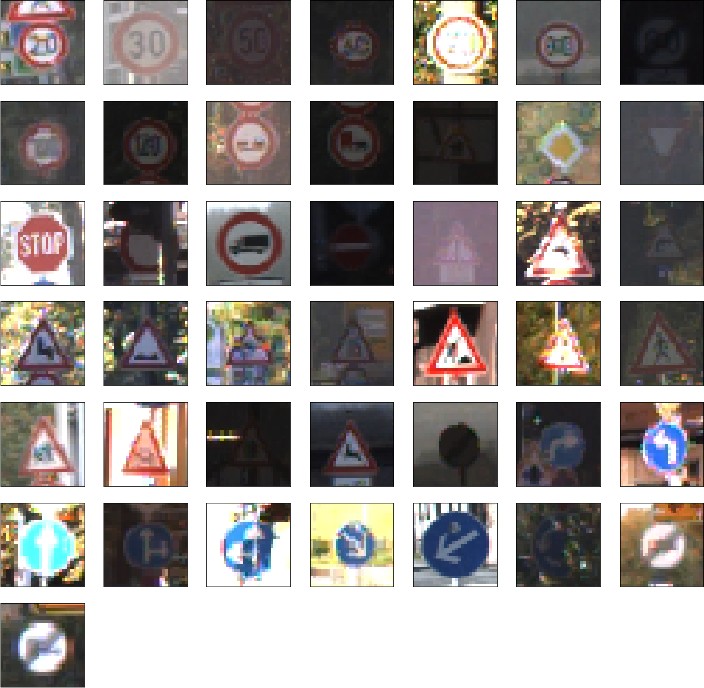
plt.yticks([])

sign = list(img\_dir.glob(f'{j}/\*'))[0]

img = load\_img(sign,target\_size = (img\_width,img\_height)) plt.imshow(img)

plt.show() num\_categories

Output interpréteur:



43

## Création du model CNN à 3 couches :



#creating the model model = Sequential()

#first convolutional layer

model.add(Conv2D(filters = 32,kernel\_size = 3, activation = 'relu', input\_shape = (img\_height, img\_width,3))) model.add(MaxPool2D(pool\_size = (2,2)))

model.add(Dropout(rate = 0.25))

#second convolutional Layer

model.add(Conv2D(filters = 64,kernel\_size = 3, activation = 'relu')) model.add(MaxPool2D(pool\_size = (2,2)))

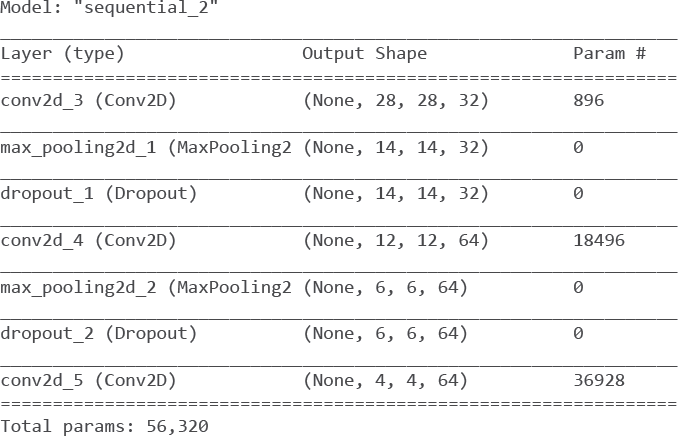
model.add(Dropout(rate = 0.25))

#Third convolutional Layer

model.add(Conv2D(filters = 64,kernel\_size = 3, activation = 'relu'))

# printing details model.summary()

Output interpréteur :



## Aplanissement du couches et l’ajout d’une couche dense :

*CNN* se compose de (conv − pool)𝑛 − (flatten ou globalpool) − (Dense)𝑚, où la partie (conv − pool)𝑛 extrait les caractéristiques d'un signal 2D et (Dense)𝑚 sélectionne les caractéristiques des couches précédentes.

La sortie de la dernière couche est (4 4 64) qui sont 64 -*feature maps*- de taille 4 × 4 (signaux 2D). Nous les aplatissons ensuite pour obtenir un vecteur de dimension 4 × 4 × 64 = 1024 (à la place, nous pouvons également utiliser -*global max/avg pool-* pour obtenir un vecteur de dimension 64).

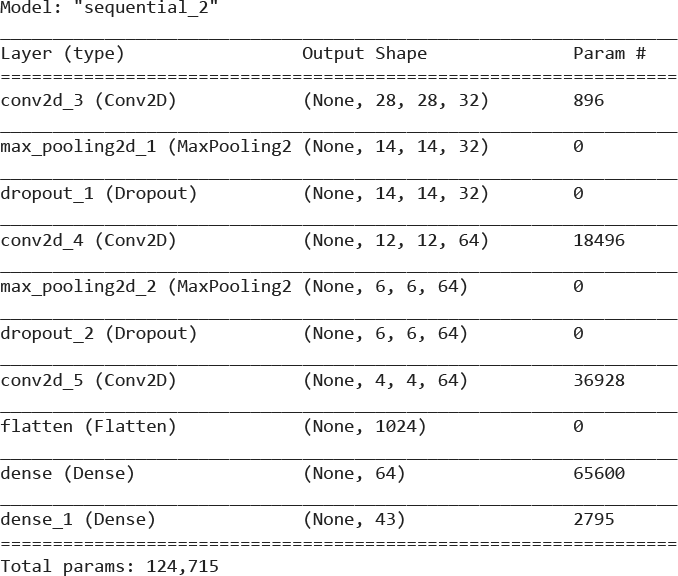


# Flattening the layer and adding Dense Layer model.add(Flatten())

model.add(Dense(units = 64, activation = 'relu')) model.add(Dense(num\_categories, activation = 'softmax'))

# Printing details model.summary()

Output interpréteur :



## Compilation du model :



#compiling the model model.compile(

loss = 'categorical\_crossentropy', optimizer = 'adam',

metrics = ['accuracy']

)

* Diviser les données en -training data- et -validation data- :



def load\_data(train\_dir): images = list() labels = list()

for cat in range(num\_categories):

cats\_train = os.path.join(train\_dir, str(cat)) for img in os.listdir(cats\_train):

img = load\_img(os.path.join(cats\_train, img),target\_size =

(30,30))

image = img\_to\_array(img) images.append(image) labels.append(cat)

return images, labels

images, labels = load\_data(train\_dir)

# Converting labels to categorical data matrix labels = to\_categorical(labels)

# Splitting the dataset into training and test set

x\_train, x\_test, y\_train, y\_test = train\_test\_split(np.array(images), labels, test\_size = 0.3)

## Fitting the model :



## Fitting the model Epochs = 30

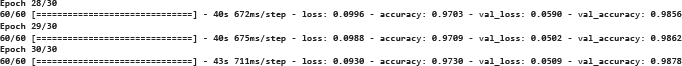
history = model.fit( x\_train, y\_train,

validation\_data = (x\_test, y\_test), epochs = Epochs,

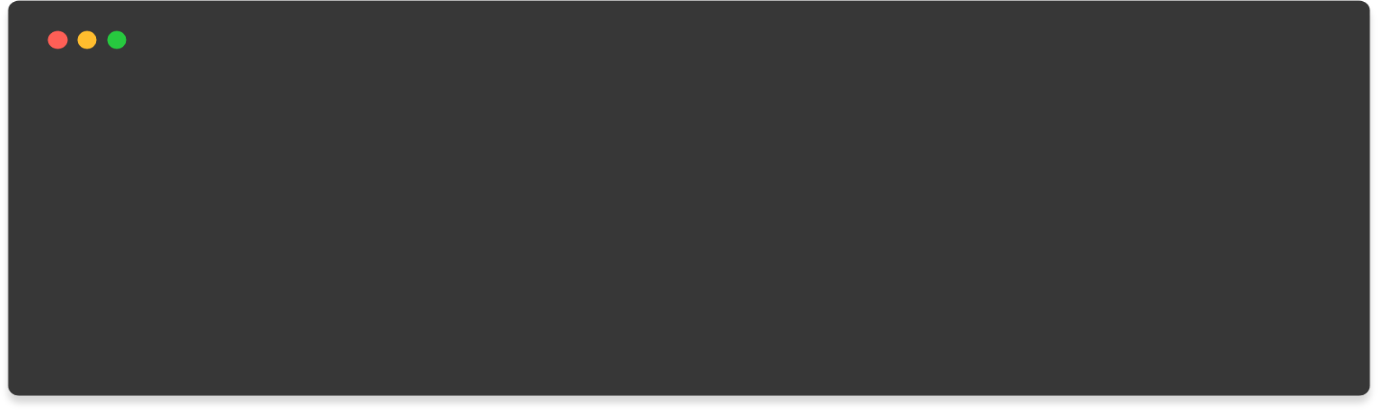
steps\_per\_epoch = 60

)

Output interpréteur (last 3 epochs) :



## Validation du model et performances :



# printing the val\_loss and val\_accuracy loss, accuracy = model.evaluate(x\_test,y\_test)

print('test set accuracy : ', accuracy\* 100)accuracy : ', accuracy\* 100)

Output interpréteur :

368/368 [==============================] - 9s 25ms/step - loss: 0.0509 -

accuracy: 0.9878

test set accuracy : 98.7758219242096

## Traçage du fonctions loss et accuracy :



# Plotting the accuracy and loss values with the training data accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

loss = history.history['loss'] val\_loss = history.history['val\_loss']

epochs\_range = range(Epochs) plt.figure(figsize = (8,8)) plt.subplot(1,2,1)

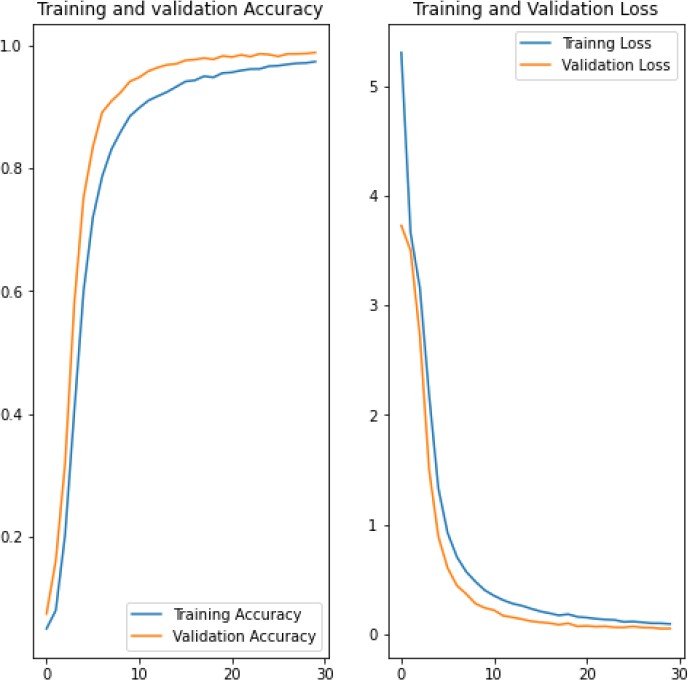
plt.plot(epochs\_range, accuracy, label = 'Training Accuracy') plt.plot(epochs\_range, val\_accuracy, label = 'Validation Accuracy') plt.legend(loc = 'lower right')

plt.title('Training and validation Accuracy') plt.subplot(1,2,2)

plt.plot(epochs\_range, loss, label = 'Trainng Loss') plt.plot(epochs\_range, val\_loss, label = 'Validation Loss') plt.legend(loc = 'upper right')

plt.title('Training and Validation Loss') plt.show()

Output interpréteur :



## Phase du test :



Y\_test = pd.read\_csv(test\_dir+'Test.csv') test\_labels = Y\_test["ClassId"].values test\_images = Y\_test["Path"].values

output = list()

for img in test\_images :

image = load\_img(os.path.join(test\_dir, img), target\_size = (30,30)) output.append(np.array(image))

X\_test = np.array(output)

pred = model.predict\_classes(X\_test)

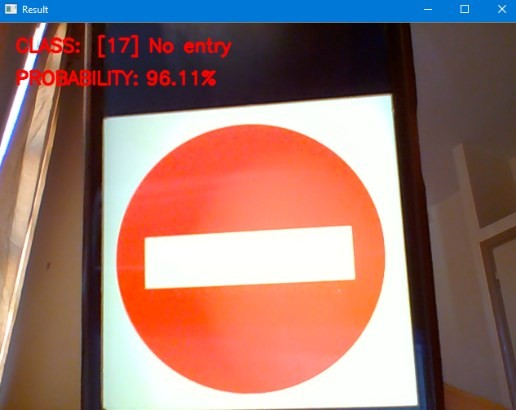
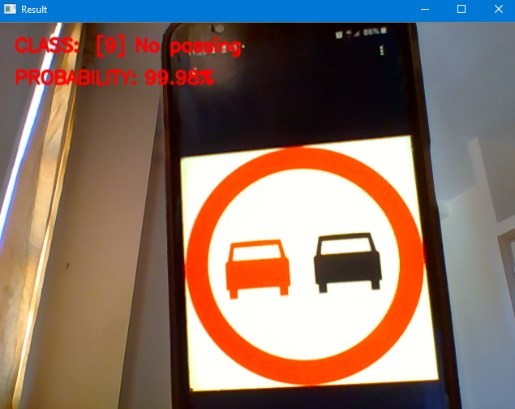
# Printing Accuracy with the test data

print('Test Data Accuracy : ', accuracy\_score(test\_labels, pred) \* 100)

Output interpréteur :

Test Data Accuracy : 94.87727632620744

* Test PAR Webcam :



# Classification et détection des panneaux de signalisation par YOLO V5 :

## Importation des bibliothèques



# install dependencies as necessary

!pip install -qr requirements.txt # install dependencies (ignore errors)

import torch

from IPython.display import Image, clear\_output # to display images

from utils.google\_utils import gdrive\_download # to download models/datasets

# clear\_output()

print('Setup complete. Using torch %s %s' % (torch. version , torch.cuda.get\_ device\_properties(0) if torch.cuda.is\_available() else 'CPU'))

* Importation du modèle YOLO V5 de roboflow



#follow the link below to get your download code from from Roboflow

!pip install -q roboflow from roboflow import Roboflow

rf = Roboflow(model\_format="yolov5", notebook="roboflow-yolov5"

## Téléchargement de la data générée par roboflow



%cd /content/yolov5

#after following the link above, recieve python code with these fields filledin

from roboflow import Roboflow

rf = Roboflow(api\_key="AFTOPTikJBOv6TMlqQDF") project = rf.workspace().project("tsr-system") dataset = project.version(8).download("yolov5")

* Vérification des classes



# this is the YAML file Roboflow wrote for us that we're loading into this not

ebook with our data

%cat {dataset.location}/data.yam

names:

* speed limit 120 km-h
* speed limit 20 km-h
* speed limit 60 km-h
* speed limit 80 km-h
* stop nc: 5

train: TSR-System-8/train/images val: TSR-System-8/valid/images

## Configuration et architecture du modèle



# parameters

nc: {num\_classes} # number of classes depth\_multiple: 0.33 # model depth multiple width\_multiple: 0.50 # layer channel multiple # anchors

anchors:

- [10,13, 16,30, 33,23] # P3/8

- [30,61, 62,45, 59,119] # P4/16

- [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone backbone:

# [from, number, module, args]

[[-1, 1, Focus, [64, 3]], # 0-P1/2

]



#this is the model configuration we will use for our tutorial

%cat /content/yolov5/models/yolov5s.yaml

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [-1, | 1, | Conv, [128, 3, | 2]], # | 1-P2/4 |
| [-1, | 3, | BottleneckCSP, | [128]], |  |
| [-1, | 1, | Conv, [256, 3, | 2]], # | 3-P3/8 |
| [-1, | 9, | BottleneckCSP, | [256]], |  |
| [-1, | 1, | Conv, [512, 3, | 2]], # | 5-P4/16 |
| [-1, | 9, | BottleneckCSP, | [512]], |  |
| [-1, | 1, | Conv, [1024, 3, 2]], # 7-P5/32 | | |
| [-1, | 1, | SPP, [1024, [5, 9, 13]]], | | |
| [-1, | 3, | BottleneckCSP, [1024, False]], # 9 | | |



# YOLOv5 head

head:

[[-1, 1, Conv, [512, 1, 1]],

[-1, 1, nn.Upsample, [None, 2, 'nearest']],

[[-1, 6], 1, Concat, [1]], # cat backbone P4

[-1, 3, BottleneckCSP, [512, False]], # 1

[-1, 1, Conv, [256, 1, 1]],

[-1, 1, nn.Upsample, [None, 2, 'nearest']],

[[-1, 4], 1, Concat, [1]], # cat backbone P3

[-1, 3, BottleneckCSP, [256, False]], # 17 (P3/8-small)

[-1, 1, Conv, [256, 3, 2]],

[[-1, 14], 1, Concat, [1]], # cat head P4

[-1, 3, BottleneckCSP, [512, False]], # 20 (P4/16-medium)

[-1, 1, Conv, [512, 3, 2]],

[[-1, 10], 1, Concat, [1]], # cat head P5

[-1, 3, BottleneckCSP, [1024, False]], # 23 (P5/32-large)

[[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)

]

* Training du modèle



# train yolov5s on custom data for 100 epochs # time its performance

%%time

%cd /content/yolov5/

!python train.py --img 416 --batch 16 --epochs 100 --

data {dataset.location}/data.yaml --cfg ./models/custom\_yolov5s.yaml -- weights '' --name yolov5s\_results --cache

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch | gpu\_mem | box | obj | cls | total | targets | img\_size |
| 0/99 | 1.79G | 0.09273 | 0.02085 | 0.05035 | 0.1639 | 2 | 416: 100% 88/88 [00:55<00:00, 1.59it/s] |

## Test sur des images

Starting training for 100 epochs...

Class

Images

Targets

P

R

mAP@.5 mAP@.5:.95: 100% 4/4 [00:04<00:00,

1.08s/it]

all

108

110

0.00081

0.279 0.000751 0.000116

Epoch gpu\_mem box obj cls total targets img\_size

1/99 1.84G 0.08241 0.02215 0.04753 0.1521 3 416: 100% 88/88 [00:50<00:00, 1.75it/s]

Class Images Targets P R mAP@.5 mAP@.5:.95: 100% 4/4 [00:01<00:00,

2.11it/s]

all 108 110 0.0112 0.125 0.00446 0.000786

Epoch gpu\_mem box obj cls total targets img\_size

2/99 1.84G 0.08033 0.02243 0.046 0.1488 1 416: 100% 88/88 [00:48<00:00, 1.80it/s]

Class Images Targets P R mAP@.5 mAP@.5:.95: 100% 4/4 [00:01<00:00,

2.13it/s]

all 108 110 0.096 0.0434 0.0252 0.0175

Epoch gpu\_mem box obj cls total targets img\_size

3/99 1.84G 0.0773 0.02261 0.04476 0.1447 1 416: 100% 88/88 [00:48<00:00, 1.80it/s]

Class Images Targets P R mAP@.5 mAP@.5:.95: 100% 4/4 [00:01<00:00,

2.12it/s]

all 108 110 0.422 0.0211 0.0125 0.0016

Epoch gpu\_mem box obj cls total targets img\_size

4/99 1.84G 0.0711 0.02436 0.04425 0.1397 1 416: 100% 88/88 [00:48<00:00, 1.80it/s]

Class Images Targets P R mAP@.5 mAP@.5:.95: 100% 4/4 [00:01<00:00,

2.16it/s]

all 108 110 0.00848 0.366 0.0106 0.00212

…





# when we ran this, we saw .007 second inference time. That is 140 FPS on a TE SLA P100!

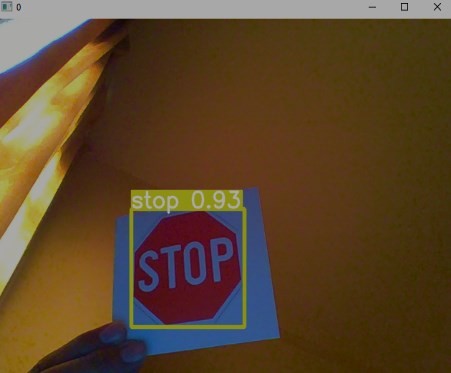
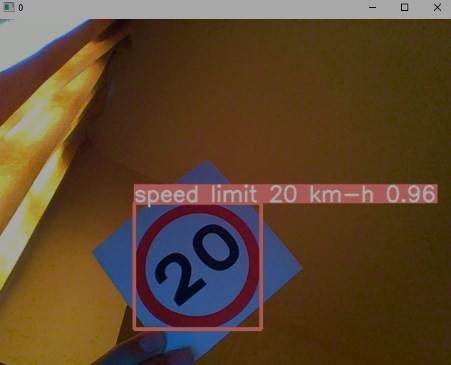
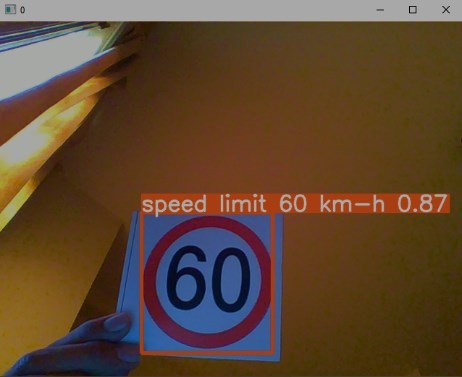
# use the best weights!

%cd /content/yolov5/

!python detect.py --weights runs/train/yolov5s\_results/weights/best.pt - img 416 --conf 0.4 --source ../images



* Test avec webcam



**CONTRIBUTORS:**

