National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Fathima Bi Rafi
Student ID: 21217084

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Fathima Bi Rafi		
Student ID:	21217084		
Programme:	MSc Data Analytics	**Year:**	2023-2024
Module:	Academic Internship		
Lecturer:	Vladimir Milosavljevic		
Submission Due Date:	31 January 2024		
Project Title:	Configuration Manual		
Word Count:	1343 **Page Count:** 13		

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: Fathima Bi Rafi

Date: 31/1/2024

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

Fathima Bi Rafi
Student ID: 21217084

1 INTRODUCTION

The Configuration Manual serves as a comprehensive guide for implementing Artwork Classification using Transfer Learning. It details the setup and optimization of the transfer learning model, outlining the specific neural network architecture, hyperparameters, and dataset configurations. This manual equips users with step-by-step instructions to seamlessly replicate the project, ensuring accurate and efficient classification of artwork through the utilization of pre-trained models and transfer learning techniques.

2 SYSTEM SPECIFICATIONS

- **Platform**: Google Colaboratory with T4 GPU and 12.7 GB RAM.
- **Integrated Development Environment (IDE)**: Visual Studio Code (VSCode)
- **Programming Language**: Python version 3.7
- **Web Browser**: Google Chrome

3 DATASET

Dataset contains European artworks from 3rd to19th century collected from a Web Gallery of Art. It contains 3 files:

- artists.csv: dataset of information for each artist
- images.zip: collection of images (full size), divided in folders and sequentially numbered.
- resized.zip: same collection but images have been resized and extracted from folder structure.

4 IMPLEMENTATIONS

STEP 1: Mount the Google Drive in Google Colab for accessing the code from the specified location. Unzip the dataset stored in drive to specified directory for subsequent image loading.

STEP 2: Importing all necessary libraries and modules for artwork classification as given below in Figure 1.

```
import numpy as np
import pickle
import os
from tqdm import tqdm
import cv2
import pandas as pd
import seaborn as sns
from os import listdir
import tensorflow as tf
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from sklearn.utils import shuffle
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam,SGD
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import img_to_array
from keras.utils import to_categorical
import imblearn
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from keras.models import Model
from tensorflow.keras.layers import Input
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization, AveragePooling2D, GlobalAveragePooling2D
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras import layers
from tensorflow.keras.applications.xception import Xception
from sklearn.metrics import classification_report,confusion_matrix
import efficientnet.keras as effnet
```

Figure 1: Importing Required Libraries

This section imports various Python libraries essential for machine learning and image processing tasks, including NumPy, Pandas, TensorFlow, Keras, OpenCV, and others. These libraries collectively support data manipulation, model development, training, and evaluation.

STEP 3: Set default size for image, retrieve labels from dataset directory and calculate the number of classes by defining batch size as shown in Figure 2.

```
image_size=128
default_image_size = tuple((128, 128))
labels = os.listdir('/content/drive/MyDrive/historic_art_classification/Data')
directory_root = '/content/drive/MyDrive/historic_art_classification/Data'
classes = len(labels)
BATCHZ_SIZE=16
```

Figure 2: Dataset Initialization

STEP 4: Identify all the class directories, count the number of image files in each class, and filter classes with file counts between 200 and 1000, excluding 328. The results are printed as Figure 3.

```
        if len(blist) > 200 and len(blist) < 1000 and len(blist) != 328:
            i50 = i50+1;
            print(i50, ". Found ", len(blist), ' files for ', img ,'(',i+1,')')
            classlist.append(img)
            flist.append(blist)
        blist=[]
    print(classlist)
```

Figure 3: Filtering class

STEP 5:

```
def image_array(dir):
    try:
        img = cv2.imread(dir)
        if img is not None :
            img = cv2.resize(img, default_image_size)
            return img_to_array(img)
        else :
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

Figure 4: Converting image into array.

In figure 4, code defines a function `image_array` to read, resize, and convert images to array.

STEP 6:

In next step, append each image's array along with its corresponding label to separate lists. The results will be shown like Figure 5.

```
[INFO] Loading images ...
[INFO] Processing Marc_Chagall ...
[INFO] Processing Pablo_Picasso ...
[INFO] Processing Paul_Gauguin ...
[INFO] Processing Pierre_Auguste_Renoir ...
[INFO] Processing Rembrandt ...
[INFO] Processing Titian ...
[INFO] Processing Vincent_van_Gogh ...
[INFO] Processing Alfred_Sisley ...
[INFO] Processing Edgar_Degas ...
[INFO] Processing Francisco_Goya ...
[INFO] Image loading completed
```

Figure 5: Image loading and Categorization

STEP 7:

Data visualization is done by creating count plot to visualize the distribution of classes in the dataset. Also performing data balancing using SMOTE to tackle imbalance class as Figure 6.

```
#data balancing
X = X.reshape(-1, image_size * image_size * 3)
X.shape
oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
X = X.reshape(-1,image_size,image_size,3)
```

Figure 6: Balancing data using SMOTE.

STEP 8:
First, Split the dataset into training and testing sets (90-10%). Further split the dataset into (80-20%) as figure 7.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1)
```

```
[ ] X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train)
```

Figure 7: Dataset split.

STEP 9:
Generate augmented images by rotating up to 20 degrees and flipping horizontally with batch size 32 as figure 8.

```
datagen = ImageDataGenerator(rotation_range=20,
                             horizontal_flip=True)

training_gen = datagen.flow(X_train, y_train, batch_size=32)
validation_gen = datagen.flow(X_val, y_val, batch_size=32)
```

Figure 8: Data Augmentation

4 TRAINING MODELS

For this project, training models like CNN, VGG16/Xception, ResNet50/Inception-V3, and EfficientNet-B2 are deployed for better performance providing insights into model accuracy and effectiveness across different classes.

4.1 CNN MODEL

STEP 10:

- Figure 9, code defines image depth, sets input shape, and adjusts channel dimensions for compatibility.

```
#CNN
depth=3
model = Sequential()
inputShape = (image_size, image_size, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, image_size, image_size)
    chanDim = 1
```

Figure 9: Configuration of image

- The images are trained using the activation function='relu' as shown in figure 10.

```
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(32, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Dense(cls))
model.add(Activation("sigmoid"))
model.summary()
```

Figure 10: sample of trained images using 'relu'

4.2 RESNET50 MODEL

STEP 11:

- Import necessary libraries as figure 11.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D, BatchNormalization, Dropout
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, CSVLogger
```

Figure 11: Required libraries for RESNET50 model.

- Load the pre-trained model excluding the top classification layer as Figure 12.

```
resnet = ResNet50(weights= 'imagenet', include_top=False, input_shape= X_train.shape[1:])
```

Figure 12: Load model

- Enhance model with average pooling, dropout, and a softmax layer for multi-class classification as figure 13.

```
x = resnet.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
predictions = Dense(cls, activation= 'softmax')(x)
model = Model(inputs = resnet.input, outputs = predictions)
```

Figure 13: Model modification

4.3 XCEPTION MODEL

STEP 12:

- Using xception base, pre-learned patterns add layers for understanding images and predict art categories for optimizing its accuracy as figure 14.

```
# Xception Backbone
xception = Xception(input_shape=(image_size,image_size,3), weights='imagenet', include_top=False)

# Freeze the model weights
xception.trainable = True

# The Xception Model baseline
model = Sequential([
    xception,
    GlobalAveragePooling2D(),
    Dropout(0.5),
    Dense(n_classes, activation='softmax')
])

# Compile the Baseline
model.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(),
    metrics=['accuracy']
)
```

Figure 14: Xception model

4.4 EFFICIENTNETB2

STEP 13:

- Employ EfficientNet B2 with pre-learned features and add layers for image understanding and predict art categories as figure 15.

```
base_model = effnet.EfficientNetB2(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
model = base_model.output
model = GlobalAveragePooling2D()(model)
model = Dense(cls, activation='softmax')(model)
model = Model(inputs = base_model.input, outputs=model)
```

Figure 15: EfficentNetB2 Model

5 MODEL EVALUATION

STEP 14: UNIFIED CODES FOR ALL MODELS

Classification Reports

- Images are trained with batch size 10 rounds and display the progress as figure 16.

```
history = model.fit(training_gen,batch_size=BATCHZ_SIZE,validation_data=validation_gen,epochs=10, verbose=1)
```

Figure 16: Model Training

- Plot accuracy and loss over epochs to visualize the model's learning process on both training and validation datasets as figure 17.

```
#accuracy and loss plot
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','val'], loc='upper left')
plt.show()


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

Figure 17: Accuracy and loss plot visualization

Confusion Matrix

- Plot heatmap displaying the confusion matrix for comparing actual vs. predicted values as figure 18.

```
cmat = confusion_matrix(y_test_new,pred)
plt.figure(figsize=(10,10))
sns.heatmap(cmat, annot = True, cbar = False, cmap='crest', fmt="d");
```

Figure 18: Confusion matrix Visualization

Specificity and Sensitivity

- Summary of precision, recall, and F1-score for model predictions on the test data as figure 19.

```
print(classification_report(y_test_new,pred))
```

Figure 19: Model predictions

- Compute and store sensitivity (true positive rate) and specificity (true negative rate) for each class as figure 20.

```
from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(cls):
    prec,recall,_,_ = precision_recall_fscore_support(y_test_new==l,
                                                      pred==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

Figure 20: Specificity and sensitivity prediction

STEP 15: Comparison

- Generate a bar chart using seaborn to visually compare the performance graphs of all models by representing their accuracy values on the y-axis as in figure 21.

```
sns.barplot(x=['Cnn', 'Resnet50', 'Xception', 'EfficientB2'], y=[acc1,acc2,acc3,acc4])
```

Figure 21: Bar chart comparison

6 TESTING

STEP 16: Model prediction

Load the best model (EfficientNetB2) after comparison, predict class probabilities for a resized image, find the class with the highest probability, and print the predicted class label for a historic art image as shown in figure 22.

```
maxi_ele = max(pred)
idx = pred.index(maxi_ele)
final_class=label
class_name= final_class[idx]
print("Predicted Class Is : " + str(class_name))
```

Figure 22: Prediction of class

7 GRAPHICAL USER INTERFACE (GUI)

STEP 17:

Here, The GUI defines a Flask web application for an image classifier using EfficientNetB2 model for predicting the artist of a given image.

- Import all necessary libraries required including Flask, TensorFlow and others as shown in figure 23.

```
# Flask utils
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from flask import Flask, request, render_template
import matplotlib.pyplot as plt
import efficientnet.keras as effnet
from PIL import Image
from tqdm import tqdm
import tensorflow
import numpy as np
import pandas as pd
from os import listdir
import seaborn as sns
import pickle
import re
import os
import string
import cv2
```

Figure 23: Import necessary libraries.

- Set up file paths and constants, load a pre-trained EfficientNetB2 model, read class labels from a pickled file, and print the available classes as shown in figure 24.

```
THIS_DIR = os.path.dirname(os.path.realpath(__file__))
save_path = os.path.join(THIS_DIR, 'static', 'img')
image_path = './static/img/input.jpeg'
default_image_size = tuple((128, 128))
image_size = 128
model =load_model('./Models/efficientnetB2.h5')

class_labels = pd.read_pickle('label_transform.pkl')
classes = class_labels.classes_
print(classes)
```

Figure 24: Setup and Model loading

- Create a Flask web application and define allowed file extensions, checking if a filename has a valid extension as shown in figure 25.

```
app = Flask(__name__)

ALLOWED_EXTENSIONS = {'png','jpg','jpeg'}


def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
```

Figure 25: Flask configuration

- Define routes for home page and classifier page.
- For file prediction, upload an image, preprocess it, predict the artist using the model, and render the result on the classifier page as shown in figure 26.

```
prediction=model.predict(image)
pred_= prediction[0]
pred=[]
for ele in pred_:
  pred.append(ele)
maxi_ele = max(pred)
idx = pred.index(maxi_ele)
final_class=classes
class_name= final_class[idx]
class_text = "Predicted Artist Is : " + class_name
class_text = class_text.upper()
print(class_text)
```

Figure 26: Prediction of Artist using Model

FINAL STEP

- Web application homepage that predicts the artist of a given image using flask screens is shown in figure 27 and a classifier page (subpage) which process the workflow is shown in figure 28.

Figure 27: GUI of Web Application using Flask Screens (Homepage)

- **Workflow**

1. Click classify art.
2. Insert an image from any given class.
3. Predict uploaded image.

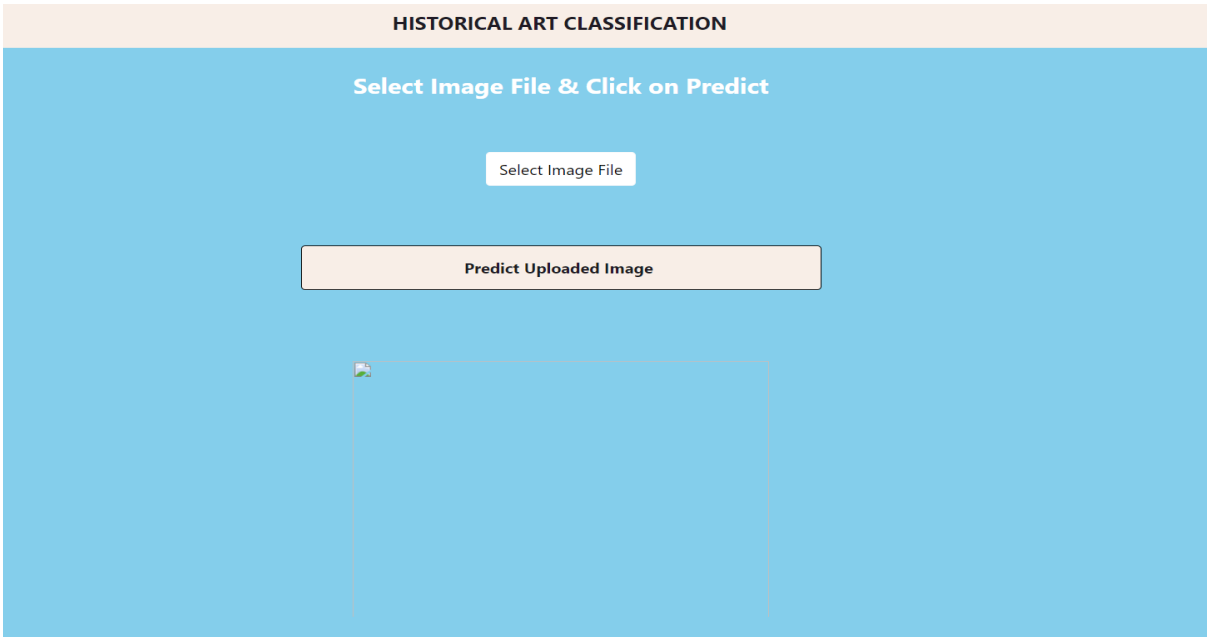

Figure 28: Final Artist Prediction (Classifier page)