1. Import dependencies:

In [ ]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
from PIL import Image
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import cv2
```

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2. Loading data:

A- Loading the both data:

In [ ]:
```
data_highway = os.listdir('data\Class0')
data_regional = os.listdir('data\Class1')
data_tunel =  os.listdir('data\Class2')
data_urbain =  os.listdir('data\Class3')
```

B- Verify the data;

In [ ]:
```
print(data_highway[0:5])
print(data_regional[0:5])
print(data_tunel[0:5])
print(data_urbain[0:5])
```

```
['0.png', '1.png', '10.png', '100.png', '101.png']
['0.png', '1.png', '10.png', '100.png', '101.png']
['0.png', '1.png', '10.png', '100.png', '101.png']
['0.png', '1.png', '10.png', '100.png', '101.png']
```

C- Number of images:

In [ ]:
```
print(len(data_highway))
print(len(data_regional))
print(len(data_tunel))
print(len(data_urbain))
```

```
570
735
410
995
```

3. Label of images:

A- Creating the labels:

In [ ]:
```
highway_label = [0]*570
regional_label = [1]*735
tunel_label = [2]*410
urbain_label = [3]*995
```

In [ ]:
```
print(highway_label[0:5])
print(regional_label[0:5])
print(tunel_label[0:5])
print(urbain_label[0:5])
```

```
[0, 0, 0, 0, 0]
[1, 1, 1, 1, 1]
[2, 2, 2, 2, 2]
[3, 3, 3, 3, 3]
```

In [ ]:
```
print(len(highway_label))
print(len(regional_label))
print(len(tunel_label))
print(len(urbain_label))
```

```
570
735
410
995
```

B- Creatiing the label of the both datas:

In [ ]:
```
labels = highway_label + regional_label + tunel_label + urbain_label
print(len(labels))
print(labels[0:5])
print(labels[-5:])
```

```
2710
[0, 0, 0, 0, 0]
[3, 3, 3, 3, 3]
```

4. Image preprocessing:

A- Displaying an highway images:

In [ ]:
```
img = mpimg.imread('data/Class0/7.png')
plt.imshow(img)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x1c82a6a75e0>
```

C- Converting & resising image to numpy array:

In [ ]:
```
highway = 'C:/Deep_learning Python/projetSalman/data/Class0/'
regional = 'C:/Deep_learning Python/projetSalman/data/Class1/'
tunel = 'C:/Deep_learning Python/projetSalman/data/Class2/'
urbain = 'C:/Deep_learning Python/projetSalman/data/Class3/'
data = []

for img in data_highway:
    image = Image.open(highway + img)
    image = image.resize( (256 ,256) )
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)

for img in data_regional:
    image = Image.open(regional + img)
    image = image.resize( (256 ,256) )
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)

for img in data_tunel:
    image = Image.open(tunel + img)
    image = image.resize( (256 ,256) )
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)

for img in data_urbain:
    image = Image.open(urbain + img)
    image = image.resize( (256 ,256) )
    image = image.convert('RGB')
    image = np.array(image)
    data.append(image)
```

D- Type of the data:

In [ ]:
```
type(data)
```

Out[ ]:
```
list
```

E- Number of images in data:

In [ ]:
```
len(data)
```

Out[ ]:
```
2710
```

F- View some images:

In [ ]:
```
data[0:2]
```

Out[ ]:
```
[array([[[146, 203, 251],
         [146, 203, 251],
         [148, 203, 251],
         ...,
         [123, 179, 232],
         [123, 179, 232],
         [123, 179, 234]],

        [[146, 203, 251],
         [146, 203, 251],
         [148, 203, 251],
         ...,
         [123, 179, 232],
         [123, 179, 232],
         [123, 179, 233]],

        [[146, 203, 251],
         [146, 203, 251],
         [148, 203, 251],
         ...,
         [123, 179, 232],
         [123, 179, 232],
         [123, 179, 233]],

        ...,

        [[ 29,  45,  65],
         [ 29,  45,  64],
         [ 29,  45,  64],
         ...,
         [ 10,  22,  38],
         [ 12,  27,  43],
         [ 18,  34,  50]],

        [[ 30,  45,  65],
         [ 29,  43,  63],
         [ 28,  43,  61],
         ...,
         [ 10,  22,  37],
         [ 12,  28,  44],
         [ 19,  36,  52]],

        [[ 28,  42,  63],
         [ 25,  40,  59],
         [ 24,  39,  57],
         ...,
         [  9,  22,  37],
         [ 12,  28,  43],
         [ 20,  37,  52]]], dtype=uint8),
 array([[[119, 171, 225],
         [119, 171, 225],
         [119, 171, 225],
         ...,
         [104, 161, 212],
         [104, 161, 212],
         [104, 161, 212]],
```

```
       [[119, 171, 225],
        [119, 171, 225],
        [119, 171, 225],
        ...,
        [104, 161, 212],
        [104, 161, 212],
        [104, 161, 212]],

       [[120, 171, 226],
        [120, 171, 226],
        [120, 171, 226],
        ...,
        [104, 161, 212],
        [104, 161, 212],
        [104, 161, 212]],

       ...,

       [[115, 103,  88],
        [112, 100,  85],
        [110,  97,  84],
        ...,
        [ 11,  26,  44],
        [ 13,  28,  46],
        [ 16,  30,  49]],

       [[114, 104,  90],
        [114, 103,  90],
        [112, 101,  87],
        ...,
        [  8,  22,  41],
        [ 10,  25,  43],
        [ 13,  28,  48]],

       [[112, 102,  88],
        [111, 100,  88],
        [110,  98,  85],
        ...,
        [  5,  20,  40],
        [  7,  21,  41],
        [ 10,  24,  45]]], dtype=uint8)]
```

G- Type & dimmenssion of the images:

In [ ]:
```
print(data[0].shape)
print(type(data[0]))
```

```
(256, 256, 3)
<class 'numpy.ndarray'>
```

5. To numpy array:

A- Separating the variables:

In [ ]:
```
X = np.array(data)
Y = np.array(labels)
```

B- Type of X & Y:

In [ ]:
```
print(type(X))
print(type(Y))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

C- Shape of X & Y:

In [ ]:
```
print(X.shape)
print(Y.shape)
```

```
(2710, 256, 256, 3)
(2710,)
```

6. Train test split:

A- Creating the variables of testing & training:

In [ ]:
```
X_train , X_test , Y_train , Y_test = train_test_split(X,Y , test_size=0.2, random_
```

B- Shape of X & X_train & X_test:

In [ ]:
```
print(X.shape , X_train.shape , X_test.shape)
```

```
(2710, 256, 256, 3) (2168, 256, 256, 3) (542, 256, 256, 3)
```

C- Scaling the data:

In [ ]:
```
X_trainStd = X_train / 255
X_testStd = X_test / 255
```

In [ ]:
```
X_trainStd[0][0][0]
```

Out[ ]:
```
array([0.57254902, 0.5254902 , 0.47843137])
```

7. Building our neural network (CNN) :

A- Number of classes:

In [ ]:
```
numClasses = 4
```

B- Creating the model:

In [ ]:
```
#Initialise le model (model couche par couche)
model = keras.Sequential()
#1er cocuhe de convolution avec:
```

```
    #32: 32 noyuax ou nombre de filtres
    #(3,3) : taille de chaque filre (carré 3*3)
    #La fonction d'activation : ajouter la nom linéarité au model
    #Input shape: definit la forme des donées d'entre (image 128*128 avec format (R
model.add(keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu', input_shape

#1er couche de poling
    #Pol size (2,2): taille de fenetre de poling qui reduit chaque dim de la carte
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

#2eme couche de convolution:
    #64: 64 noyuax ou nombre de filtres
    #(3,3) : taille de chaque filre (carré 3*3)
    #La fonction d'activation : ajouter la nom linéarité au model
model.add(keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'))

#2eme couche de poling
    #Pol size (2,2): taille de fenetre de poling qui reduit chaque dim de la carte
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))

#Applatissement des donées:
    #Converir les caractéristique 2D en un vecteur 1D
model.add(keras.layers.Flatten())

#1er Couche dense:
    #Ajout une couche dense avec 128 unité
    #Utilisation de la fonction d'activation Relu
model.add(keras.layers.Dense(128, activation='relu'))

#1er couche de dropout:
    #Couche qui abondnne 50% des caractéristique (diversité de caratérisiques)
model.add(keras.layers.Dropout(0.5))

#2eme couche dense:
    #Couche avec 64 unités
model.add(keras.layers.Dense(64, activation='relu'))

#2eme couche dropout:
model.add(keras.layers.Dropout(0.5))

#Couche de sortie:
    #Identifier le nombre de class a predire + la fnct d'activation
model.add(keras.layers.Dense(numClasses, activation='sigmoid'))
```

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.
```

C- Compiling the model:

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

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

D- Training the model:

In [ ]:
```
history = model.fit(X_trainStd, Y_train, validation_split=0.1, epochs=5)
```

```
Epoch 1/5
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

61/61 [==============================] - 42s 657ms/step - loss: 2.6671 - acc: 0.4485 - val_loss: 0.7870 - val_acc: 0.6912
Epoch 2/5
61/61 [==============================] - 39s 634ms/step - loss: 0.8360 - acc: 0.6535 - val_loss: 0.4394 - val_acc: 0.7972
Epoch 3/5
61/61 [==============================] - 39s 631ms/step - loss: 0.5527 - acc: 0.7888 - val_loss: 0.4767 - val_acc: 0.9355
Epoch 4/5
61/61 [==============================] - 38s 616ms/step - loss: 0.4516 - acc: 0.8365 - val_loss: 0.2071 - val_acc: 0.9263
Epoch 5/5
61/61 [==============================] - 38s 616ms/step - loss: 0.3208 - acc: 0.8898 - val_loss: 0.1952 - val_acc: 0.9355
```

E- Accuracy & score:

In [ ]:
```
loss , accuracy = model.evaluate(X_testStd , Y_test)
print("Accuracy score:", accuracy)
print("Loss score: ", loss)
```

```
17/17 [==============================] - 2s 110ms/step - loss: 0.1485 - acc: 0.9557
Accuracy score: 0.955719530582428
Loss score:  0.14845524728298187
```

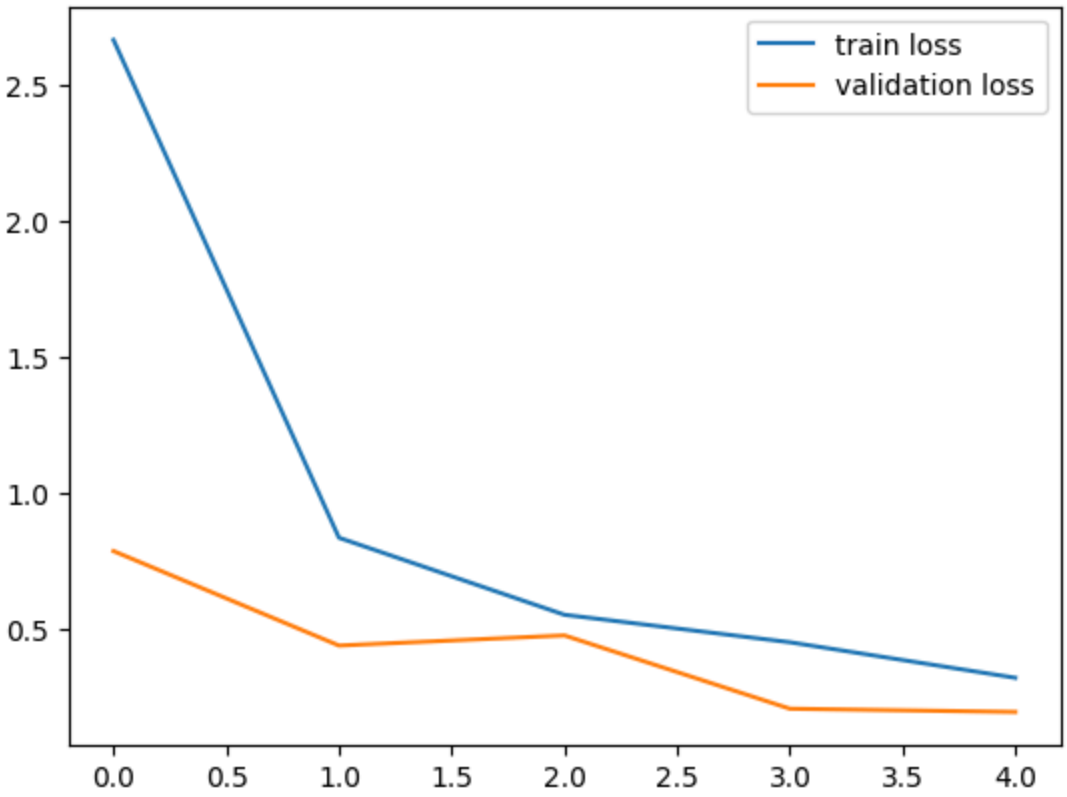
F- Visulasiation of score & accuracy:

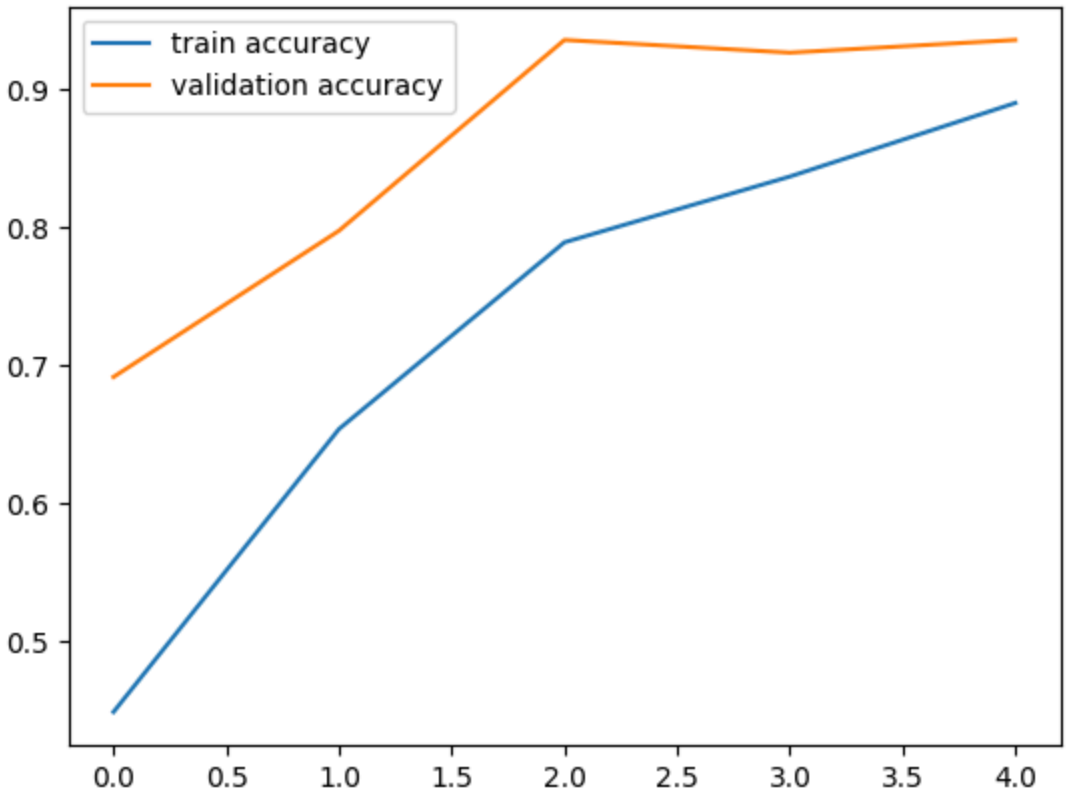
In [ ]:
```
h = history

# plot the loss value
plt.plot(h.history['loss'], label='train loss')
plt.plot(h.history['val_loss'], label='validation loss')
plt.legend()
```

```
plt.show()

# plot the accuracy value
plt.plot(h.history['acc'], label='train accuracy')
plt.plot(h.history['val_acc'], label='validation accuracy')
plt.legend()
plt.show()
```





G- Save the model:

In [ ]:
```
# After training the model, save it to a file:
model.save('path_to_my_model.h5')
```

```
c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engine\training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

8. Example:

A- Loading the model:

In [ ]:
```
model = keras.models.load_model('path_to_my_model.h5')
```

```
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.
```

In [ ]:
```
def maskPrediction(path):
    input_image = Image.open(path)
    # Redimensionner l'image à 128x128 pixels
```

```
    image_resized = input_image.resize((256, 256))
    # Convertir l'image PIL en un array numpy
    image_np = np.array(image_resized)
    # Normaliser les pixels
    image_np_normalized = image_np / 255.0
    image_reshaped = np.reshape(image_np_normalized, [1, 256, 256, 3])
    # Prédire la classe
    predict = model.predict(image_reshaped)
    print(predict)
    predict_mask = np.argmax(predict)
    # Afficher l'image originale
    plt.imshow(input_image)
    plt.show()
    print('The predicted number is:', predict_mask)
    if (predict_mask == 0):
        print("The type of road is: Highway")
    elif (predict_mask == 1):
        print("The type of road is: Regional")
    elif (predict_mask == 2):
        print("The type of road is: Tunnel")
    elif (predict_mask == 3):
        print("The type of road is: Urbain")

# Exemple d'utilisation
path = "data/Class2/14.png"
maskPrediction(path)
```

```
1/1 [==============================] - 0s 160ms/step
[[0.24879716 0.7094808  1.         0.54563075]]
```


```
The predicted number is: 2
The type of road is: Tunnel
```