National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Siddharud Tevaramani
Student ID: x21156549

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Siddharud Tevaramani		
Student ID:	X21156549		
Programme:	Data Analytics	**Year:**	2022
Module:	MSc Research Project		
Lecturer:	Noel Cosgrave		
Submission Due Date:	15/12/2022		
Project Title:	Configuration Manual		
Word Count:	671	**Page Count:**	14

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

Siddharud Tevaramani
Student ID: x21156549

1 Introduction

The purpose of this guide is to provide a detailed explanation of the implementation, configuration, and setup of the research experiment. This documentation includes information on the software and hardware configuration, as well as the libraries used in the project. It also outlines the coding process and the steps needed to run the code.

2 Local Machine System Configuration

Device Specification:

Device name	LAPTOP-846CFMGM
Processor	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz
Installed RAM	16.0 GB (15.8 GB usable)
Device ID	D4E8B9F1-3F23-409C-896C-D8417DDCF5AE
Product ID	00327-35198-60377-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows(OS) Specification:

Edition	Windows 11 Home Single Language
Version	21H2
Installed on	17-06-2022
OS build	22000.1219
Experience	Windows Feature Experience Pack 1000.22000.1219.0

3 Dataset Collection

The dataset used in this project was obtained from the Harvard Dataverse website and consists of 10015 images.[1]

[1] https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/DBW86T

4 Google Colab SetUp

```
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 79
model name  : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping    : 0
microcode   : 0x1
cpu MHz          : 2200.000
cache size  : 56320 KB
physical id : 0
siblings    : 2
core id          : 0
cpu cores   : 1
apicid           : 0
initial apicid   : 0
fpu         : yes
fpu_exception    : yes
cpuid level : 13
wp          : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb
rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc eagerfpu pni
pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave
avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase
tsc_adjust bmi1 hle avx2 smep bmi2 erms rtm rdseed adx xsaveopt
bugs        :
bogomips    : 4400.00
clflush size     : 64
cache_alignment  : 64
address sizes    : 46 bits physical, 48 bits virtual
power management:

processor   : 1
vendor_id   : GenuineIntel
cpu family  : 6
model       : 79
model name  : Intel(R) Xeon(R) CPU @ 2.20GHz
stepping    : 0
microcode   : 0x1
cpu MHz          : 2200.000
cache size  : 56320 KB
physical id : 0
siblings    : 2
core id          : 0
cpu cores   : 1
apicid           : 1
initial apicid   : 1
fpu         : yes
fpu_exception    : yes
cpuid level : 13
wp          : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb
rdtscp lm constant_tsc rep_good nopl xtopology nonstop_tsc eagerfpu pni
pclmulqdq ssse3 fma cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave
```

```
avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase
tsc_adjust bmi1 hle avx2 smep bmi2 erms rtm rdseed adx xsaveopt
bugs        :
bogomips    : 4400.00
clflush size      : 64
cache_alignment   : 64
address sizes     : 46 bits physical, 48 bits virtual

Filesystem      Size  Used Avail Use% Mounted on
overlay          46G   13G   33G  28% /
tmpfs           6.4G     0  6.4G   0% /dev
tmpfs           6.4G     0  6.4G   0% /sys/fs/cgroup
/dev/sda1        46G   13G   33G  28% /content
shm              64M     0   64M   0% /dev/shm
tmpfs           6.4G     0  6.4G   0% /sys/firmware.
```

5 Importing Libraries:

```
[ ] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    from tqdm import tqdm
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    import keras
    from keras.models import Sequential, load_model
    from keras.callbacks import EarlyStopping, ModelCheckpoint
    from keras.layers.core import Dropout, Activation
    from keras.layers import Conv2D, BatchNormalization, MaxPool2D, Flatten, \
        Dense, Input, Activation, Dropout, GlobalAveragePooling2D, AveragePooling2D
    from keras.utils.np_utils import to_categorical
    from sklearn.model_selection import train_test_split
    from keras.models import Model
    from sklearn.utils.class_weight import compute_class_weight
    from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
    import cv2
    from cv2 import imread, resize # manipulating the images
    from tensorflow.keras.optimizers import Adam
    import os
    import tensorflow as tf
    from sklearn.model_selection import train_test_split
    from sklearn.feature_extraction.text import CountVectorizer
    from sklearn.feature_extraction.text import TfidfTransformer
    from sklearn.naive_bayes import MultinomialNB
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import confusion_matrix
    from keras.utils import np_utils
    import keras
    from keras.utils.np_utils import to_categorical # used for converting labels to one-hot-encoding
    from keras.models import Sequential
    from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
    from keras import backend as K
    import itertools
    from tensorflow.keras.layers import BatchNormalization
    from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
    from skimage.feature import greycomatrix, greycoprops
    import numpy as np
    from skimage.util import img_as_ubyte
```

6 Data Understanding & Preparation:

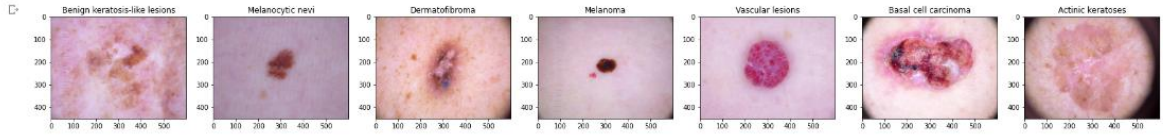
```
lesion_names = ['Melanocytic nevi','Melanoma','Benign keratosis-like lesions ',
                'Basal cell carcinoma','Actinic keratoses','Vascular lesions',
                'Dermatofibroma']
lesion_names_short = ['nv','mel','bkl','bcc','akiec','vasc','df']

lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}

lesion_ID_dict = {
    'nv': 0,
    'mel': 1,
    'bkl': 2,
    'bcc': 3,
    'akiec': 4,
    'vasc': 5,
    'df': 6
}

df_skin['lesion_type']=df_skin['dx'].map(lesion_type_dict)
df_skin['lesion_ID'] = df_skin['dx'].map(lesion_ID_dict)

# Listing all files in the part_1, part_2 dirs
lista = os.listdir('/content/HAM10000_images_part_1/')
lista.extend(os.listdir('/content/HAM10000_images_part_2/'))


def read_image(url):
    return imread(url)

def resize_image(img):
    return resize(img,(140, 140))

def display_image(url):
  img = read_image(url)
  img2 = resize_image(img)
  plt.figure(figsize = (10, 5))
  plt.subplot(1, 2, 1)
  plt.imshow(img[:, :, ::-1])
  plt.title('Original image')
  plt.subplot(1, 2, 2)
  plt.imshow(img2[:, :, ::-1])
  plt.title('Resized image to 140 x 140')
  plt.show()

# Listing all files in the part_1, part_2 dirs
lista = os.listdir('/content/HAM10000_images_part_1/')
lista.extend(os.listdir('/content/HAM10000_images_part_2/'))


df_skin['lesion_type'].value_counts()
```

```
Melanocytic nevi                6705
Melanoma                        1113
Benign keratosis-like lesions   1099
Basal cell carcinoma             514
Actinic keratoses                327
Vascular lesions                 142
Dermatofibroma                   115
Name: lesion_type, dtype: int64
```

7 EDA:

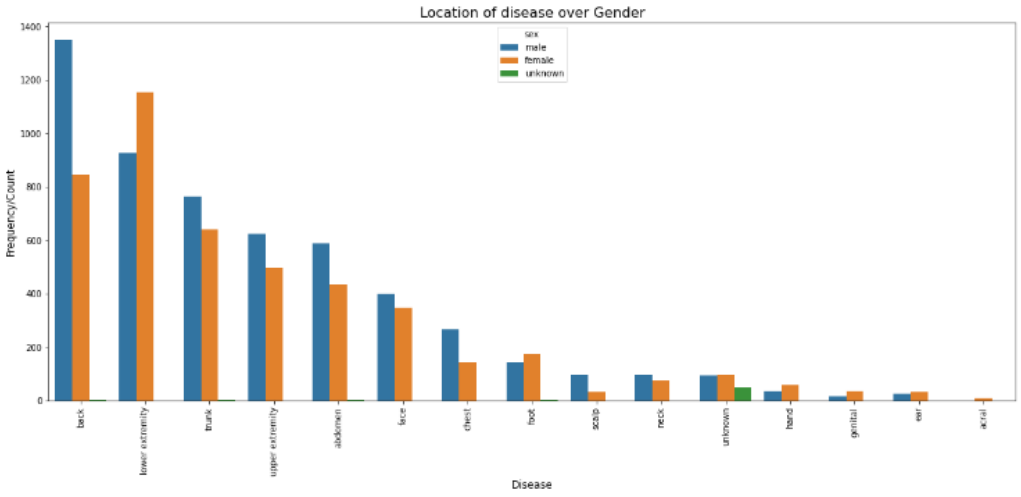
Below code displays each category of the image.

```
fig, axs = plt.subplots(1,7)
fig.set_figheight(20)
fig.set_figwidth(30)
#plt.subplot(121)
axs[0].set_title('Benign keratosis-like lesions')
axs[0].imshow(read_image('/content/HAM10000_images_part_1/ISIC_0027419.jpg')[:, :, ::-1])

#plt.subplot(122)
axs[1].set_title('Melanocytic nevi')
axs[1].imshow(read_image('/content/HAM10000_images_part_1/ISIC_0024698.jpg')[:, :, ::-1])

#plt.subplot(211)
axs[2].set_title('Dermatofibroma')
axs[2].imshow(read_image('/content/HAM10000_images_part_1/ISIC_0027008.jpg')[:, :, ::-1])

axs[3].set_title('Melanoma')
axs[3].imshow(read_image('/content/HAM10000_images_part_1/ISIC_0025964.jpg')[:, :, ::-1])

axs[4].set_title('Vascular lesions')
axs[4].imshow(read_image('/content/HAM10000_images_part_2/ISIC_0031197.jpg')[:, :, ::-1])

axs[5].set_title('Basal cell carcinoma')
axs[5].imshow(read_image('/content/HAM10000_images_part_1/ISIC_0028155.jpg')[:, :, ::-1])

axs[6].set_title('Actinic keratoses')
axs[6].imshow(read_image('/content/HAM10000_images_part_2/ISIC_0029417.jpg')[:, :, ::-1])
plt.show()
```



▾ Location of disease over Gender

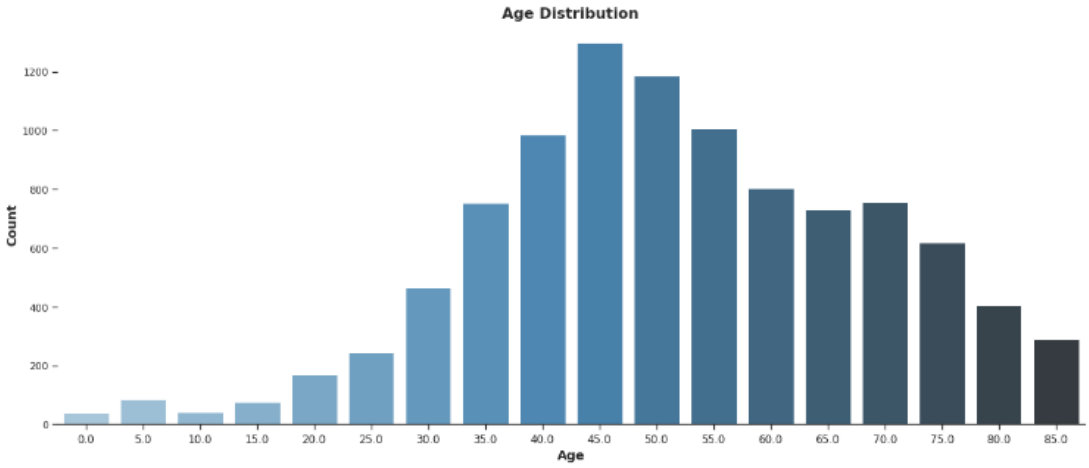
```
[ ] plt.figure(figsize=(20,8))
    value = df_skin[['localization', 'sex']].value_counts().to_frame()
    value.reset_index(level=[1,0 ], inplace=True)
    temp = value.rename(columns = {'localization':'location', 0: 'count'})
    sns.barplot(x = 'location',  y='count', hue = 'sex', data = temp)
    plt.title('Location of disease over Gender', size = 16)
    plt.xlabel('Disease', size=12)
    plt.ylabel('Frequency/Count', size=12)
    plt.xticks(rotation = 90)
    plt.show()
```



Age Distribution

```
[ ] #@title Age Distribution
```

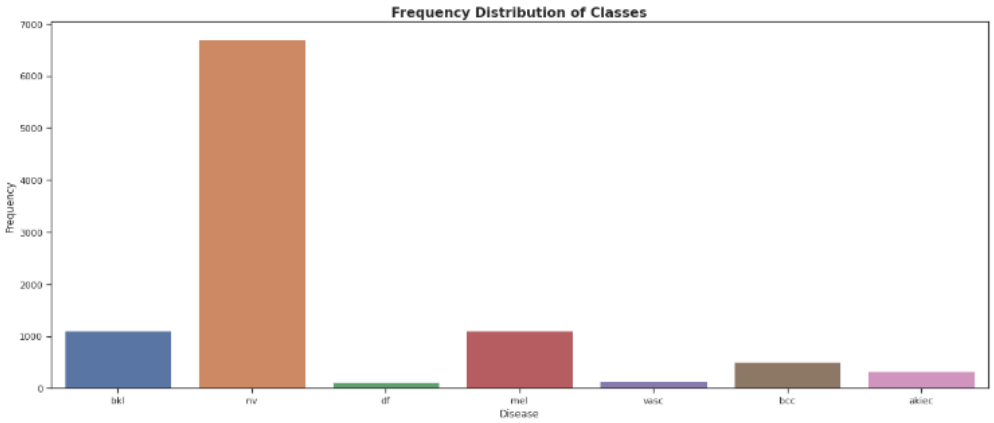
```
plt.figure(figsize=(20,8))
sns.set(style="ticks", font_scale = 1)
ax = sns.countplot(data = df_skin,x='age',palette="Blues_d")#, hue = 'sex')
sns.despine(top=True, right=True, left=True, bottom=False)
plt.xticks(rotation=0,fontsize = 12)
ax.set_xlabel('Age',fontsize = 14,weight = 'bold')
ax.set_ylabel('Count',fontsize = 14,weight = 'bold')
plt.title('Age Distribution', fontsize = 16 ,weight = 'bold');
```



Frequency Distribution

```
#@title Frequency Distribution
```

```
[ ] plt.figure(figsize=(20,8))
sns.countplot(x = 'dx', data = df_skin)
plt.xlabel('Disease', size=12)
plt.ylabel('Frequency', size=12)
plt.title('Frequency Distribution of Classes', size=16, weight = 'bold')
plt.show()
```



8 Label Encoding:

This code translates each of the 7 labels to numeric values ranging from 0 to 1.

```
# label encoding to numeric values from text
le = LabelEncoder()
le.fit(df_skin['dx'])
LabelEncoder()
print(list(le.classes_))

df_skin['label'] = le.transform(df_skin["dx"])
print(df_skin.sample(10))
```

```
['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']
        lesion_id      image_id     dx    dx_type   age     sex  \
9695  HAM_0005282  ISIC_0028730  akiec      histo  65.0    male
6653  HAM_0006795  ISIC_0031892     nv  follow_up  55.0  female
5710  HAM_0006819  ISIC_0025590     nv  follow_up  40.0    male
5504  HAM_0001779  ISIC_0031439     nv  follow_up  65.0    male
2435  HAM_0006291  ISIC_0026490   vasc  consensus  65.0  female
5221  HAM_0005108  ISIC_0026078     nv  follow_up  50.0  female
2836  HAM_0003191  ISIC_0028050    bcc      histo  65.0  female
6960  HAM_0003222  ISIC_0026382     nv      histo  45.0  female
1367  HAM_0006152  ISIC_0027238    mel      histo  55.0  female
6971  HAM_0001623  ISIC_0029488     nv      histo  85.0  female

         localization          lesion_type  lesion_ID  label
9695  lower extremity     Actinic keratoses          4      0
6653             back      Melanocytic nevi          0      5
5710            trunk      Melanocytic nevi          0      5
5504  upper extremity      Melanocytic nevi          0      5
2435            trunk      Vascular lesions          5      6
5221  lower extremity      Melanocytic nevi          0      5
2836             hand  Basal cell carcinoma          3      1
6960  lower extremity      Melanocytic nevi          0      5
1367  upper extremity              Melanoma          1      4
6971          unknown      Melanocytic nevi          0      5
```

9 Image Augmentation:

This code does down and upscaling of images and caps it to 500 per category with the use of resampling method.

```
[ ]  n_samples=500
     df_0_balanced = resample(df_0, replace=True, n_samples=n_samples, random_state=42)
     df_1_balanced = resample(df_1, replace=True, n_samples=n_samples, random_state=42)
     df_2_balanced = resample(df_2, replace=True, n_samples=n_samples, random_state=42)
     df_3_balanced = resample(df_3, replace=True, n_samples=n_samples, random_state=42)
     df_4_balanced = resample(df_4, replace=True, n_samples=n_samples, random_state=42)
     df_5_balanced = resample(df_5, replace=True, n_samples=n_samples, random_state=42)
     df_6_balanced = resample(df_6, replace=True, n_samples=n_samples, random_state=42)
```

10 CNN Model:

```
from keras.applications import ResNet50
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import BatchNormalization

data_dir = './image_data'
batch_size = 64
epochs = 1000

# load the pre-trained ResNet50 model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(224, 224, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
# model.add(Dropout(0.3))

model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
# model.add(Dropout(0.3))

model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
# model.add(Dropout(0.3))

model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
# model.add(Dropout(0.4))

model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
# model.add(Dropout(0.3))

model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
# model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
model.summary()

model.summary()
# Compile the model
optimizer = Adam(learning_rate = 0.0001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-3)
model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy',f1_m,precision_m, recall_m])

# Train the model on the new data
train_datagen = ImageDataGenerator(rescale=1./255,
```

Contd.

```
# Train the model on the new data
train_datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training') # set as training data

validation_generator = train_datagen.flow_from_directory(
    data_dir, # same directory as training data
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation') # set as validation data

# datagen.fit(X_train)



es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
mcp_save = ModelCheckpoint('/content/drive/MyDrive/ML data.h5', save_best_only=True, monitor='val_loss', mode='min')
# org model result data
# history = model.fit(datagen.flow(X_train,y_train),
#                     epochs = epochs,
#                     batch_size = batch_size,
#                     shuffle = True,
#                     validation_data = (X_test, y_test),
#                     callbacks=[es, mcp_save]
#                 )

history = model.fit(
    train_generator,
    steps_per_epoch = train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs = epochs,
    shuffle = True,
    callbacks=[es, mcp_save])
```

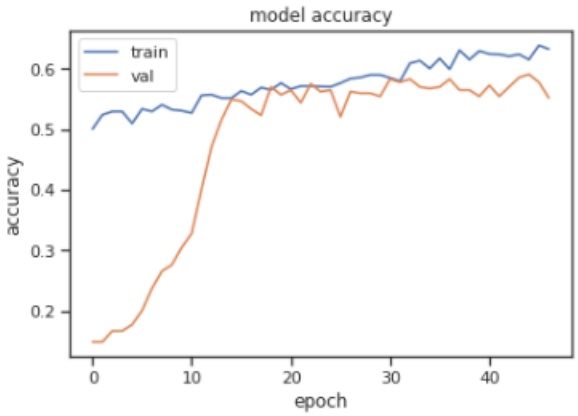
11 Base CNN Model Output:

```
batch_normalization_10 (Bat  (None, 10, 10, 32)       128
chNormalization)

activation_10 (Activation)  (None, 10, 10, 32)       0

max_pooling2d_9 (MaxPooling  (None, 5, 5, 32)        0
2D)

flatten_1 (Flatten)         (None, 800)              0

dense_2 (Dense)             (None, 512)              410112

batch_normalization_11 (Bat  (None, 512)             2048
chNormalization)

activation_11 (Activation)  (None, 512)              0

dropout_1 (Dropout)         (None, 512)              0

dense_3 (Dense)             (None, 7)                3591

=================================================================
Total params: 454,279
Trainable params: 452,935
Non-trainable params: 1,344
_________________________________________________________________
Found 1682 images belonging to 7 classes.
Found 416 images belonging to 7 classes.
Epoch 1/1000
26/26 [==============================] - 39s 1s/step - loss: 2.5316 - accuracy: 0.1799 - f1_m: 0.1008 - precision_m: 0.2048 - recall_m: 0.0673 - val_loss: 1.9400 - val_accuracy: 0.1927 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 2/1000
26/26 [==============================] - 35s 1s/step - loss: 2.0450 - accuracy: 0.3109 - f1_m: 0.2337 - precision_m: 0.4039 - recall_m: 0.1651 - val_loss: 1.9548 - val_accuracy: 0.1823 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 3/1000
26/26 [==============================] - 35s 1s/step - loss: 1.8667 - accuracy: 0.3665 - f1_m: 0.2865 - precision_m: 0.4577 - recall_m: 0.2096 - val_loss: 1.9962 - val_accuracy: 0.1510 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 4/1000
26/26 [==============================] - 34s 1s/step - loss: 1.7310 - accuracy: 0.3986 - f1_m: 0.3242 - precision_m: 0.5109 - recall_m: 0.2390 - val_loss: 2.0528 - val_accuracy: 0.1484 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 5/1000
26/26 [==============================] - 34s 1s/step - loss: 1.6527 - accuracy: 0.4116 - f1_m: 0.3379 - precision_m: 0.4991 - recall_m: 0.2561 - val_loss: 2.1415 - val_accuracy: 0.1484 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 6/1000
26/26 [==============================] - 34s 1s/step - loss: 1.5847 - accuracy: 0.4308 - f1_m: 0.3746 - precision_m: 0.5390 - recall_m: 0.2883 - val_loss: 2.2036 - val_accuracy: 0.1536 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 7/1000
26/26 [==============================] - 35s 1s/step - loss: 1.5592 - accuracy: 0.4477 - f1_m: 0.3805 - precision_m: 0.5392 - recall_m: 0.2951 - val_loss: 2.3096 - val_accuracy: 0.1484 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 8/1000
26/26 [==============================] - 34s 1s/step - loss: 1.5053 - accuracy: 0.4524 - f1_m: 0.3897 - precision_m: 0.5590 - recall_m: 0.3006 - val_loss: 2.3461 - val_accuracy: 0.1484 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 9/1000
26/26 [==============================] - 34s 1s/step - loss: 1.4755 - accuracy: 0.4753 - f1_m: 0.4046 - precision_m: 0.5666 - recall_m: 0.3156 - val_loss: 2.3936 - val_accuracy: 0.1510 - val_f1_m: 0.0000e+00 - val_precision_m: 0.0000e+00 - val_reca
Epoch 10/1000
26/26 [==============================] - 34s 1s/step - loss: 1.3906 - accuracy: 0.5019 - f1_m: 0.4367 - precision_m: 0.5925 - recall_m: 0.3470 - val_loss: 2.4279 - val_accuracy: 0.1484 - val_f1_m: 0.0103 - val_precision_m: 0.3333 - val_recall_m: 0
Epoch 11/1000
26/26 [==============================] - 34s 1s/step - loss: 1.3832 - accuracy: 0.5056 - f1_m: 0.4473 - precision_m: 0.6120 - recall_m: 0.3541 - val_loss: 2.4792 - val_accuracy: 0.1484 - val_f1_m: 0.0682 - val_precision_m: 0.2869 - val_recall_m: 0
Epoch 11: early stopping
```

12 Base CNN Model Graph:

Below code gives the graph that plots accuracy against epoch

```
[ ]  ## Plotting the training and Validation accuracy
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'val'], loc='upper left')
     plt.savefig("/content/drive/MyDrive/ML data/cnn_train_val_acc.jpg")
     plt.show()
```



Optimized CNN model's output:

```
optimizer = Adam(learning_rate = 0.00005, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e-3)
model.compile(optimizer = optimizer, loss = 'categorical_crossentropy', metrics = ['accuracy',f1_m,precision_m, recall_m])

history = model.fit(
    train_generator,
    steps_per_epoch = train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs = epochs,
    shuffle = True,
    callbacks=[es, mcp_save])
```

```
Epoch 1/1000
26/26 [==============================] - 37s 1s/step - loss: 1.3557 - accuracy: 0.5006 - f1_m: 0.4420 - precision_m: 0.5890 - recall_m: 0.3553 - val_loss: 2.4703 - val_accuracy: 0.1484 - val_f1_m: 0.0774 -
Epoch 2/1000
26/26 [==============================] - 35s 1s/step - loss: 1.3133 - accuracy: 0.5241 - f1_m: 0.4838 - precision_m: 0.6451 - recall_m: 0.3891 - val_loss: 2.4835 - val_accuracy: 0.1484 - val_f1_m: 0.1152 -
Epoch 3/1000
26/26 [==============================] - 35s 1s/step - loss: 1.2983 - accuracy: 0.5297 - f1_m: 0.4771 - precision_m: 0.6313 - recall_m: 0.3847 - val_loss: 2.4657 - val_accuracy: 0.1667 - val_f1_m: 0.1266 -
Epoch 4/1000
26/26 [==============================] - 35s 1s/step - loss: 1.2913 - accuracy: 0.5297 - f1_m: 0.4733 - precision_m: 0.6281 - recall_m: 0.3805 - val_loss: 2.4597 - val_accuracy: 0.1667 - val_f1_m: 0.1139 -
Epoch 5/1000
26/26 [==============================] - 34s 1s/step - loss: 1.3036 - accuracy: 0.5099 - f1_m: 0.4670 - precision_m: 0.6161 - recall_m: 0.3773 - val_loss: 2.3407 - val_accuracy: 0.1771 - val_f1_m: 0.1264 -
Epoch 6/1000
26/26 [==============================] - 35s 1s/step - loss: 1.2998 - accuracy: 0.5340 - f1_m: 0.4703 - precision_m: 0.6154 - recall_m: 0.3820 - val_loss: 2.2331 - val_accuracy: 0.2005 - val_f1_m: 0.1362 -
Epoch 7/1000
26/26 [==============================] - 34s 1s/step - loss: 1.2824 - accuracy: 0.5297 - f1_m: 0.4720 - precision_m: 0.6284 - recall_m: 0.3793 - val_loss: 2.1511 - val_accuracy: 0.2370 - val_f1_m: 0.1463 -
Epoch 8/1000
26/26 [==============================] - 34s 1s/step - loss: 1.2479 - accuracy: 0.5408 - f1_m: 0.4864 - precision_m: 0.6318 - recall_m: 0.3974 - val_loss: 2.0455 - val_accuracy: 0.2656 - val_f1_m: 0.1396 -
Epoch 9/1000
26/26 [==============================] - 34s 1s/step - loss: 1.2533 - accuracy: 0.5328 - f1_m: 0.4790 - precision_m: 0.6195 - recall_m: 0.3924 - val_loss: 1.9738 - val_accuracy: 0.2760 - val_f1_m: 0.1784 -
Epoch 10/1000
26/26 [==============================] - 35s 1s/step - loss: 1.2636 - accuracy: 0.5309 - f1_m: 0.4961 - precision_m: 0.6391 - recall_m: 0.4068 - val_loss: 1.8185 - val_accuracy: 0.3047 - val_f1_m: 0.1714 -
```

Extracting CNN Features for furthure modeling

Extract CNN features for Meta Learner

```
feature_extractor = keras.Model(
   inputs=model.inputs,
   outputs=model.get_layer(name="dense_2").output,
)
```

```
[ ] from tensorflow.keras.preprocessing import image
    from tensorflow.keras.applications.resnet50 import preprocess_input
```

```
[ ] CNN_features = []

    for index, row in df_skin_balanced.iterrows():
        # retrieve the file path and label
        file_path = row['path']
        img = image.load_img(file_path, target_size=(224, 224))
        img_array = image.img_to_array(img)/255.0
        img_batch = np.expand_dims(img_array, axis=0)
        features = feature_extractor.predict(img_batch)
        CNN_features.append(features.flatten())
```

```
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
```

Generating & Extracting GLCM Features:

```
def grey_scale_prep(img):
    """
    prepare a scale picture for comatrix feature exraction
    """
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # h, w = gray.shape
    # ymin, ymax, xmin, xmax = h//3, h*2//3, w//3, w*2//3
    # crop = gray[ymin:ymax, xmin:xmax]
    resize = cv2.resize(gray, (512,512))
    return resize



def calc_glcm_all_agls(img, props, dists=[5], agls=[18, 36, 54, 72, 90, 108, 126, 144, 162, 180], lvl=256, sym=True, norm=True):
    """
    calculate greycomatrix() & greycoprops() for angle 0, 45, 90, 135
    """
    glcm = greycomatrix(img,
                        distances=dists,
                        angles=agls,
                        levels=lvl,
                        symmetric=sym,
                        normed=norm)
    feature = []
    glcm_props = [propery for name in props for propery in greycoprops(glcm, name)[0]]
    for item in glcm_props:
            feature.append(item)
    #feature.append(label)

    return feature


properties = ['dissimilarity', 'correlation', 'homogeneity', 'contrast', 'ASM', 'energy']

columns = []
angles = ["18", "36", "54", "72", "90", "108", "126", "144", "162", "180"]
for name in properties :
    for ang in angles:
        columns.append(name + "_" + ang)


def get_glcm(x):
    """
    input : grey scaled images
    output : dataframe with all GLCM properties
    """
    glcm_all_agls = []
    for img in x:
        glcm_all_agls.append(
                calc_glcm_all_agls(img,
                                   props=properties)
                                )
    glcm_df = pd.DataFrame(glcm_all_agls,
                           columns = columns)
    return glcm_df
```

Later stage the features from GLCM and CNN are combined and are fed to meta learner.

Building Meta model and running through Logistic regression and output of the Logistic regression.

Make Meta Model

```
[ ] X_train, X_test, y_train, y_test = train_test_split(X,df_skin_balanced["label"], random_state = 0)
```

Meta Learner

```
[ ] from sklearn.linear_model import LogisticRegression
    meta_learner_lr = LogisticRegression(random_state=0).fit(X_train, y_train)
    meta_learner_lr.score(X_test, y_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
0.6034285714285714
```

```
[ ]
    from sklearn.neural_network import MLPClassifier
    meta_learner_mlp = MLPClassifier(hidden_layer_sizes=[500,300,100], random_state=0, max_iter=300).fit(X_train, y_train)
    meta_learner_mlp.score(X_test, y_test)
```

```
0.7965714285714286
```

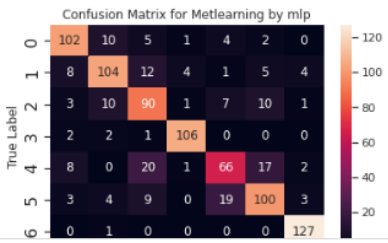
Running through MLP classifier and its output along with confusion matrix.

```
[ ]
    from sklearn.neural_network import MLPClassifier
    meta_learner_mlp = MLPClassifier(hidden_layer_sizes=[500,300,100], random_state=0, max_iter=300).fit(X_train, y_train)
    meta_learner_mlp.score(X_test, y_test)
```

```
0.7965714285714286
```

```
[ ] import seaborn as sns
    from sklearn.metrics import confusion_matrix

    # Fit the logistic regression model
    meta_learner_mlp = MLPClassifier(random_state=0).fit(X_train, y_train)

    # Generate predictions on the test set
    y_pred = meta_learner_mlp.predict(X_test)

    # Compute the confusion matrix
    confusion_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap using Seaborn's heatmap() function
    sns.heatmap(confusion_matrix, annot=True, fmt="d")

    # Add labels and adjust font size
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix for Metlearning by mlp")
    plt.tick_params(labelsize=16)

    # Display the heatmap
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum ite
  warnings.warn(
```



Similarly Random Forest:

```
[ ] import numpy as np
    from sklearn.ensemble import RandomForestClassifier
    meta_learner_RFC = RandomForestClassifier(random_state=0).fit(X_train, y_train)
    meta_learner_RFC.score(X_test, y_test)
```

```
0.8034285714285714
```

```
[ ] import seaborn as sns
    from sklearn.metrics import confusion_matrix

    # Fit the logistic regression model
    meta_learner_rfc = RandomForestClassifier(random_state=0).fit(X_train, y_train)

    # Generate predictions on the test set
    y_pred = meta_learner_rfc.predict(X_test)

    # Compute the confusion matrix
    confusion_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap using Seaborn's heatmap() function
    sns.heatmap(confusion_matrix, annot=True, fmt="d")

    # Add labels and adjust font size
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix for Metlearning by rfc")
    plt.tick_params(labelsize=16)
```