Configuration Manual

MSc Research Project
Data Analytics

Aishwarya Ghongane
Student ID: x20177259

School of Computing
National College of Ireland

Supervisor: Dr. Prashanth Nayak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aishwarya Ghongane
Student ID:	x20177259
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Prashanth Nayak
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1116
Page Count:	15

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:	12th August 2022

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 keepa 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

Aishwarya Ghongane
x20177259

1 Introduction

The objective of the research project to hierarchically classify the Yoga poses. This configuration manual outlines the hardware and software requirements essential to help future researchers to replicate the project. The different stages of project implementation, starting from data acquisition to model testing and evaluation are discussed in detail. The reference code repositories are provided in footnotes wherever applicable.

2 System Configurations

This section outlines the hardware and software setup used during implementation.

2.1 Hardware Configurations

The hardware configuration of the computer machine that was used to implement this project is shown in Figure 1. The M1 chip has 8-core CPU and 8-core GPU which proved powerful to process image dataset.

Figure 1: Hardware Configuration

2.2 Software Confugurations

To implement the coding part, Google Colaboratory IDE was used. Due to huge corpus of data, CPU power was not enough, hence, Google Colab Pro was used to access the GPUsand enhance RAM capacity. The dataset was uploaded on Google drive and the data was accessed by mounting the drive to Google Colab as shown in Figure 2. Python programming language was used throughout the implementation part.

Connecting to Google Drive to access all the data files

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Figure 2: Mounting Google Drive

3 Importing required libraries

To implement this project, different libraries were downloaded as and when required during the step-by-step implementation as shown in Figure 3

```
import os
import socket
import urllib.request
import pandas as pd
import PIL
from pathlib import Path
from PIL import UnidentifiedImageError
import seaborn as sns
import matplotlib.pyplot as plt
import cv2
import numpy as np
import tensorflow
from tensorflow.keras.optimizers import SGD,Adam
from keras.utils import np_utils
import keras
import random
from keras.callbacks import ModelCheckpoint,CSVLogger
import tensorflow as tf
import keras.backend as K
from PIL import Image
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")
from keras.layers import Dense,Dropout,Conv2D,Input,MaxPool2D,Flatten,Add,Activation,GlobalAveragePooling2D,BatchNormalization,MaxPooling2D,Conv2D
from keras.models import Model
keras.backend.set_image_data_format('channels_last')
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from tensorflow.keras import layers
from tensorflow.keras import backend
from tensorflow.keras import models
from tensorflow.keras import utils as keras_utils
from keras_applications import imagenet_utils
from keras_applications.imagenet_utils import decode_predictions
from keras_applications.imagenet_utils import _obtain_input_shape
```

Figure 3: Installing Important Libraries

4 Data Collection and Pre-processing

4.1 Data Collection

For the purpose of this research project, the Yoga-82 dataset was used which is available here [1]. The dataset comprised of three text files, namely, a train data file, test data file, and another file containing the URLs of all the images belonging to different yoga classes. The train and test files consist name of the yoga class folder, image name, and the hierarchical label of the yoga pose. All the images were downloaded using Python programming as shown in Figure 4. The invalid URLs were ignored while downloading the images.

```
file_dir = f"/content/drive/MyDrive/Yoga_Image_Dataset_Links/"    #Path to text file containg Image URLs
download_img_dir = f"/content/drive/MyDrive/Downloaded_Images/"   #New directory to store downloaded images

socket.setdefaulttimeout(10)          #default timeout parameter set, if the website fails to respond
def check_folder_exists(dir_name):    #function to check if yoga class directory exists, if not creates new directory
    if not os.path.isdir(dir_name):
       os.makedirs(dir_name)

all_files = os.listdir(file_dir)
txt_files = filter(lambda x: x[-4:] == '.txt', all_files)   #To select only text files
try:
    for file in txt_files:
        folder_name = os.path.splitext(file)[0]
        image_directory = file_dir+file
        with open(image_directory,'r') as data_file:
             check_folder_exists(download_img_dir+folder_name)
             print("opening file",folder_name)
             for line in data_file:
                 data = line.split()
                 file_name = data[0].replace(" ","_")
                 url = data[1]
                 try:
                   urllib.request.urlretrieve(url, download_img_dir+file_name)
                   print(" Status", download_img_dir+file_name,' :downloaded successfully')
                 except Exception as e:
                   print(" File download error",e)
                   continue
except Exception as e:
       print(" An exception occurred",e)
```

Figure 4: Downloading the dataset

4.2 Data Cleaning

After downloading the images, some of the images were corrupted, while some images were not in readable format. Such images were deleted from the directory using the Python Imaging Library (PIL) module in Python as shown in Figure 5.

[1]https://sites.google.com/view/yoga-82/home

```python
import PIL
from pathlib import Path
from PIL import UnidentifiedImageError
rootdir = f"/content/drive/MyDrive/Downloaded_Images/"
path = Path(rootdir).rglob("*.jpg")                         #Filter image URLs with jpg image format
for img_p in path:
    try:
        img = PIL.Image.open(img_p)
    except PIL.UnidentifiedImageError:
        print("error image",img_p)
        os.remove(img_p)                                    #remove corrupt image
```

Figure 5: Removing corrupt images

4.3 Data Balancing

The dataset was highly imbalanced. This imbalance ofthe dataset was impacting model performance. Hence, to have enough data and adequate samples from each class, 110 images were selected from each class as shown in Figure 6.

```python
from shutil import copy

src_dir = f"/content/drive/MyDrive/Downloaded_Images/"   #Source Directory
dst_dir = f"/content/drive/MyDrive/Yoga_82/"             #Destination Directory

try:
    for subdir, dirs, files in os.walk(src_dir):
        dst_sub = dst_dir + subdir.split("/")[-1]
        for f in files[:110]:                            #Select only 110 images from each class
            if f.endswith('.jpg'):
                copy(os.path.join(subdir, f), dst_sub)
except Exception as e:
    print("Error occurred while copying the files", e)
```

Figure 6: Balancing the Dataset

4.4 Creating train data and test data

The Yoga-82 dataset had around 28k images. However, the data cleaning step left us with around 15k images. Hence, the train and test files provided cannot be used directly.So to maintain data consistency, new train and test data files were created to ensure, that the entries of deleted images while data cleaning are removed from both train and test data files. This was achieved using the Python's Pandas series function as shown in Figure 7.

```
image_list = []
dst_dir = f"/content/drive/MyDrive/Yoga_82/"
for subdir, dirs, files in os.walk(dst_dir):
    for file in files:
        image_list.append(subdir.split("/")[-1]+"/"+file)

image_series = pd.Series(image_list)
print("Total Number of Images in new dataset:", len(image_series))
```

Figure 7: Filtering the dataset

The images retrieved using series function were then filtered by comparing with the image entries present in train and test files.

```
train_df = pd.read_csv(f"/content/drive/MyDrive/Train_Test_Data/yoga_train.txt",header= None)
train_df.head()
print("Total number of images in used baseline project train dataset:", len(train_df))

filter_df = train_df[train_df[0].isin(image_series)]

print("Total number of images in used current project train dataset:", len(filter_df))
filter_df.to_csv(f"/content/drive/MyDrive/Train_Test_Data/Train_82.txt",index = False,header = None)
```

Figure 8: Creating train data file

```
test_df = pd.read_csv(f"/content/drive/MyDrive/Train_Test_Data/yoga_test.txt",header= None)
test_df.head()
print("Total number of images in used baseline project test dataset:", len(test_df))

filter_test_df = test_df[test_df[0].isin(image_series)]

print("Total number of images in used current project train dataset:", len(filter_test_df))
filter_test_df.to_csv(f"/content/drive/MyDrive/Train_Test_Data/Test_82.txt",index = False,header = None)
```

Figure 9: Creating test data file

4.5 Data Transformation

After reducing the dataset for the purpose of data balance, the train and test datasetsize was not enough to solve the research problem. This is because, image classifica- tion requires massive image data. Hence, image augmentation techniques were employed.

The augmentation techniques such as rotation, and flipping the image were applied using Open CV library [2] as shown in Figure 10 respectively.

```
dir_R45_R90 = f"/content/drive/MyDrive/Yoga_82_R45_R90/"
list_items_in_dir = os.listdir(dir_R45_R90)
c = 0
for subdir, dirs, files in os.walk(dir_R45_R90):
    try:
        for index, item in enumerate(files):
            image, ext = item.split(".")
            read_img = (subdir+"/"+item)
            img = cv2.imread(read_img)
            (h, w) = img.shape[:2]
            (cX, cY) = (w // 2, h // 2)

            #Flip the image at vertical axis
            img_FV= cv2.flip(img, 1)
            save_FV_img = (subdir + "/"+image+"_FV"+"."+ ext)
            cv2.imwrite(save_FV_img, img_FV)

            #Rotate image by 45 degree angle
            img_matrix_R45 = cv2.getRotationMatrix2D((cX, cY), 45, 1.0)
            img_R45 = cv2.warpAffine(img, img_matrix_R45, (w, h))
            save_img = (subdir + "/"+image+"_45"+"."+ ext)
            cv2.imwrite(save_img, img_R45)
            #Rotate image by 90 degree angle
            img_matrix_R90 = cv2.getRotationMatrix2D((cX, cY), 90, 1.0)
            img_R90 = cv2.warpAffine(img, img_matrix_R90, (w, h))
            save_90_img = (subdir + "/"+image+"_90"+"."+ ext)
            cv2.imwrite(save_90_img, img_R90)
    except Exception as e:
              print("Error:",e)
              continue
```

Figure 10: Data Augmentation

5 Creating Augmented train data and batches for processing

The next step was to create the train data file with the newly augmented images as shown in Figure 11. Before proceeding with the model building process, the train, validation, and test files are processed to generate data in batch size of 32. Figure 12 shows data generator function for train data.Similarly, it was done for validation data and test data as shown in Figure 13 and Figure 14 respectively.

[2]https://towardsdatascience.com/top-python-libraries-for-image-augmentation-in-computer-vision-2566bed0533e

```python
Train_R45_R90 = f"/content/drive/MyDrive/Train_Test_Data/yoga_train_R45_R90.txt"
ff = open(Train_R45_R90,'r')
lines = ff.readlines()

cnt = 0
for l in lines:
    img_name = l.split(',')[0]
    x1_label = l.split(',')[1]
    x2_label = l.split(',')[2]
    x3_label = l.split(',')[-1]
    image, ext = img_name.split(".")
    image_R45 = (image + "_45" + "." + ext + "," + x1_label + "," + x2_label + "," + x3_label)
    image_R90 = (image + "_90" + "." + ext + "," + x1_label + "," + x2_label + "," + x3_label)
    write_file = open(Train_R45_R90,'a')
    if cnt==0:
        write_file.write("\n")
        cnt=1
    write_file.write(image_R45)
    write_file.write(image_R90)
write_file.close()
```

Figure 11: Creating augmented train data file

```python
def generator_train_batch(train_txt,batch_size,num_classes,img_path):
    ff = open(train_txt, 'r')
    lines = ff.readlines()
    num = len(lines)
    class_6 = num_classes[0]
    class_20 = num_classes[1]
    class_82 = num_classes[2]
    try:
      while True:
          new_line = []
          index = [n for n in range(num)]
          random.shuffle(index)
          for m in range(num):
              new_line.append(lines[index[m]])
          try:
            for i in range(int(num/batch_size)):
                a = i*batch_size
                b = (i+1)*batch_size
                x_train, x1_labels, x2_labels, x3_labels = process_batch(new_line[a:b],img_path,train=True)
                x = preprocess(x_train)
                y1 = np_utils.to_categorical(np.array(x1_labels), class_6)
                y2 = np_utils.to_categorical(np.array(x2_labels), class_20)
                y3 = np_utils.to_categorical(np.array(x3_labels), class_82)
                y = [y1,y2,y3]
                print(y)
                yield x, y
          except Exception as e:
            print("Error:", e)

    except Exception as e:
      print("Error:", e)
```

Figure 12: Data Generator function to create batches of training data images

```python
def generator_val_batch(val_txt,batch_size,num_classes,img_path):
    f = open(val_txt, 'r')
    lines = f.readlines()
    num = len(lines)
    class_6 = num_classes[0]
    class_20 = num_classes[1]
    class_82 = num_classes[2]
    while True:
        new_line = []
        index = [n for n in range(num)]
        #random.shuffle(index)
        for m in range(num):
            new_line.append(lines[index[m]])
        for i in range(int(num / batch_size)):
            a = i * batch_size
            b = (i + 1) * batch_size
            y_test,y1_labels, y2_labels, y3_labels = process_batch(new_line[a:b],img_path,train=False)
            x = preprocess(y_test)
            y1 = np_utils.to_categorical(np.array(y1_labels), class_6)
            y2 = np_utils.to_categorical(np.array(y2_labels), class_20)
            y3 = np_utils.to_categorical(np.array(y3_labels), class_82)
            test_data = x
            y = [y1,y2,y3]
            yield test_data, y
```

Figure 13: Data Generator function to create batches of validation data images

```python
def generator_test_batch(test_txt,batch_size,num_classes,img_path):
    f = open(test_txt, 'r')
    lines = f.readlines()
    num = len(lines)
    class_6 = num_classes[0]
    class_20 = num_classes[1]
    class_82 = num_classes[2]
    while True:
        new_line = []
        index = [n for n in range(num)]
        #random.shuffle(index)
        for m in range(num):
            new_line.append(lines[index[m]])
        for i in range(int(num / batch_size)):
            a = i * batch_size
            b = (i + 1) * batch_size
            y_test,y1_labels, y2_labels, y3_labels = process_batch(new_line[a:b],img_path,train=False)
            x = preprocess(y_test)
            y1 = np_utils.to_categorical(np.array(y1_labels), class_6)
            y2 = np_utils.to_categorical(np.array(y2_labels), class_20)
            y3 = np_utils.to_categorical(np.array(y3_labels), class_82)
            test_data = x
            y = [y1,y2,y3]
            yield test_data, y
```

Figure 14: Data Generator function to create batches of testing data images

6 Model Building

6.1 Model Building

In this step, the DenseNet-201 and ResNet-50 architectures were modified to as- sist hierarchical classification. The state-of-the-art model by Verma et al. (2020) using DenseNet-201 has been implemented to form a baseline for comparing the results. The ResNet-50 network has been modified as a part of this research. The code for building the basic structure of DenseNet 3 and ResNet4 networks is available on Keras github re- pository. The code for modified architecture of DenseNet-201 is available on Yoga-82 github repository 5. The code blocks for modfified ResNet-50 architecture are shown in Figure 15, Figure 16, and Figure 17.

```
def identity_block(input_tensor, kernel_size, filters, stage, block):
    filters1, filters2, filters3 = filters
    if backend.image_data_format() == 'channels_last':
        bn_axis = 3
    else:
        bn_axis = 1
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = layers.Conv2D(filters1, (1, 1),
                      kernel_initializer='he_normal',
                      name=conv_name_base + '2a')(input_tensor)
    x = layers.BatchNormalization(axis=bn_axis, name=bn_name_base + '2a')(x)
    x = layers.Activation('relu')(x)

    x = layers.Conv2D(filters2, kernel_size,
                      padding='same',
                      kernel_initializer='he_normal',
                      name=conv_name_base + '2b')(x)
    x = layers.BatchNormalization(axis=bn_axis, name=bn_name_base + '2b')(x)
    x = layers.Activation('relu')(x)

    x = layers.Conv2D(filters3, (1, 1),
                      kernel_initializer='he_normal',
                      name=conv_name_base + '2c')(x)
    x = layers.BatchNormalization(axis=bn_axis, name=bn_name_base + '2c')(x)

    x = layers.add([x, input_tensor])
    x = layers.Activation('relu')(x)
    return x
```

Figure 15: Identity block for ResNet-50 network

```python
#Build Convolutional Block for ResNet-50
def conv_block(input_tensor,kernel_size,filters,stage,block,strides=(2, 2)):
    filters1, filters2, filters3 = filters
    if backend.image_data_format() == 'channels_last':
        bn_axis = 3
    else:
        bn_axis = 1
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'
    x = layers.Conv2D(filters1, (1, 1), strides=strides,
                      kernel_initializer='he_normal',
                      name=conv_name_base + '2a')(input_tensor)
    x1 = layers.BatchNormalization(axis=bn_axis,
                                   name=bn_name_base + '2a')(x)
    x1 = layers.Activation('relu')(x)
    x1 = layers.Conv2D(filters2, kernel_size,
                       padding='same', kernel_initializer='he_normal',
                       name=conv_name_base + '2b')(x1)
    x1 = layers.BatchNormalization(axis=bn_axis,
                                   name=bn_name_base + '2b')(x1)
    x1 = layers.Activation('relu')(x1)
    x1 = layers.Conv2D(filters3, (1, 1), kernel_initializer='he_normal',
                       name=conv_name_base + '2c')(x1)
    x1 = layers.BatchNormalization(axis=bn_axis,
                                   name=bn_name_base + '2c')(x1)
    shortcut = layers.Conv2D(filters3, (1, 1), strides=strides,
                             kernel_initializer='he_normal',
                             name=conv_name_base + '1')(input_tensor)
    shortcut = layers.BatchNormalization(axis=bn_axis,
                                         name=bn_name_base + '1')(shortcut)
    x = layers.add([x1, shortcut])
    x = layers.Activation('relu')(x1)
    return x
```

Figure 16: Convolutional block for ResNet-50 network

```python
def ResNet50_hir_new(
        input_shape = (224,224,3),
        class_6=6,
        class_20=20,
        class_82=82):

    inputs = Input(input_shape)
    base_model= ResNet50_hir(include_top=False, weights=None,
                         input_tensor = inputs,
                         backend = keras.backend,
                         layers = keras.layers,models = keras.models,
                         utils = keras.utils)
    [x1,x2,x] = base_model.output
    x1 = BatchNormalization( epsilon=1.001e-5, name = 'bn_class6_last')(x1)
    x1 = Activation('relu', name='relu_class6_last')(x1)
    x1 = GlobalAveragePooling2D(name='GAvgPool_class6_last')(x1)
    x2 = BatchNormalization( epsilon=1.001e-5, name = 'bn_class20_last')(x2)
    x2 = Activation('relu', name='relu_class20_last')(x2)
    x2 = GlobalAveragePooling2D(name='GAvgPool_class20_last')(x2)
    x = GlobalAveragePooling2D()(x)

    x1 = Dense(class_6, activation= 'softmax')(x1)

    x2 = Dense(class_20, activation= 'softmax')(x2)

    x = Dense(class_82, activation='softmax')(x)

    model = Model(inputs, [x1,x2,x])

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

    return model
```

Figure 17: Modified ResNet-50 network

7 Model Training

Finally, the model is compiled, trained, and saved for evaluation purpose as shown in Figure 18.

```
path = f'/content/drive/MyDrive/Yoga_82_R45_R90/'
img_path = path
path_test = f'/content/drive/MyDrive/Downloaded_Images/'
train_file = f'/content/drive/MyDrive/Train_Test_Data/Train_R45_R90.txt'
val_file = f'/content/drive/MyDrive/Train_Test_Data/Validation.txt'
test_file = f'/content/drive/MyDrive/Train_Test_Data/Test.txt'
f1 = open(train_file, 'r')
lines = f1.readlines()
f1.close()
train_samples = len(lines)
f2 = open(test_file, 'r')
lines = f2.readlines()
f2.close()
test_samples = len(lines)
num_classes = [6,20,82]
batch_size = 32
epochs = 30

model = ResNet50_hir_new()

lr = 0.003
sgd = SGD(lr=lr, momentum=0.9, nesterov=False)
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

model.compile(loss=['categorical_crossentropy','categorical_crossentropy','categorical_crossentropy'], loss_weights=[1,1,1], optimizer= sgd,
              metrics=['accuracy'])
model.summary()
output_files = f'/content/drive/MyDrive/OutputFiles/'
checkpointer = ModelCheckpoint(filepath=output_files+'weights_betweenhirarModify_lw111_dense32_nopre_mix_.0003.hdf5', verbose=1, save_best_only= True,
                               monitor='val_loss')
csv_logger= CSVLogger(output_files+'ResNet50_With_Augmentation.csv')

train_val_fit = model.fit_generator(generator_train_batch(train_file, batch_size, num_classes,img_path),
                      steps_per_epoch=train_samples // batch_size,
                      epochs=epochs,
                      callbacks=[checkpointer, csv_logger],
                      validation_data=generator_val_batch(val_file, batch_size,num_classes,path_test),
                      validation_steps=test_samples // batch_size)
model.save("ResNet50_WithAugmentation.h5")
```

Figure 18: Model compilation and training

[3]https://github.com/keras-team/keras-applications/blob/master/keras$_a$*pplications/densenet.py*
[4]https://github.com/keras-team/keras-applications/blob/master/keras$_a$*pplications/resnet*50*.py*
[5]https://github.com/maniver7/yoga-82

8 Model Evaluation

The model has been evaluated on validation dataset using evaluate function as shown in figure Figure 19. Post this, accuracy for all the levels is derived as shown in Figure 20.

```
#Model Evaluation
val_file = f'/content/drive/MyDrive/Train_Test_Data/Validation.txt'
score = model.evaluate_generator(generator_val_batch(val_file,8,
                                            num_classes,path_test),
                          steps=test_samples // 8, verbose=1)

metrics1 = model.metrics_names
print(metrics1)
```

Figure 19: Model evaluation

```
#Deriving training and validation Accuracy
acc_CL_1 = train_val_fit.history['dense_1_accuracy']        #Coarse level 1
acc_CL_2 = train_val_fit.history['dense_2_accuracy']        #Coarse Level 2
acc_FL = train_val_fit.history['dense_accuracy']            #Fine Level

val_acc_CL1 = train_val_fit.history['val_dense_accuracy']
val_acc_CL2 = train_val_fit.history['val_dense_1_accuracy']
val_acc_FL = train_val_fit.history['val_dense_2_accuracy']

loss = train_val_fit.history['loss']
val_loss = train_val_fit.history['val_loss']
```

Figure 20: Calculating Model Accuracy

9 Model Prediction

For predicting and classifying the Yoga classes, a custom function has been defined. This is because the default function to plot the confusion matrix does not support hierarchical classification. This is shown in Figure 21.

```
#To plot confusion matrix for all the three levels of hierarchy
def PlotConfusionMatrix(preds,test_file):
    level_1 = preds[0]
    level_2 = preds[1]
    level_3 = preds[2]
    predicted_class_indices_1 = np.argmax(level_1,axis=1)
    print(predicted_class_indices_1)
    predicted_class_indices_2 = np.argmax(level_2,axis=1)
    print(predicted_class_indices_2)
    predicted_class_indices_3 = np.argmax(level_3,axis=1)
    print(predicted_class_indices_3)
    length = len(predicted_class_indices_1)
    true_labels_1 = []
    true_labels_2 = []
    true_labels_3 = []
    count = 0
    with open(test_file, 'r') as f:
        for line in f:
            if (count < length):
                count = count + 1
                stripped_line = line.strip()
                label_indices = stripped_line.split(',')
                true_labels_1.append(int(label_indices[1]))
                true_labels_2.append(int(label_indices[2]))
                true_labels_3.append(int(label_indices[3]))
    print("Total test Images: ",count)
    print(true_labels_1)
    print(true_labels_2)
    print(true_labels_3)
    true_labels = [true_labels_1, true_labels_3, true_labels_3]
    x = [i for i in range(6)]
    print(x)
    y = [i for i in range(20)]
    print(y)
    z = [i for i in range(82)]
    print(z)
    cm1 = metrics.confusion_matrix(true_labels_1,predicted_class_indices_1, labels=x)
    cr1 = metrics.classification_report(true_labels_1, predicted_class_indices_1, labels=x)
    cm2 = metrics.confusion_matrix(true_labels_2,predicted_class_indices_2, labels = y)
    cr2 = metrics.classification_report(true_labels_2, predicted_class_indices_2, labels=y)
    cm3 = metrics.confusion_matrix(true_labels_3,predicted_class_indices_3, labels = z)
    cr3 = metrics.classification_report(true_labels_3, predicted_class_indices_3, labels=z)
    #Confusion Matrix for Coarse Level 1 classes
    cmd1 = metrics.ConfusionMatrixDisplay(cm1, display_labels=x)
    cmd1.plot()
    cmd1.ax_.set(title = "Confusion Matrix for Coarse Level 1 classes",xlabel='Predicted Labels', ylabel='True Labels')
    print(cr1)
    #Confusion Matrix for Coarse Level 2 classes
    cmd2 = metrics.ConfusionMatrixDisplay(cm2, display_labels=y)
    cmd2.plot()
    cmd2.ax_.set(title = "Confusion Matrix for Coarse Level 2 classes",xlabel='Predicted Labels', ylabel='True Labels')
    print(cr2)
    print(cm3)
    print(cr3)
    return true_labels
```

Figure 21: Confusion Matrix

Post this, the model prediction is carried out on test data as shown in Figure 22.

```
#Prediction with test images
preds = model.predict_generator(generator=generator_test_batch(test_file,8,num_classes,path_test),
                                steps=test_samples // 8, verbose=1)
```

Figure 22: Model Prediction

10 Calculating Top-N Accuracy

Finally, using predicted labels and true labels, Top-N accuracy is calculated for all three levels of hierarchy as shown in Figure 23.

All the code blocks shown above remain same for all the models, except for the modelbeing called and the data used, that is, augmented and not augmented.

```
from sklearn import metrics
true_labels = PlotConfusionMatrix(preds,test_file)

# Top − n accuracy for all the three levels of hierarchy
preds_level_1 = preds[0]
preds_level_2 = preds[1]
preds_level_3 = preds[2]

true_labels_1 = true_labels[0]
true_labels_2 = true_labels[1]
true_labels_3 = true_labels[2]

# Top−1, Top−3, and Top−5 accuracy for Coarse level 1 with 6 classes
top_1_acc_1 = top_n_accuracy(true_labels_1,preds_level_1,1)
top_3_acc_1 = top_n_accuracy(true_labels_1,preds_level_1,3)
top_5_acc_1 = top_n_accuracy(true_labels_1,preds_level_1,5)
print("Top−1 Accuracy for Coarse level 1 with 6 classes: ", top_1_acc_1)
print("Top−3 Accuracy for Coarse level 1 with 6 classes: ", top_3_acc_1)
print("Top−5 Accuracy for Coarse level 1 with 6 classes: ", top_5_acc_1)

# Top−1, Top−3, and Top−5 accuracy for Coarse level 2 with 20 classes
top_1_acc_2 = top_n_accuracy(true_labels_2,preds_level_2,1)
top_3_acc_2 = top_n_accuracy(true_labels_2,preds_level_2,3)
top_5_acc_2 = top_n_accuracy(true_labels_2,preds_level_2,5)
print("Top−1 Accuracy for Coarse Level 2 with 20 classes: ", top_1_acc_2)
print("Top−3 Accuracy for Coarse Level 2 with 20 classes: ", top_3_acc_2)
print("Top−5 Accuracy for Coarse Level 2 with 20 classes: ", top_5_acc_2)

# Top−1, Top−3, and Top−5 accuracy for Fine level with 82 classes
top_1_acc_3 = top_n_accuracy(true_labels_3,preds_level_3,1)
top_3_acc_3 = top_n_accuracy(true_labels_3,preds_level_3,3)
top_5_acc_3 = top_n_accuracy(true_labels_3,preds_level_3,5)
print("Top−1 Accuracy for Fine level with 82 classes ", top_1_acc_3)
print("Top−3 Accuracy for Fine level with 82 classes ", top_3_acc_3)
print("Top−5 Accuracy for Fine level with 82 classes ", top_5_acc_3)
```

Figure 23: Calculating Top-N accuracy

References

Verma, M., Kumawat, S., Nakashima, Y. and Raman, S. (2020). Yoga-82: A new dataset for fine-grained classification of human poses, *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 4472–4479.