Model Training Manual

Via JupyterNotebook :

1. Import necessary libraries

1. Imports

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# file management imports
import os  ### only for count of images from dir, can be removed later

# model imports for deep learning
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Activation
import keras
from keras import optimizers
from tensorflow.keras.utils import to_categorical

# image processing imports
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,  img_to_array, load_img

from sklearn import metrics
#for confusion matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
import itertools


#for displaying images when predicting class
from PIL import Image, ImageOps
#for rounding up fitting model for steps_per_epoch
import math

from keras.models import load_model

from xplique.attributions import GradientInput
```

2. File management

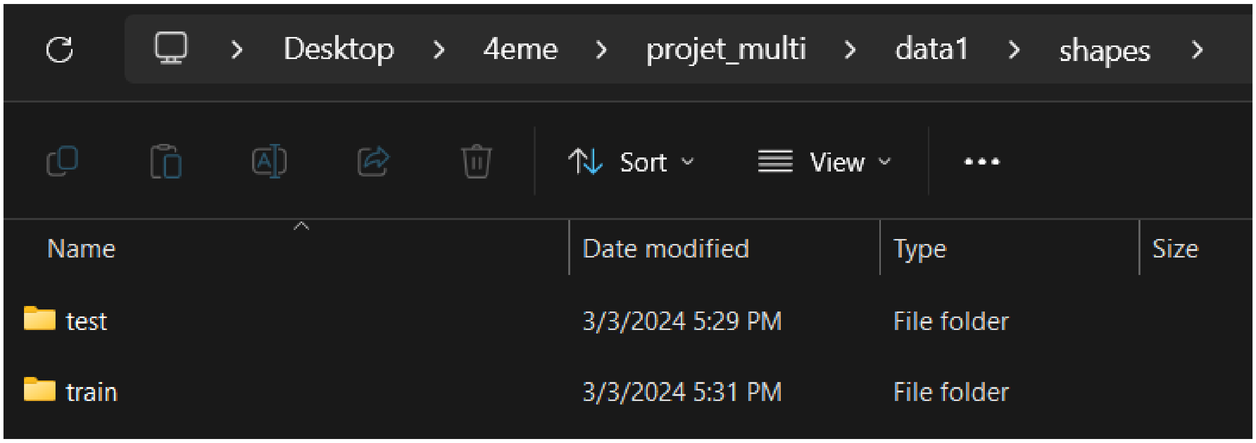
2. Loading data

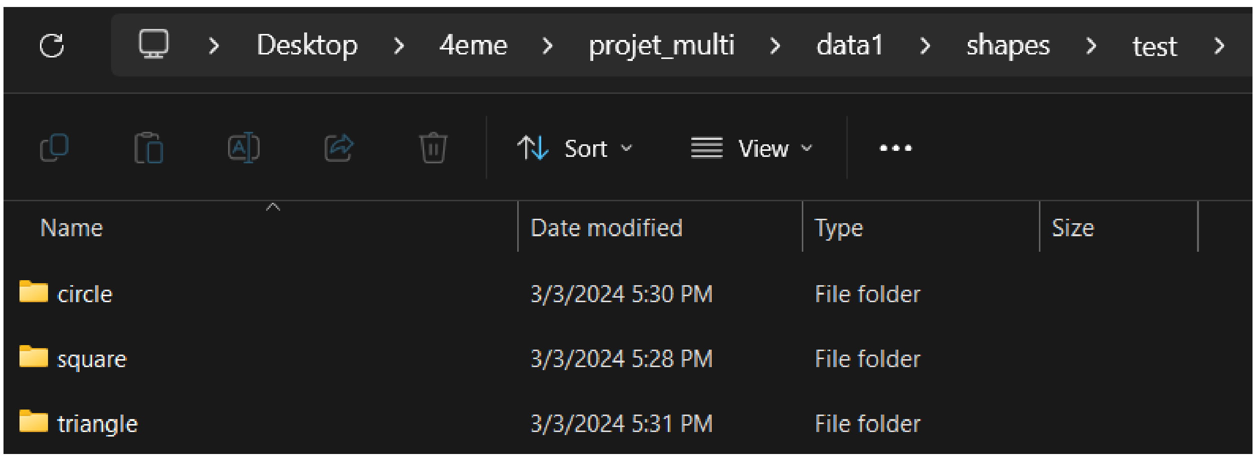
```python
#Loading train and test datasets from directory ../shapes
train_dir = "C:/Users/ACER/OneDrive/Desktop/4eme/projet_multi/data1/shapes/train"
test_dir = "C:/Users/ACER/OneDrive/Desktop/4eme/projet_multi/data1/shapes/test"


#Location of train and test sets of circle
train_dir_circle = train_dir+"/circle"
test_dir_circle = test_dir+"/circle"
#Location of train and test sets of square
train_dir_square = train_dir+"/square"
test_dir_square = test_dir+"/square"
#Location of train and test sets of triangle
train_dir_triangle = train_dir+"/triangle"
test_dir_triangle = test_dir+"/triangle"
#Location of train and test sets of stars
# train_dir_stars = train_dir+'/stars/JPEG'
# test_dir_stars = test_dir+'/stars'

### i used a terminal command for spliting the stars dataset into train and test sets ### similar to this - %mv $(ls | sort -R | head -920) (directory)
```

Make sure your data correspond well to the loaded path.





The folders' name in test and train folders have to be the same and with the same order.

You can check the number of images in each set.

3. EDA

```python
#checking the amount of images in each set
print('Total number of images in circle training set: ', len(os.listdir(train_dir_circle)))
print('Total number of images in circle test set:     ', len(os.listdir(test_dir_circle)))
print('Total number of images in square training set: ', len(os.listdir(train_dir_square)))
print('Total number of images in square test set:     ', len(os.listdir(test_dir_square)))
print('Total number of images in triangle training set:', len(os.listdir(train_dir_triangle)))
print('Total number of images in triangle test set:     ', len(os.listdir(test_dir_triangle)))
# print('Total number of images in stars training set:', len(os.listdir(train_dir_stars)))
# print('Total number of images in stars test set:     ', len(os.listdir(test_dir_stars)))
```

```
Total number of images in circle training set:  3348
Total number of images in circle test set:      372
Total number of images in square training set:  3372
Total number of images in square test set:      393
Total number of images in triangle training set: 3324
Total number of images in triangle test set:      396
```

3. *Data generator and fit to training, testing set*

Generator for Image Augmentation.

```python
data_generator  = ImageDataGenerator(
                            rescale = 1.0/255.0,
                            fill_mode = 'nearest',
                            width_shift_range=0.3,
                            height_shift_range=0.3,
                            zoom_range = 0,      # can randomly apply a zoom to the image (from 0 to 20%)
                            rotation_range = 90,   # can randomly apply a rotation to the image (from 0 to 45 degrees)
                            horizontal_flip=True,  # can randomly flips the image on the x-axis
                            vertical_flip=True)
```

Data generator created a virtual data only for the training process (no impact on local files). The document for data generator can be found on Tensorflow document.

Fit generator to training and testing sets

```
# classes= ['Circle','Square','Triangle', 'Stars']
batch_size = 64

#making tensorflow for training data
training_data = data_generator.flow_from_directory(directory= train_dir,       # dataset
                                                    target_size = (128,128),     # desired dimensions for the images
                                                    batch_size = batch_size,    # it will be run in groups of(batch_size specified above)
                                                    class_mode = 'categorical', # type of classifiaction
                                                    color_mode='rgb',
                                                    shuffle=False)
#making tensorflow for testing data
testing_data = data_generator.flow_from_directory(directory= test_dir,
                                                    target_size = (128,128),
                                                    batch_size = batch_size,
                                                    class_mode = 'categorical',
                                                 color_mode='rgb',
                                                    shuffle=False)
training_data.image_shape
```

```
Found 10044 images belonging to 3 classes.
Found 1161 images belonging to 3 classes.

(128, 128, 3)
```

```
# preparing model for final layer in the Neural Net
num_classes = len(set(training_data.classes))
num_classes
```

```
3
```

Load the dataset with the right *directory* parameter input. *Target_size* and *color_mode ('rgb' or 'grayscale')* should correspond to the input layer of the model. Loading the dataset with *class_mode* = '*categorical'* means that the classes are labeled based on the order of the dataset folders.

4. Build the model

The example of building a sequential model is as following.

Building CNN model

```
model = Sequential()
    #first block
#first convolutional layer
model.add(Conv2D(filters = 8,          # first layer should have the lowest filter size
                 kernel_size = (4, 4),  # kernel_size = (3, 3) b/c image size smaller then 128x128
                 activation = 'relu',
                 padding="same",
                 input_shape = training_data.image_shape))  # first layer needs a input_shape (set it to size of training image)
                 # input_shape = (128,128,3)))

#second convolutional layer
model.add(Conv2D(filters = 8, kernel_size = (3,3), padding="same",activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2)))  #divides 1 pixel into 4 (2x2)
model.add(Dropout(rate = 0.2))
    #second block
#third convolutional layer
model.add(Conv2D(filters = 16, kernel_size = (3, 3), padding="same", activation = 'relu'))
#fourth convolutional layer
model.add(Conv2D(filters = 16, kernel_size = (3, 3), padding="same", activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2)))
model.add(Dropout(rate = 0.2))
    #third block
#fifth convolutional layer
model.add(Conv2D(filters = 8, kernel_size = (3, 3), padding="same", activation = 'relu'))
#last convolutional layer (last layer should have the highest filter size)
model.add(Conv2D(filters = 8, kernel_size = (3, 3), padding="same", activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2), strides=(2,2)))
model.add(Dropout(rate = 0.3))
# this converts our 3D feature maps to 1D feature vectors
model.add(Flatten())

model.add(Dense(units = 64, activation = 'relu'))
model.add(Dropout(rate = 0.5))

model.add(Dense(units = 64, activation = 'relu'))
model.add(Dropout(rate = 0.5))


#output layer
model.add(Dense(units = num_classes, activation = 'softmax'))
#optimizer for compiler
keras.optimizers.Adam(learning_rate=1e-5)

model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['accuracy','mse'])
```

The model is built by first calling *tensorflow.keras.model.Sequential().* Then we add the necessary layer to it, the layers are linearly stacked on each other. At the end, we compile the model and calibrate the learning rate if needed.

Call *model.summary()* to verity its architecture.

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 128, 128, 8)	392
conv2d_7 (Conv2D)	(None, 128, 128, 8)	584
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 8)	0
dropout_5 (Dropout)	(None, 64, 64, 8)	0
conv2d_8 (Conv2D)	(None, 64, 64, 16)	1,168
conv2d_9 (Conv2D)	(None, 64, 64, 16)	2,320
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 16)	0
dropout_6 (Dropout)	(None, 32, 32, 16)	0
conv2d_10 (Conv2D)	(None, 32, 32, 8)	1,160
conv2d_11 (Conv2D)	(None, 32, 32, 8)	584
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 8)	0
dropout_7 (Dropout)	(None, 16, 16, 8)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_3 (Dense)	(None, 64)	131,136
dropout_8 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 64)	4,160
dropout_9 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 3)	195

Total params: 141,699 (553.51 KB)

Trainable params: 141,699 (553.51 KB)

Non-trainable params: 0 (0.00 B)

5. Train the model

```
total_training_size = len(os.listdir(train_dir_circle))+len(os.listdir(train_dir_square))+len(os.listdir(train_dir_triangle))
total_testing_size = len(os.listdir(test_dir_circle))+len(os.listdir(test_dir_square))+len(os.listdir(test_dir_triangle))
print(f"Total # of images in training set: {total_training_size}")
print(f"Total # of images in testing set: {total_testing_size}")
```

```
Total # of images in training set: 10044
Total # of images in testing set: 1161
```

```
#Trains the model on the data batch-by-batch (bit-by-bit)
fitted_model = model.fit(training_data,
                         epochs = 50,
                         validation_data = (testing_data))
```

Input the number of epochs.

6. Save and load the model

```
model.save('C:/Users/ACER/OneDrive/Desktop/4eme/projet_multi/models/Model2.h5')  # save your weights and model after training
```

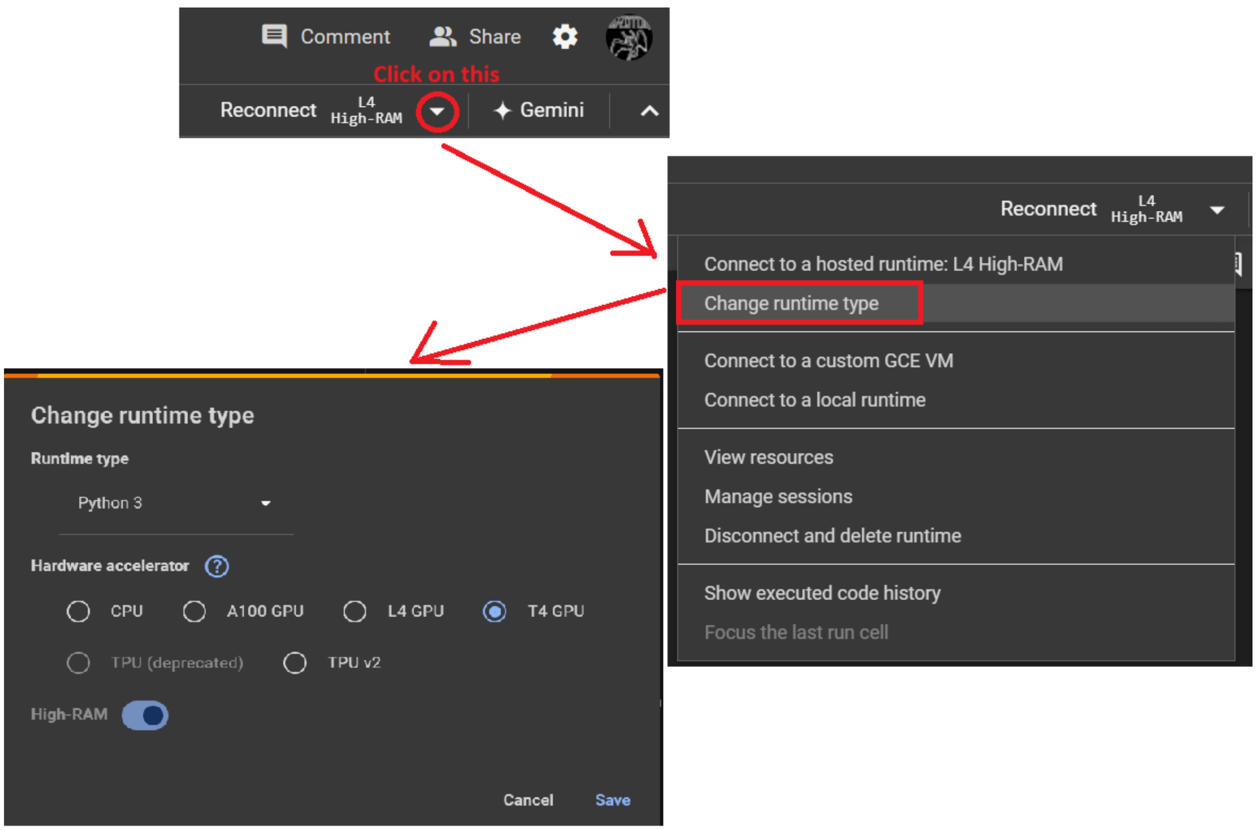
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

```
model_load = tf.keras.models.load_model('C:/Users/ACER/OneDrive/Desktop/4eme/projet_multi/models/Model2.h5')
```

Via Google Colab :

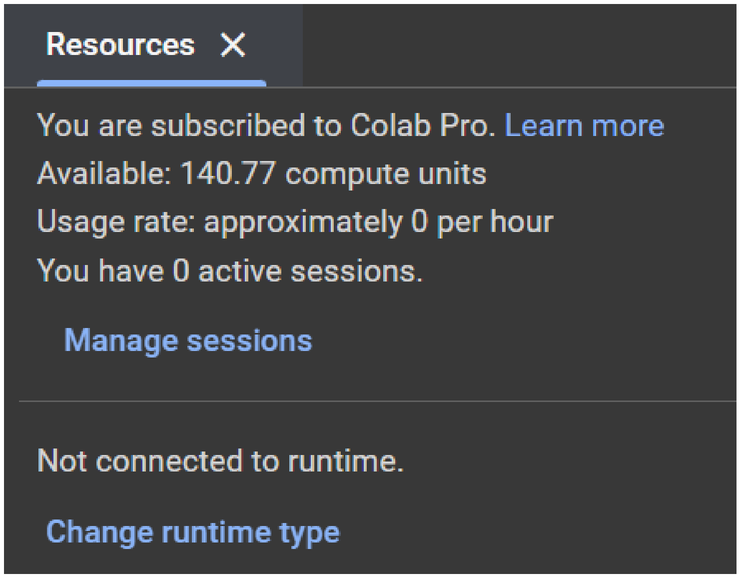
The training and building process via Google Colab follow the same method as the Jupyter one. However the environment set up and file management is a bit different. Google Colab provide cloud computing which means the code is run without any ressource from your own machine.

1. *Set up computational hardware*



The free version only gives you the access to CPU and T4 GPU without High-RAM mode.
The CPU option is totally free. However, the GPU option cost some compute units. T4 GPU takes around 1,5 unit/h, L4 takes around 4,5 units/h, A100 GPU and TPU v2 take around 14 units/h. Free version give 10 units, and paid version give 100 unit expired in 100 days at a cost of 11 euro/month.

To track your computational unit, click on *View resources.*



Since the file routing in Colab is very complicated, the first epoch usually takes a lot of time to load the dataset into the virtual Notebook's environment. To fix this, we zip the dataset and manually load it directly into the virtual environment before training the model.

To do this, first we connect to the Drive

```
drive.mount('/content/drive')

Mounted at /content/drive
```

Next, we zip the dataset and up load it on Google Drive. The dataset management and order of the folder are identical to the previous step. Then, we direct to the folder containing the Notebook and unzip the dataset

```
# Path to the folder containing this Notebook
! cd "/content/drive/My Drive/Colab_Notebooks/"
# Path to the zipped dataset
! unzip "/content/drive/My Drive/Colab_Notebooks/IMGDB_data_8_classes.zip"
```

2. *Build a Functional Model*

The difference between a Functional Model and a Sequential one can be analogically considered as the difference between pointers and table in C/C++.

```python
input = Input(shape=(256, 256, 1))

x = layers.Conv2D(16, kernel_size=(3, 3), activation='relu', padding = 'same',
                  input_shape=(256, 256, 1), kernel_regularizer=regularizers.L2(0.01))(input)
x = layers.Conv2D(16, kernel_size=(3, 3), activation='relu', padding = 'same')(x)
x = layers.MaxPool2D((2, 2), strides = 2)(x)

x = layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding = 'same')(x)
x = layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding = 'same')(x)
x = layers.MaxPool2D((2, 2), strides = 2)(x)

x = layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding = 'same')(x)
x = layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding = 'same')(x)
x = layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding = 'same')(x)
x = layers.MaxPool2D((2, 2), strides = 2)(x)


x = layers.Flatten()(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(8, activation='softmax')(x)

model = models.Model(input, x)


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

Firstly, we call an Input layer with *keras.Input()*. Then we call a layer named *x* and connect this layer to the Input layer as shown in the image. Next, we build layers on this layer *x* as the same way between *x* and Input. In the example, we only name the layer *x* to simplify the code. You can name the layer and connect them differently as you want. We finish the model by calling *tensorflow.keras.models.Model(Input layer, Model architecture)*.

The rest is identical to the JupyterNotebook.

3. Download the Model

Wait some time till the download is finished.

```
model.save('Model_256x256x1_IMGDB_8_classes.h5')
```

```
ing.py:3103: UserWarning: You are saving your model as a
```

```
from google.colab import files
files.download('Model_256x256x1_IMGDB_8_classes.h5')
```

For more information, please check our Notebook and Tensorflow keras documentation.