```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Fri Mar 18 15:05:20 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   62C    P8    11W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
#Import Stuff here

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import tensorflow.keras as keras
from tensorflow.keras import layers, Model, optimizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, Glo
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
import numpy as np
import time
from datetime import datetime
from tensorflow.keras.utils import plot_model
from tensorflow.keras.optimizers import RMSprop



from numpy.random import seed
seed(1)
import tensorflow
```

```
import tensorflow
tensorflow.random.set_seed(2)
```

```
from google.colab import drive, files

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
def load_data():

    datagen = ImageDataGenerator(rescale = 1/255)

    train_it = datagen.flow_from_directory ('/content/drive/My Drive/Colab Notebooks/I
                                            class_mode = 'categorical', color_mode="rg

    val_it = datagen.flow_from_directory ('/content/drive/My Drive/Colab Notebooks/IFI
                                            class_mode = 'categorical', color_mode="rg

    test_it = datagen.flow_from_directory ('/content/drive/My Drive/Colab Notebooks/II
                                            class_mode = 'categorical', color_mode="rg

    return train_it, val_it, test_it

train_it, val_it, test_it = load_data()
```

```
Found 13600 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
```

```
batchX, batchy = train_it.next()
print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()
```

```
Batch shape=(64, 256, 256, 3), min=0.000, max=1.000
```

```
from keras.applications.vgg16 import VGG16
# load model
#model = VGG16()
# summarize the model
#model.summary()
```

```
# Fine-tune
# Part 1: Pre train

nclass = len(train_it.class_indices)
```

```
# create the base pre-trained model
base_model = VGG16(input_shape = (224, 224, 3), # Shape of our images
                          include_top = False, # Leave out the last fully connected La
                          weights = 'imagenet',
                          classes = nclass)
base_model.trainable = False

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
predictions = Dense(nclass, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional layers
# for layer in base_model.layers:
#     layer.trainable = False


# model.compile(loss='categorical_crossentropy',
#               # optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),
#               optimizer = 'Adam',
#               metrics=['AUC', 'accuracy'])
model.summary()
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
```

```
def compile_model(model):

  model.compile(optimizer = 'Adam',
              loss=keras.losses.CategoricalCrossentropy(),
              metrics=['AUC', 'accuracy'])

  return model

def train_model(model, train_it, val_it):

    callback = keras.callbacks.EarlyStopping(monitor='loss', patience = 5, restore_bes
    history = model.fit(train_it, epochs = 24, steps_per_epoch = 107, verbose = 2, val
    return model, history

model = compile_model(model)

model, history = train_model(model, train_it, val_it)
print (model.summary())
```

```
Epoch 1/24
107/107 - 4077s - loss: 2.5861 - auc: 0.6950 - accuracy: 0.1907 - val_loss: 2
Epoch 2/24
107/107 - 1381s - loss: 2.1175 - auc: 0.8364 - accuracy: 0.3303 - val_loss: 1
Epoch 3/24
107/107 - 763s - loss: 1.8767 - auc: 0.8780 - accuracy: 0.3881 - val_loss: 1.
Epoch 4/24
107/107 - 469s - loss: 1.7380 - auc: 0.8970 - accuracy: 0.4340 - val_loss: 1.
Epoch 5/24
107/107 - 295s - loss: 1.6371 - auc: 0.9089 - accuracy: 0.4673 - val_loss: 1.
Epoch 6/24
107/107 - 222s - loss: 1.5821 - auc: 0.9155 - accuracy: 0.4827 - val_loss: 1.
Epoch 7/24
107/107 - 185s - loss: 1.5228 - auc: 0.9225 - accuracy: 0.5009 - val_loss: 1.
Epoch 8/24
107/107 - 157s - loss: 1.4554 - auc: 0.9296 - accuracy: 0.5171 - val_loss: 1.
Epoch 9/24
107/107 - 156s - loss: 1.4157 - auc: 0.9339 - accuracy: 0.5251 - val_loss: 1.
Epoch 10/24
107/107 - 151s - loss: 1.3959 - auc: 0.9356 - accuracy: 0.5353 - val_loss: 1.
Epoch 11/24
107/107 - 149s - loss: 1.3262 - auc: 0.9426 - accuracy: 0.5549 - val_loss: 1.
Epoch 12/24
107/107 - 146s - loss: 1.3129 - auc: 0.9435 - accuracy: 0.5560 - val_loss: 1.
Epoch 13/24
107/107 - 145s - loss: 1.3019 - auc: 0.9447 - accuracy: 0.5625 - val_loss: 1.
Epoch 14/24
107/107 - 145s - loss: 1.2532 - auc: 0.9487 - accuracy: 0.5803 - val_loss: 1.
Epoch 15/24
107/107 - 145s - loss: 1.2501 - auc: 0.9489 - accuracy: 0.5717 - val_loss: 1.
Epoch 16/24
107/107 - 145s - loss: 1.1969 - auc: 0.9538 - accuracy: 0.5936 - val_loss: 1.
Epoch 17/24
107/107 - 146s - loss: 1.1982 - auc: 0.9530 - accuracy: 0.5898 - val_loss: 1.
Epoch 18/24
107/107 - 147s - loss: 1.1690 - auc: 0.9556 - accuracy: 0.6002 - val_loss: 1.
Epoch 19/24
107/107 - 145s - loss: 1.1300 - auc: 0.9590 - accuracy: 0.6170 - val_loss: 1.
Epoch 20/24
107/107 - 146s - loss: 1.1338 - auc: 0.9585 - accuracy: 0.6081 - val_loss: 1.
Epoch 21/24
107/107 - 145s - loss: 1.0995 - auc: 0.9608 - accuracy: 0.6250 - val_loss: 1.
Epoch 22/24
107/107 - 147s - loss: 1.0882 - auc: 0.9619 - accuracy: 0.6291 - val_loss: 1.
Epoch 23/24
107/107 - 146s - loss: 1.0737 - auc: 0.9627 - accuracy: 0.6376 - val_loss: 1.
Epoch 24/24
107/107 - 147s - loss: 1.0346 - auc: 0.9655 - accuracy: 0.6455 - val_loss: 1.
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
```

```
| (None, 56, 56, 256) | (None, 56, 56, 256) |
```

```
def eval_model(model, test_it):
    # TODO: evaluate the model

    test_loss, test_auc, test_accuracy = model.evaluate (test_it, steps = 27)

    return test_loss, test_auc, test_accuracy

test_loss, test_auc, test_accuracy = eval_model(model, test_it)
```

```
27/27 [==============================] - 1333s 51s/step - loss: 1.8299 - auc: 0.
```

```
test_it.reset()
preds = model.predict(test_it, steps = 27)


y_predict = np.argmax(preds,axis=1)


print(classification_report(test_it.classes, y_predict, target_names=test_it.class_inc
```

```
              precision    recall  f1-score   support

        1977       0.38      0.58      0.46       100
       Amaro       0.35      0.49      0.41       100
     Brannan       0.48      0.36      0.41       100
   Clarendon       0.28      0.53      0.36       100
     Gingham       0.38      0.27      0.31       100
       He-Fe       0.50      0.22      0.31       100
      Hudson       0.37      0.45      0.40       100
       Lo-Fi       0.33      0.40      0.36       100
     Mayfair       0.20      0.17      0.19       100
   Nashville       0.80      0.85      0.83       100
    Original       0.27      0.15      0.19       100
    Perpetua       0.42      0.50      0.45       100
       Sutro       0.64      0.54      0.59       100
     Toaster       0.76      0.74      0.75       100
    Valencia       0.37      0.18      0.24       100
      Willow       0.96      0.90      0.93       100
     X-ProII       0.53      0.48      0.50       100

    accuracy                           0.46      1700
   macro avg       0.47      0.46      0.45      1700
weighted avg       0.47      0.46      0.45      1700
```

```
y_predict
```

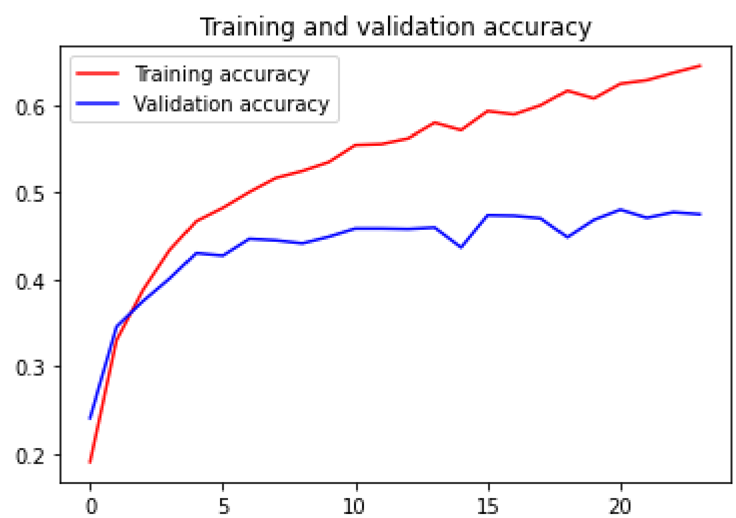
```
array([ 6,  0,  0, ..., 16, 12, 16])
```

```
error_count = 0

for x in range (0,len(y_predict)):
    if y_predict[x] != test_it.classes[x]:
        error_count += 1
error_count
```

```
919
```

```
def plot_result(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.plot(epochs, acc, 'r', label='Training accuracy')
    plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.legend(loc=0)
    plt.figure()
    plt.show()
```

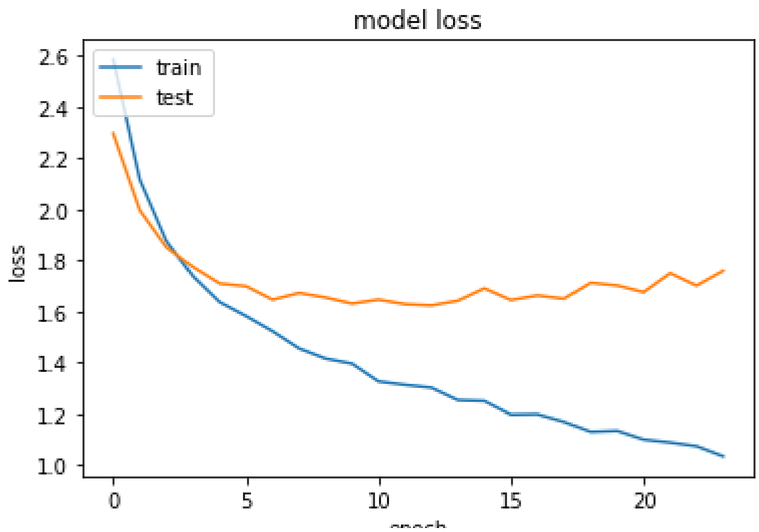
```
plot_result(history)
```




```
<Figure size 432x288 with 0 Axes>
```

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
#Fine-Tune part 2
# Visualize layer names and layer indices to see how many layers need to be freeze
for i, layer in enumerate(base_model.layers):
   print(i, layer.name)
```

```
0 input_1
1 block1_conv1
2 block1_conv2
3 block1_pool
4 block2_conv1
5 block2_conv2
6 block2_pool
7 block3_conv1
8 block3_conv2
9 block3_conv3
10 block3_pool
11 block4_conv1
12 block4_conv2
13 block4_conv3
14 block4_pool
15 block5_conv1
16 block5_conv2
17 block5_conv3
18 block5_pool
```

```
# we chose to train the top 2 blocks, try freeze the first 3 layers and unfreeze the r
for layer in model.layers[:3]:
   layer.trainable = False
for layer in model.layers[3:]:
   layer.trainable = True



# we need to recompile the model for these modifications to take effect
# try a low learning rate
```

```
model.compile(optimizer=optimizers.Adam(learning_rate = 0.0001),
              loss='categorical_crossentropy',
              metrics=['AUC', 'accuracy'])


# Train our model again (fine-tuning the top 2 blocks alongside the top Dense layers
model, history = train_model(model, train_it, val_it)
print (model.summary())
```

```
Epoch 1/24
107/107 - 158s - loss: 3.0452 - auc: 0.5826 - accuracy: 0.0822 - val_loss: 2.
Epoch 2/24
107/107 - 149s - loss: 2.0223 - auc: 0.8458 - accuracy: 0.2985 - val_loss: 1.
Epoch 3/24
107/107 - 152s - loss: 1.1837 - auc: 0.9539 - accuracy: 0.5695 - val_loss: 1.
Epoch 4/24
107/107 - 150s - loss: 0.8030 - auc: 0.9779 - accuracy: 0.7167 - val_loss: 0.
Epoch 5/24
107/107 - 150s - loss: 0.6126 - auc: 0.9871 - accuracy: 0.7849 - val_loss: 0.
Epoch 6/24
107/107 - 150s - loss: 0.5154 - auc: 0.9903 - accuracy: 0.8091 - val_loss: 0.
Epoch 7/24
107/107 - 149s - loss: 0.4852 - auc: 0.9914 - accuracy: 0.8264 - val_loss: 0.
Epoch 8/24
107/107 - 150s - loss: 0.4028 - auc: 0.9936 - accuracy: 0.8568 - val_loss: 0.
Epoch 9/24
107/107 - 150s - loss: 0.3333 - auc: 0.9951 - accuracy: 0.8787 - val_loss: 0.
Epoch 10/24
107/107 - 149s - loss: 0.2972 - auc: 0.9962 - accuracy: 0.8946 - val_loss: 0.
Epoch 11/24
107/107 - 150s - loss: 0.3063 - auc: 0.9961 - accuracy: 0.8925 - val_loss: 0.
Epoch 12/24
107/107 - 151s - loss: 0.2909 - auc: 0.9963 - accuracy: 0.8935 - val_loss: 0.
Epoch 13/24
107/107 - 151s - loss: 0.2571 - auc: 0.9970 - accuracy: 0.9039 - val_loss: 0.
Epoch 14/24
107/107 - 150s - loss: 0.2163 - auc: 0.9978 - accuracy: 0.9247 - val_loss: 0.
Epoch 15/24
107/107 - 150s - loss: 0.2049 - auc: 0.9981 - accuracy: 0.9263 - val_loss: 0.
Epoch 16/24
107/107 - 150s - loss: 0.2280 - auc: 0.9973 - accuracy: 0.9175 - val_loss: 0.
Epoch 17/24
107/107 - 151s - loss: 0.1951 - auc: 0.9979 - accuracy: 0.9338 - val_loss: 0.
Epoch 18/24
107/107 - 151s - loss: 0.1420 - auc: 0.9985 - accuracy: 0.9520 - val_loss: 0.
Epoch 19/24
107/107 - 151s - loss: 0.1122 - auc: 0.9992 - accuracy: 0.9613 - val_loss: 0.
Epoch 20/24
107/107 - 150s - loss: 0.1693 - auc: 0.9981 - accuracy: 0.9375 - val_loss: 0.
Epoch 21/24
107/107 - 149s - loss: 0.1259 - auc: 0.9990 - accuracy: 0.9553 - val_loss: 0.
Epoch 22/24
107/107 - 150s - loss: 0.1060 - auc: 0.9991 - accuracy: 0.9632 - val_loss: 0.
Epoch 23/24
107/107 - 149s - loss: 0.0829 - auc: 0.9995 - accuracy: 0.9743 - val_loss: 0.
```

```
Epoch 24/24
107/107 - 150s - loss: 0.1164 - auc: 0.9986 - accuracy: 0.9614 - val_loss: 0.
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
```

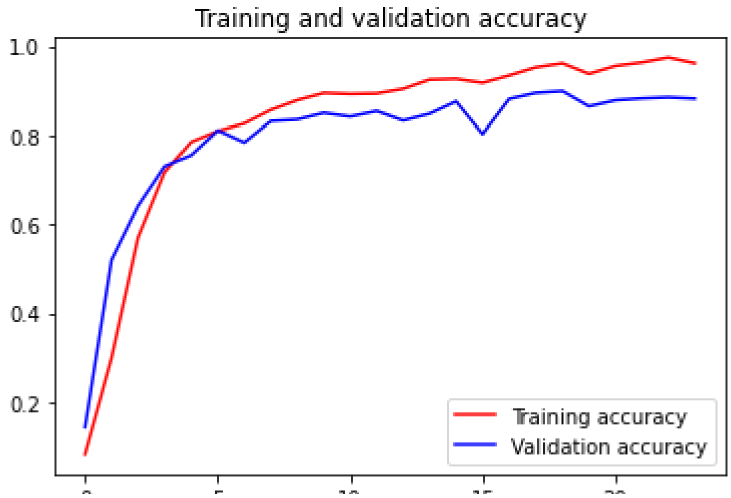
```
test_loss, test_auc, test_accuracy = eval_model(model, test_it)
```

```
27/27 [==============================] - 45s 2s/step - loss: 0.3376 - auc: 0.991
```

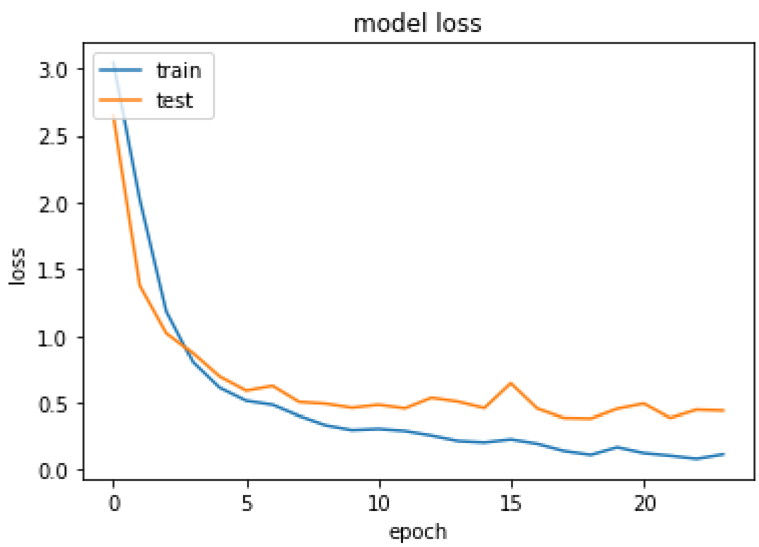
```
test_it.reset()
preds = model.predict(test_it, steps = 27)
y_predict = np.argmax(preds,axis=1)
print(classification_report(test_it.classes, y_predict, target_names=test_it.class_inc
```

```
              precision    recall  f1-score   support

        1977       0.88      0.99      0.93       100
       Amaro       0.75      0.96      0.84       100
     Brannan       0.99      0.91      0.95       100
   Clarendon       0.74      0.85      0.79       100
     Gingham       0.88      0.91      0.90       100
       He-Fe       0.98      0.95      0.96       100
      Hudson       0.98      0.89      0.93       100
       Lo-Fi       0.93      0.71      0.81       100
     Mayfair       0.75      0.93      0.83       100
   Nashville       0.95      1.00      0.98       100
    Original       0.85      0.66      0.74       100
    Perpetua       0.97      0.92      0.94       100
       Sutro       0.96      0.95      0.95       100
     Toaster       0.97      0.99      0.98       100
    Valencia       0.96      0.73      0.83       100
      Willow       1.00      1.00      1.00       100
     X-ProII       0.91      0.97      0.94       100

    accuracy                           0.90      1700
   macro avg       0.91      0.90      0.90      1700
weighted avg       0.91      0.90      0.90      1700
```

```
plot_result(history)
```




```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
model.save('/content/drive/My Drive/Colab Notebooks/VGG16.h5', overwrite = True,
           include_optimizer=True)
```