National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Pooja Rakate
Student ID: x19241267

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes, Dr Pramod Pathak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Pooja Rakate		
Student ID:	X19241267		
Programme:	MSc Data Analytics	**Year:**	2020 - 2021
Module:	MSc Research Project		
Lecturer:	Dr Paul Stynes, Dr Pramod Pathak		
Submission Due Date:	16/08/2021		
Project Title:	A Deep Learning Framework to classify Yoga poses Hierarchically		
Word Count:	2038	**Page Count:**	21

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: Pooja Rakate

Date: 16/08/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Configuration Manual

Pooja Rakate
Student ID: x19241267

1 Introduction

This configuration manual provides information regarding the data source, software, system specification, hardware specifications, tools, libraries, and code used for executing the models implemented for the research project "**A Deep Learning Framework to classify Yoga poses Hierarchically**". It also describes the steps undertaken to process the data to make it ready for implementing the proposed models.

2 System Specifications

This section provides details about the system configurations used for implementation in this project.

2.1 System Hardware for processing data

Operating System:	Windows 10
Processor:	AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10GHz
Installed RAM:	8.00 GB (5.88 GB is usable)
System Type:	64-bit operating system x64based processor
Hard Disk:	1 TB

2.2 System Hardware for training the deep learning models

Training a deep learning model on the system mentioned in section 2.1 took very long hours to run. Approximately 2 hours per epoch and a total of 30 epochs were to be run. Hence, an instance of a Microsoft Azure virtual CPU machine (VM) was created[1]. Data was stored on this machine so that the model could access it. Each time the model had to be run, the VM was started and once the training ended it was stopped. Initially, credits provided by the National College of Ireland were used to create a machine, but as it was a student subscription a machine with large RAM could not be created and hence, the subscription was updated from Azure-student to "pay-as-you-go" to be able to use a machine with the below-specified configuration.

Operating System:	Windows (Windows Server 2016)
RAM:	112 Gib (approximately 120GB)
vCPUs	8

[1] https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal

3 Softwares Used

All the software and programming languages used for implementing this project are listed in this section. JAVA 8 was used to download the images from the dataset. Python programming language was used for processing the downloaded data and running the deep learning models. The python code was written and executed in Jupyter notebook from the Anaconda Navigator.

- JAVA 8
- JET BRAINS IntelliJ IDEA (Version 2021.2)
- Python 3.7
- Jupyter Notebook

IntelliJ IDEA can be downloaded from here[2] for the windows system. To use Jupyter notebook. Install Anaconda navigator version 1.9.12 was used.

4 Project Implementation

4.1 Data collection

The Yoga-82 dataset was downloaded from their home page[3]. The downloaded data had three text files and one readme file. There was one text file containing the image name and its corresponding web URL to download it from. The other two text files were the train label file and the test label file. It consisted of the image name and their corresponding labels on the three levels of hierarchy. There are a total of three levels in the hierarchy. The first level classifies a yoga pose into 6 classes, the second level classifies it into 20 classes and the third fine level classifies them into 82 classes(82 Yoga poses). One can either download the original data or directly used the processed data that is ready for modelling directly from here[4]. The processed data has 4 datasets. One is normal image data, second is its balanced version, third is the skeleton image dataset created by using OpenPose model and the last one is the balanced version of the third dataset. Each folder also includes the train, validation and test files.

4.2 Data Extraction & Cleaning

Downloading images using Python 3.7 took time, hence JAVA was used to download the images. The GIF format images were excluded while downloading. Entry of images not found was made into a text file and entry for these images was removed from the train and test label files. All the images were stored using the JAVA program in to their respective class folders for each hierarchical level. As this structure would have been difficult to retrieve the images. All the images were later stored once in a single foler and second time in two separate folders as test and train images. The JAVA code written for this step can be found

[2] https://www.jetbrains.com/idea/download/#section=windows
[3] https://sites.google.com/view/yoga-82/home
[4] https://studentncirl-my.sharepoint.com/:f:/g/personal/x19241267_student_ncirl_ie/Et30BQwnBeZEujd1iMLFniUBuC16k8Jb3ztuEUSo3bLHsA?e=ABb8Ls

here[5]. Download the “Data_extraction_Java_code.zip” file unzip it and run application 3 in your IDE. This same code is available in the “Java_Code” folder in the cade artefacts.

Hereafter, the Python programming language was used. Install OpenCV (library for computer vision tasks) and import all the other libraries. At this stage we start running the “YOGA_main.ipynb” file cell by cell.

```
!pip3 install opencv-python
```

```
import requests
from pathlib import Path
import os
import glob
import cv2
from PIL import Image
import numpy as np
import random
import pandas as pd
import shutil
import cv2
import time
import math
import re
import pickle
import matplotlib.pyplot as plt
```

Figure 1: Import Required Libraries

Images that were downloaded by the JAVA code but were still unreadable were downloaded again using Python and saved locally.

```
def downloadImage(imageUrl,filename,extension,Train_or_Test,FolderPath):

    url = imageUrl

    try:
        response = requests.get(url)
        if response.status_code == 200:  ##If image is found
            if Train_or_Test == "TRAIN":

                Path("D:/Pooja_Thesis/2.0/image_downloaded_not_redable_train/").mkdir(parents=True, exist_ok=True)
                f = open("D:/Pooja_Thesis/2.0/image_downloaded_not_redable_train/"+filename+extension, "x")
                with open("D:/Pooja_Thesis/2.0/image_downloaded_not_redable_train/"+filename+extension, "wb") as f:
                    f.write(response.content)
            else:
                Path("D:/Pooja_Thesis/2.0/image_downloaded_not_redable_test/").mkdir(parents=True, exist_ok=True)
                f = open("D:/Pooja_Thesis/2.0/image_downloaded_not_redable_test/"+filename+extension, "x")
                with open("D:/Pooja_Thesis/2.0/image_downloaded_not_redable_test/"+filename+extension, "wb") as f:
                    f.write(response.content)
        ##If image is not found make an entry in respective text file
        else:
            if Train_or_Test == "TRAIN":
                with open("D:/Pooja_Thesis/2.0/not_found_train.txt","a+") as f:
                    f.write(filename + '\t')
                    f.write(FolderPath + '\t')
                    f.write(url)
                    f.write('\n')
            else:
                with open("D:/Pooja_Thesis/2.0/not_found_test.txt","a+") as f:
                    f.write(filename + '\t')
                    f.write(FolderPath + '\t')
                    f.write(url)
                    f.write('\n')

            print(filename,response.status_code)


    except Exception as e:
        print("An exception occurred",e)
```

Figure 2: Download unreadable images again – Part 1

[5] https://github.com/PoojaRakate26/DataAnalytics_Thesis/blob/main/Data_extraction_Java_code.zip

```
substring1 = 'TRAIN'
substring2 = 'TEST'

with open("D:/Pooja_Thesis/2.0/image_downloaded_not_redable.txt", "r") as a_file:
        for line in a_file:
            stripped_line = line.strip()

            FileNameString = stripped_line[0:stripped_line.find(".jpg")]
            #print(FileNameString)

            stripped_line_2 = stripped_line[stripped_line.find("  ")+1:]
            imageUrl = stripped_line_2.strip()
            Url = imageUrl[0:imageUrl.find(" ")]
            #print(Url)

            ext = Url[-4:]
            #print(ext)

            stripped_line_3 = stripped_line_2.strip()
            FolderPath = stripped_line_3[stripped_line_3.find(" "):]
            FolderPath = FolderPath.strip()
            #print(FolderPath)

            if substring1 in line:
                Train_or_Test = substring1
            else:
                Train_or_Test = substring2
            #print(Train_or_Test)

            downloadImage(Url,FileNameString,ext,Train_or_Test,FolderPath)
```

Figure 2: Download unreadable images again – Part 2

Images that were still unreadable were deleted and they were noted.

```
try:
    imdir = 'D:/Pooja_Thesis/2.0/image_downloaded_not_redable_test/check_again/'
    len_imdir = len(imdir)
    #ext = ['png', 'jpg']    # Add image formats here

    files = []
    #[files.extend(glob.glob(imdir + '*.' + e)) for e in ext]
    [files.extend(glob.glob(imdir + '*.*'))]

    for file in files:
        im = cv2.imread(file)  ##returns none if image is not readable

        if im is None:
            stripped_line = file.strip()
            filename = stripped_line[len_imdir:]

            ###make a note of images that are stille unreadable and delete them from folder
            with open("D:/Pooja_Thesis/2.0/delete_entreis_from_main_test_data.txt","a+") as f:
                    f.write(filename)
                    f.write('\n')
            os.remove(file)
            print("Removed:"+' '+filename+ ' ' + file)

except Exception as e:
    print("An exception occurred",e)
```

Figure 3: Delete Unreadable images

Make sure the images saved in the local drive are in sync with the entries in the train and test label files. This was done by matching all downloaded images and entries in test and train text files. Keep only those entries in text files of which corresponding images are present.

```
try:
    imdir = 'D:/Pooja_Thesis/2.0/TRAIN/'
    len_imdir = len(imdir)
    #ext = ['png', 'jpg']    # Add image formats here

    count = 0
    imageName = []
    retain = []

    for file in glob.glob(imdir + '*.*'):
        stripped_line = file.strip()
        imName = stripped_line[len_imdir:]
        imageName.append(imName)

    print(len(imageName))
    with open('D:/Pooja_Thesis/Train.txt', 'r') as f:
        for line in f:
            for name in imageName:
                if name.lower() in line.lower():
                    with open('D:/Pooja_Thesis/Train_final.txt', 'a+') as newfile:  ## make a newfile with updated entries
                        newfile.write(line)
                        retain.append(name)
                    count = count + 1

    print("number of matches found: "+ str(count))
    print("retain: "+ str(len(retain)))


except Exception as e:
    print("An exception occurred",e)
```

Figure 4: Synching the images saved and the data in the train and test label files

Due to system constraints, the entire dataset was not used for model training. Hence, the data was reduced by 50% by first reducing the entries in the two label files and then using these updated text files to keep only 50% of the images.

```
###First Reducing the labels file of train and test images
```

```
def reducedata(imdir, className):
    new_lines = []
    with open(imdir+'Train_final.txt','r') as file:
            for line in file:
                if className in line:
                    line = line[:line.find("/n")]
                    new_lines.append(line)

            num = len(new_lines)
            half = int(round(num*0.5))
            print(half)
            index = [n for n in range(num)]
            random.shuffle(index)

            count = 0
            with open(imdir+'Train_final_reduced.txt','a+') as ff:
                for m in range(half):
                    count = count + 1
                    ff.write(new_lines[index[m]])
                    ff.write('\n')

            #print("Entries added to " + className + str(count))
            #print("Entries for " + className + ": " + str(num))

try:
    list_classes = []
    imdir = 'D:/Pooja_Thesis/2.0/Reducing_Data/'

    with open('D:/Pooja_NCI/3_Semester/Class_Names.txt','r') as file:
        for f in file:
            f = f[:f.find("/n")]
            list_classes.append(f)

    for i in list_classes:
        #print("class " + i.split('/n')[-1] +':')
        reducedata(imdir, i)

except Exception as e:
    print("An exception occurred",e)
```

Figure 5: Reducing the data in the label files by 50%

```
### Reducing the image data using the reduced train and test label files
```

```
try:
    imdir = 'D:/Pooja_Thesis/2.0/TRAIN/'

    with open(imdir+'Train_final_reduced.txt','r') as ff:
        lines = ff.readlines()

    path = imdir + 'Image_data_reduced/'
    len_path = len(path)

    new_path = imdir + 'Image_data_reduced/Train/'

    for file in glob.glob(path+'*.*'):
        stripped_line = file.strip()
        path = stripped_line[:len_path]
        imFullName = stripped_line[len_path:]

        for line in lines:
            if imFullName in line:
                print(line)
                shutil.move(path+imFullName, new_path+imFullName)

except Exception as e:
    print("An exception occurred",e)
```

Figure 6: Reducing the images by 50%

4.3 Data Transformation

In this step, firstly Skeleton images were extracted from the normal images using the OpenPose model. Secondly, Data augmentation was carried out to balance the dataset.

4.3.1 OpenPose

The normal images were converted into skeleton images with a black background using the OpenPose model. The "pose_deploy_linevec.prototxt" and the "pose_iter_440000.caffemodel" files to run the OpenPose model can be downloaded from here[6]. Since there were images in the dataset in which the yoga poses were in silhouette form of sketches form, key points for them could not be identified. Such images were bypassed. Required libraries were imported and then the OpenPose model was run to create the skeleton images. This code is present in the "OpenPose_create_skeleton.ipynb" file.

```
import cv2
import time
import numpy as np
import os
import matplotlib.pyplot as plt
import math
import re
import glob
import pickle
import pandas as pd
```

Figure 7: Importing the required libraries

[6] https://github.com/foss-for-synopsys-dwc-arc-processors/synopsys-caffe-models/tree/master/caffe_models/openpose/caffe_model

```
def getSkeletonImages(imdir,file):
    nPoints = 18
    POSE_PAIRS = [ [1,0],[1,2],[1,5],[2,3],[3,4],[5,6],[6,7],[1,8],[8,9],[9,10],[1,11],[11,12],[12,13],[0,14],[0,15],[14,16],[15,
    t = time.time()
    TEXT_COLOR = (0,0,0)
    inWidth = 256
    inHeight = 256
    threshold = 0.1
    pot = []
    #pot_train = []
    #pot_test = []
    #pot_val = []
    #imageNames = []

    splits = file.split(".")
    splits = splits[0]
    dataset = splits[1:splits.find("_")]

    with open(imdir+file,"r") as f1:
        for line in f1:
            stripped_line = line.strip()
            splitted = stripped_line.split(",")
            print("Image being processed: ",splitted[0])

            path = imdir +'Images/'+splitted[0]

            frame = cv2.imread(path)
            #print(frame)
            if (frame.shape[1] > 600 and frame.shape[0] > 600):
                scale_percent = 50 # percent of original size
                width = int(frame.shape[1] * scale_percent / 100)
                height = int(frame.shape[0] * scale_percent / 100)
                dim = (width, height)
                frame = cv2.resize(frame, dim)
                print(frame.shape)

            frameCopy = np.copy(frame)

            count_none = 0

            frameWidth = frame.shape[1]
            print('frameWidth: '+ str(frameWidth))
            frameHeight = frame.shape[0]
            print('frameHeight: '+ str(frameHeight))

            inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),(0, 0, 0), swapRB=False, crop=False)
            net.setInput(inpBlob)
            output = net.forward()

            H = output.shape[2]
            #print(H)
            W = output.shape[3]
            #print(W)
            # Empty list to store the detected keypoints
            points = []
```

Figure 8: Creating Skeleton Images using OpenPose model – part 1

```
for i in range(nPoints):
    # confidence map of corresponding body's part.
    probMap = output[0, i, :, :]

    # Find global maxima of the probMap.
    minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

    # Scale the point to fit on the original image
    x = (frameWidth * point[0]) / W
    print("x:"+ str(x))
    y = (frameHeight * point[1]) / H
    print("y:"+ str(y))

    if prob > threshold :
        cv2.circle(frameCopy, (int(x), int(y)), 8, (0, 255, 255), thickness=-1, lineType=cv2.FILLED)
        cv2.putText(frameCopy, "{}".format(i), (int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, lineTyp

        # Add the point to the list if the probability is greater than the threshold
        points.append((int(x), int(y)))
    else :
        count_none+=1
        points.append((None,None))

print(points)
print(count_none)

if (count_none == 18 or count_none >15):
    print("Key-points cannot be detected by OpenPose for: "+ splitted[0])

    ##Images for which key points could not be generated
    with open(imdir+dataset+"_OpenPose_failed.txt","a+") as f2:
        f2.write(line)

else:
    points.append(splitted[0])  ##Append image name to the points array in case needed
    points.append(splitted[1])   ##Append image label 1 to the points array in case needed
    points.append(splitted[2])    ##Append image label 2 the points array in case needed
    points.append(splitted[3])    ##Append image label 3 the points array in case needed

    print(points)
```

Figure 8: Creating Skeleton Images using OpenPose model – part 2

```
            print(points)

            frame = np.zeros((frameHeight,frameWidth,3),np.uint8)  # For making the background black
            TEXT_COLOR = (255,255,255)

            for pair in POSE_PAIRS:
                partA = pair[0]
                partB = pair[1]

                if points[partA][0] and points[partB][0]:
                    cv2.line(frame, points[partA], points[partB], (0, 255, 255), 1, lineType=cv2.LINE_AA)
                    cv2.ellipse(frame, points[partA], (4, 4), 0, 0, 360, (255, 255, 255), cv2.FILLED)
                    cv2.ellipse(frame, points[partB], (4, 4), 0, 0, 360, (255, 255, 255), cv2.FILLED)
                    cv2.putText(frame, str(partA), points[partA], cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),2,cv2.LINE_AA)
                    cv2.putText(frame, str(partB), points[partB], cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255),2,cv2.LINE_AA)

            pot.append(points)   ##Add key points for all images to a list, in case needed
            #print("pot array: ", pot)

            destpath = imdir+"/Skeleton_Images/"+splitted[0]
            cv2.imwrite(destpath,frame)     ##Save the skeleton images with black-background to local drive
            cv2.waitKey(2)
            cv2.waitKey(0)
            cv2.destroyAllWindows()

    return pot
```

Figure 9: Creating Skeleton Images using OpenPose model – part 3

```
protoFile = "D:/Pooja_Thesis/OpenPose/pose_deploy_linevec.prototxt"
weightsFile = "D:/Pooja_Thesis/OpenPose/pose_iter_440000.caffemodel"
imdir = "D:/Pooja_Thesis/OpenPose/"
len_imdir = len(imdir)

net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)

##adding new
list = ["/Test_final.txt","/Train_final.txt","/Validation.txt"]
#list = ["/trial_final.txt"]

for items in list:
    pot = getSkeletonImages(imdir,items)

    splits = items.split(".")
    splits = splits[0]
    data_file = splits[1:splits.find("_")]

    if data_file == "Train":
        with open(imdir+"Train_pickle.txt", 'wb') as fp:
            pickle.dump(pot, fp) #Pickling so that it can be loaded back in case needed
            print("Train pickle file created.")

    elif data_file == "Test":
        with open("Test_pickle.txt", 'wb') as fp:
            pickle.dump(pot, fp)
            print("Test pickle file created.")

    else:
        with open("Validaition_pickle.txt", 'wb') as fp:
            pickle.dump(pot, fp)
            print("Validation pickle file created.")
```

Figure 10: Creating Skeleton Images using OpenPose model – part 4

After applying the OpenPose model, few images had to be deleted as no key points for them were returned by the openPose model. Thus, to maintain uniformity, these images were removed from the folder of normal images too. Reference to build the code for extracting key points and plotting it on the image was taken from here[7].

[7] https://github.com/shahil1993/Thesis/blob/main/Final_code.py

4.3.2 Data Augmentation

The Yoga-82 dataset was highly imbalanced. So we tried to balance the data on the third level which has 82 classes. On checking the number of images per class it was found that there the number of images in the classes is varying with the maximum and minimum value being 230 and 13 respectively. Hence, we balanced images in these classes by keeping the utmost 65 images per class. Classes having more than 65 images were down-sampled while those having fewer images were upsampled by adding vertically flipped and rotated versions of the original image. The below code was used to achieve all this simultaneously. The same code was run for the normal image dataset and the skeleton image dataset. The code for this is present in the "Data_Balancing_using_Augmentation_techniques.ipynb" file.

```
import requests
from pathlib import Path
import os
import glob
import cv2
from PIL import Image
import numpy as np
import random
import pandas as pd
import shutil
```

Figure 11: Import required libraries

```
def augmentData(imdir, className):
    new_lines = []
    default = 65 ##number of images should be present in each class on the third level of hierarchy which has a total of 82 class
    angle = 45  ##value for rotating images
    src = "D:/Pooja_Thesis/OpenPose/Skeleton_Images/"
    dest=imdir+"OpenPose_Augmented_Images/"

    with open(imdir+'Train_OpenPose.txt','r') as file:
            for line in file:
                if className in line:
                    line = line[:line.find("/n")]
                    new_lines.append(line)

            num = len(new_lines)
            lst = [n for n in range(num)]
            index = random.sample(lst, num)
            #print(new_lines)

            count = 0
            if (num > default):   ## Classes that need down sampling to be done since they have more than 65 images
                print("Category 1: ",className)
                with open(imdir+'Train_OpenPose_Augnmented.txt','a+') as ff:
                    for m in range(default):
                        count = count+1
                        ff.write(new_lines[index[m]])
                        ff.write("\n")

                        entry = new_lines[index[m]]
                        fileName = entry.split(",")
                        #print(fileName[0])
                        shutil.copy(src + fileName[0], dest)

                print("Augmented Images for " + className + ": " + str(count))
```

Figure 12: Data Augmentation – Part 1

```
elif (num < default and num > 20):  ## up sampling needed
    print("Category 2: ",className)
    diff = default - num
    num_images_to_be_augmented = int(diff/2)
    print("Category 2 - Images to be augmented: ",num_images_to_be_augmented)

    with open(imdir+'Train_OpenPose_Augnmented.txt','a+') as ff:
        for m in range(num):
            count = count+1
            ff.write(new_lines[m])
            ff.write("\n")
            entry = new_lines[m]
            fileName = entry.split(",")
            #print(fileName[0])
            shutil.copy(src + fileName[0], dest)

        for m in range(num_images_to_be_augmented):
            entry = new_lines[index[m]]
            fileName = entry.split(",")
            imgName = fileName[0]
            ext = imgName[imgName.find('.'):]
            img_wo_ext=imgName[:imgName.find('.')]
            image = cv2.imread(src+imgName)

            # 1st augmentation technique: Flipping the image upon the y-axis
            image_flipped_vertically= cv2.flip(image, 1)
            destpath_FV = dest+img_wo_ext+"_FV"+ext
            cv2.imwrite(destpath_FV,image_flipped_vertically)
            count = count+1

            # 2nd augmentation technique: Rotating the image by 45 degrees
            angle = int(random.uniform(-angle, angle))
            h, w = image.shape[:2]
            M = cv2.getRotationMatrix2D((int(w/2), int(h/2)), angle, 1)
            rotated_image = cv2.warpAffine(image, M, (w, h))
            destpath_R = dest+img_wo_ext+"_R"+ext
            cv2.imwrite(destpath_R,rotated_image)
            count = count+1
            ## appending these newly created images entry into the label file
            FV_line = img_wo_ext+"_FV"+ext+","+fileName[1]+","+fileName[2]+","+fileName[3]
            ff.write(FV_line)
            ff.write("\n")
            R_line = img_wo_ext+"_R"+ext+","+fileName[1]+","+fileName[2]+","+fileName[3]
            ff.write(R_line)
            ff.write("\n")
```

Figure 13: Data Augmentation – Part 2

```
elif (num == default):   ## classes which already have only 65 images
    print("Category 3: ",className)

    with open(imdir+'Train_OpenPose_Augnmented.txt','a+') as ff:
        for m in range(num):
            count = count+1
            ff.write(new_lines[m])
            ff.write("\n")

            entry = new_lines[m]
            fileName = entry.split(",")
            imgName = fileName[0]

            shutil.copy(src + imgName, dest)
```

Figure 14: Data Augmentation – Part 3

```
        else:
            print("Category 4: ",className)
            with open(imdir+'Train_OpenPose_Augnmented.txt','a+') as ff:
                for m in range(num):
                    count = count+3
                    entry = new_lines[index[m]]
                    fileName = entry.split(",")
                    imgName = fileName[0]
                    shutil.copy(src + imgName, dest)  #Copying the image to new(dest) folder
                    ext = imgName[imgName.find('.'):]
                    img_wo_ext=imgName[:imgName.find('.')]

                    image = cv2.imread(src+imgName)

                    # 1st augmentation technique: Flipping the image upon the y-axis
                    image_flipped_vertically= cv2.flip(image, 1)
                    destpath_FV = dest+img_wo_ext+"_FV"+ext
                    cv2.imwrite(destpath_FV,image_flipped_vertically)

                    # 2nd augmentation technique: Rotating the image by 45 degrees
                    angle = int(random.uniform(-angle, angle))
                    h, w = image.shape[:2]
                    M = cv2.getRotationMatrix2D((int(w/2), int(h/2)), angle, 1)
                    rotated_image = cv2.warpAffine(image, M, (w, h))
                    destpath_R = dest+img_wo_ext+"_R"+ext
                    cv2.imwrite(destpath_R,rotated_image)

                    ## appending these newly created images entry into the label file
                    ff.write(entry)
                    ff.write("\n")
                    FV_line = img_wo_ext+"_FV"+ext+","+fileName[1]+","+fileName[2]+","+fileName[3]
                    ff.write(FV_line)
                    ff.write("\n")
                    R_line = img_wo_ext+"_R"+ext+","+fileName[1]+","+fileName[2]+","+fileName[3]
                    ff.write(R_line)
                    ff.write("\n")

    return num, count
```

Figure 15: Data Augmentation – Part 4

```
try:
    list_classes = []
    imdir = 'D:/Pooja_Thesis/Augmentation/'

    before_lst = []
    after_lst = []

    with open('D:/Pooja_NCI/3_Semester/Class_Names.txt','r') as file:
        for f in file:
            f = f[:f.find("/n")]
            list_classes.append(f)

    for i in list_classes:
        before, after = augmentData(imdir, i)
        print("class " + i.split('/n')[-1] +' total images before and after augmentation:', before, after)

        before_lst.append(before)
        after_lst.append(after)
        #print(val)

    print("Max number of images in a class Before Augmentation",max(before_lst))
    print("Max number of images in a class After Augmentation",max(after_lst))
    print("Before Augmentation",sorted(before_lst, reverse = True))
    print("After Augmentation",sorted(after_lst, reverse = True))

    print("Total number of images before augmentation",sum(before_lst))
    print("Total number of images after augmentation",sum(after_lst))

except Exception as e:
    print("An exception occurred",e)
```

Figure 16: Data Augmentation – Part 5

Below is the output after running the above code.

```
Max number of images in a class Before Augmentation 230
Max number of images in a class After Augmentation 65
Before Augmentation [230, 220, 163, 145, 125, 110, 97, 97, 97, 96, 94, 93, 92, 91, 90, 90, 90, 88, 86, 86, 85, 85, 84, 84, 8
0, 79, 77, 74, 73, 71, 69, 68, 66, 65, 65, 64, 63, 63, 63, 62, 61, 60, 58, 58, 58, 57, 57, 54, 54, 54, 52, 52, 50, 49, 49, 4
6, 45, 44, 43, 42, 41, 40, 39, 39, 38, 36, 30, 29, 29, 28, 27, 27, 27, 24, 20, 19, 18, 18, 16, 15, 14, 13]
After Augmentation [65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 6
5, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 64, 64, 64, 64, 6
4, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 60, 57, 54, 54, 48, 45, 42, 39]
Total number of images before augmentation 5350
Total number of images after augmentation 5188
```

Figure 17: Output after running Data Augmentation code

4.4 Data Modelling & Evaluation

In this step, the SOTA model (DenseNet201), the modified version of the SOTA network, and the modified VGG19 networks were implemented. Before that, since the data had only the train and test split files, the test label file was divided into validation and test files. The data was generated in batches and the data generator function was the same for the train, validation, and test data, Only difference being the train data was shuffled and randomly generated. Below are code snippets for importing necessary libraries, packages, and the data generator function. Code for building the SOTA model was referred from Verma et al. (2020).

```
import os
#os.environ["CUDA_DEVICE_ORDER"]= "PCI_BUS_ID"
#os.environ["CUDA_VISIBLE_DEVICES"]= "0"

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import warnings    ## Added
warnings.filterwarnings("ignore")

import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models
from tensorflow.keras import Input
from tensorflow.keras import backend

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense, Dropout, Activation, Reshape
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, UpSampling2D
from tensorflow.keras.layers import AveragePooling2D, MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.optimizers import SGD,Adam, RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from tensorflow.keras.utils import get_file, to_categorical, get_source_inputs
from keras_applications.imagenet_utils import _obtain_input_shape
from tensorflow.keras.applications.imagenet_utils import decode_predictions
from tensorflow.keras.layers import Dense,Dropout,Conv2D,MaxPool2D,Flatten,Activation, GlobalAveragePooling2D
from tensorflow.keras.layers import BatchNormalization, MaxPooling2D, Concatenate
keras.backend.set_image_data_format('channels_last')
from pathlib import Path

import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
from PIL import Image
from PIL import ImageFile
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

Figure 18: Importing necessary libraries and packages for model building and training

```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]
    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
            x_train, x1_labels, x2_labels, x3_labels = process_batch(new_line[a:b],img_path,train=True)
            x = preprocess(x_train)

            y1 = to_categorical(np.array(x1_labels), class_6)
            y2 = to_categorical(np.array(x2_labels), class_20)
            y3 = to_categorical(np.array(x3_labels), class_82)

            y = [y1,y2,y3]
            yield x, y
```

Figure 19: Data Generator function

```python
def process_batch(lines,img_path,train=True):
    num = len(lines)
    batch = np.zeros((num,224,224,3),dtype='float32')
    x1_labels = np.zeros(num,dtype='int')
    x2_labels = np.zeros(num,dtype='int')
    x3_labels = np.zeros(num,dtype='int')
    for i in range(num):
        path = lines[i].split(',')[0]
        x1_label = lines[i].split(',')[1]
        x2_label = lines[i].split(',')[2]
        x3_label = lines[i].split(',')[-1]
        x3_label = x3_label.strip('\n')

        x1_label = int(x1_label)
        x2_label = int(x2_label)
        x3_label = int(x3_label)
        imgs = img_path+path

        if train:
            image = Image.open(imgs).convert("RGB")

            image = image.resize((224,224), Image.ANTIALIAS)
            batch[i][:][:][:] = image
            x1_labels[i] = x1_label
            x2_labels[i] = x2_label
            x3_labels[i] = x3_label
        else:
            image = Image.open(imgs).convert("RGB")

            image = image.resize((224,224), Image.ANTIALIAS)
            batch[i][:][:][:] = image
            x1_labels[i] = x1_label
            x2_labels[i] = x2_label
            x3_labels[i] = x3_label

    return batch, x1_labels, x2_labels, x3_labels
```

Figure 20: Function for processing the batches of images and returning the array format of images and their corresponding one-hot encoded labels

```python
def preprocess(inputs):
    inputs /=255.

    return inputs
```

Figure 21: Function for normalizing the values of the input array of images

The original architecture of DenseNet201 with just one output branch was modified to build the SOTA model with three output branches. Which was further modified in this project as proposed to gain better classification accuracy. Similarly, the VGG19 architecture was modified for hierarchical classification purposes in this project. The original code .py files of both the DenseNet201[8] and VGG19[9] were used to modify their architecture. Models are built in the functions and then the functions are called before compiling the model. Below are snippets showing the modified/ added parts to the model.

4.4.1 DenseNet201 – Hierarchical – Using connectivity pattern

```
bn_axis = 3 if keras.backend.image_data_format() == 'channels_last' else 1
x = layers.ZeroPadding2D(padding=((3, 3), (3, 3)))(img_input)
x = layers.Conv2D(64, 7, strides=2, use_bias=False, name='conv1/conv')(x)
x = layers.BatchNormalization(
    axis=bn_axis, epsilon=1.001e-5, name='conv1/bn')(x)
x = layers.Activation('relu', name='conv1/relu')(x)
x = layers.ZeroPadding2D(padding=((1, 1), (1, 1)))(x)
x = layers.MaxPooling2D(3, strides=2, name='pool1')(x)
x = dense_block(x, blocks[0], name='conv2')
x = transition_block(x, 0.5, name='pool2')
x = dense_block(x, blocks[1], name='conv3')

x = transition_block(x, 0.5, name='pool3')

x1=x        ##Output branch for coarse level 1 in hierarchy

x = dense_block(x, blocks[2], name='conv4')

x = transition_block(x, 0.5, name='pool4')

x2=x       ##Output branch for coarse level 2 in hierarchy

x = dense_block(x, blocks[3], name='conv5')
x = layers.BatchNormalization(
    axis=bn_axis, epsilon=1.001e-5, name='bn')(x)
x = layers.Activation('relu', name='relu')(x)    ##Output branch for fine level in hierarchy

if include_top:
    x = layers.GlobalAveragePooling2D(name='avg_pool')(x)
    x = layers.Dense(classes, activation='softmax', name='fc1000')(x)
else:
    if pooling == 'avg':
        x = layers.GlobalAveragePooling2D(name='avg_pool')(x)
    elif pooling == 'max':
        x = layers.GlobalMaxPooling2D(name='max_pool')(x)

# Ensure that the model takes into account
# any potential predecessors of `input_tensor`.
if input_tensor is not None:
    inputs = get_source_inputs(input_tensor)
else:
    inputs = img_input

# Create model.

model = models.Model(inputs, [x1,x2,x], name='densenet201_hir')
```

Figure 22: Modified the original Keras densent.py file here in the function “DenseNet”

```
def DenseNet201_hir(include_top=True,
                    weights='imagenet',
                    input_tensor=None,
                    input_shape=None,
                    pooling=None,
                    classes=1000,
                    **kwargs):
    return DenseNet([6, 12, 48, 32],
                        include_top, weights,
                        input_tensor, input_shape,
                        pooling, classes,
                        **kwargs)
```

Figure 23: Function for modified DenseNet201 architecture (SOTA model)

[8] https://github.com/keras-team/keras-applications/blob/master/keras_applications/densenet.py
[9] https://github.com/keras-team/keras-applications/blob/master/keras_applications/vgg19.py

```
n [17]: def dense201_hirar_concat(
            input_shape = (224,224,3),
            class_6=6,
            class_20=20,
            class_82=82):


        inputs = Input(input_shape)
        base_model= DenseNet201_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 = dense_block_hir(x1, 32, name='denseblockClass6')
        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)
        coarse_1_pred = Dense(class_6, activation= 'softmax', name = 'coarse_1_pred')(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)
        x2 = keras.layers.concatenate([x2,x1])
        coarse_2_pred = Dense(class_20, activation= 'softmax', name = 'coarse_2_pred')(x2)

        x = GlobalAveragePooling2D()(x)
        x = keras.layers.concatenate([x,x2])
        fine_pred = Dense(class_82, activation='softmax', name = 'fine_pred')(x)

        model = Model(inputs, [coarse_1_pred,coarse_2_pred,fine_pred])
        for layer in base_model.layers:
            layer.trainable = True

        return model
```

Figure 24: Building the SOTA model with fully connected layers and concatenation layers in a function

4.4.2 VGG19 – Hierarchical – Using connectivity pattern

```
    # Block 1
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(img_input)
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)
    # Block 2
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
    # Block 3
    x = layers.Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
    x = layers.Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
    x = layers.Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
    x = layers.Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv4')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
    ##Coarse level 1 Output - Output Branch 1
    x1 = x
    # Block 4
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv4')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)
    ##Coarse level 2 Output - Output Branch 2
    x2 = x
    # Block 5
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
    x = layers.Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv4')(x)
    x = layers.MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
    if include_top:
        # Classification block
        x = layers.Flatten(name='flatten')(x)
        x = layers.Dense(4096, activation='relu', name='fc1')(x)
        x = layers.Dense(4096, activation='relu', name='fc2')(x)
        x = layers.Dense(classes, activation='softmax', name='predictions')(x)
    else:
        if pooling == 'avg':
            x = layers.GlobalAveragePooling2D()(x)
        elif pooling == 'max':
            x = layers.GlobalMaxPooling2D()(x)

    # Ensure that the model takes into account
    # any potential predecessors of `input_tensor`.
    if input_tensor is not None:
        inputs = get_source_inputs(input_tensor)
    else:
        inputs = img_input

    # Create model.
    model = models.Model(inputs, [x1, x2, x], name='hir_vgg19')
```

Figure 25: Modified original VGG19 architecture to have a model with 3 output branches in the VGG19 function

```
def Hir_VGG19(include_top=True,
              weights='imagenet',
              input_tensor=None,
              input_shape=None,
              pooling=None,
              classes=1000,
              **kwargs):
    return VGG19(include_top, weights,
                 input_tensor, input_shape,
                 pooling, classes, **kwargs)
```

Figure 26: Function for calling the modified VGG19 architecture

```
def Yoga_Hir_Concat_VGG19(
          input_shape = (224,224,3),
        class_6=6,
        class_20=20,
        class_82=82):

    inputs = Input(input_shape)
    base_model= Hir_VGG19(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 = layers.Flatten(name='coarse_1_flatten')(x1)
    x1 = layers.Dense(256, activation='relu', name='coarse_1_fc1')(x1)
    x1 = BatchNormalization()(x1)
    x1 = Dropout(0.5)(x1)
    coarse_1_pred = layers.Dense(class_6, activation='softmax', name='coarse_1_pred')(x1)

    x2 = layers.Flatten(name='coarse_2_flatten')(x2)
    x2 = layers.Dense(512, activation='relu', name='coarse_2_fc1')(x2)
    x2 = BatchNormalization()(x2)
    x2 = Dropout(0.5)(x2)
    x2 = keras.layers.concatenate([x2,x1])
    coarse_2_pred = layers.Dense(class_20, activation='softmax', name='coarse_2_pred')(x2)

    x = layers.Flatten(name='fine_flatten')(x)
    x = layers.Dense(512, activation='relu', name='fine_fc1')(x)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)
    x = keras.layers.concatenate([x2,x])
    fine_pred = layers.Dense(class_82, activation='softmax', name='fine_pred')(x)

    model = Model(inputs, [coarse_1_pred,coarse_2_pred,fine_pred])

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


    return model
```

Figure 27: Building the modified VGG19 model with fully connected layers and concatenation layers in a function

4.4.3 Model Compiling, Training, and Predicting

On completing the model training, the training and validation accuracy and training and validation loss plots are plotted for each output level and a total loss plot for the entire model is plotted. This is functionality is coded in the "PlotConfusionMatrix" function. For evaluation purposes, the top-1, 3, and 5 accuracy rates for predicted samples for all the three output levels are calculated using the functions "top_n_accuracy" and "arg_sort_desc". Below are the code snippets of these functions.

```python
##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("")
    print("Total test Images: ",count)
    print(true_labels_1)
    print(true_labels_2)
    print(true_labels_3)

    true_labels = [true_labels_1, true_labels_2, true_labels_3]

    x = [i for i in range(6)]
    y = [i for i in range(20)]
    z = [i for i in range(82)]

    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)

    cmd1 = metrics.ConfusionMatrixDisplay(cm1, display_labels=x)
    cmd1.plot()
    cmd1.ax_.set(title = "Confusion Matrix for Level 1 classes(coarse level)",xlabel='Predicted Labels', ylabel='True Labels')
    print(cr1)
    cmd2 = metrics.ConfusionMatrixDisplay(cm2, display_labels=y)
    cmd2.plot()
    cmd2.ax_.set(title = "Confusion Matrix for Level 2 classes(intermediate classes)",xlabel='Predicted Labels', ylabel='True Lab
    print(cr2)
    print(cm3)
    print(cr3)
    return true_labels
```

Figure 28: The “PlotConfusionMatrix” function

```python
## Functions to calculate Top-n accuracy of the model

def arg_sort_desc(ar):
    P=ar.argsort(axis=1)
    Q=[]
    for i in range(P.shape[0]):
        p=list(P[i,:][::-1])
        Q.append(p)

    R=np.array(Q)
    return(R)

def top_n_accuracy(A,C,n):
    B=arg_sort_desc(C)
    topn = B[:,:n]
    A = np.array(A)

    return np.mean(np.array([1 if A[k] in topn[k] else 0 for k in range(len(topn))]))
```

Figure 29: Function for calculating top-n accuracy

In the code artifacts, all the models are implemented in different Jupyter notebook files for simplicity. Finally, all the variables are initialized. The code snippets below remain the same for both models. The only variation is which model function is being called.

```
path = 'C:/Users/x19241267VM/Desktop/VGG_19_concat_Imbalanced_Normal/'      #give your folder path
img_path = path+'Image_data/'
train_file = path+'Train_Normal.txt'
test_file = path+'Test_Normal.txt'
val_file = path+'Validation_Normal.txt'
#test_test ='yoga_test.txt'
f1 = open(train_file, 'r')
f2 = open(val_file, 'r')
f3 = open(test_file, 'r')
lines1 = f1.readlines()
f1.close()
train_samples = len(lines1)
#print(train_samples)
lines2 = f2.readlines()
f2.close()
val_samples = len(lines2)
lines3 = f3.readlines()
f3.close()
test_samples = len(lines3)
num_classes = [6,20,82]
batch_size = 32
epochs = 30

model = Yoga_Hir_Concat_VGG19()   ##Call the required model
```

Figure 30: Load the model for the training phase

Values for the callbacks, optimizer, loss function were defined. The model was compiled and trained using the fit_generator function of Keras. The trained model was saved so that its weight and architecture both could be retained to be loaded in the future.

```
lr = 0.003 # orig= 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)
rmsprop = RMSprop(learning_rate = lr, momentum=0.9)

model.compile(loss=['categorical_crossentropy','categorical_crossentropy','categorical_crossentropy'],
              loss_weights=[1,1,1], optimizer= sgd, metrics=['accuracy'])#, 'top_k_categorical_accuracy'])

model.summary()

checkpointer = ModelCheckpoint(filepath=path+'weights_VGG19_Hir_Concat_Normal_Imb.hdf5',
                               verbose=1, save_best_only= True, monitor='val_loss')
csv_logger= CSVLogger(path+'log_VGG19_Hir_Concat_Normal_Imb.csv')
early_Stop = EarlyStopping(monitor='val_loss', patience=15, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, cooldown=1, verbose=1)

train_val = 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,early_Stop,reduce_lr],
                          validation_data=generator_val_batch(val_file, batch_size,num_classes,img_path),
                          validation_steps=val_samples // batch_size,
                          verbose=1)

model.save(path+"VGG19_Hir_Concat_Normal_Imb_model.h5")

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

score = model.evaluate_generator(generator_val_batch(val_file,8,num_classes,img_path),steps=val_samples // 8, verbose=1)
print(score)
print("")
```

Figure 31: Model compilation, training, and saving

```
accuracy_Level1 = train_val.history['coarse_1_pred_accuracy']
accuracy_Level2 = train_val.history['coarse_2_pred_accuracy']
accuracy_Level3 = train_val.history['fine_pred_accuracy']

val_acc_Level1 = train_val.history['val_coarse_1_pred_accuracy']
val_acc_Level2 = train_val.history['val_coarse_2_pred_accuracy']
val_acc_Level3 = train_val.history['val_fine_pred_accuracy']

loss_Level1 = train_val.history['coarse_1_pred_loss']
loss_Level2 = train_val.history['coarse_2_pred_loss']
loss_Level3 = train_val.history['fine_pred_loss']

val_loss_Level1 = train_val.history['val_coarse_1_pred_loss']
val_loss_Level2 = train_val.history['val_coarse_2_pred_loss']
val_loss_Level3 = train_val.history['val_fine_pred_loss']

loss = train_val.history['loss']
val_loss = train_val.history['val_loss']

epochs = range(len(accuracy_Level1))
plt.plot(epochs, accuracy_Level1, 'r', label='Training Level 1 accuracy')
plt.plot(epochs, val_acc_Level1, 'b', label='Validation Level 1 accuracy')
plt.title('Training and validation accuracy for classes in Level 1 (coarse Level 1)')
plt.legend()
plt.figure()

plt.plot(epochs, accuracy_Level2, 'r', label='Training Level 2 accuracy')
plt.plot(epochs, val_acc_Level2, 'b', label='Validation Level 2 accuracy')
plt.title('Training and validation accuracy for classes in Level 2 (coarse Level 2)')
plt.legend()
plt.figure()

plt.plot(epochs, accuracy_Level3, 'r', label='Training Level 3 accuracy')
plt.plot(epochs, val_acc_Level3, 'b', label='Validation Level 3 accuracy')
plt.title('Training and validation accuracy for classes in Level 3 (Fine Level)')
plt.legend()
plt.figure()

plt.plot(epochs, loss_Level1, 'r', label='Training Level 1 loss')
plt.plot(epochs, val_loss_Level1, 'b', label='Validation Level 1 loss')
plt.title('Training and validation loss for classes in Level 1 (coarse Level 1)')
plt.legend()
plt.figure()

plt.plot(epochs, loss_Level2, 'r', label='Training Level 2 loss')
plt.plot(epochs, val_loss_Level2, 'b', label='Validation Level 2 loss')
plt.title('Training and validation loss for classes in Level 2 (coarse Level 2)')
plt.legend()
plt.figure()

plt.plot(epochs, loss_Level3, 'r', label='Training Level 3 loss')
plt.plot(epochs, val_loss_Level3, 'b', label='Validation Level 3 loss')
plt.title('Training and validation loss for classes in Level 3 (Fine Level)')
plt.legend()
plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')
```

Figure 32: Plotting all the accuracy and loss graphs

```
#Prediction with test images
preds_VGG19 = model.predict_generator(generator=generator_test_batch(test_file,8,num_classes,img_path),
                                      steps=test_samples // 8, verbose=1)
#print(preds)
true_labels = PlotConfusionMatrix(preds_VGG19,test_file)

### Top - n accuracy for all the three levels of hierarchy
preds_level_1 = preds_VGG19[0]
preds_level_2 = preds_VGG19[1]
preds_level_3 = preds_VGG19[2]

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

## Top-1 and Top-5 accuracy for Level 1 of Hierarchy - Coarse level 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 of model for Level 1 of Hierarchy - Coarse level with 6 classes: ", top_1_acc_1)
print("Top-3 Accuracy of model for Level 1 of Hierarchy - Coarse level with 6 classes: ", top_3_acc_1)
print("Top-5 Accuracy of model for Level 1 of Hierarchy - Coarse level with 6 classes: ", top_5_acc_1)

## Top-1 and Top-5 accuracy for Level 2 of Hierarchy - intermediate level 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 of model for Level 2 of Hierarchy - intermediate level with 20 classes: ", top_1_acc_2)
print("Top-3 Accuracy of model for Level 2 of Hierarchy - intermediate level with 20 classes: ", top_3_acc_2)
print("Top-5 Accuracy of model for Level 2 of Hierarchy - intermediate level with 20 classes: ", top_5_acc_2)

## Top-1 and Top-5 accuracy for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses)
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 of model for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses): ", top_1_acc_3)
print("Top-3 Accuracy of model for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses): ", top_3_acc_3)
print("Top-5 Accuracy of model for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses): ", top_5_acc_3)
```

Figure 33: Prediction on test images and calculating the top-n test accuracy

Below is a sample of how the evaluation plots look like and the calculated top-1, 3, and 5 test accuracy values for all three levels of hierarchy for the DenseNet201 – Hierarchical – Using connectivity pattern model (Densenet201_hir_Concat)

Below are the train and validation accuracy plots for all three levels during training.

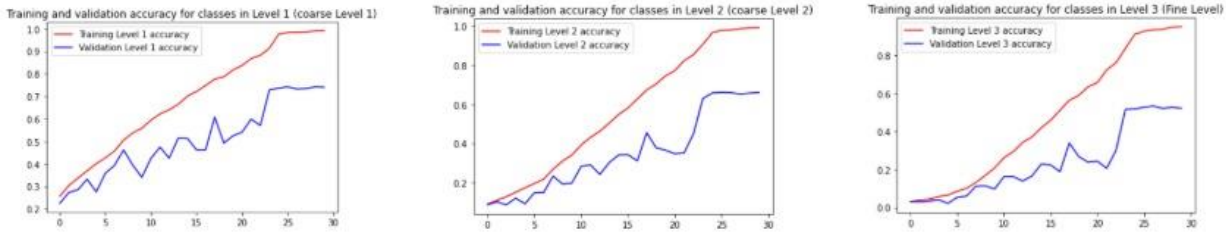


Figure 34: Train and validation accuracy plots for all three levels

Below are the train and validation loss plots for all three levels during training.

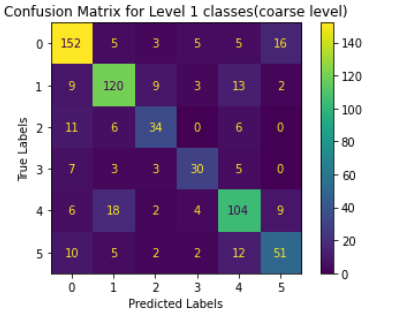
Figure 35: Train and validation loss plots for all three levels

Below is the train and validation loss plot of the entire model during training.

Figure 36: train and validation loss plot for all the entire model

Below are the top-1, top-3, and top-5 accuracy rates on the predicted test set for all three levels of the model and the confusion matrix for the first and second levels of hierarchy.

```
Top-1 Accuracy of model for Level 1 of Hierarchy - Coarse level with 6 classes:  0.7306547619047619
Top-3 Accuracy of model for Level 1 of Hierarchy - Coarse level with 6 classes:  0.9375
Top-5 Accuracy of model for Level 1 of Hierarchy - Coarse level with 6 classes:  0.9880952380952381
Top-1 Accuracy of model for Level 2 of Hierarchy - intermediate level with 20 classes:  0.6354166666666666
Top-3 Accuracy of model for Level 2 of Hierarchy - intermediate level with 20 classes:  0.8497023809523809
Top-5 Accuracy of model for Level 2 of Hierarchy - intermediate level with 20 classes:  0.9151785714285714
Top-1 Accuracy of model for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses):  0.5193452380952381
Top-3 Accuracy of model for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses):  0.6994047619047619
Top-5 Accuracy of model for Level 3 of Hierarchy - Fine level with 82 classes(82 Yoga Poses):  0.7619047619047619
```



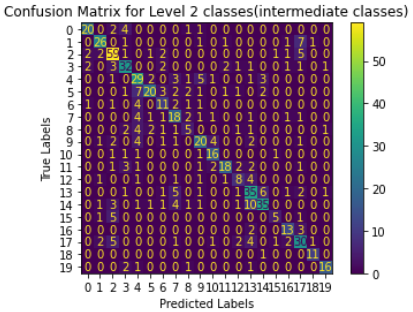


Figure 37: Top-n test accuracy and confusion matrix for the predicted images

References

Verma, M. et al. (2020) 'Yoga-82: A New Dataset for Fine-grained Classification of Human Poses', in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 4472–4479. doi: 10.1109/CVPRW50498.2020.00527.