Data Importing

Importing Datasets from Google Drive

```
from google.colab import drive

drive.mount('/content/gdrive')
root_path = 'gdrive/My Drive/Datasets/COMPLETED DATASETS TTV CLASSES/'  #change dir to you
```

```
Mounted at /content/gdrive
```

Check the Root directory

```
print(root_path)
```

```
gdrive/My Drive/Datasets/COMPLETED DATASETS TTV CLASSES/
```

Initialize variable for directories (Training, Testing, and Validation)

```
train_data_dir = root_path + "Train"
#gdrive/My Drive/Datasets/COMPLETED DATASETS TTV CLASSES/Train/

valid_data_dir = root_path + "Validation"
#gdrive/My Drive/Datasets/COMPLETED DATASETS TTV CLASSES/Validation/

test_data_dir = root_path + "Test"
#gdrive/My Drive/Datasets/COMPLETED DATASETS TTV CLASSES/Test/
```

Initialized Image Size

```
img_height, img_width = (224,224)
# img_height = 224
# img_width = 224
```

Initialized Batch Size

The batch size defines the number of samples that will be propagated through the network.

For instance, let's say you have 1050 training samples and you want to set up a batch_size equal to 100. The algorithm takes the first 100 samples (from 1st to 100th) from the training dataset and trains the network. Next, it takes the second 100 samples (from 101st to 200th) and trains the network again. We can keep doing this procedure until we have propagated all samples through of the network. Problem might happen with the last set of samples. In our example, we've used 1050 which is not divisible by 100 without remainder. The simplest solution is just to get the final 50 samples and train the network.

Advantages of using a batch size < number of all samples:

- It requires less memory. Since you train the network using fewer samples, the overall training procedure requires less memory. That's especially important if you are not able to fit the whole dataset in your machine's memory.
- Typically networks train faster with mini-batches. That's because we update the weights after each propagation. In our example we've propagated 11 batches (10 of them had 100 samples and 1 had 50 samples) and after each of them we've updated our network's parameters. If we used all samples during propagation we would make only 1 update for the network's parameter.

Disadvantages of using a batch size < number of all samples:

- The smaller the batch the less accurate the estimate of the gradient will be.

```
batch_size = 200
```

Initialized Epochs

```
epochs_num = 10
```

Data Augmentation

- To balance the dataset and to increase the size of training nd testing dataset
- Regularization technique to minimize overfitting
- Rescaling the image is Normalizing the image

https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844

ImageDataGenerator

Import Module

- ImageDataGenerator

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator #
```

Model preprocess_input

use as a value of preprocessing_function

Using **resnet50**

```
from tensorflow.keras.applications.resnet50 import preprocess_input
```

Using **VGG16**

```
#from keras.applications.vgg16 import preprocess_input
```

Using ImageDataGenerator

Parameters:

- horizontal_flip=True,
- preprocessing_function=preprocess_input,
- shear_range=0.2,
- zoom_range=0.2,
- validation_split=0.4
- brightness_range=(0.1, 0.6),
- height_shift_range=0.3,
- width_shift_range=0.3,
- fill_mode='reflect',

Image data generator for Training

```
# training ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   width_shift_range=0.3,
                                   fill_mode='reflect',
                                   height_shift_range=0.3,
                                   brightness_range=(0.1, 0.6),
                                   shear_range=45.0,
                                   zoom_range=[0.5, 1.5])

  # Generator from Training Directory
train_generator = train_datagen.flow_from_directory(train_data_dir,
                                                    target_size=(img_height, img_width),
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    subset='training') #set as training da

```

```
Found 7180 images belonging to 9 classes.
```

Image data generator for Validation

```
# validation ImageDataGenerator
valid_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   width_shift_range=0.3,
                                   fill_mode='reflect',
                                   height_shift_range=0.3,
                                   brightness_range=(0.1, 0.6),
                                   shear_range=45.0,
                                   zoom_range=[0.5, 1.5],
                                   validation_split=0.99)

  # Generator from Validation Directory
valid_generator = valid_datagen.flow_from_directory(valid_data_dir,
                                                    target_size=(img_height, img_width),
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=False,
                                                    subset='validation')#set as validatior

```

```
Found 891 images belonging to 9 classes.
```

Image data generator for Validation

```
# testing ImageDataGenerator
test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input,
                                   width_shift_range=0.3,
                                   fill_mode='reflect',
                                   height_shift_range=0.3,
                                   brightness_range=(0.1, 0.6),
                                    shear_range=45.0,
                                    zoom_range=[0.5, 1.5],
                                    validation_split=0.99)

  # Generator from Testing Directory
test_generator = test_datagen.flow_from_directory(test_data_dir,
                                                  target_size=(img_height, img_width),
                                                  batch_size=batch_size, # dati 1 lang r
                                                  class_mode='categorical',
                                                  shuffle=False,
                                                  subset='validation') # set as testing
```

```
Found 891 images belonging to 9 classes.
```

To print number of batches per epoch

```
print('Number of Batches per epoch train=%d, valid=%d ,test=%d' % (len(train_generator), l
```

```
Number of Batches per epoch train=36, valid=5 ,test=5
```

Double-click (or enter) to edit

```
x,y = test_generator.next()
x.shape
```

```
(200, 224, 224, 3)
```

Getting the number of classes

```
train_generator.num_classes
```

```
9
```

Import Model

Using **ResNet50**

```
from tensorflow.keras.applications.resnet50 import ResNet50

base_model = ResNet50(input_shape=[img_height, img_width] + [3], weights='imagenet', inclu
#base_model = ResNet50(input_shape = (224, 224, 3), include_top = False, weights = 'imager
```

Using **VGG16**

```
#from keras.applications.vgg16 import VGG16

#base_model = VGG16(input_shape=[img_height, img_width] + [3], weights='imagenet', include
```

Note:

[img_height, img_width] + [3]

is the same as

[224, 224, 3]

Layers

```
x = base_model.output

from tensorflow.keras.layers import GlobalAveragePooling2D
x = GlobalAveragePooling2D()(x)

## our layers - you can add more if you want

from tensorflow.keras.layers import Flatten
x = Flatten()(base_model.output) #added

from tensorflow.keras.layers import Dense
x = Dense(1024, activation='relu')(x)

from tensorflow.keras.layers import Dropout
x = Dropout(0.5)(x)

x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(train_generator.num_classes, activation='softmax')(x)
```

???

```
#from tensorflow.keras.preprocessing.image import load_img
#from tensorflow.keras.applications.resnet50 import decode_predictions

#from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization

#
#from tensorflow.keras.preprocessing import image
```

```
#from tensorflow.keras.models import Sequential

#import numpy as np
```

Create Model

```
from tensorflow.keras.models import Model

# create a model object
model = Model(inputs=base_model.input, outputs=predictions) #transfer learning model
```

```
# don't train existing weights
for layer in base_model.layers:
    layer.trainable = False
```

```
# tell the model what cost and optimization method to use
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics = ['accuracy']
)
# model.compile(optimizer = Adam(learning_rate=0.000001), loss = 'categorical_crossentropy
```

```
# view the structure of the model
# model.summary()
```

Train Model

```
myModel = model.fit(train_generator,epochs = epochs_num,validation_data = valid_generator
```

Save the Model

```
model.save('ResNet50-1-epochs.h5')
```

Visualization

Accuracy

```python
import matplotlib.pyplot as plt

plt.plot(myModel.history['accuracy'])
plt.plot(myModel.history['val_accuracy'])
plt.legend(['training','validation'])
plt.title("Accuracy")
```

Loss

```python
plt.plot(myModel.history['loss'])
plt.plot(myModel.history['val_loss'])
plt.legend(['training','validation'])
plt.title("Loss")
```

Test Model

```python
test_loss, test_acc = model.evaluate(test_generator, verbose=2)
print('\nTest accuracy:', test_acc)
```

Save Model as JSON

```python
model_json = model.to_json()
```

```python
with open("gdrive/My Drive/resnet50-10-epocs-model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("gdrive/My Drive/resnet50-10-epocs-model_weights.h5")
print("Saved model to drive")
```

Validate Model

Confusion Matrix