Configuration Manual

MSc Research Project
MSc in Data Analytics

Priya Prakashbhai Patel
Student ID: x20119097

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Priya Prakashbhai Patel
Student ID:	x20119097
Programme:	MSc in Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Jorge Basilio
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th December 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Forename Surname
XXX

1 Introduction

The system specification, as well as the software and hardware utilized in the project's execution, are detailed in this paper. It also outlines the measures used to carry out the research study "Automated Identification of Painters Based on Style of Art Using Algorithms for Machine Learning"

2 System Configuration

2.1 Software configuration

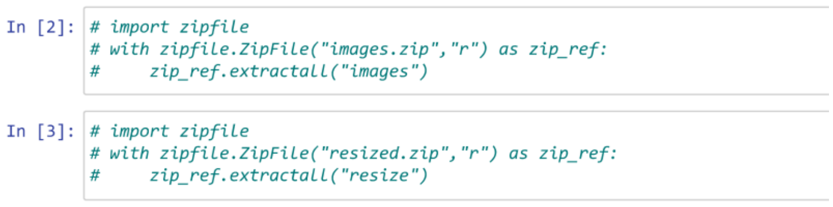
- anaconda Version 2.1.1 should be installed with jupyter notebook version 6.1.4.
- python version 3.8.5 is used in this project.

2.2 Hardware configuration

- Asus Tuf 15 with 1TB HHD, 256 GB SSD, and 8Gb RAM is used.
- intel core i5 with 2.50GHz Gpu

3 data gathering

go to kaggle.com and download the dataset in zip format. the using zipfile extarct all the data to be used in jupyter file

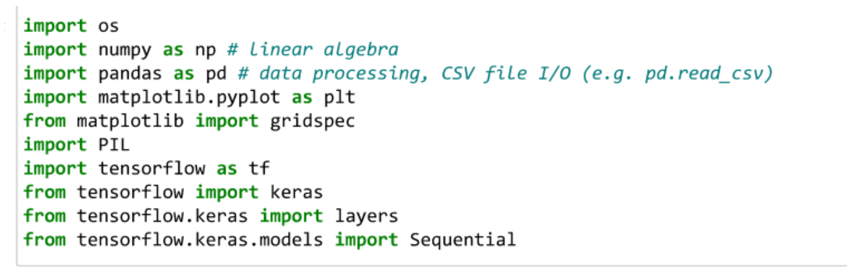


```
In [2]: # import zipfile
        # with zipfile.ZipFile("images.zip","r") as zip_ref:
        #     zip_ref.extractall("images")

In [3]: # import zipfile
        # with zipfile.ZipFile("resized.zip","r") as zip_ref:
        #     zip_ref.extractall("resize")
```

4 implementation

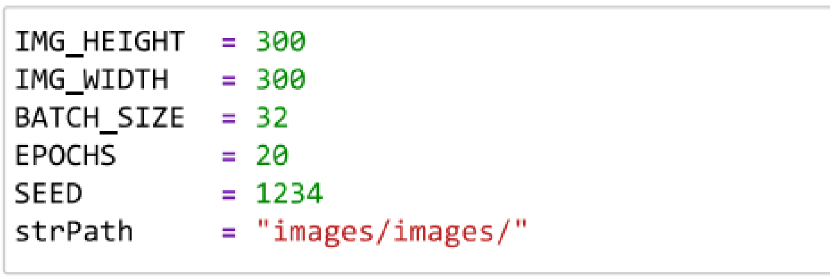
4.1 CNN model

Import all the important libraries



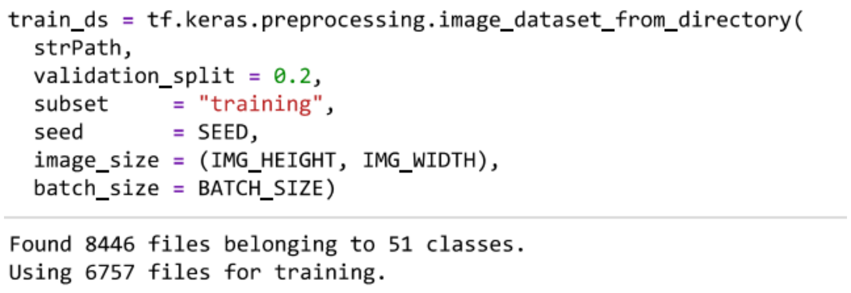
```python
import os
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from matplotlib import gridspec
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

define the image dimensions



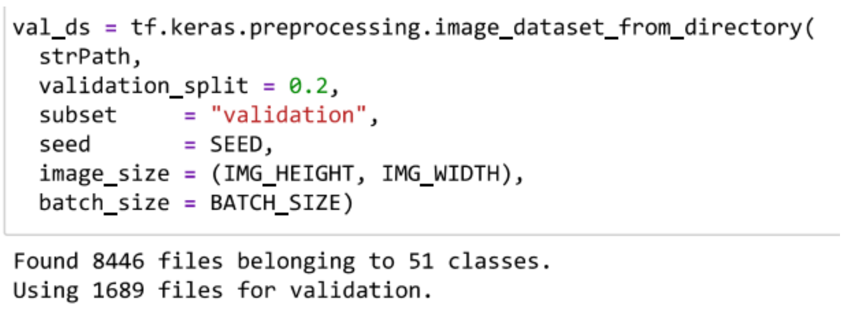
```python
IMG_HEIGHT  = 300
IMG_WIDTH   = 300
BATCH_SIZE  = 32
EPOCHS      = 20
SEED        = 1234
strPath     = "images/images/"
```

split the data for training and testing



```python
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  strPath,
  validation_split = 0.2,
  subset     = "training",
  seed       = SEED,
  image_size = (IMG_HEIGHT, IMG_WIDTH),
  batch_size = BATCH_SIZE)
```

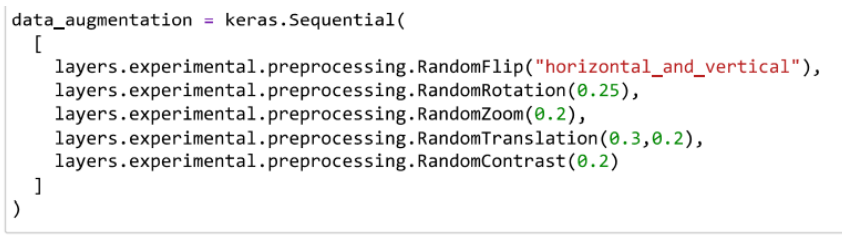
```
Found 8446 files belonging to 51 classes.
Using 6757 files for training.
```



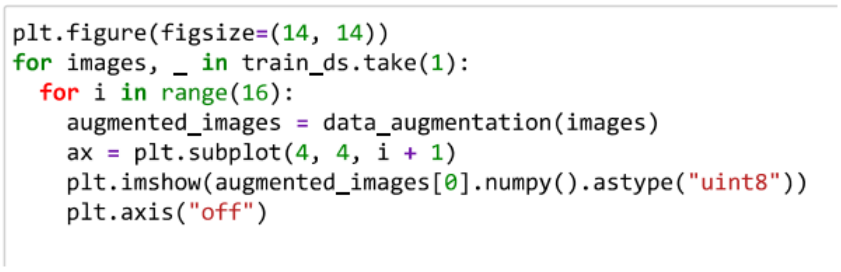
```python
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  strPath,
  validation_split = 0.2,
  subset     = "validation",
  seed       = SEED,
  image_size = (IMG_HEIGHT, IMG_WIDTH),
  batch_size = BATCH_SIZE)
```

```
Found 8446 files belonging to 51 classes.
Using 1689 files for validation.
```

data augmentation and plot those augmented images

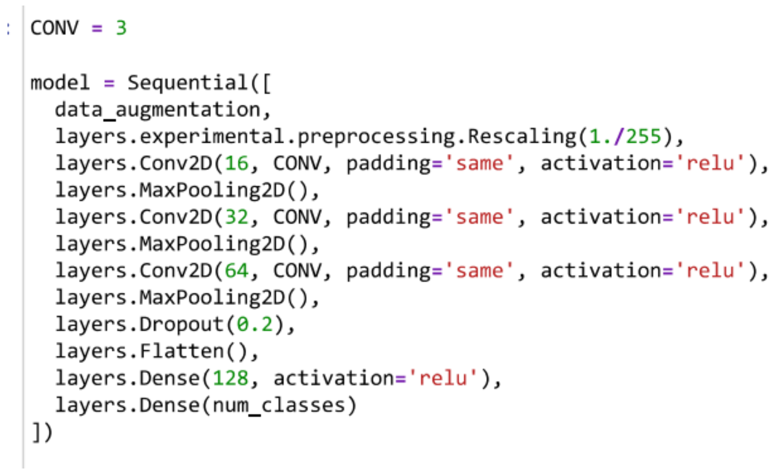


```python
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.25),
    layers.experimental.preprocessing.RandomZoom(0.2),
    layers.experimental.preprocessing.RandomTranslation(0.3,0.2),
    layers.experimental.preprocessing.RandomContrast(0.2)
  ]
)
```



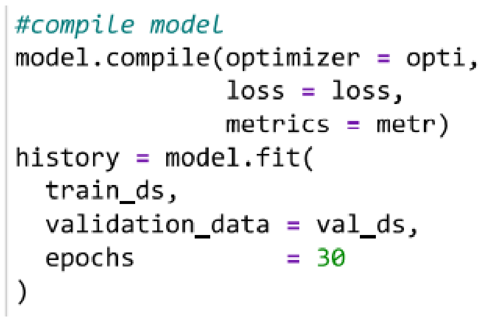
```python
plt.figure(figsize=(14, 14))
for images, _ in train_ds.take(1):
  for i in range(16):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

builds the model



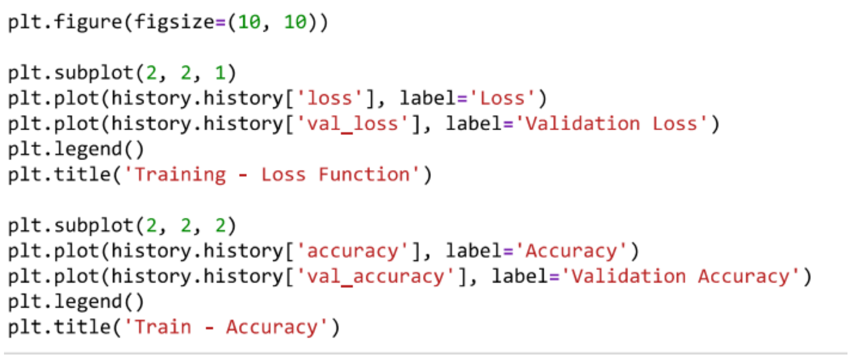
```python
CONV = 3

model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, CONV, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, CONV, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, CONV, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

run the model



```python
#compile model
model.compile(optimizer = opti,
              loss = loss,
              metrics = metr)
history = model.fit(
  train_ds,
  validation_data = val_ds,
  epochs          = 30
)
```

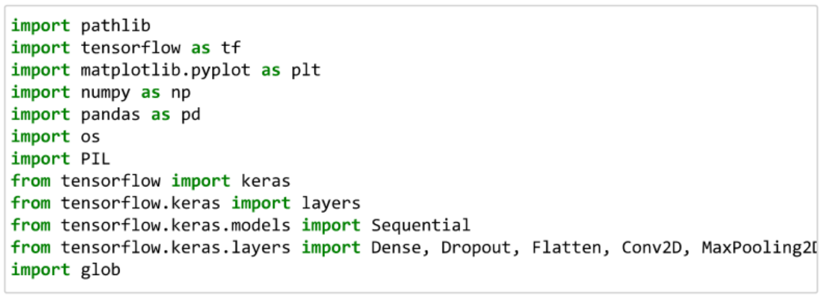
plot the training graphs



```python
plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training - Loss Function')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Train - Accuracy')
```

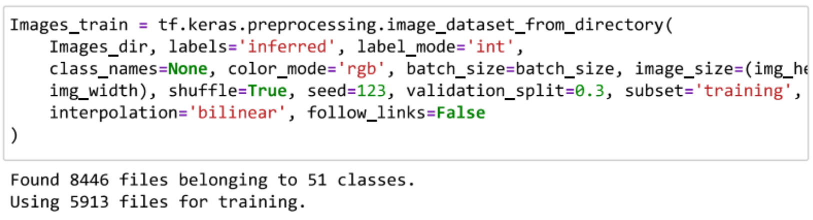
4.2 AlexNet model

Import all the necessary Libraries



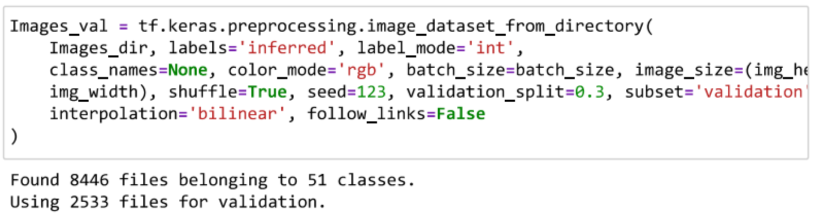
```
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import PIL
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2I
import glob
```

Split data into training and testing



```
Images_train = tf.keras.preprocessing.image_dataset_from_directory(
    Images_dir, labels='inferred', label_mode='int',
    class_names=None, color_mode='rgb', batch_size=batch_size, image_size=(img_he
    img_width), shuffle=True, seed=123, validation_split=0.3, subset='training',
    interpolation='bilinear', follow_links=False
)
```

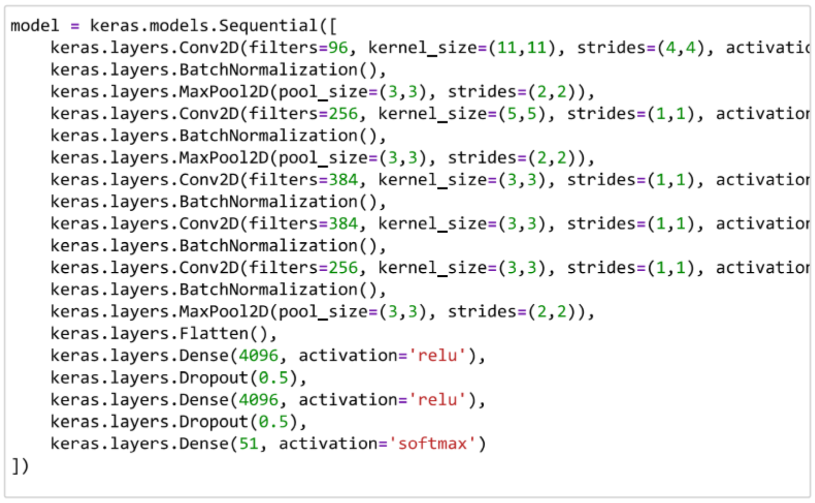
```
Found 8446 files belonging to 51 classes.
Using 5913 files for training.
```



```
Images_val = tf.keras.preprocessing.image_dataset_from_directory(
    Images_dir, labels='inferred', label_mode='int',
    class_names=None, color_mode='rgb', batch_size=batch_size, image_size=(img_he
    img_width), shuffle=True, seed=123, validation_split=0.3, subset='validation'
    interpolation='bilinear', follow_links=False
)
```

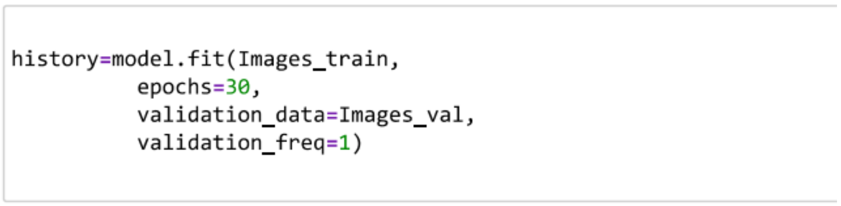
```
Found 8446 files belonging to 51 classes.
Using 2533 files for validation.
```

Build the model



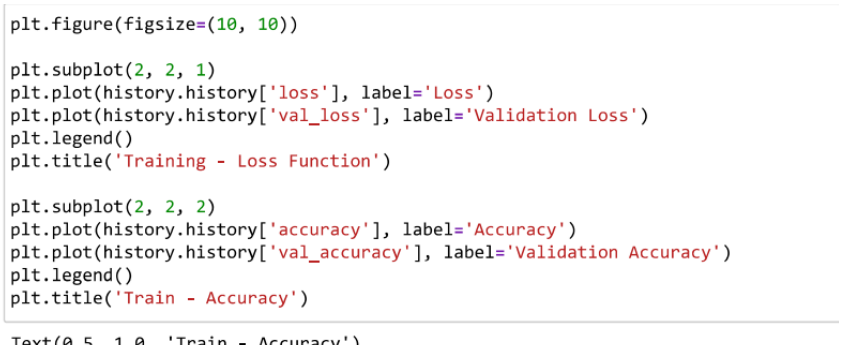
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activatic
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activatior
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activatior
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activatior
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activatior
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(51, activation='softmax')
])
```

run the model



```
history=model.fit(Images_train,
          epochs=30,
          validation_data=Images_val,
          validation_freq=1)
```

the the training and testing curves

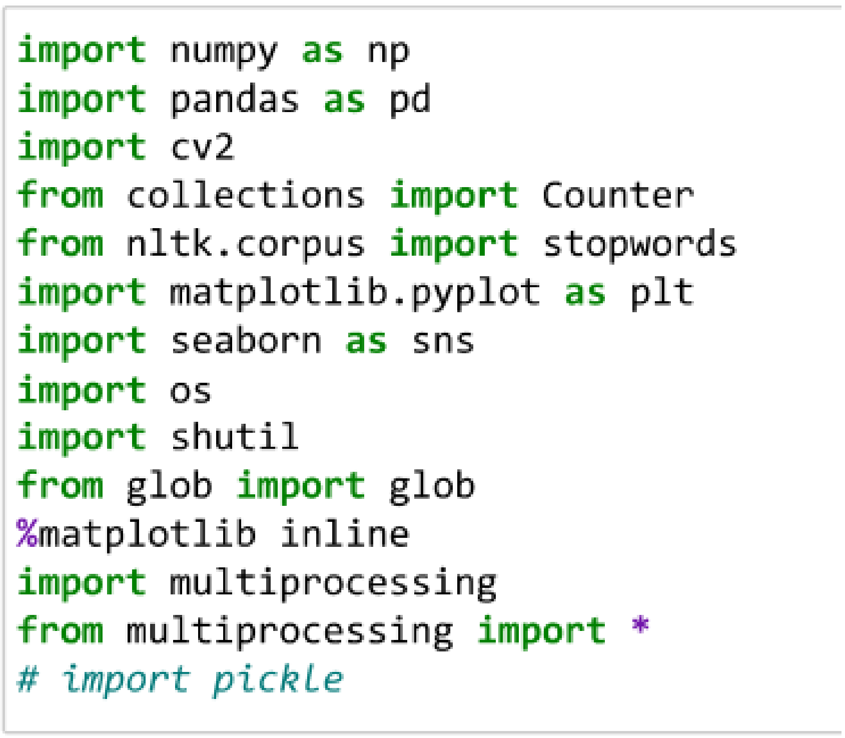


```
plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training - Loss Function')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Train - Accuracy')
```

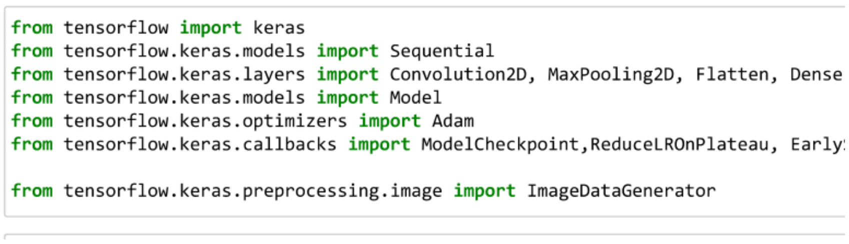
```
Text(0.5, 1.0, 'Train - Accuracy')
```

4.3 ResNet-50 with imagent Weights

fist Import all the libraries

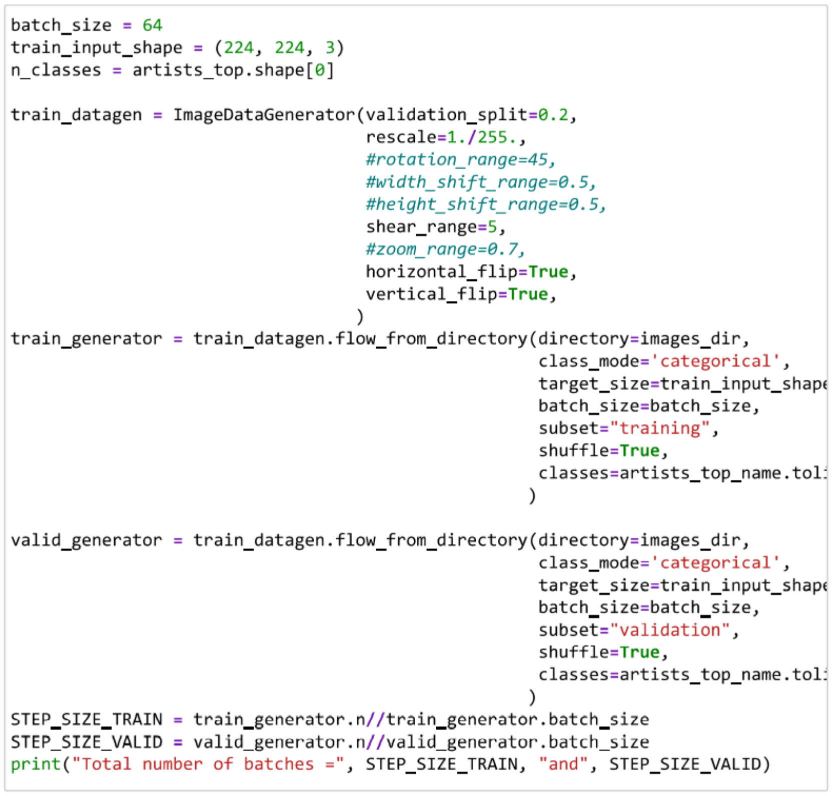


```
import numpy as np
import pandas as pd
import cv2
from collections import Counter
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
import seaborn as sns
import os
import shutil
from glob import glob
%matplotlib inline
import multiprocessing
from multiprocessing import *
# import pickle
```



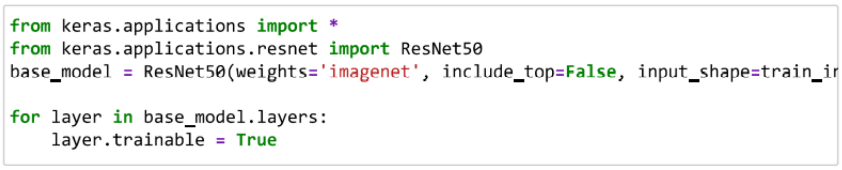
```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint,ReduceLROnPlateau, Early

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

next split the data for training and testing and validation



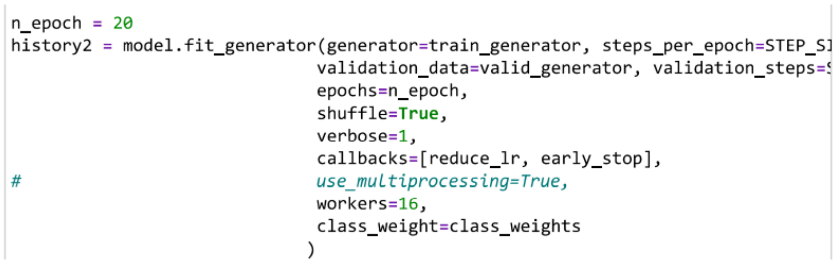
```
batch_size = 64
train_input_shape = (224, 224, 3)
n_classes = artists_top.shape[0]

train_datagen = ImageDataGenerator(validation_split=0.2,
                                   rescale=1./255.,
                                   #rotation_range=45,
                                   #width_shift_range=0.5,
                                   #height_shift_range=0.5,
                                   shear_range=5,
                                   #zoom_range=0.7,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                  )
train_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                    target_size=train_input_shape
                                                    batch_size=batch_size,
                                                    subset="training",
                                                    shuffle=True,
                                                    classes=artists_top_name.tol
                                                   )

valid_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                    target_size=train_input_shape
                                                    batch_size=batch_size,
                                                    subset="validation",
                                                    shuffle=True,
                                                    classes=artists_top_name.tol
                                                   )
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print("Total number of batches =", STEP_SIZE_TRAIN, "and", STEP_SIZE_VALID)
```

after that implement the model



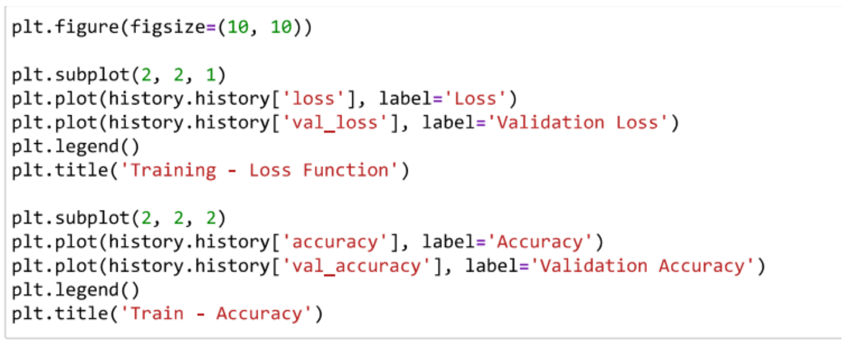
```
from keras.applications import *
from keras.applications.resnet import ResNet50
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=train_ir

for layer in base_model.layers:
    layer.trainable = True
```

run said epochs of the models



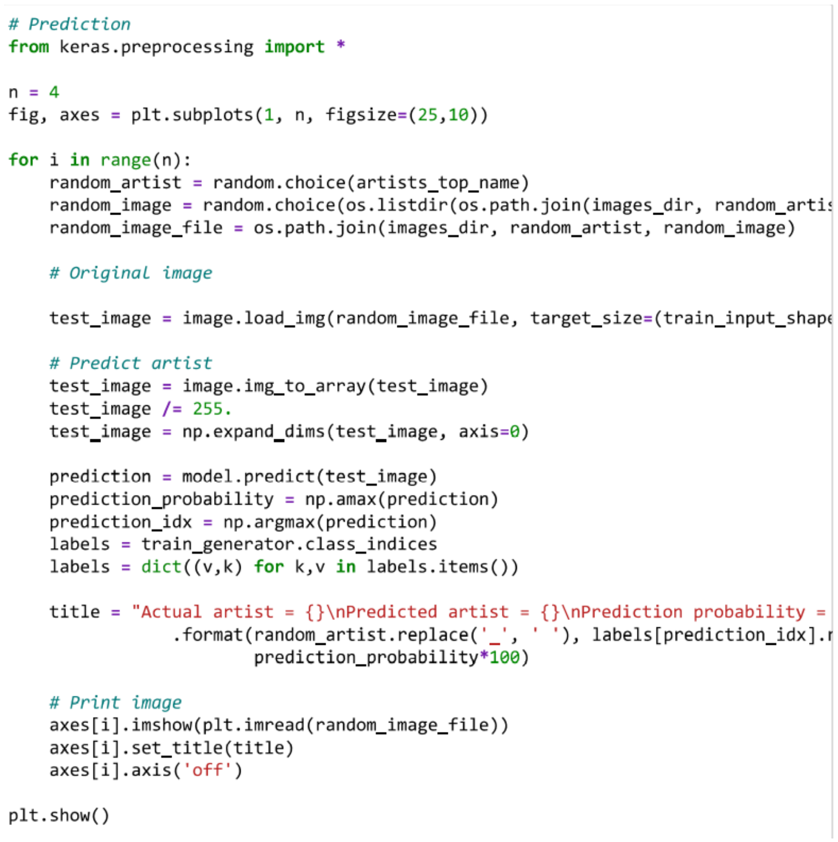
```
n_epoch = 20
history2 = model.fit_generator(generator=train_generator, steps_per_epoch=STEP_SI
                               validation_data=valid_generator, validation_steps=S
                               epochs=n_epoch,
                               shuffle=True,
                               verbose=1,
                               callbacks=[reduce_lr, early_stop],
#                              use_multiprocessing=True,
                               workers=16,
                               class_weight=class_weights
                              )
```

plot the graphs



```
plt.figure(figsize=(10, 10))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Training - Loss Function')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Train - Accuracy')
```

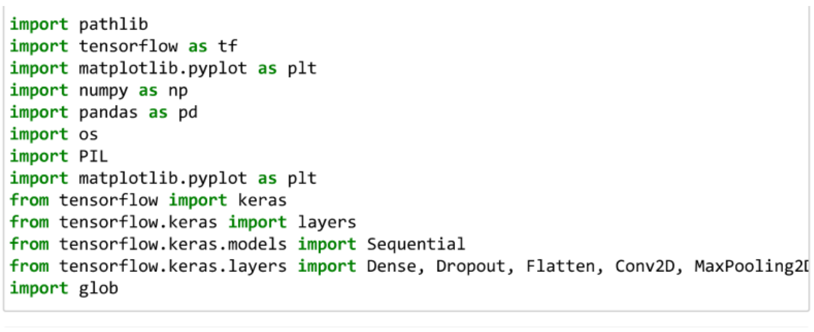
plot the predicted images



```
# Prediction
from keras.preprocessing import *

n = 4
fig, axes = plt.subplots(1, n, figsize=(25,10))

for i in range(n):
    random_artist = random.choice(artists_top_name)
    random_image = random.choice(os.listdir(os.path.join(images_dir, random_artis
    random_image_file = os.path.join(images_dir, random_artist, random_image)

    # Original image

    test_image = image.load_img(random_image_file, target_size=(train_input_shape

    # Predict artist
    test_image = image.img_to_array(test_image)
    test_image /= 255.
    test_image = np.expand_dims(test_image, axis=0)

    prediction = model.predict(test_image)
    prediction_probability = np.amax(prediction)
    prediction_idx = np.argmax(prediction)
    labels = train_generator.class_indices
    labels = dict((v,k) for k,v in labels.items())

    title = "Actual artist = {}\nPredicted artist = {}\nPrediction probability =
                .format(random_artist.replace('_', ' '), labels[prediction_idx].r
                        prediction_probability*100)

    # Print image
    axes[i].imshow(plt.imread(random_image_file))
    axes[i].set_title(title)
    axes[i].axis('off')

plt.show()
```

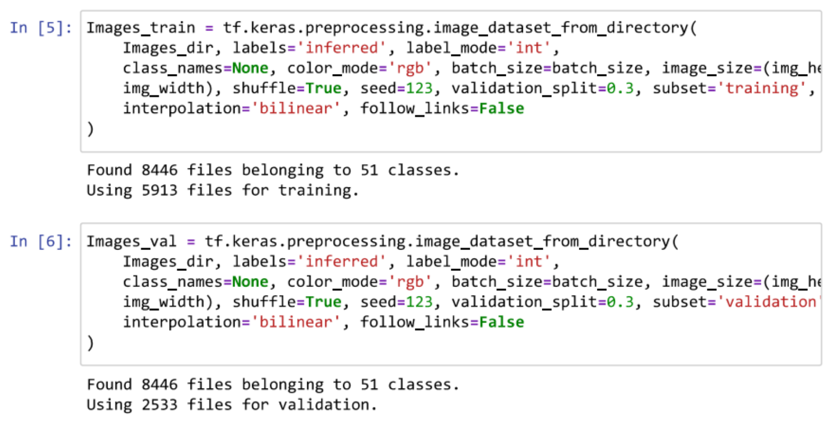
4.4 ResNet-50 With Xception layer

Import the necessary libraries



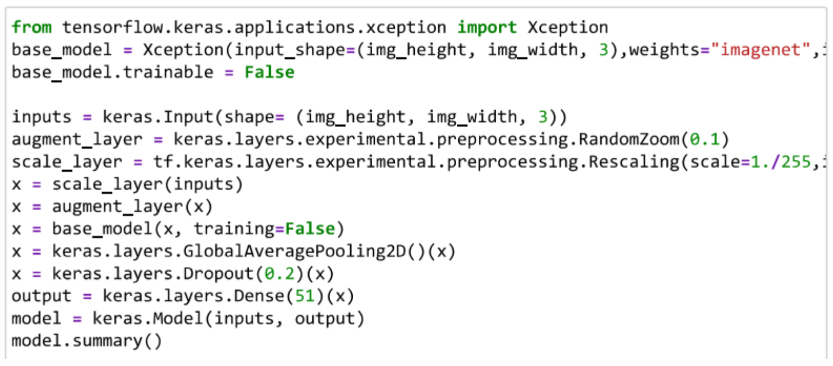
```
import pathlib
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import PIL
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2I
import glob
```

Split the data into test and train



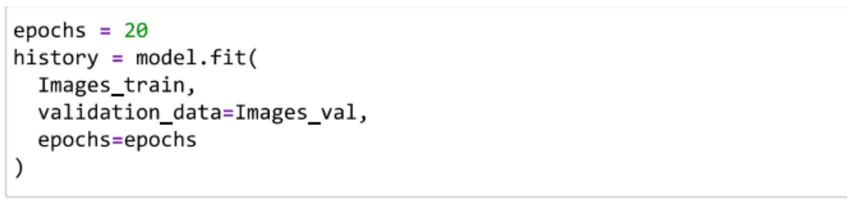
```
In [5]: Images_train = tf.keras.preprocessing.image_dataset_from_directory(
            Images_dir, labels='inferred', label_mode='int',
            class_names=None, color_mode='rgb', batch_size=batch_size, image_size=(img_he
            img_width), shuffle=True, seed=123, validation_split=0.3, subset='training',
            interpolation='bilinear', follow_links=False
        )

        Found 8446 files belonging to 51 classes.
        Using 5913 files for training.

In [6]: Images_val = tf.keras.preprocessing.image_dataset_from_directory(
            Images_dir, labels='inferred', label_mode='int',
            class_names=None, color_mode='rgb', batch_size=batch_size, image_size=(img_he
            img_width), shuffle=True, seed=123, validation_split=0.3, subset='validation'
            interpolation='bilinear', follow_links=False
        )

        Found 8446 files belonging to 51 classes.
        Using 2533 files for validation.
```

Build the model



```
from tensorflow.keras.applications.xception import Xception
base_model = Xception(input_shape=(img_height, img_width, 3),weights="imagenet",i
base_model.trainable = False

inputs = keras.Input(shape= (img_height, img_width, 3))
augment_layer = keras.layers.experimental.preprocessing.RandomZoom(0.1)
scale_layer = tf.keras.layers.experimental.preprocessing.Rescaling(scale=1./255,i
x = scale_layer(inputs)
x = augment_layer(x)
x = base_model(x, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)
output = keras.layers.Dense(51)(x)
model = keras.Model(inputs, output)
model.summary()
```

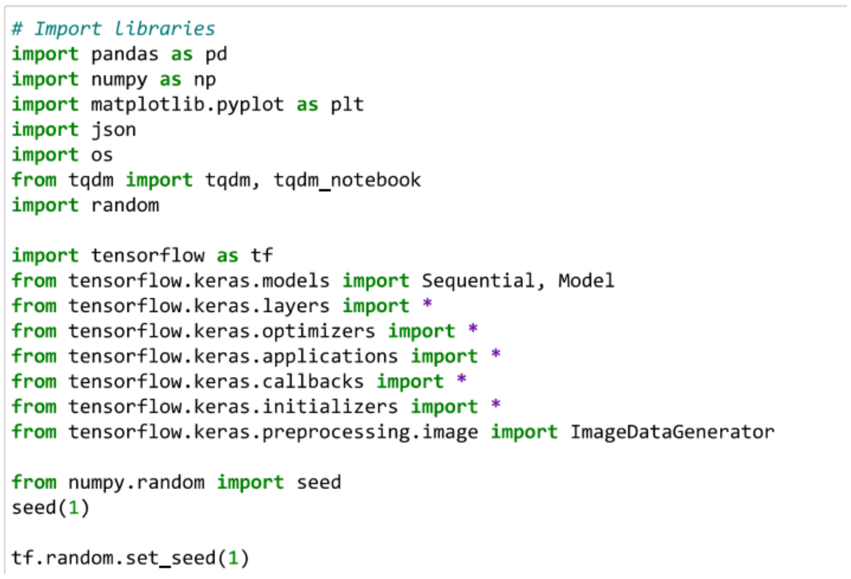
Run the Model



```
epochs = 20
history = model.fit(
  Images_train,
  validation_data=Images_val,
  epochs=epochs
)
```

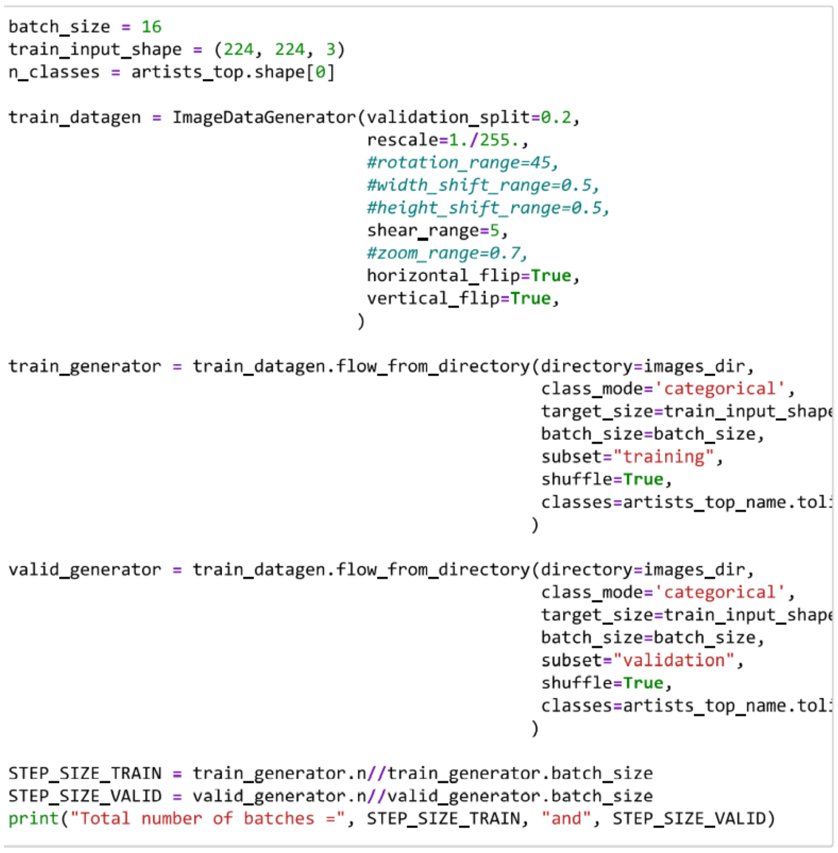
4.5 Resnet-50 fine-tuned

Import all the required libraries



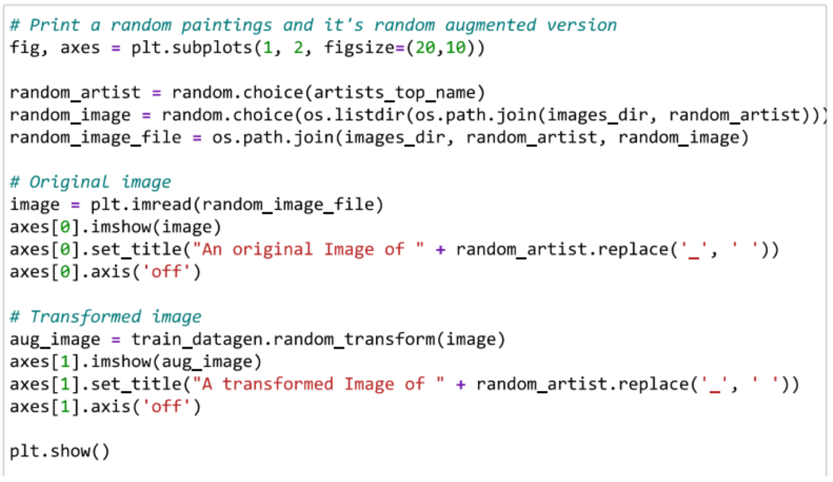
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
import os
from tqdm import tqdm, tqdm_notebook
import random

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import *
from tensorflow.keras.applications import *
from tensorflow.keras.callbacks import *
from tensorflow.keras.initializers import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from numpy.random import seed
seed(1)

tf.random.set_seed(1)
```

split the data for training and testing



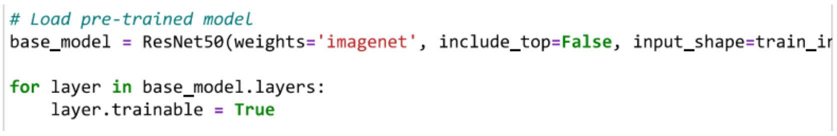
```
batch_size = 16
train_input_shape = (224, 224, 3)
n_classes = artists_top.shape[0]

train_datagen = ImageDataGenerator(validation_split=0.2,
                                   rescale=1./255.,
                                   #rotation_range=45,
                                   #width_shift_range=0.5,
                                   #height_shift_range=0.5,
                                   shear_range=5,
                                   #zoom_range=0.7,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                  )

train_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                    target_size=train_input_shape
                                                    batch_size=batch_size,
                                                    subset="training",
                                                    shuffle=True,
                                                    classes=artists_top_name.tol
                                                   )

valid_generator = train_datagen.flow_from_directory(directory=images_dir,
                                                    class_mode='categorical',
                                                    target_size=train_input_shape
                                                    batch_size=batch_size,
                                                    subset="validation",
                                                    shuffle=True,
                                                    classes=artists_top_name.tol
                                                   )

STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
print("Total number of batches =", STEP_SIZE_TRAIN, "and", STEP_SIZE_VALID)
```

Augment the data and plot the augmented images



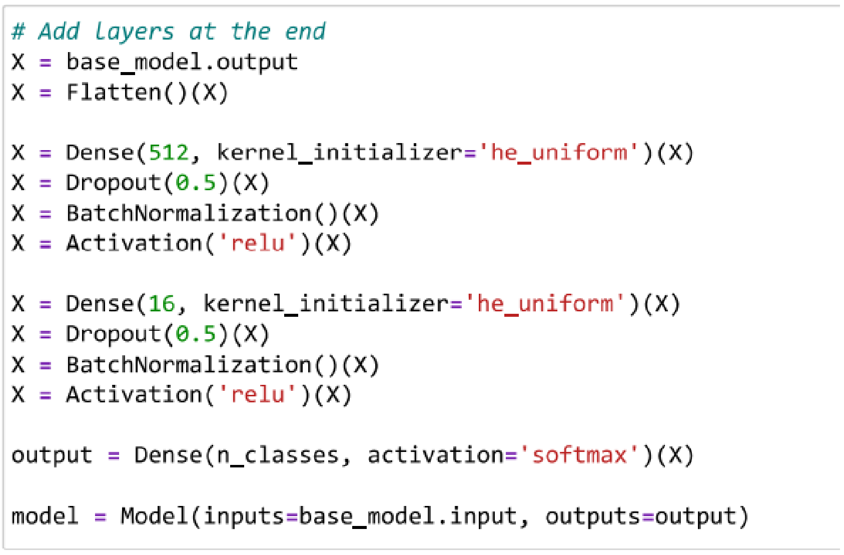
```
# Print a random paintings and it's random augmented version
fig, axes = plt.subplots(1, 2, figsize=(20,10))

random_artist = random.choice(artists_top_name)
random_image = random.choice(os.listdir(os.path.join(images_dir, random_artist)))
random_image_file = os.path.join(images_dir, random_artist, random_image)

# Original image
image = plt.imread(random_image_file)
axes[0].imshow(image)
axes[0].set_title("An original Image of " + random_artist.replace('_', ' '))
axes[0].axis('off')

# Transformed image
aug_image = train_datagen.random_transform(image)
axes[1].imshow(aug_image)
axes[1].set_title("A transformed Image of " + random_artist.replace('_', ' '))
axes[1].axis('off')

plt.show()
```

Load pre-trained model



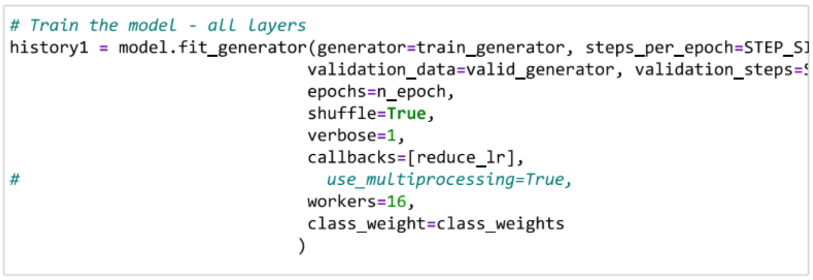
```
# Load pre-trained model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=train_ir

for layer in base_model.layers:
    layer.trainable = True
```

add the final images in the end



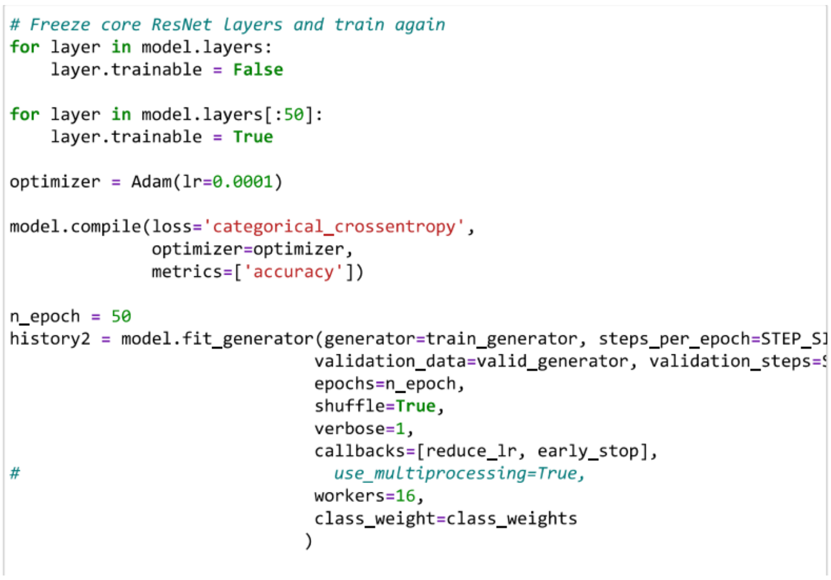
```
# Add layers at the end
X = base_model.output
X = Flatten()(X)

X = Dense(512, kernel_initializer='he_uniform')(X)
X = Dropout(0.5)(X)
X = BatchNormalization()(X)
X = Activation('relu')(X)

X = Dense(16, kernel_initializer='he_uniform')(X)
X = Dropout(0.5)(X)
X = BatchNormalization()(X)
X = Activation('relu')(X)

output = Dense(n_classes, activation='softmax')(X)

model = Model(inputs=base_model.input, outputs=output)
```

train the model



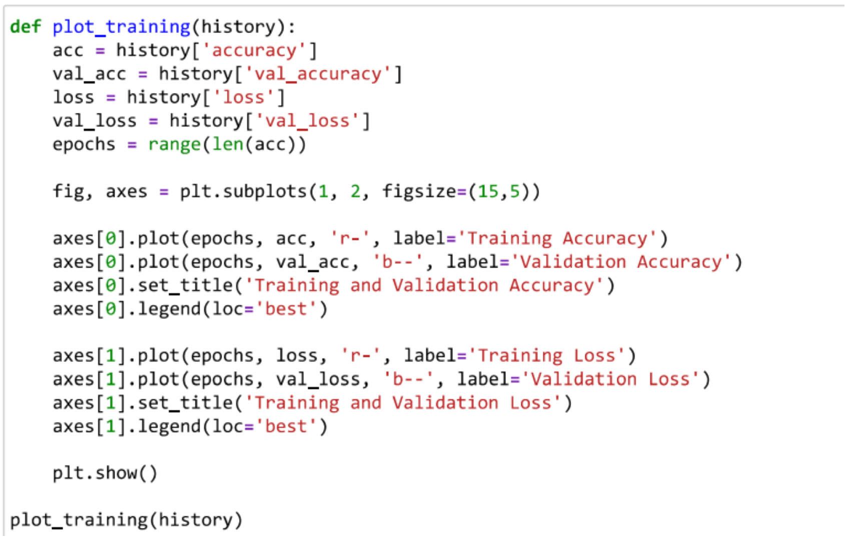
```
# Train the model - all layers
history1 = model.fit_generator(generator=train_generator, steps_per_epoch=STEP_SI
                              validation_data=valid_generator, validation_steps=S
                              epochs=n_epoch,
                              shuffle=True,
                              verbose=1,
                              callbacks=[reduce_lr],
#                               use_multiprocessing=True,
                              workers=16,
                              class_weight=class_weights
                             )
```

Freeze resnet layer and train the model again



```
# Freeze core ResNet layers and train again
for layer in model.layers:
    layer.trainable = False

for layer in model.layers[:50]:
    layer.trainable = True

optimizer = Adam(lr=0.0001)

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

n_epoch = 50
history2 = model.fit_generator(generator=train_generator, steps_per_epoch=STEP_S
                              validation_data=valid_generator, validation_steps=
                              epochs=n_epoch,
                              shuffle=True,
                              verbose=1,
                              callbacks=[reduce_lr, early_stop],
#                               use_multiprocessing=True,
                              workers=16,
                              class_weight=class_weights
                             )
```

plot the testing curves and graph



```
def plot_training(history):
    acc = history['accuracy']
    val_acc = history['val_accuracy']
    loss = history['loss']
    val_loss = history['val_loss']
    epochs = range(len(acc))

    fig, axes = plt.subplots(1, 2, figsize=(15,5))

    axes[0].plot(epochs, acc, 'r-', label='Training Accuracy')
    axes[0].plot(epochs, val_acc, 'b--', label='Validation Accuracy')
    axes[0].set_title('Training and Validation Accuracy')
    axes[0].legend(loc='best')

    axes[1].plot(epochs, loss, 'r-', label='Training Loss')
    axes[1].plot(epochs, val_loss, 'b--', label='Validation Loss')
    axes[1].set_title('Training and Validation Loss')
    axes[1].legend(loc='best')

    plt.show()

plot_training(history)
```

import image fro outer source and apply the model to predict the accuracy

```
url = 'https://www.gpsmycity.com/img/gd/2081.jpg'

import imageio
import cv2

web_image = imageio.imread(url)
web_image = cv2.resize(web_image, dsize=train_input_shape[0:2], )
web_image = image.img_to_array(web_image)
web_image /= 255.
web_image = np.expand_dims(web_image, axis=0)


prediction = model.predict(web_image)
prediction_probability = np.amax(prediction)
prediction_idx = np.argmax(prediction)

print("Predicted artist =", labels[prediction_idx].replace('_', ' '))
print("Prediction probability =", prediction_probability*100, "%")

plt.imshow(imageio.imread(url))
plt.axis('off')
plt.show()
```