National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Mrinmoy Dutta Choudhury
Student ID: x18182658

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mrinmoy Dutta Choudhury
Student ID:	x18182658
Programme:	MSc in Data Analytics **Year:** 2019-20
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	……
Project Title:	Automated Identification of Painters Over WikiArt Image Data Using Machine Learning Algorithms

Word Count: …… **Page Count**……..

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: ……

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

Mrinmoy Dutta Choudhury
X18182658

1 Introduction

This document elaborates the system specification; software and hardware used for the implementation of the project. It also lays out the steps carried out in the implementation of the research project, "Automated Identification of Painters Based on Style of Art Using Machine Learning Algorithms".

2 System Configuration

2.1 Software Specification

- Linux environment set up using Ubuntu plug in for windows to download required image data of the relevant artists from wikiart files.
- Jupyter Notebook: Open source application, was used to split the downloaded data into train, test, and validation splits.
- A Gmail account to access data uploaded to google drive.
- Google Colab, a python environment that uses google cloud.

2.2 Hardware Specification

- Lenovo IdeaPad C340, 256 GB SSD, 8 GB RAM.
- Processor: 1.8 GHz, Intel Core, i5
-

3 Data Generation Steps

- Download the scripts provided by Lucas David[1] to your local machine.
- Execute the following lines of code in Linux shell from the same directory.

[1] https://github.com/lucasdavid/wikiart

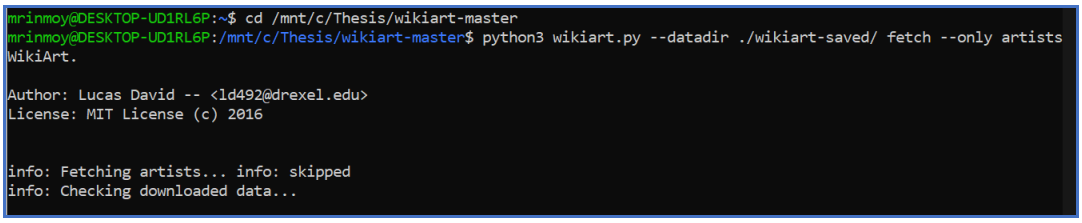

Figure 1 Download JSON File

- This downloads a JSON file containing the name of all the artists in the wikiart database. Keep the names of artists of interest and remove others from the JSON file and save the file.
- Then run the following piece of code that downloads all the paintings of the specified artists from the wikiart database.

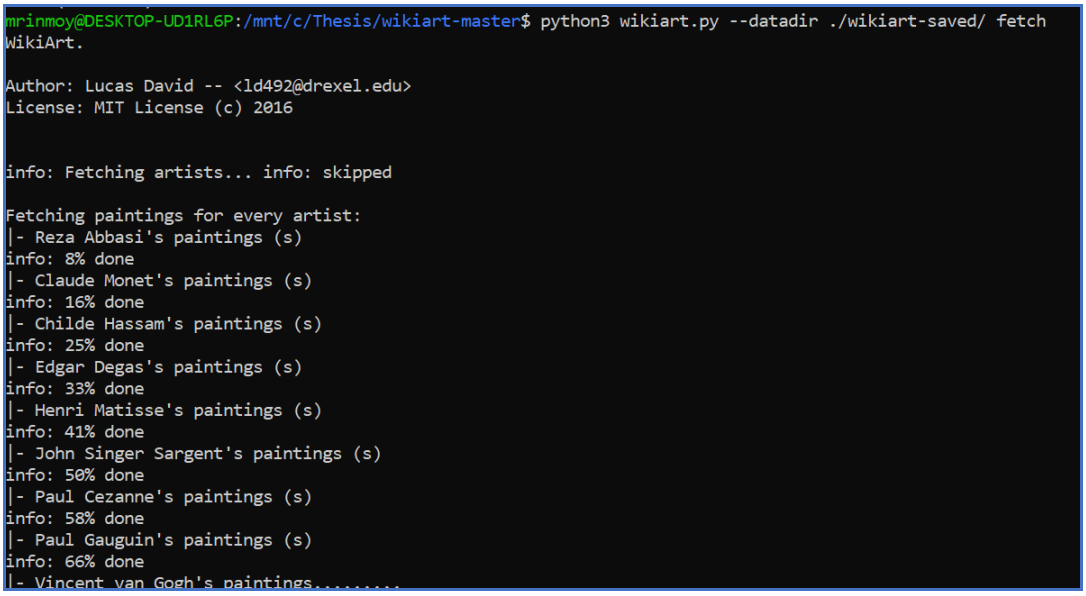

Figure 2 Fetching Images of Relevant Artists

- Run the script “Extract_Copy_Impressionist.ipynb” using a jupyter notebook. This copies all the images of an artist inside their respective folders. Create the folders for individual artists in the destination folder before running the script.

```python
#function to copy files from individual folders into one folder for every artists


def copy_files(src_artist, dest_artist):


    artist_src_dir = os.path.join(root_dir, src_artist)
    artist_destination_dir = os.path.join(destination_dir, dest_artist)

    print(f"Copying files for {src_artist}")
    for root, dirs, files in os.walk(artist_src_dir):
        for file in tqdm(files):
            path_file = os.path.join(root, file)
            copy2(path_file, artist_destination_dir)

    print("Done")


if __name__ == "__main__":

    #specify the source and destination directories, the folder names must be created in the destination folder

    src_artist_lst = ['camille-pissarro', 'childe-hassam', 'claude-monet', 'edgar-degas', 'henri-matisse',
                      'john-singer-sargent', 'paul-cezanne', 'paul-gauguin', 'pierre-auguste-renoir',
                      'vincent-van-gogh']

    dest_artist_lst = ['Pissarro', 'Hassam', 'Monet', 'Degas', 'Matisse',
                       'Sargent', 'Cezanne', 'Gauguin', 'Renoir', 'VanGogh']

    # specify directories specific to local machines
    root_dir = 'C:/Users/mrinm/wikiart-master/wikiart-saved/images1/'
    destination_dir = 'C:/Thesis/wikiart-master/wikiart-saved/impressionist_thesis/'

    zip_lst = list(zip(src_artist_lst, dest_artist_lst))

    for (src, dest) in zip_lst:
        copy_files(src, dest)
```

Figure 3 Script to Extract and Copy Files

- The downloaded data was imbalanced in the sense that it contained non-uniform distribution of images for each artist as shown in Figure 4.

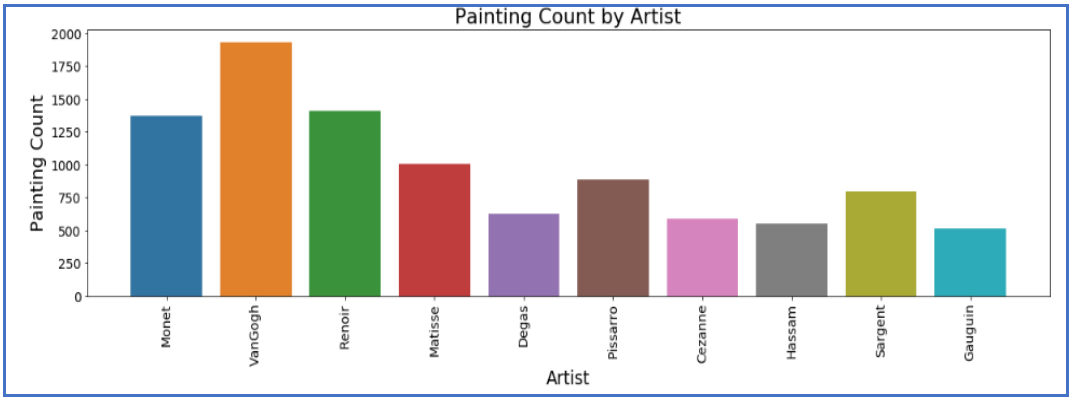

Figure 4 Original Distribution of Images

- Create training, testing and validation folders in the destination path and create folders for each artist inside each of these three folders.
- Now, run the script “impressionist_train_validation_split_impressionist.ipynb” from jupyter notebook. This splits the dataset into specified numbers of train, test, and validation images for each artist. The total number of images considered for the project was 5000 (500 each of 10 painters). Inside this script, firstly a function is defined that splits the data into train, test, validation splits.

```python
#function to create train,test,validation splits
def split_data(root, artist_name, train_split_idx, valid_split_idx, test_split_idx):

    print(f"Creating train and validation splits for {artist_name}")

    #specify the directories

    src = os.path.join(root, artist_name)
    train_dir = os.path.join(root, 'training', artist_name)
    valid_dir = os.path.join(root, 'validation', artist_name)
    test_dir = os.path.join(root, 'test', artist_name)

    # Check for zero length files
    files = []
    for file in os.listdir(src):
        if os.path.getsize(f"{src}/{file}") > 0:
            files.append(file)
        else:
            print(f"{file} is zero length, so ignoring.")

    # Shuffle files in the train, validation and test sets
    files_shuffled = random.sample(files, len(files))

    for file in tqdm(files_shuffled[:train_split_idx]):
        path_file = os.path.join(src, file)
        copy2(path_file, train_dir)

    for file in tqdm(files_shuffled[-valid_split_idx:]):
        path_file = os.path.join(src, file)
        copy2(path_file, valid_dir)

    for file in tqdm(files_shuffled[-test_split_idx:]):
        path_file = os.path.join(src, file)
        copy2(path_file, test_dir)
```

- Secondly, a function is created to check if the files in train, test and validation folders are unique.

```python
#function to check the files in the train,test and validation folders are unique

def check_files(root, artist_name):


    print(f"Checking for duplicates for {artist_name}")

    src_dir = os.path.join(root, artist_name)
    test_dir = os.path.join(root, 'files_for_testing', artist_name)

    train_dir = os.path.join(root, 'training', artist_name)
    valid_dir = os.path.join(root, 'validation', artist_name)
    testing_dir = os.path.join(root, 'test', artist_name)

    all_files = os.listdir(src_dir)
    train_files = os.listdir(train_dir)
    valid_files = os.listdir(valid_dir)
    test_files = os.listdir(testing_dir)

    duplicate_files = len(set(train_files) & set(valid_files) & set(test_files))

    print(f"{duplicate_files} duplicates for {artist_name}")

    not_train_valid__test_lst = []

    for file in all_files:
        if (file not in train_files) and (file not in valid_files) and (file not in valid_files):
            not_train_valid_lst.append(file)

    for file in not_train_valid_test_lst:
        path_file = os.path.join(src_dir, file)
        copy2(path_file, test_dir)

    print(f"Finished creating non train and non valid and non test samples for {artist_name}")
```

```
if __name__ == "__main__":

    # Root directory specific to my local machine
    root_dir = 'C:/Thesis/wikiart-master/wikiart-saved/impressionist_thesis'

    artist_lst = ['Pissarro', 'Hassam', 'Monet', 'Degas', 'Matisse',
                  'Sargent', 'Cezanne', 'Gauguin', 'Renoir', 'VanGogh']

    # specify the number of train, test and validation images for each artist
    train_idx = 380
    valid_idx = 50
    test_idx = 70

    for artist in artist_lst:
        split_data(root_dir, artist, train_idx, valid_idx, test_idx)

    for artist in artist_lst:
        check_files(root_dir, artist)
```

- Figure 5 shows the distribution of data per class after pre-processing and balancing the dataset.

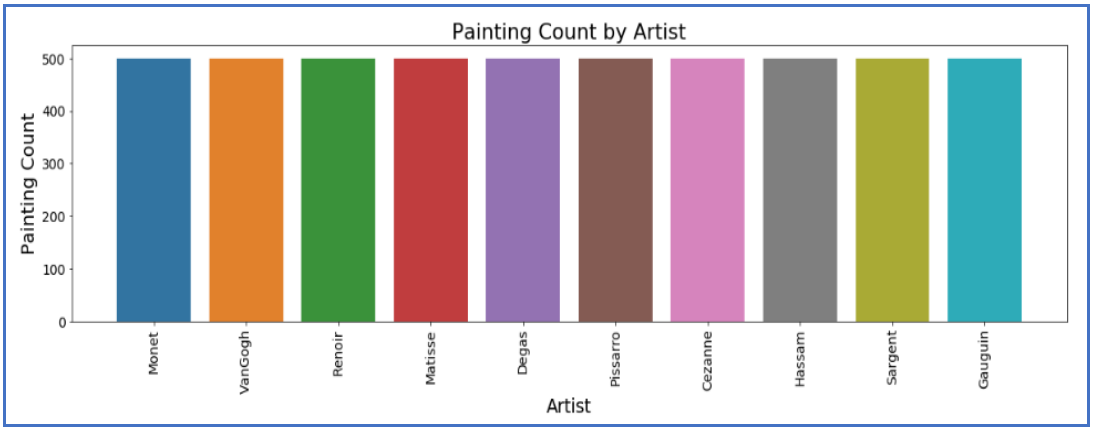

Figure 5 Data Distribution Considered for Implementation

4 Upload Data to Google Drive

- The train, test and Validation folders were uploaded to google drive for data processing using Google Colab.

5 Implemented Models

5.1 Implementation of Random Forest

Step 1: Import all the required libraries

```
#import required libraries

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import mahotas
import cv2
import os
import h5py
import glob
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from google.colab import drive
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sns
```

Step 2: Mount the Google Drive

```
#mount the drive containing the dataset

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

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

Step 3: Define the training directory and image size and specify the size of the test set.

```
# make a fix file size
fixed_size  = tuple((500,500))

#train path
train_path = "drive/My Drive/Project/Total_Images/training"

# no of trees for Random Forests
num_tree = 100

# bins for histograms
bins = 8

# train_test_split size
test_size = 0.10

# seed for reproducing same result
seed = 9
```

Step 4: Write the feature extraction steps for Random Forest Classifier

```
#feature extraction techniques

# features description -1:  Hu Moments

def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

```
# feature-descriptor -2 Haralick Texture

def fd_haralick(image):
    # conver the image to grayscale
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    # Ccompute the haralick texture fetature ve tor
    haralic = mahotas.features.haralick(gray).mean(axis=0)
    return haralic
```

```
# feature-description -3 Color Histogram

def fd_histogram(image, mask=None):
    # conver the image to HSV colors-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    #COMPUTE THE COLOR HISTPGRAM
    hist  = cv2.calcHist([image],[0,1,2],None,[bins,bins,bins], [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist,hist)
    # return the histog....
    return hist.flatten()
```

Step 5: Generate the training labels and initializing empty lists to contain global features and labels

```
# generate the training labels
train_labels = os.listdir(train_path)

# sort the training labels
train_labels.sort()
print(train_labels)

# initiate empty lists to store the values of global features and labels
global_features = []
labels = []

i, j = 0, 0
k = 0

# num of images per class
images_per_class = 500
```

Step 6: Club all the global features into one single list and append the labels and global features list

```
# lop over the training data sub folder

for training_name in train_labels:
    # join the training data path and each species training folder
    dir = os.path.join(train_path, training_name)

    # get the current training label
    current_label = training_name

    k = 1
    # loop over the images in each sub-folder

    for file in os.listdir(dir):

        file = dir + "/" + os.fsdecode(file)

        # read the image and resize it to a fixed-size
        image = cv2.imread(file)

        if image is not None:
            image = cv2.resize(image,fixed_size)
            fv_hu_moments = fd_hu_moments(image)
            fv_haralick   = fd_haralick(image)
            fv_histogram  = fd_histogram(image)
        # Concatenate global features
        global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])

        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)

        i += 1
        k += 1
    print("[STATUS] processed folder: {}".format(current_label))
    j += 1

print("[STATUS] completed Global Feature Extraction...")
```

Step 7: Encode the target labels and save the feature vectors and trained labels

```
# get the overall feature vector size
print("[STATUS] feature vector size {}".format(np.array(global_features).shape))

# get the overall training label size
print("[STATUS] training Labels {}".format(np.array(labels).shape))

# encode the target labels
targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)
print("[STATUS] training labels encoded...{}")
# normalize the feature vector in the range (0-1)
scaler = MinMaxScaler(feature_range=(0, 1))
rescaled_features = scaler.fit_transform(global_features)


print("[STATUS] target labels: {}".format(target))
print("[STATUS] target labels shape: {}".format(target.shape))

# save the feature vector using HDF5
# ensure that the output folder is created in the specified path
h5f_data = h5py.File('drive/My Drive/Project/Total_Images/output/data.h5', 'w')
h5f_data.create_dataset('dataset_1', data=np.array(rescaled_features))

h5f_label = h5py.File('drive/My Drive/Project/Total_Images/output/labels.h5', 'w')
h5f_label.create_dataset('dataset_1', data=np.array(target))

h5f_data.close()
h5f_label.close()
```

Step 8: Import the feature vector and trained labels into the session.

```
# import the feature vector and trained labels

h5f_data = h5py.File('drive/My Drive/Project/Total_Images/output/data.h5', 'r')
h5f_label = h5py.File('drive/My Drive/Project/Total_Images/output/labels.h5', 'r')

global_features_string = h5f_data['dataset_1']
global_labels_string = h5f_label['dataset_1']

global_features = np.array(global_features_string)
global_labels = np.array(global_labels_string)
```

Step 9: Split the data into train and test sets. Because the implementation of Random Forest does not require a separate validation set. Hence, the training and test sets were set to 4500 and 500 images, respectively.

```
# split the training and testing data
(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = train_test_split(np.array(global_features),
                                                                                          np.array(global_labels),
                                                                                          test_size=test_size,
                                                                                          random_state=seed)
```

Step 10: Create and fit the model on the train data. Also, specify the number of decision trees (estimators)

```
# create the model
rf = RandomForestClassifier(n_estimators=100)
rf.fit(trainDataGlobal,trainLabelsGlobal)
# Predictions on training and validation
y_pred_train = rf.predict(trainDataGlobal)

# training metrics
print("Training metrics:")
print(classification_report(y_true= trainLabelsGlobal, y_pred= y_pred_train))
```

Step 11: Make predictions on the test set and plot the classification report showing precision, recall, F1 score and accuracy.

```
#make predictions on the test set

y_pred_test = rf.predict(testDataGlobal)
```

```
#print the classification report showing precision, recall, F1 score and accuracy

print(classification_report(testLabelsGlobal,y_pred_test))
```

5.2 Implementation of SVM

Step 1: The same feature extraction techniques and train test splits (as in the case of Random Forest) were used for the implementation of SVM. Step 1 to Step 9 to be followed as per section 5.1 for the implementation of SVM.

Step 2: Specifying the C values, kernel and gamma parameters and defining and training the model.

```
#defining the kernel, model and training the model
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
svc = svm.SVC()
clf = GridSearchCV(svc, param_grid)
clf.fit(trainDataGlobal, trainLabelsGlobal)
```

Step 3: Make predictions on the test split and plot the classification report

```
#testing on the model for predictions on the test set
y_pred_test = clf.predict(testDataGlobal)
```

```
#print out the classification report

print("Classification report for - \n{}:\n{}\n".format(
    clf, metrics.classification_report(testLabelsGlobal, y_pred_test)))
```

5.3 Implementation of CNN

Step 1: Import all the required libraries and specify the input directory

```
#importing necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import cv2
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Flatten
from tensorflow.keras.layers import Conv2D,MaxPool2D
from tensorflow.keras.layers import Input, Dense
from keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from tensorflow.keras.applications.vgg16 import VGG16
import os
import pandas as pd
import seaborn as sns

#print the content of the working directory
print(os.listdir("drive/My Drive/Project/Total_Images"))
```

Step 2: Specify the image dimensions, batch size, and initialize empty lists to contain image data and class names

```
#specifying the image height, width and depth (Channels)
img_rows=32
img_cols=32
num_channel=3

#setting the number of epoch and batch size
num_epoch = 20
batch_size = 32

#initializing empty lists to contain image data and class names
img_data_list=[]
classes_names_list=[]
target_column=[]
```

Step 3: Fetch images from each of the classes.

```
#looping through each of the folders in the specified path to fetch images from each of these folders

for dataset in data_dir_list:
    classes_names_list.append(dataset)
    print("Getting images from {} folder\n".format(dataset))
    img_list = os.listdir(PATH+'/'+ dataset)
    for img in img_list:
        input_img = cv2.imread(PATH + '/' + dataset + '/' + img)
        input_img_resize=cv2.resize(input_img,(img_rows,img_cols))
        img_data_list.append(input_img_resize)
        target_column.append(dataset)
```

```
Getting images from Hassam folder

Getting images from Matisse folder

Getting images from Sargent folder

Getting images from VanGogh folder

Getting images from Gauguin folder

Getting images from Monet folder

Getting images from Degas folder
```

Step 4: Convert the image data and target column lists into numpy arrays.

```
#converting the lists to numpy arrays and printing their shape

img_data_list = np.array(img_data_list)
target_column = np.array(target_column)
print('the shape of X is: ', img_data_list.shape, 'and that of Y is: ', target_column.shape)
```

```
the shape of X is:  (5000, 32, 32, 3) and that of Y is:  (5000,)
```

Step 5: Standardizing the input data and encoding the labels using LabelEncoder function.

```
#stadardizing the input data
img_data_list = img_data_list.astype('float32')/255
```

```
#converting the y_data into categorical:
from sklearn.preprocessing import LabelEncoder
y_encoded = LabelEncoder().fit_transform(target_column)
from keras.utils import to_categorical
y_categorical = to_categorical(y_encoded)
```

Step 6: Splitting the data into train and test splits in a 90-10 train-test ratio.

```
#using skikit library, dividing the data into train and test splits
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.10)
```

Step 7: Defining the CNN model

```
#structuring the CNN model
from keras import models, layers
model = models.Sequential()
model.add(layers.Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(10, activation='softmax'))
```

Step 8: Compile the model

```
#let's compile the model
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy'])
```

Step 9: Train and fit the model by assigning 20 percent of the training data to validation split

```
#fitting the model
history = model.fit(X_train, Y_train, epochs=num_epoch, validation_split=0.2)
```

Step 10: Once the model has been trained, make predictions on the test set

```
#apply predictions on the test set
Y_pred = model.predict_classes(X_test)
```

Step 11: Print the accuracy obtained on the test set

```
#converting over Y test to actual labels.
Y_test = np.argmax(Y_test, axis = 1)
```

```
#print the accuracy
from sklearn.metrics import accuracy_score
print('the accuracy obtained on the test set is:', accuracy_score(Y_pred,Y_test))
```

Step 12: Print the classification report.

```
#print the classification report to display the precision, recall and f1 scores
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred))
```

5.4 Implementation of Resnet-18 Transfer Learning

Step 1: Import all required libraries

```
#import all the required libraries

import numpy as np
import torchvision
%matplotlib inline
import matplotlib.pyplot as plt
from PIL import Image
import torch
from torchvision import datasets, models, transforms
import torch.nn as nn
from torch.nn import functional as F
import torch.optim as optim
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import pandas as pd
import seaborn as sns
from keras.callbacks import EarlyStopping
```

Step 2: Specify the input path, normalize the inputs with precomputed standard deviation and mean and perform relevant data transformations and augmentations on the train set. No data augmentation has been performed on the validation set.

```
#specify the input path
input_path = 'drive/My Drive/Project/impressionist_train_test_valid/'

#normalize the data and on precomputed standard deviation and mean

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
#perform data augmentation and resizing
data_transforms = {
    'train':
    transforms.Compose([
        transforms.Resize((224,224)),
        #transforms.RandomRotation(degrees=15),
        #transforms.CenterCrop(size=224),
        transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize
    ]),
    'validation':
    transforms.Compose([
        transforms.Resize((224,224)),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        normalize
    ])
}
```

Step 3: Use the dataset.ImageFolder functionality to create the test and validation sets and dataloaders to yield batches of images and labels.

```
#create datasets using datasets.ImageFolder functionality of Pytorch

image_datasets = {
    'train':
    datasets.ImageFolder(input_path + 'training', data_transforms['train']),
    'validation':
    datasets.ImageFolder(input_path + 'validation',
                         data_transforms['validation'])
}

#use dataloader to yeild batches of images and labels

dataloaders = {
    'train':
    torch.utils.data.DataLoader(image_datasets['train'],
                                batch_size=32,
                                shuffle=True,
                                num_workers=0),
    'validation':
    torch.utils.data.DataLoader(image_datasets['validation'],
                                batch_size=32,
                                shuffle=False,
                                num_workers=0)
}
```

Step 4: Check the availability of GPU, else work with CPU.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

Step 5: Download the pretrained ResNet-18 model to the device and define the last layer classifier based on the needs of the project.

```
#download the pretrained resnet18 model to device

model = models.resnet18(pretrained=True).to(device)

#freeze all the weights of the model initially

for param in model.parameters():
    param.requires_grad = False
#Defining the custom classifier and with a softmax classifier at the end
#the size of the last layer is 10, meaning the number of classes

model.fc = nn.Sequential(
               nn.Linear(512, 10),
               nn.LogSoftmax(dim=1).to(device))
```

Step 6: Specify the loss function, optimizers, and early stop parameters.

```
#define the loss function, optimizer and early stop parameter

criterion = nn.CrossEntropyLoss()
#optimizer = optim.Adam(model.fc.parameters())
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
es = EarlyStopping(monitor='validation loss', mode='min', verbose=1, patience=3)
```

Step 7: Define a function to train the model on the train and validation sets, that were defined in the earlier steps.

```
def train_model(model, criterion, optimizer, es, num_epochs=3):
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs))
        print('-' * 10)

        for phase in ['train', 'validation']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                outputs = model(inputs)
                loss = criterion(outputs, labels)

                if phase == 'train':
                    optimizer.zero_grad()
                    loss.backward()
                    optimizer.step()

                _, preds = torch.max(outputs, 1)
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / len(image_datasets[phase])
            epoch_acc = running_corrects.double() / len(image_datasets[phase])

            print('{} loss: {:.4f}, acc: {:.4f}'.format(phase,
                                                        epoch_loss,
                                                        epoch_acc))
    return model
```

Step 8: Train the model

```
model_trained = train_model(model, criterion, optimizer, es, num_epochs=20)
```

```
Epoch 1/20
----------
train loss: 2.2479, acc: 0.1689
validation loss: 2.0997, acc: 0.3580
Epoch 2/20
----------
train loss: 2.0317, acc: 0.3068
validation loss: 1.8396, acc: 0.4680
Epoch 3/20
----------
train loss: 1.8339, acc: 0.3884
validation loss: 1.6340, acc: 0.5060
Epoch 4/20
----------
train loss: 1.6647, acc: 0.4453
validation loss: 1.4722, acc: 0.5520
Epoch 5/20
----------
train loss: 1.5668, acc: 0.4813
validation loss: 1.3824, acc: 0.5580
Epoch 6/20
```

Step 9: Save and load the model in session for prediction using the test set.

```
#save the trained model


PATH="drive/My Drive/Project/impressionist_train_test_valid/model_new.pth"
print("\nSaving the model...")
torch.save(model_trained, PATH)

#load the saved model

EVAL_MODEL1='drive/My Drive/Project/impressionist_train_test_valid/model_new.pth
model = torch.load(EVAL_MODEL1)
model.eval()
```

Step 10: Specify the path to the test set, create dataset and fetch images and labels in batches

```
#define the test directory, create dataset and load data in batches
bs = 8
EVAL_DIR='drive/My Drive/Project/impressionist_train_test_valid/test'


# Prepare the eval data loader
eval_transform=transforms.Compose([
        transforms.Resize(size=256),
        transforms.CenterCrop(size=224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])])

eval_dataset=datasets.ImageFolder(root=EVAL_DIR, transform=eval_transform)
eval_loader=torch.utils.data.DataLoader(eval_dataset, batch_size=bs, shuffle=True, pin_memory=True)

# Enable gpu mode, if cuda available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Number of classes and dataset-size
num_classes=len(eval_dataset.classes)
dsize=len(eval_dataset)

# Class label names
class_names=['Cezanne', 'Degas', 'Gauguin', 'Hassam', 'Matisse', 'Monet', 'Pissarro', 'Renoir', 'Sargent', 'VanGogh']

# Initialize the prediction and label lists
predlist=torch.zeros(0,dtype=torch.long, device='cpu')
lbllist=torch.zeros(0,dtype=torch.long, device='cpu')
```

Step 11: Evaluate the model accuracy on the test dataset.

```
# Evaluate the model accuracy on the dataset
correct = 0
total = 0
with torch.no_grad():
    for images, labels in eval_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        predlist=torch.cat([predlist,predicted.view(-1).cpu()])
        lbllist=torch.cat([lbllist,labels.view(-1).cpu()])
```

Step 12: Check the model performance and print the confusion matrix and classification report.

```
# Overall accuracy
overall_accuracy=100 * correct / total
print('Accuracy of the network on the {:d} test images: {:.2f}%'.format(dsize,
    overall_accuracy))

# Confusion matrix
conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
print('Confusion Matrix')
print('-'*16)
print(conf_mat,'\n')

print(classification_report(lbllist.numpy(),predlist.numpy()))
```

5.5 Implementation of ResNet-50 Transfer Learning

Step 1: The data transformation, data creation and steps involving fetching of images and labels remain like the ones stated in the implementation of ResNet-18 transfer learning. Follow Step 1 to Step 4 in section 5.4 for the steps.

Step 2: Download the pretrained ResNet-50 model to the device and define the last layer classifier based on the specific requirements of the project.

```
#download the pretrained resnet50 model and define the classifier

model = models.resnet50(pretrained=True).to(device)

for param in model.parameters():
    param.requires_grad = False

model.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, 10),
               nn.LogSoftmax(dim=1).to(device))
```

Step 3: Specify the optimizer, loss function and early stop parameters.

```
#specify the loss function, optimizer and early stopping
criterion = nn.CrossEntropyLoss()
#optimizer = optim.Adam(model.fc.parameters())
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
es = EarlyStopping(monitor='validation loss', mode='min', verbose=1, patience=3)
```

Step 4: The creation of the train function to train the model on the test and validation splits remains the same as in the implementation of ResNet-18 transfer learning model (Step 7 in section 5.4).

Step 5: Train the model by specifying the number of epochs.

```
#train the model
model_trained = train_model(model, criterion, optimizer, es, num_epochs=20)
```

```
Epoch 1/20
----------
train loss: 2.2099, acc: 0.2763
validation loss: 2.0611, acc: 0.4460
Epoch 2/20
----------
train loss: 1.9144, acc: 0.4821
validation loss: 1.7364, acc: 0.5060
Epoch 3/20
----------
train loss: 1.6349, acc: 0.5258
validation loss: 1.4762, acc: 0.5400
Epoch 4/20
----------
train loss: 1.4214, acc: 0.5795
validation loss: 1.3271, acc: 0.5860
Epoch 5/20
----------
train loss: 1.2813, acc: 0.6105
validation loss: 1.2151, acc: 0.5980
Epoch 6/20
```

Step 6: Save the model and then load it again to test its performance against the test set.

```
#save the model


PATH="drive/My Drive/Project/impressionist_train_test_valid/model_resnet50.pth"
print("\nSaving the model...")
torch.save(model_trained, PATH)

#load the saved model
EVAL_MODEL='drive/My Drive/Project/impressionist_train_test_valid/model_resnet50.pth'
model = torch.load(EVAL_MODEL)
model.eval()
```

Step 7: Specifying the path to the test set, creating the test dataset, and fetching images and labels in batches remain exactly like the implementation of the ResNet-18 model (refer Step 10 in section 5.4).

Step 8: Evaluate the model accuracy on the test dataset.

```
# Evaluate the model accuracy on the dataset
correct = 0
total = 0
with torch.no_grad():
    for images, labels in eval_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        predlist=torch.cat([predlist,predicted.view(-1).cpu()])
        lbllist=torch.cat([lbllist,labels.view(-1).cpu()])
```

Step 8: Calculate the accuracy and plot the confusion matrix.

```
# Overall accuracy
overall_accuracy=100 * correct / total
print('Accuracy of the network on the {:d} test images: {:.2f}%'.format(dsize,
    overall_accuracy))

# Confusion matrix
conf_mat=confusion_matrix(lbllist.numpy(), predlist.numpy())
print('Confusion Matrix')
print('-'*16)
print(conf_mat,'\n')
```

Step 9: Print the classification report showing the precision, recall and F1- score.

```
print(classification_report(lbllist.numpy(),predlist.numpy()))
```

```
              precision    recall  f1-score   support

           0       0.77      0.71      0.74        70
           1       0.80      0.70      0.75        70
           2       0.77      0.71      0.74        70
           3       0.66      0.79      0.72        70
           4       0.85      0.87      0.86        70
           5       0.69      0.59      0.64        70
           6       0.62      0.64      0.63        70
           7       0.84      0.80      0.82        70
           8       0.74      0.86      0.79        70
           9       0.77      0.83      0.80        70

    accuracy                           0.75       700
   macro avg       0.75      0.75      0.75       700
weighted avg       0.75      0.75      0.75       700
```