

Human Pose Estimation

B.E. Project Report

Session 2021-22

Submitted in partial fulfilment of requirements for the award of the degree of

Integrated Bachelor of Technology and MBA

(Computer Science)

Submitted to

Lovely Professional University Phagwara, Punjab



L OVELY
P ROFESSIONAL
U NIVERSITY

Submitted By: Shivanshu Yadav

Registration No: 11902839

Subject: INT247

Signature:

Declaration

I the undersigned solemnly declare that the project report titled **Human Pose Estimation** is based on my own work carried out during my study under the supervision of dr. Sagar Pande

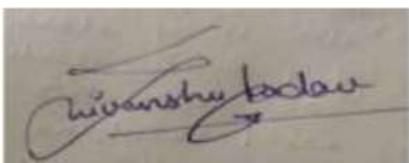
I assert the statements made and conclusions drawn are an outcome of my research work. I further certify that

- The work contained in the report is original and has been done by me under the general supervision of my supervisor.
- The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.
- We have followed the guidelines provided by the university in authoring the report.
- Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

Name: V Shivanshu Yadav

Reg. No: 11902839

Signature:

A photograph of a handwritten signature in blue ink on a light-colored surface. The signature is cursive and appears to read 'Shivanshu Yadav'.

Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to dr. Sagar Pande for their guidance and constant supervision as well as for providing necessary information about the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & fellow mates for their kind co-operation and encouragement which help me in completion of this project.

Name: V Shivanshu Yadav

Reg. No: 11902839

Index

Serial No	Title	Page No
1	Abstract	5
2	Introduction	6
3	Introduction to MediaPipe	12
4	Implementation of Project	16
5	Results	30
6	Application and Future Scope	38
7	Conclusion	41
8	References	42

Abstract

Human Pose Estimation(HPE) is a computer vision task that is usually tackled through deep learning. It is one of the most interesting areas of research that has gained a lot of traction because of its usefulness and versatility—it finds applications in a wide range of fields including gaming, healthcare, AR, and sports.

This report will give you a vast and logical overview of what human pose estimation is and how it works. Under this document one I'll explain about what Human Pose Estimation(HPE).

Human pose estimation locates body key points to accurately recognizing the postures of individuals given an image. This step is a important to multiple tasks of computer vision(CV) which include human tracking, human action recognition, gaming, sign languages, human-computer interaction, and video surveillance. Thus, I show this subject project report to fill the knowledge gap and shed light on the research of 2D human pose estimation. A brief is followed by classifying it as a single or multi-person pose estimation based on the number of people needed to be tracked. Then gradually the approaches used in human pose estimation are described before listing some applications and flaws facing in pose estimation. Following that, a centre of attention is given on briefly discussing research with a significant effect on human pose estimation and examine the novelty, motivation, architecture, the procedures (working principles) of each model together with its practical application and drawbacks, datasets implemented, as well as the evaluation metrics used to evaluate the model. This review is presented as a base for new developers and guides researchers to find new models by seeing the procedure and architectural bugs of present research.

Introduction

Human Pose Estimation (HPE) is a way of finding and classifying the joints in the human body. Essentially it is a way to capture a set of coordinates for each joint (arm, head, torso, etc.,) which is known as a key point that can describe a pose of a person. The connection between these points is known as a pair. The connection formed between the points must be significant, which means not all points can form a pair. From the outset, the aim of HPE is to form a skeleton-like representation of a human body and then process it further for task-specific applications.

However— There are three types of approaches to model the human body:

- Skeleton-based model
- Contour-based model
- Volume-based model

approaches are primarily around computer vision, and it is used to understand geometric and motion information of the human body, which can be very intricate.

This section explores the two approaches: the classical approach and the deep learning-based approach to Human pose estimation. We will also explain how classical approaches do not capture the geometric and motion information of the human body, and how deep learning algorithms such as the CNNs excel at it.

Bottom-up vs. Top-down methods

All approaches for pose estimation can be grouped into bottom-up and top-down methods.

- Bottom-up methods estimate each body joint first and then group them to form a unique pose. Bottom-up methods were pioneered with Deep Cut (a method we will cover later in more detail).
- Top-down methods run a person detector first and estimate body joints within the detected bounding boxes.

Classical approaches to 2D Human Pose Estimation

Classical approaches usually refer to techniques and methods involving shallow machine learning algorithms. For instance, the earlier work to estimate human pose included the implementation of random forest within a “pictorial structure framework”. This was used to predict joints in the human body.

The pictorial structure framework (PSF) is commonly referred to as one of the traditional methods to estimate human pose. PSF had two components:

- **Discriminator:** It models the likelihood of a certain part present at a particular location. In other words, it identifies the body parts.
- **Prior:** It is referred to as modelling the probability distribution over pose using the output from the discriminator; the modelled pose should be realistic.

In essence, the pictorial structure framework objective is to represent the human body as a collection of coordinates for each body part in each input image. pictorial structure framework uses nonlinear joint regressors, ideally a two-layered random forest regressor.

These models work well when the input image has clear and visible limbs, however, they do not capture and model limbs that are hidden or not visible from a certain angle.

To overcome these issues, feature building methods like histogram oriented gaussian (HOG), contours, histograms, etc., were used. Despite using these methods, the classical model lacked accuracy, correlation, and generalization capabilities, so adopting a better approach was just a matter of time.

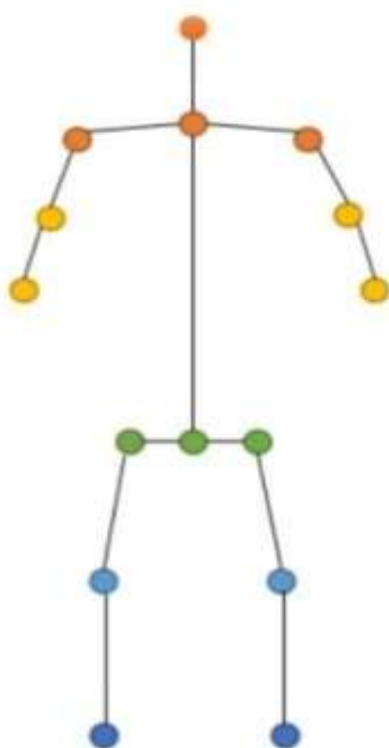
3d Human Body Modelling

In human pose estimation, the location of human body parts is used to build a human body representation (such as a body skeleton pose) from visual input data. Therefore, human body modelling is an important aspect of human pose estimation. It is used to stand for features and key points extracted from visual input data. Typically, a model-based approach is used to describe and infer human body poses and render 2D or 3D poses.

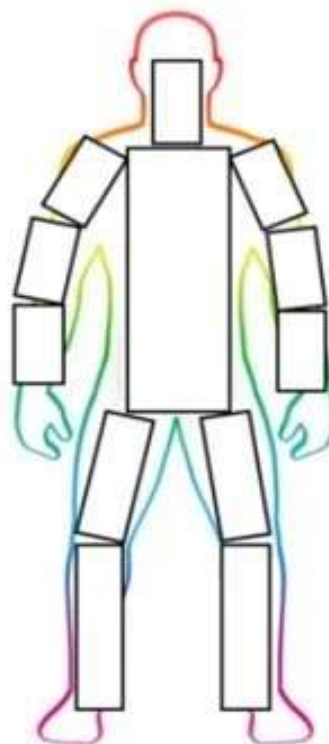
Most methods use an N-joints rigid kinematic model where a human body is represented as an entity with joints and limbs, having body kinematic structure and body shape information.

There are three types of models for human body modelling:

- **Kinematic Model**, also called skeleton-based model, is used for 2D pose estimation as well as 3D pose estimation. This flexible and intuitive human body model includes a set of joint positions and limb orientations to stand for the human body structure. Therefore, skeleton pose estimation models are used to capture the relations between different body parts. However, kinematic models are limited in standing for texture or shape information.
- **Planar Model**, or contour-based model, which is used for 2D pose estimation. The planar models are used to stand for the appearance and shape of a human body. Usually, body parts are represented by multiple rectangles approximating the human body contours. A popular example is the Active Shape Model (ASM) that is used to capture the full human body graph and the silhouette deformations using principal part analysis.
- **Volumetric model**, which is used for 3D pose estimation. There exist multiple popular 3D human body models used for deep learning-based 3D human pose estimation for recovering 3D human mesh. For example, GHUM & GHUML(ite), are fully trainable end-to-end deep learning pipelines trained on a high-resolution dataset of full-body scans of over 60'000 human configurations to model statistical and articulated 3D human body shape and pose. It can be used to infer



(a) Kinematic

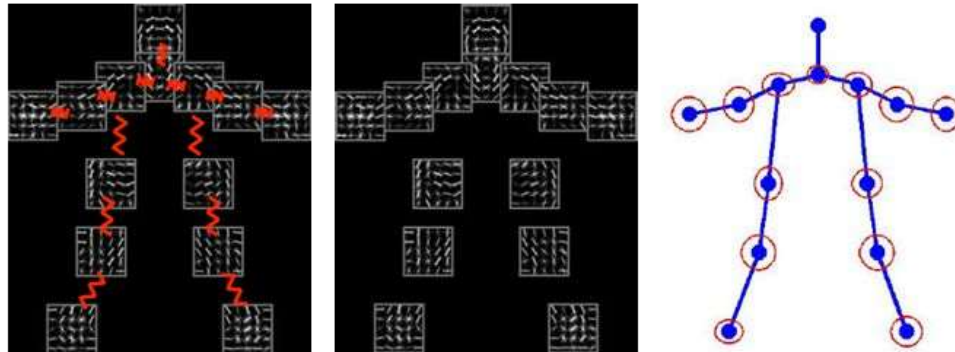


(b) Planar



(c) Volumetric

Pictorial Structure Model



$$S(I, L) = \sum_{i \in V} \alpha_i \cdot \phi(I, l_i) + \sum_{ij \in E} \beta_{ij} \cdot \psi(l_i, l_j)$$

- $\psi(l_i, l_j)$: Spatial features between l_i and l_j
- β_{ij} : Pairwise springs between part i and part j

Deep Learning-based approaches to 2D Human Pose Estimation

Deep learning-based approaches are well defined by their ability to generalize any function (if enough nodes are present in the given hidden layer). When it comes to computer vision tasks, deep convolutional neural networks (CNN) surpass all other algorithms, and this is true in HUMAN POSE ESTIMATION as well.

CNN has the ability to extract patterns and representations from the given input image with more precision and accuracy than any other algorithm; this makes CNN especially useful for tasks such as classification, detection, and segmentation. Unlike the classical approach, where the features were handcrafted; CNN can learn complex features when provided with enough training data.

Toshev et al in 2014 initially used the CNN to estimate human pose, switching from the classical-based approach to the deep learning-based approach, and they named it Deep Pose: Human Pose Estimation via Deep Neural Networks. In the paper that they had released, they defined the whole problem as a CNN-based regression problem towards body joints.

The authors also proposed an added method where they implemented the cascade of such regressors in order to get more precise and consistent results. They argued that the proposed Deep Neural Network can model the given data in a holistic fashion, i.e., the network has the capability to model hidden poses, which was not true for the classical approach.

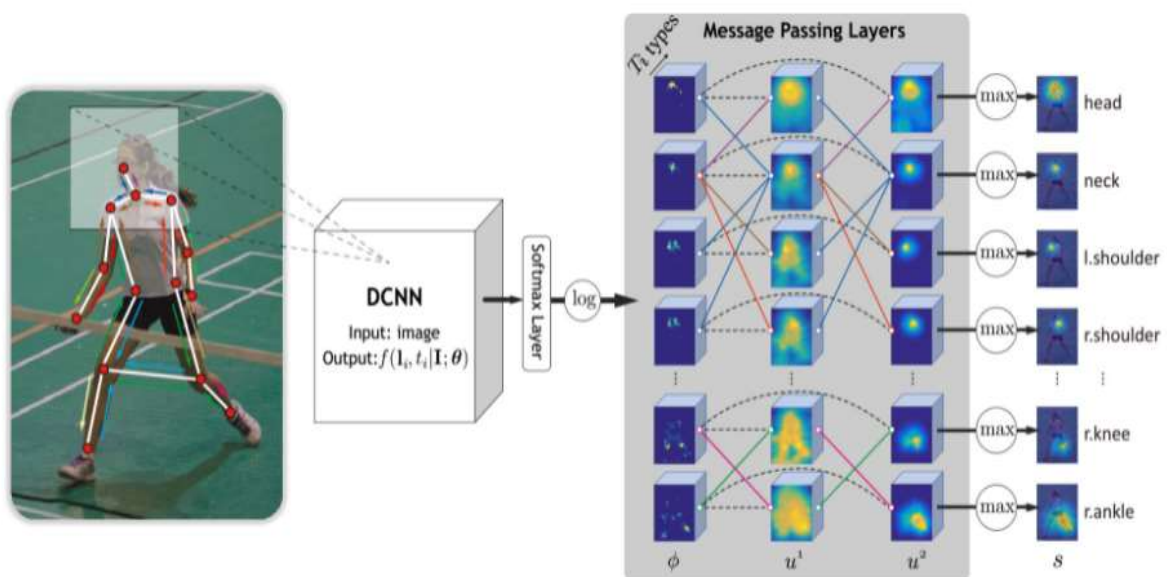
With strong and promising results shown by Deep Pose, the human pose estimation research naturally gravitated towards the deep learning-based approaches.

Human Pose Estimation using Deep Neural Networks

As the research and development started to take off in human pose estimation, it brought forth new challenges. One of them was to tackle the multi-person pose estimation. DNNs are very proficient in estimating single human pose but when it comes to estimating multi-human they struggle because:

1. An image can hold multiple numbers of people in various positions.
2. As the count of people adds, the interaction between increases leads to computational complexities.
3. An hike in computational complexities can often lead to an increment in inference time in real-time.

To tackle these problems, the researchers introduced two approaches:



Top-down: Localize the humans in the image or video and then estimate the parts followed by calculating the pose.

Bottom-up: Estimate the human body parts in the image followed by calculating the pose.

Here I'll be using Media Pipe an opensource library to perform deep neural network-based pose estimation on given image or video frames.

Main Challenges of Pose Detection

- Human pose estimation is a challenging task as the body's appearance joins changes dynamically due to diverse forms of clothes, arbitrary occlusion, occlusions due to the viewing angle, and background contexts.
- Pose estimation needs to be robust to challenging real-world variations such as are lighting and weather.

Therefore, it is challenging for image processing models to find the fine-grained joint coordinates. It is especially difficult to track small and barely visible joints.

Introduction to Media-Pipe

Google open-source MediaPipe was first introduced in June 2019. It aims to make our life easy by supplying some integrated computer vision and machine learning features. Media Pipe is a framework for building multimodal(e.g., video, audio or any time series data),cross-platform (i.e., Android, IOS, web, edge devices) applied ML pipelines. MediaPipe also eases the deployment of machine learning technology into demos and applications on a wide variety of different hardware platforms.

Notable Applications

Face Detection

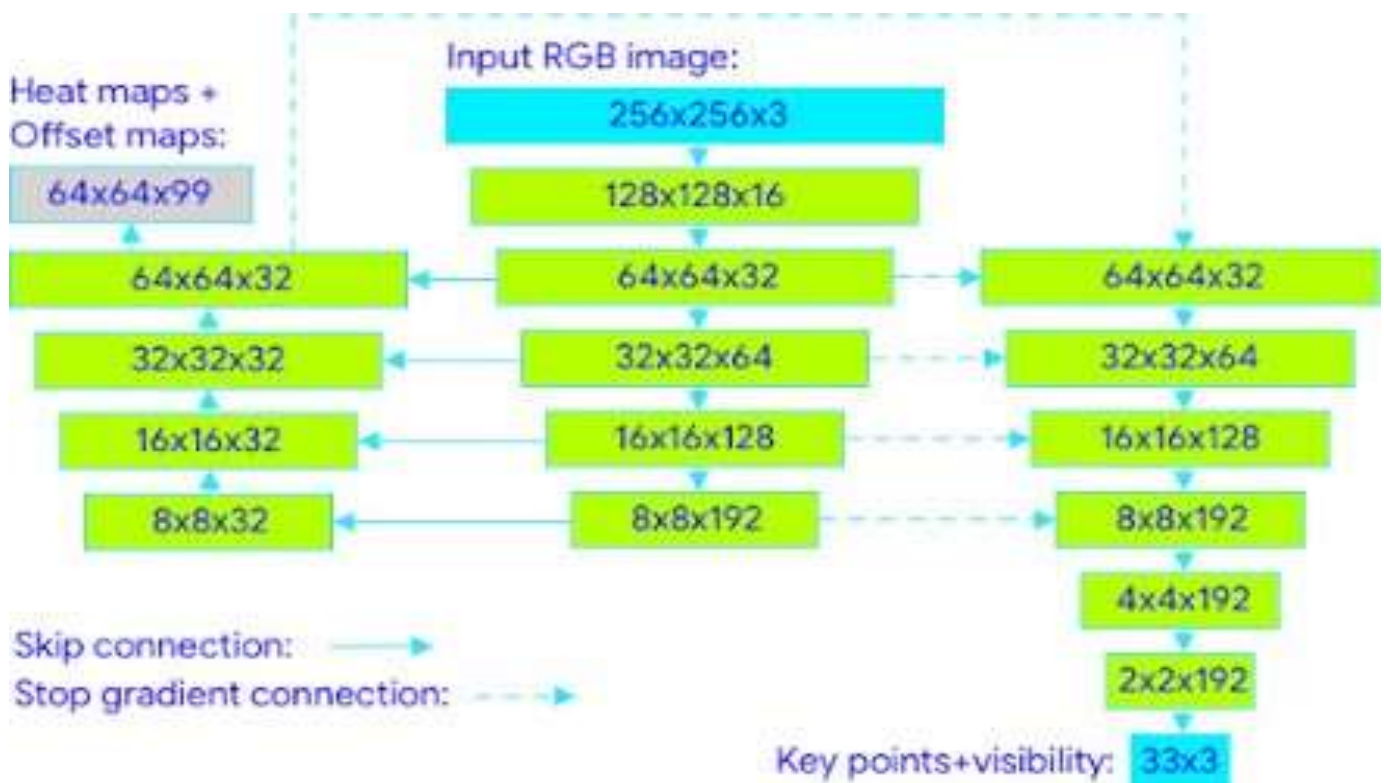
Multi-hand Tracking

Hair Segmentation

Object Detection and Tracking

Objection: 3D Object Detection and Tracking

Auto Flip: Automatic video cropping pipeline



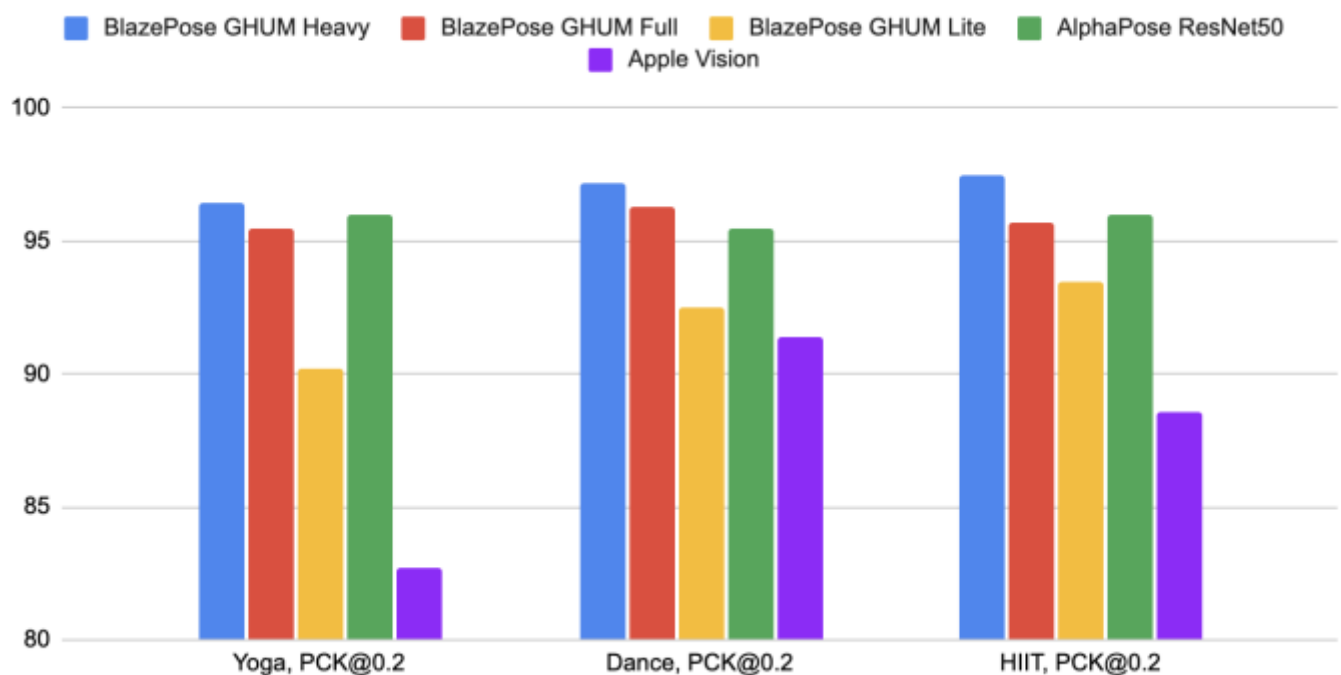
The image above shows the architecture of Blaze Pose which is a multi-stage CNN.

ML Pipeline

The solution uses a two-step detector-tracker ML pipeline, proven to be effective in our MediaPipe Hands and MediaPipe Face Mesh solutions. Using a detector, the pipeline first finds the person/pose region-of-interest (ROI) within the frame. The tracker subsequently predicts the pose landmarks and segmentation mask within the ROI using the ROI-cropped frame as input. Note that for video use cases the detector is invoked only as needed, i.e., for the very first frame and when the tracker could no longer find body pose presence in the previous frame. For other frames, the pipeline simply derives the ROI from the previous frame's pose landmarks.

The pipeline is implemented as a MediaPipe graph that uses a pose landmark subgraph from the pose landmark module and makes using a dedicated pose renderer subgraph. The pose landmark subgraph internally uses a pose detection subgraph from the pose detection module.

Note: To visualize a graph, copy the graph and paste it into MediaPipe Visualizer. For more information on how to visualize its associated subgraphs, please see visualizer documentation.

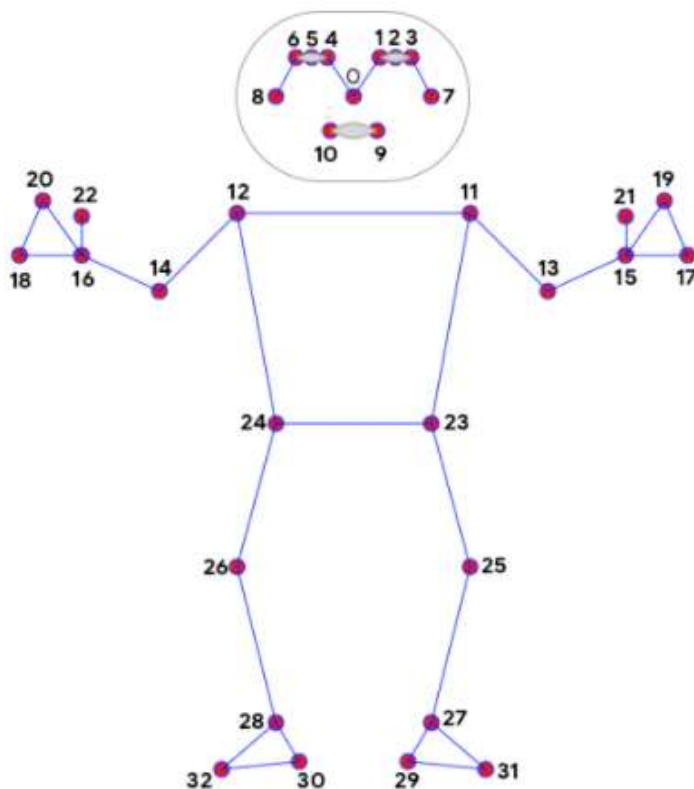


Process of Estimation

- In first step the image is passed through baseline CNN network to extract the feature maps of the input in the paper. In this paper the authors used first 10 layers of VGG-19 network.
- The feature map is then process in a multi-stage CNN pipeline to generate the Part Confidence Maps and Part Affinity Field
 - Part Confidence Maps:
 - Part Affinity Field

In the last step, the Confidence Maps and Part Affinity Fields that are generated above are processed by a greedy bipartite matching algorithm to obtain the poses for each person in the image.

The landmark model in MediaPipe Pose predicts the location of 33 pose landmarks (see figure below).



0. nose	17. left_pinky
1. left_eye_inner	18. right_pinky
2. left_eye	19. left_index
3. left_eye_outer	20. right_index
4. right_eye_inner	21. left_thumb
5. right_eye	22. right_thumb
6. right_eye_outer	23. left_hip
7. left_ear	24. right_hip
8. right_ear	25. left_knee
9. mouth_left	26. right_knee
10. mouth_right	27. left_ankle
11. left_shoulder	28. right_ankle
12. right_shoulder	29. left_heel
13. left_elbow	30. right_heel
14. right_elbow	31. left_foot_index
15. left_wrist	32. right_foot_index
16. right_wrist	

Optionally, MediaPipe Pose can predict a full-body segmentation mask represented as a two-class segmentation (human or background).

Model	FPS	AR Dataset, PCK@0.2	Yoga Dataset, PCK@0.2
OpenPose (body only)	0.4 ¹	87.8	83.4
BlazePose Full	10 ²	84.1	84.5
BlazePose Lite	31 ²	79.6	77.6

TABLE 1. Performance Comparison

The pose estimation component of system predicts the location of all 33-person key points and uses the person alignment proposal provided by the first stage of the pipeline. Author adopted a joint heatmap, offset, and regression approach. We use the heatmap and offset loss only in the training stage and remove the corresponding output layers from the model before running the inference. Thus, it effectively uses the heatmap to supervise the lightweight embedding, which is then used by the regression encoder network.

This approach is partially inspired by Stacked Hourglass approach of Newell et al., but in this case, we stack a tiny encoder-decoder heatmap-based network and a later regression encoder network. Author actively uses skip-connections between all the stages of the network to achieve a balance between high and low-level features. However, the gradients from the regression encoder are not propagated back to the heatmap trained features. We have found this to not only improve the heatmap predictions, but also substantially increase the coordinate regression accuracy.

Mediapipe Implements the backend on Blaze Pose architecture and Blaze Pose is written taking a base estimation as Open Pose. Thus, both are quite efficient pose landmark detectors.

Implementation

We can download image data from Kaggle datasets from this link:

#Dataset Link: kaggle.com/datasets/niharika41298/yoga-poses-dataset?resource=download

- First step is to import some libraries for data processing into our python notebook
 - Media-Pipe (Getting Pose landmarks)
 - OpenCV-python (Process Image files)
 - NumPy (To Perform mathematical operation)
 - Pandas (Creating and supporting dataset)
 - OS (Reading filesystem)
 - Matplotlib (plotting numerical optimisation data in visual format)
 - Sklearn (Data optimization)
 - XGBoost (Machine learning model).

```

1 #dataset Link: kaggle.com/datasets/niharika41298/yoga-poses-dataset?resource=download
2
3 import mediapipe as mp
4 import cv2
5 import time
6 import numpy as np
7 import pandas as pd
8 import os
9 import matplotlib.pyplot as plt
10 from sklearn.metrics import roc_auc_score, accuracy_score
11 from sklearn import preprocessing
12 from sklearn.model_selection import train_test_split
13 from xgboost import XGBClassifier
14 #from sklearn import tree
15 #from sklearn.datasets import make_classification
16
17 print("Numpy Version:", np.__version__)
18 print("OpenCV version:", cv2.__version__)

```

```

Numpy Version: 1.22.3
OpenCV version: 4.5.5

```


- Now we can check some example images from our datasets.

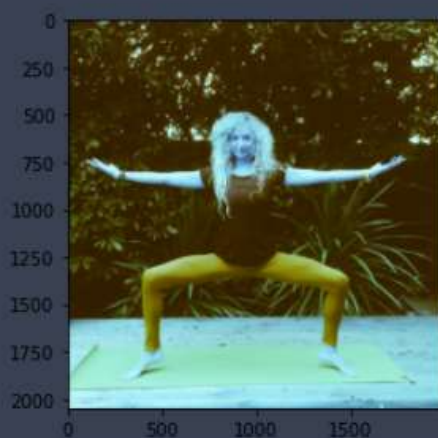
```
1 train = "DATASET/TRAIN/"
2 test = "DATASET/TEST/"
3
4 print("Example images")
5 for d in os.listdir(test):
6     print(d,end=" ")
7     img=cv2.imread(test+d+"/"+os.listdir(test+d)[np.random.randint(len(os.listdir(test+d)))])
8     plt.imshow(img)
9     plt.show()
```

Example images
downdog

Example images
downdog



goddess



- Third step is to count data present in our dataset. While reading it from file system and displaying it using the print statement under a for loop iterating over os.listdir() method.

```

1 total=0
2 for d in os.listdir(test):
3     print(d,end=" ")
4     total+=len(os.listdir(test+d))
5     print("images present: ",len(os.listdir(test+d)))
6 print("Total Test images:",total)
7 print()
8
9 tot2=0
10 for d in os.listdir(train):
11     print(d,end=" ")
12     tot2+=len(os.listdir(train+d))
13     print("images present: ",len(os.listdir(train+d)))
14 print("Total Train images:",tot2)
15 total+=tot2;
16 print("Total Images:",total)
17
18 PARTS={"Nose":0,"Neck":1,"RShoulder":2,"RElbow":3,"RWrist":4,"LShoulder":5,
19        "LElbow":6,"LWrist":7,"RHip":8,"RKnee":9,"RAnkle":10,"LHip":11,"LKnee":12,
20        "LAnkle":13,"REye":14,"LEye":15,"REar":16,"LEar":17,"Background":18}
21
22 PAIRS=[["Neck","RShoulder"],["Neck","LShoulder"],["RShoulder","RElbow"],
23         ["RElbow","RWrist"],["LShoulder","LElbow"],["LElbow","LWrist"],
24         ["Neck","RHip"],["RHip","RKnee"],["RKnee","RAnkle"],["Neck","LHip"],
25         ["LHip","LKnee"],["LKnee","LAnkle"],["Neck","Nose"],["Nose","REye"],
26         ["REye","REar"],["Nose","LEye"],["LEye","LEar"]]
27
28 print("\nBody Parts available:",len(PARTS))
29 print("Total Number of pairs:",len(PAIRS))
30

```

```
downdog images present: 97
goddess images present: 80
plank images present: 115
tree images present: 69
warrior2 images present: 109
Total Test images: 470
```

```
downdog images present: 223
goddess images present: 180
plank images present: 266
tree images present: 160
warrior2 images present: 252
Total Train images: 1081
Total Images: 1551
```

```
Body Parts available: 19
Total Number of pairs: 17
```

So, we have a total of 1551 images with 1081 train and 470 test Images to process and make our machine learning dataset model to predict the pose.

- We have 5 classes of poses to train our model on

- Down dog



- Goddess



- Plank



○ Tree



○ Warrior 2



- Loading landmark model from media pipe into a variable and make data frame with necessary headers. Our data frame must have X, Y, Z and, Visibility of coordinates for each predicted landmark also a title label for the pose. Pose is our feature and target column to get prediction on.

```

1  mpPose = mp.solutions.pose
2  pose = mpPose.Pose()
3  mpDraw = mp.solutions.drawing_utils # For drawing keypoints
4  points = mpPose.PoseLandmark # Landmarks
5
6  data = []
7  data.append("pose")
8  for p in points:
9      x = str(p)[13:]
10     data.append(x + "_x")
11     data.append(x + "_y")
12     data.append(x + "_z")
13     data.append(x + "_vis")
14  head=data;
15  data = pd.DataFrame(columns = data) # Empty dataset
16  print("Length of Dataset row:",len(head))

```

Length of Dataset row: 133

A Total of 133 dataset columns are to be made which are a collection of 33 key points/landmarks.

- Now finally let us process our data into a valid data frame and save it into a csv file to train our model on.
- PROCESS
 - Read image from folders named test and train
 - Convert BGR image to RGB format to deal with any colour interchanges
 - Get a black Image of same shape as of image that is been read from file
 - Get predicted landmark result and save them into a new row of our data frame
 - If no results are achieved from a certain image, then that image is invalid, and it will be ignored by the algorithm.

- Draw the landmarks on the black image then plot it on a graph to see the resultant figure for a given image.
- Print the percentage of work done. And eventually it will show how percentage of data is valid from supplied set of images.
- Save all the landmarks in a data frame and plot images on matplotlib pyplot, to see how data looked like.

```

1 count,p = 0,0
2 examples=[]
3 print("Processing Data:")
4 for D in [test,train]:
5     for folder in os.listdir(D):
6         for img in os.listdir(D+folder):
7             temp = []
8             img = cv2.imread(D+folder + "/" + img)
9             img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
10            imageWidth, imageHeight = img.shape[:2]
11            imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
12            blackie = np.zeros(img.shape) # Blank image
13            results = pose.process(imgRGB)
14            if results.pose_landmarks:
15
16                # mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS) +
17                mpDraw.draw_landmarks(blackie, results.pose_landmarks, mpPose.POSE_CONNECTIONS)
18                landmarks = results.pose_landmarks.landmark
19                temp=temp+[folder]
20                for i,j in zip(points,landmarks):
21                    temp = temp + [j.x, j.y, j.z, j.visibility]
22                data.loc[count] = temp
23                count +=1
24                if len(examples)<2:
25                    examples.append([img,blackie])
26
27            if int((count/total)*100) > p:
28                p=int((count/total)*100);
29                stat="#"*int(p/2)+"-"*int(50-(p/2))+ " "+"{0:.3f}%".format((count/total)*100)

```

```

29         stat="#"*int(p/2)+"-"*int(50-(p/2))+ " "+"{0:.3f}%".format((count/total)*100)
30         print(stat,end="\r", flush=True)
31         cv2.waitKey(100)
32
33
34     '''data['pose']=data['pose'].apply(lambda x:0 if x=='downdog'
35                                     else 1 if x=='goddess'
36                                     else 2 if x=='plank'
37                                     else 3 if x=='tree'
38                                     else 4)'''
39
40     data.to_csv("data.csv") # save the data as a csv file
41     print("Saved into data.csv\n\n");
42
43     print("Example data:")
44     fig=plt.figure()
45     fig.add_subplot(2,2,1)
46     plt.imshow(examples[0][0])
47
48     fig.add_subplot(2,2,2)
49     plt.imshow(examples[0][1])
50
51     fig.add_subplot(2,2,3)
52     plt.imshow(examples[1][0])
53
54     fig.add_subplot(2,2,4)
55     plt.imshow(examples[1][1])
56     plt.show()

```

- And output will look like this:

```
Processing Data:
#####----- 85.687%%

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Saved into data.csv

Example data:

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Finally save data into a file named data.csv to further use it in our training model

Model Training

- The final task is training our ML classifier to predict data classes on
- For Training model I'm using `xgboost.classifier()` a method to fit my dataset on and give us train and validation losses according
- First is to split whole data into two sets
 - Test (a set to test data)
 - Train (a set to Train data)
- For splitting data, we can use sklearn library, using `train_test_split()` method
- Make one evalset and pass it to fit method.


```

1 data=pd.read_csv('data.csv')
2 print("data shape:",data.shape)
3 num_of_classes=len(data.pose.unique())
4 print(num_of_classes)
5

```

```

data shape: (1329, 134)
5

```

```
1 data.head()
```

	Unnamed: 0	pose	NOSE_x	NOSE_y	NOSE_z	NOSE_vis	LEFT_EYE_INNER_x	LEFT_EYE_INNER_y	LEFT_EYE_INNER_z
0	0	down dog	0.508779	0.735192	-0.071366	0.997456	0.493383	0.758222	-0.112861
1	1	down dog	0.488671	0.609469	-0.751016	0.991552	0.507283	0.614835	-0.762855
2	2	down dog	0.539378	0.765981	-0.084369	0.992335	0.556045	0.778583	-0.063912
3	3	down dog	0.655291	0.535797	0.488664	0.991769	0.669238	0.543971	0.528218
4	4	down dog	0.388131	0.755049	-0.051361	0.992228	0.365101	0.744365	-0.090443

```

1 x=data.drop(axis=0, columns=['pose'])
2 y=data.pose
3
4 print(x.shape)
5 print(y.shape)
6 data_train,data_test,y_train,y_test=train_test_split(x,y,
7                                                         shuffle=True,
8                                                         test_size=0.4,
9                                                         random_state=1)
10 evalset = [(data_train, y_train), (data_test,y_test)]
11 print("Data Train:",data_train.shape)
12 print("Data Test:",data_test.shape)
13 print("label Train:",y_train.shape)
14 print("label Test:",y_test.shape)

```

```

(1329, 133)
(1329,)
Data Train: (797, 133)
Data Test: (532, 133)
label Train: (797,)
label Test: (532,)

```

- Initialize a classifier using XGBClassifier and setting some arguments
 - booster='gbtree'
 - objective='multi:softprob'
 - random_state=42
 - eval_metric="mlogloss" (Multiclass Log Loss)
 - num_class=num_of_classes (Total number of classes is 5)
- I used two classifiers one to plot losses and another to plot area under curve.
 - booster='gbtree'
 - objective='multi:softprob'
 - random_state=42
 - eval_metric="auc" (Area Under Curve)
 - num_class=num_of_classes (Total number of classes is 5)
- finally fit the data into our classifiers and get the validation results.

```

1 # Create a classifier
2 xgb = XGBClassifier(booster='gbtree', objective='multi:softprob', random_state=42, eval_metric="mlogloss")
3 e_xgb = XGBClassifier(booster='gbtree', objective='multi:softprob', random_state=42, eval_metric="auc",
4
5 # Fit the classifier with the training data
6 xgb.fit(data_train,y_train, eval_set=evalset)
7 e_xgb.fit(data_train,y_train, eval_set=evalset)

```

[5]	validation_0-mlogloss:0.20003	validation_1-mlogloss:0.28350
[6]	validation_0-mlogloss:0.15038	validation_1-mlogloss:0.23155
[7]	validation_0-mlogloss:0.11385	validation_1-mlogloss:0.19179
[8]	validation_0-mlogloss:0.08780	validation_1-mlogloss:0.16244
[9]	validation_0-mlogloss:0.06842	validation_1-mlogloss:0.13987
[10]	validation_0-mlogloss:0.05311	validation_1-mlogloss:0.11965
[11]	validation_0-mlogloss:0.04206	validation_1-mlogloss:0.10643
[12]	validation_0-mlogloss:0.03356	validation_1-mlogloss:0.09443
[13]	validation_0-mlogloss:0.02731	validation_1-mlogloss:0.08582
[14]	validation_0-mlogloss:0.02201	validation_1-mlogloss:0.07712
[15]	validation_0-mlogloss:0.01812	validation_1-mlogloss:0.07093
[16]	validation_0-mlogloss:0.01547	validation_1-mlogloss:0.06763
[17]	validation_0-mlogloss:0.01330	validation_1-mlogloss:0.06528
[18]	validation_0-mlogloss:0.01131	validation_1-mlogloss:0.06216
[19]	validation_0-mlogloss:0.01012	validation_1-mlogloss:0.06042
[20]	validation_0-mlogloss:0.00887	validation_1-mlogloss:0.05810
[21]	validation_0-mlogloss:0.00816	validation_1-mlogloss:0.05771
[22]	validation_0-mlogloss:0.00761	validation_1-mlogloss:0.05706
[23]	validation_0-mlogloss:0.00711	validation_1-mlogloss:0.05597
[24]	validation_0-mlogloss:0.00663	validation_1-mlogloss:0.05483
[25]	validation_0-mlogloss:0.00624	validation_1-mlogloss:0.05404
[26]	validation_0-mlogloss:0.00589	validation_1-mlogloss:0.05257
[27]	validation_0-mlogloss:0.00560	validation_1-mlogloss:0.05266
[28]	validation_0-mlogloss:0.00540	validation_1-mlogloss:0.05269

- We can get our predicted values using predict method from `xgboost.xgbclassifier().predict()`
 - Pass landmark data frame holding 133 cols after dropping target column from it.
 - This will return a list of predicted pose value for each row.
- Got accuracy score using `sklearn.accuracy_score` library method
 - Pass predicted labels and expected labels
 - Returns accuracy score in scaled floating value from 0 to 1
- Got area under curve score using `sklearn.auc_score` library method
 - Pass predicted labels and expected labels
 - Returns area under curve in scaled floating value from 0 to 1

```

1  # Use trained model to predict output of test dataset
2  val = xgb.predict(data_test)
3  e_val = e_xgb.predict(data_test)
4
5  acc_score=accuracy_score(y_test,val)
6  e_acc_score=accuracy_score(y_test,e_val)
7  print("multiclass logloss Accuracy:",acc_score)
8  print("multiclass error Accuracy:",acc_score)
9
10 lb = preprocessing.LabelBinarizer()
11 lb.fit(y_test)
12
13 y_test_lb = lb.transform(y_test)
14 val_lb = lb.transform(val)
15
16 roc_score=roc_auc_score(y_test_lb, val_lb, average='macro')
17
18 print("\nroc_score(mlogloss):",roc_score)
19 print()

```

```
multiclass logloss Accuracy: 0.9868421052631579
```

```
multiclass error Accuracy: 0.9868421052631579
```

```
roc_score(mlogloss): 0.9911849372289125
```

- Accuracy score: 0.9868421052631579
- roc_score(mlogloss): 0.9911849372289125

- displaying predicted vs Expected results
 - create a data frame using pandas
 - add predicted values under “Predicted” label
 - add expected values under “Expected” label
 - display data frame

```
1 output = pd.DataFrame()
2 output['Expected Output'] = y_test
3 output['Predicted Output'] = val
4 output.head()
5
```

	Expected Output	Predicted Output
453	downdog	downdog
1129	warrior2	warrior2
1118	warrior2	warrior2
115	goddess	goddess
1153	warrior2	warrior2

- from accuracy score we can get accuracy to be around 98.6%
- finally getting plots for multiclass log loss and area under curve for each training iteration

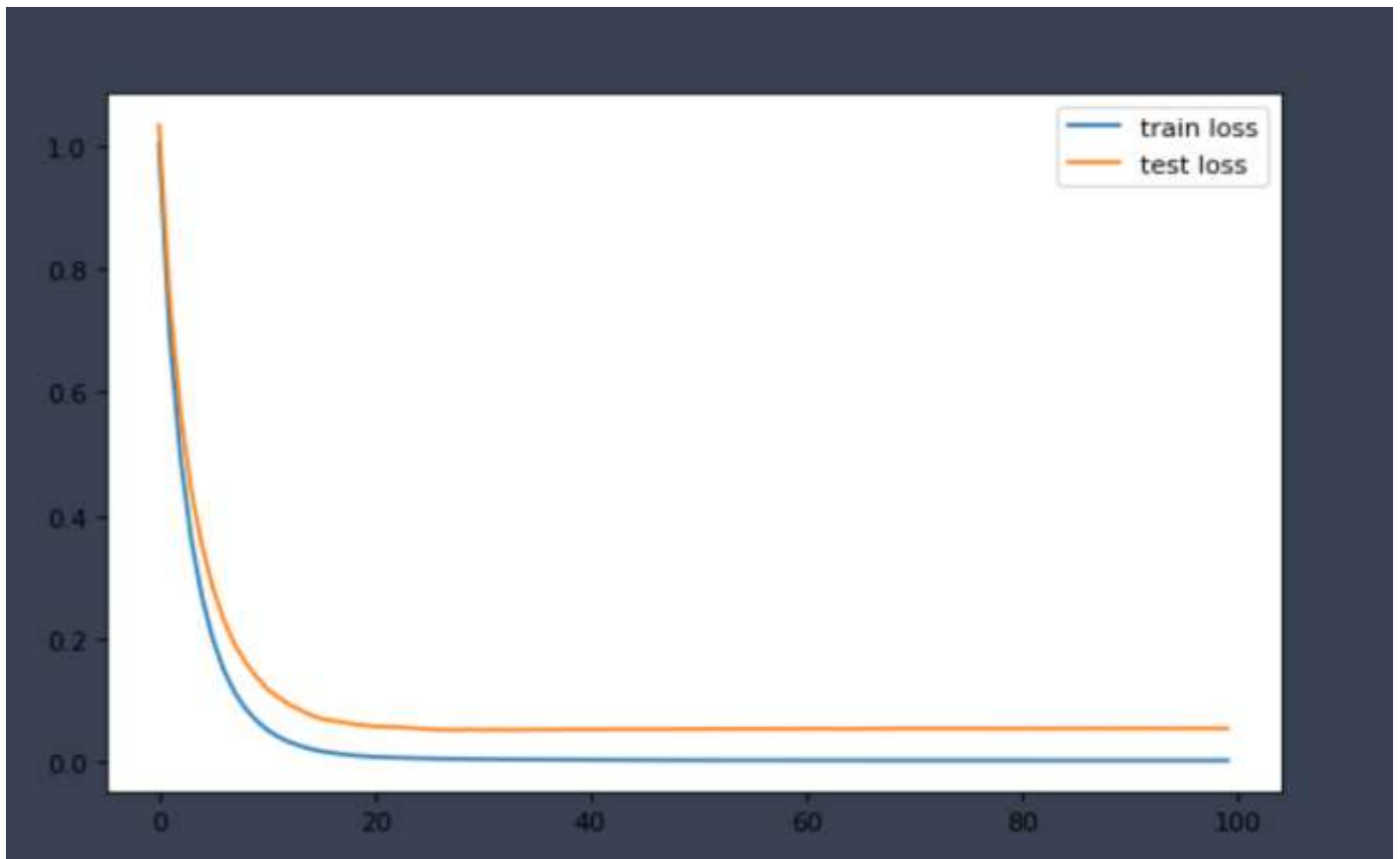


Fig - log loss curve

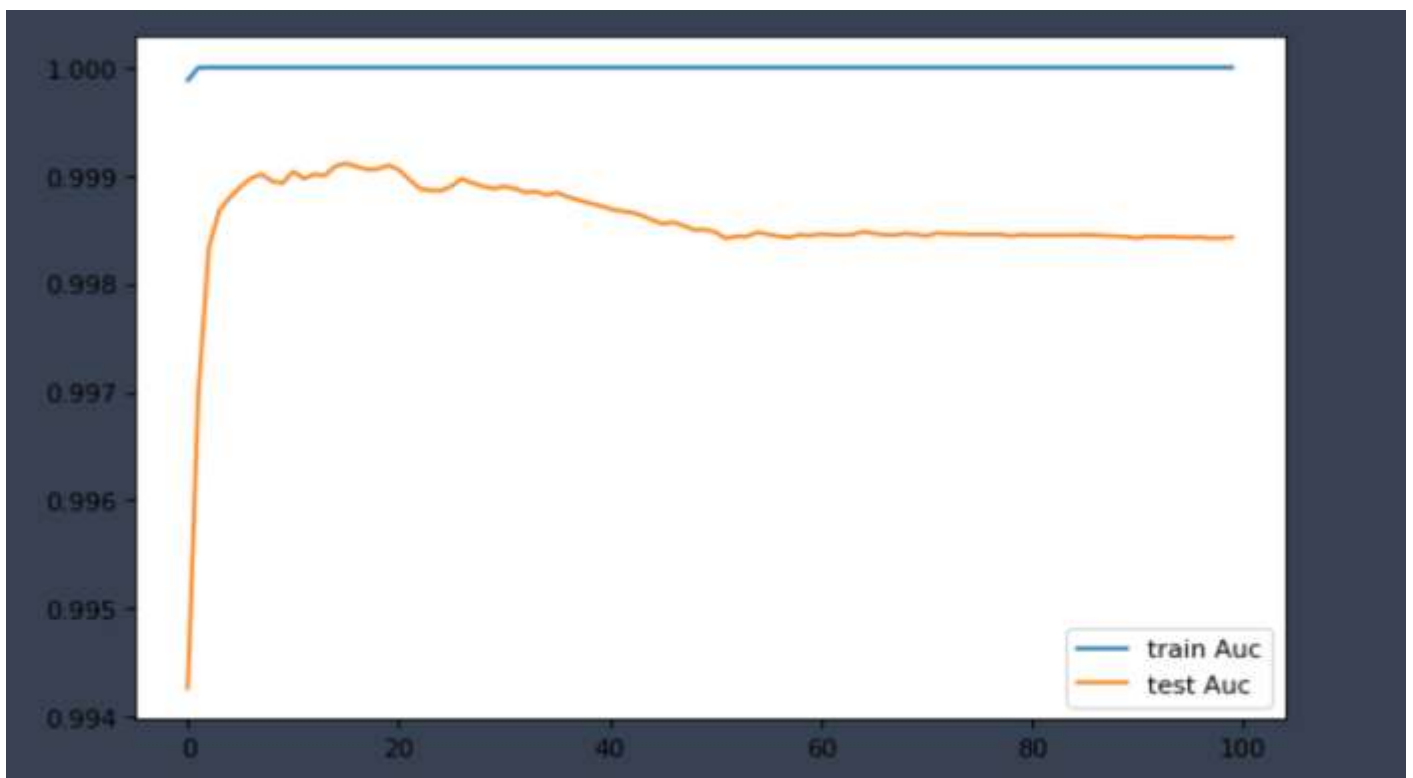


Fig – area under curve

Results

Now we'll predict and display results on images, either on camera frames or on Test set images. The first task will always be to import necessary libraries

- OpenCV
- Sklearn
- XGBoost
- NumPy
- Pandas
- Matplotlib
- MediaPipe
- OS

This will be a user driven program; user will get two options

1. Read images from test set
2. Read frames from camera

We'll loading dataset first into our XGBClassifier and initialize fit function. If user selected reading from test set, then we'll:

- We'll read data from file system where the data for test set is being saved using `os.listdir(PATH)` method, where PATH equals address to the test set directory.
- Then we'll be initializing media pipe model for pose landmark detection into a variable.
- Pass image to the media pipe landmark prediction model fetch the results into a 133-column data frame using pandas.
- Pass it to our XGBClassifier and get predicted pose as a string list.
- Put the predicted output into a blank image after drawing the landmarks from media pipe model.
- initialize a matplotlib figure and plot both the images into it one beside another for getting a good look of what data is looking like .

Code Screenshots:

```

import mediapipe as mp
import cv2
import time
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

mpPose = mp.solutions.pose
pose = mpPose.Pose()
mpDraw = mp.solutions.drawing_utils # For drawing keypoints
points = mpPose.PoseLandmark # Landmarks
test = "DATASET/TEST/"

```

```

points = mpPose.PoseLandmark # Landmarks
test = "DATASET/TEST/"

def get_prediction(results,model):
    return model.predict(results)

data=pd.read_csv('data.csv')
num_of_classes=len(data.pose.unique())
x=data.drop(axis=0, columns=['pose'])
y=data.pose

print(x.shape)
print(y.shape)
data_train,data_test,y_train,y_test=train_test_split(x,y,
                                                    shuffle=True,
                                                    test_size=0.4,
                                                    random_state=1)

evalset = [(data_train, y_train), (data_test,y_test)]
print("Data Train:",data_train.shape)
print("Data Test:",data_test.shape)
print("label Train:",y_train.shape)
print("label Test:",y_test.shape)

xgb = XGBClassifier(booster='gbtree', objective='multi:softprob', random_state=42, eval_metric="mlogloss", num_class=num_of_classes)
xgb.fit(data_train,y_train, eval_set=evalset)

```

```

font = cv2.FONT_HERSHEY_SIMPLEX
org = (50, 100)
fontScale = 4
color = (255, 0, 0)
thickness = 4

data = []
data.append("")
data.append("pose")
for p in points:
    x = str(p) [13:]
    data.append(x + "_x")
    data.append(x + "_y")
    data.append(x + "_z")
    data.append(x + "_vis")
head=data;
data = pd.DataFrame(columns = data) # Empty dataset

ch=int(input("[1] read from Test set\n[2] read from Camera frame\nChoice: "))

```

```

if ch==1:
    for folder in os.listdir(test):
        img=cv2.imread(test+folder+"/"+os.listdir(test+folder)[np.random.randint(len(os.listdir(test+folder)))])
        temp = []
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        imageWidth, imageHeight = img.shape[:2]
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        blackie = np.zeros(img.shape) # Blank image
        results = pose.process(imgRGB)
        if results.pose_landmarks:
            # mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS) #draw landmarks on image
            mpDraw.draw_landmarks(blackie, results.pose_landmarks, mpPose.POSE_CONNECTIONS) # draw landmarks on blackie
            landmarks = results.pose_landmarks.landmark
            temp=temp+[0, folder]
            for i,j in zip(points,landmarks):
                temp = temp + [j.x, j.y, j.z, j.visibility]
            data.loc[0] = temp
            data.loc[1] = data.loc[0]
            x=data.drop(axis=0, columns=["pose"])
            y=data.pose
            _,t,y_tr,y_te=train_test_split(x,y,
                                           shuffle=True,
                                           test_size=0.5,
                                           random_state=1)

            string=get_prediction( t,xgb)
            blackie=cv2.putText(blackie, string[0], org, font,fontScale, color, thickness, cv2.LINE_AA)
            fig=plt.figure(figsize=(18, 5), dpi=80)
            fig.add_subplot(1,2,1)
            plt.imshow(img)

            fig.add_subplot(1,2,2)
            plt.imshow(blackie)
            plt.show()

```

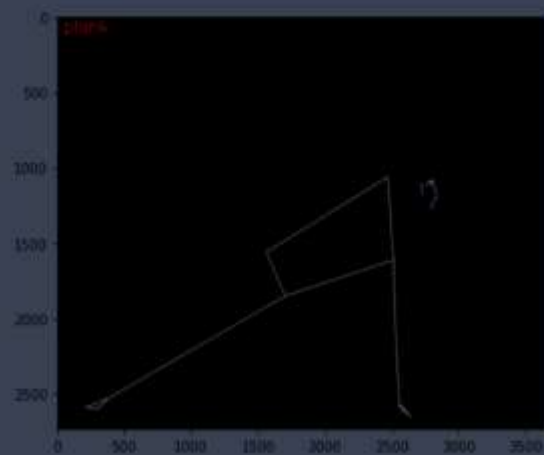

Output Images:

```
[1] read from Test set
[2] read from Camera frame
Choice: 1
```

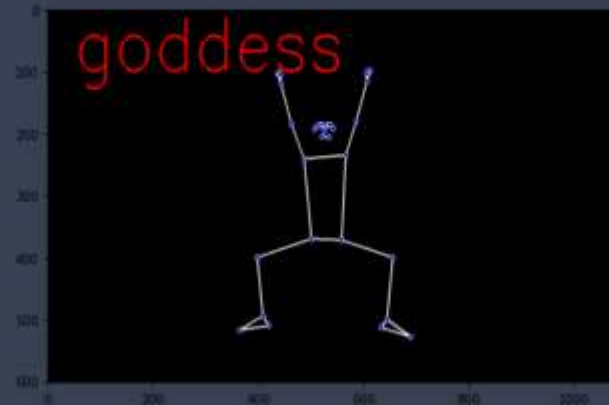
```
C:\Users\Shivanshu\AppData\Roaming\Python\Python39\site-packages\xgboost\data.py:262: FutureWarning: pandas.Int64Index is deprecated and
will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
    elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



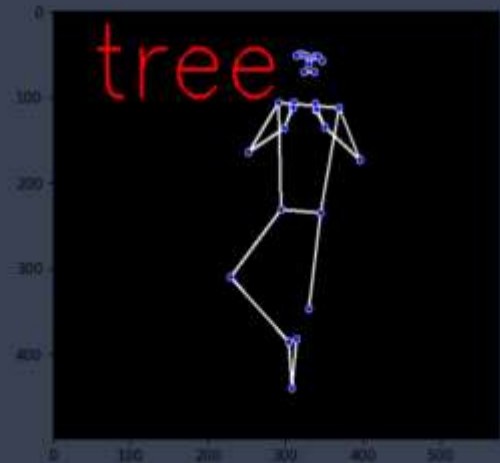
```
C:\Users\Shivanshu\AppData\Roaming\Python\Python39\site-packages\xgboost\data.py:262: FutureWarning: pandas.Int64Index is deprecated and
will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
    elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



```
C:\Users\Shivanshu\AppData\Roaming\Python\Python39\site-packages\xgboost\data.py:262: FutureWarning: pandas.Int64Index is deprecated and
will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



```
C:\Users\Shivanshu\AppData\Roaming\Python\Python39\site-packages\xgboost\data.py:262: FutureWarning: pandas.Int64Index is deprecated and
will be removed from pandas in a future version. Use pandas.Index with the appropriate dtype instead.
elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



If user selects the read from the camera option, then we'll:

- initialize a cv2.VideoCapture(0) object
- start an infinite loop
- read camera frame inside loop
- pass the camera frame to media pipe model for predicting landmarks of body parts and store them into a data frame using pandas' library
- pass that data frame to our XGBClassifier().predict() method and get predicted pose from the classifier model.
- Put that classification text to a cv2 images and display it.
- Repeat this step until user presses "q" button to break the loop and destroy all windows.

Code Images:

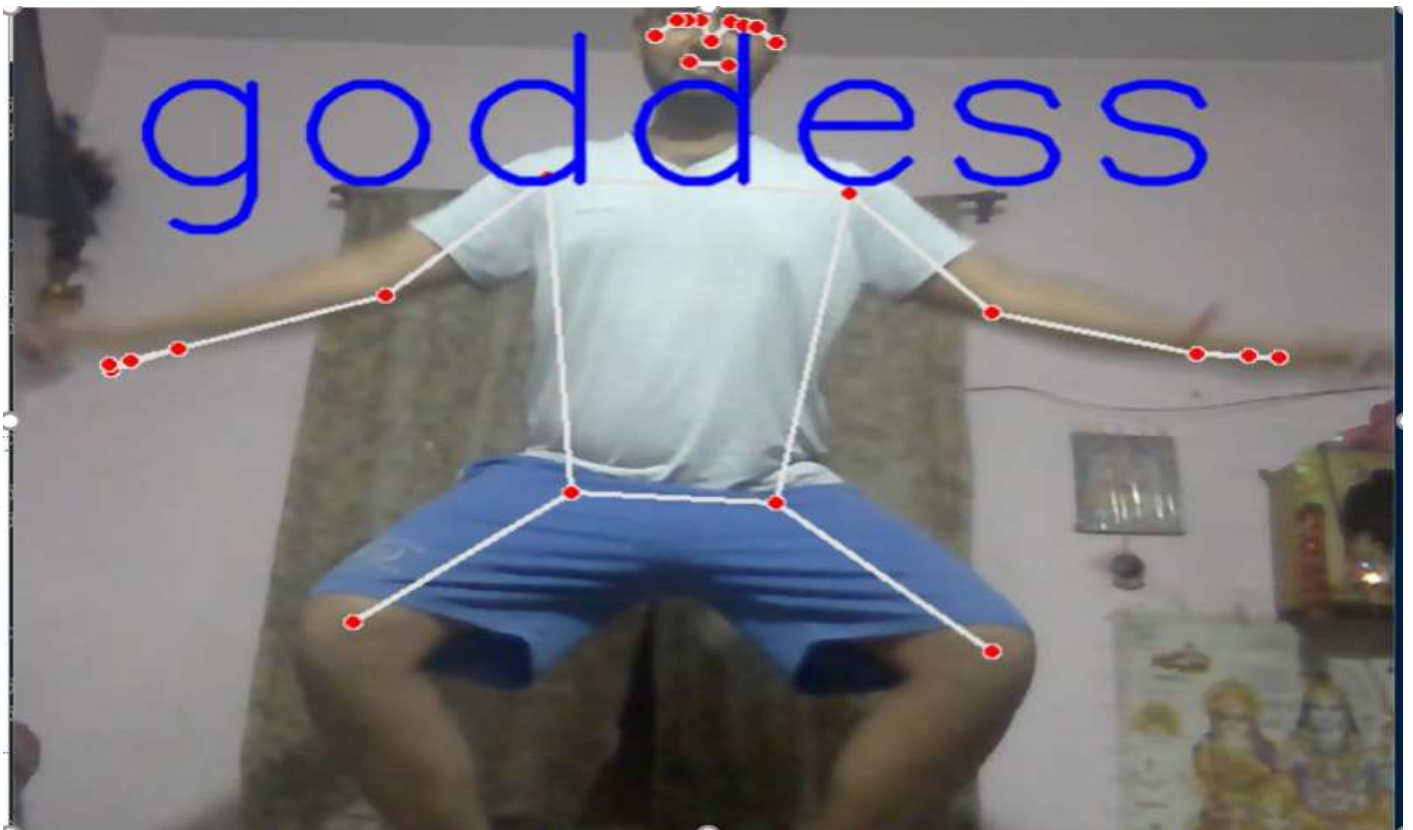
```
elif ch==2:
    cap=cv2.VideoCapture(0)
    while True:
        temp = []
        ret, img = cap.read()
        img=cv2.flip(img, 1,0)
        results = pose.process(img)
        if results.pose_landmarks:
            mpDraw.draw_landmarks(img, results.pose_landmarks, mpPose.POSE_CONNECTIONS) #draw landmarks on image
            #mpDraw.draw_landmarks(blackie, results.pose_landmarks, mpPose.POSE_CONNECTIONS) # draw landmarks on blackie
            landmarks = results.pose_landmarks.landmark
            temp=temp+[0,"NULL"]
            for i,j in zip(points,landmarks):
                temp = temp + [j.x, j.y, j.z, j.visibility]
            data.loc[0] = temp
            data.loc[1] = data.loc[0]
```

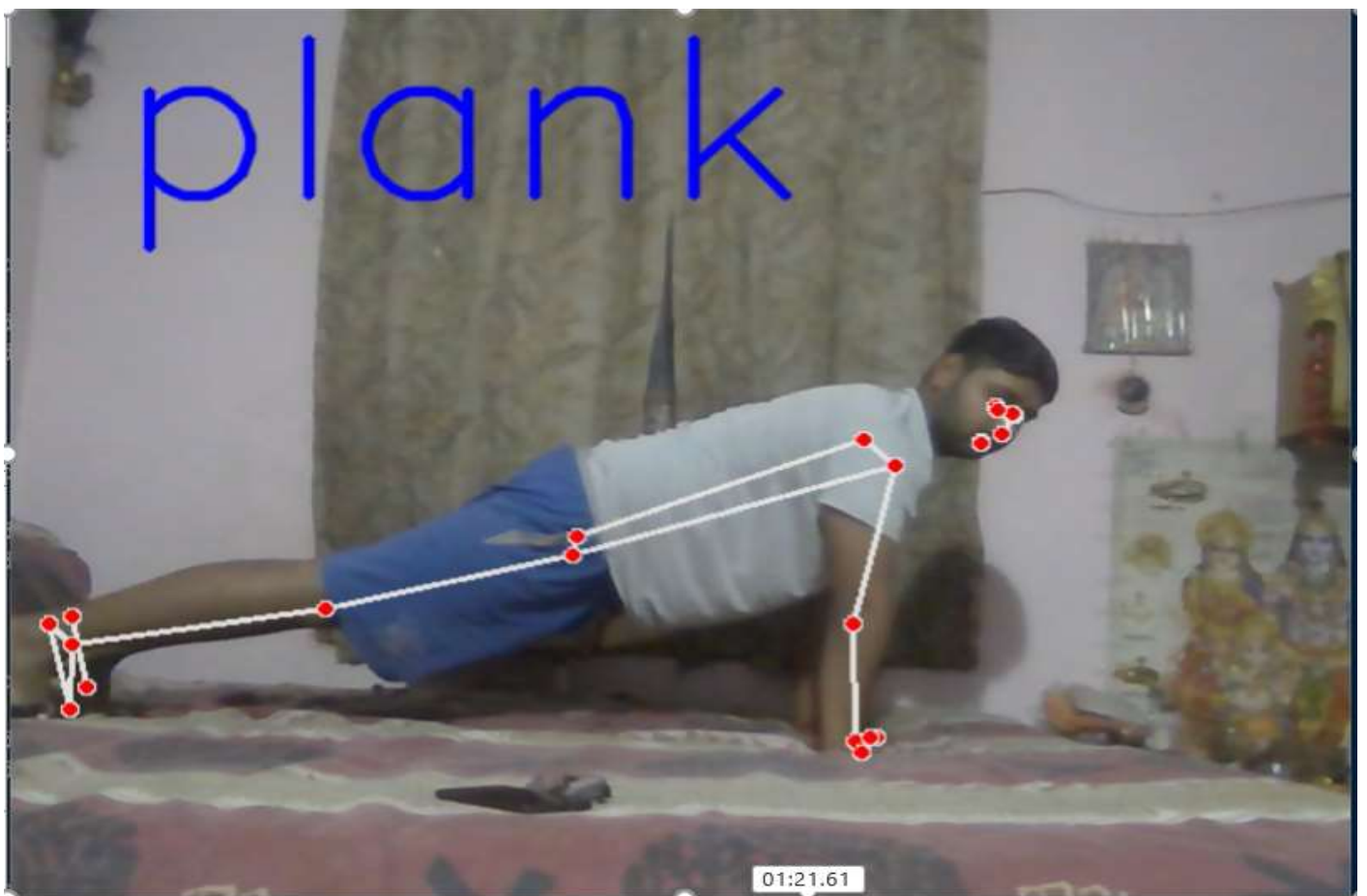
```
x=data.drop(axis=0, columns=['pose'])
y=data.pose
_,t,y_tr,y_te=train_test_split(x,y,
                                shuffle=True,
                                test_size=0.5,
                                random_state=1)

string=get_prediction(_t,xgb)
img=cv2.putText(img, string[0], org, font,fontScale, color, thickness, cv2.LINE_AA)
cv2.imshow("abc",img)
if cv2.waitKey(10) == ord('q'): # wait a bit, and see keyboard press
    break                      # if q pressed, quit

# release things before quitting
cap.release()
cv2.destroyAllWindows()
```

Output will look like this:





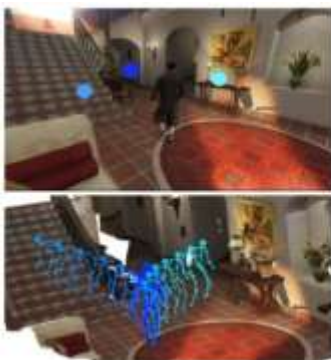
Application

Human pose estimation has been used in a wide range of applications, including human-computer interaction, motion analysis, augmented reality, and robotics.

Pose estimation has applications in lots of fields, some of which are listed below.

Most Popular Pose Estimation Applications

- Application #1: Human Activity Estimation
- Application #2: Motion Transfer and Augmented Reality
- Application #3: Training Robots
- Application #4: Motion Tracking for Consoles
- Application #5: Human Fall Detection



Action prediction



Surveillance



Cloth Parsing



Online Coaching



Movie and Game



AR and VR



Healthcare

Human Activity Estimation

A rather obvious application of pose estimation is tracking and measuring human activity and movement. Architectures like Dense Pose, Pose Net, or Open Pose are often used for activity, gesture, or gait recognition. Examples of human activity tracking via the use of pose estimation include:

Application for detecting sitting gestures

- Full body/sign language communication (for example, traffic police officer's signals)
- Applications to detect if a person has fallen or is sick
- Applications to support the analysis of football, basketball, and sports
- Applications to analyse dance techniques (for example, in ballet dances)
- Application of posture learning for body works and fitnesses
- Applications in security and surveillance enhancement

Augmented Reality and Virtual Reality

As of today, pose estimation worked with augmented and virtual reality applications gives users a better online experience. For instance, users can virtually learn how to play games like tennis via virtual tutors who are pose represented.

More so, pose estimators can also be worked with augmented reality-based applications. For example, The United States Army experiments with augmented reality programs to be used in combat. These programs aim to help soldiers distinguish between enemies and friendly troops, as well as improve night vision.

Training Robots with Human Pose Tracking

Typical use cases of pose estimators are in the application of making robots learn certain crafts. In place of manually programming robots to follow trajectories, robots can be made to learn actions and movements by following the tutor's posture look or appearance.

Human Motion Tracking for Consoles

Other applications of pose estimation are in-game applications, where human subjects auto-generate and inject poses into the game environment for an interactive gaming experience. For instance, Microsoft's Kinect used 3D pose estimation (using IR sensor data) to track the motion of the human players and to use it to give the actions of the characters virtually into the gaming environment.

Outlook and Future Trends

Pose estimation for objects is a major trend in computer vision. Object pose estimation allows gaining a more detailed understanding of objects compared to two-dimensional bounding boxes. Until now, pose estimation is still computationally very intensive and requires expensive AI hardware (often multiple NVIDIA GPUs) that is not practical for real-world use.

Edge AI technology

Modern technologies and methods make it possible to decrease the size of AI models, making pose estimation algorithms less “heavy” and much more efficient. This is the basis for the real-world implementation of human pose detection.

As a result, it becomes possible to deploy pose estimation algorithms to edge devices and perform on-device machine learning (Edge AI). Edge Inference makes the technology scalable, more robust for mission-critical applications (offline capability), and private (no visuals need to be sent to the cloud). An example of a lightweight pose estimation model for Edge ML is Lightweight Open Pose.

Conclusion

Pose detection is an active area of research in the field of machine learning and offers several real-life applications. In this project report, we tried to work on one such application and get our hands dirty with pose detection. We learned about pose detection and several models that can be used for posing detection. We selected the blaze pose model for our purpose and learned about its pros and cons over other models. In the end, we built a classifier to classify yoga poses using the support vector classifier from the sklearn library. We also built our own dataset for this purpose which could further be extended easily using more images.

You can try other machine learning algorithms instead of SVM too and compare the results accordingly.

Thank you.

References

- Blaze Pose: On-device Real-time Body Pose tracking
 - <https://arxiv.org/pdf/2006.10204.pdf>
- MediaPipe
 - <https://google.github.io/mediapipe/solutions/pose.html>
 - <https://developers.googleblog.com/search/label/MediaPipe>
- Human Pose Detection
 - <https://www.v7labs.com/blog/human-pose-estimation-guide>
 - <https://www.analyticsvidhya.com/blog/2021/10/human-pose-estimation-using-machine-learning-in-python/>
 - <https://towardsdatascience.com/human-pose-estimation-simplified-6cfd88542ab3>
- Sklearn
 - <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>
 - <https://towardsdatascience.com/getting-started-with-xgboost-in-scikit-learn-f69f5f470a97>
 - https://xgboost.readthedocs.io/en/stable/python/python_api.html
 - <https://www.kaggle.com/code/stuarthallows/using-xgboost-with-scikit-learn/notebook>
- OpenCV
 - <https://www.geeksforgeeks.org/opencv-python-tutorial/>
 - https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
- XGBoost
 - <https://www.datacamp.com/community/tutorials/xgboost-in-python>
 - <https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/>
- Comparing Datasets
 - <https://medium.com/analytics-vidhya/human-pose-comparison-and-action-scoring-using-deep-learning-opencv-python-c2bdf0ddecba>
- Open Pose
 - <https://www.geeksforgeeks.org/openpose-human-pose-estimation-method/>
 - <https://analyticsindiamag.com/guide-to-openpose-for-real-time-human-pose-estimation/>
 - <https://github.com/CMU-Perceptual-Computing-Lab/openpose>