ANURAG UNIVERSITY | SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION TECHNOLOGY

III B.TECH I SEMESTER

MACHINE LEARNING AND IT'S APPLICATIONS PROJECT

TITLE: HUMAN POSE ESTIMATION USING MACHINE LEARNING

TEAM MEMBERS:

P A SAI VAMSHI(21EG112A35)

D SUBRAMANYAM((21EG112A09)

BHANU PRAKASH ((21EG112A66)

SHARATH GOUD((21EG112A05)

SAI TEJA((21EG112A53)

CONTENTS

PROBLEM STATEMENT

- To Estimate the human pose from any video or image source provided using machine learning.
- Human pose estimation using machine learning refers to the process of training a computer system to automatically estimate the pose within images or videos.
- The primary problem revolves around achieving accurate and robust human pose estimation in diverse, uncontrolled environments.

APPROACH

Algorithm	**Convolutional Neural Networks (CNNs)**
Programming Language	Python
Tools And Libraries	VSCODE(IDE), OpenCV, matplotlib , seaborn , os
Dataset	COCO data set for images and videos

ABSTRACT

This presentation is about using smart computer programs to find and Estimate the poses of the human in pictures and videos. We'll combine two powerful tools: Machine Learning, which helps computers learn
from examples, and OpenCV, a library with special tools for seeing and understanding images. We'll learn how to handle tricky situations like the human is hidden backgrounds and human may move to quickly and etc. We'll also see how to use pre-made models and teach the computer to recognize specific pose. With these techniques, we can make cool applications like security systems, robots, and machines that can see and understand the world around them. By the end, you'll know how to use these tools to solve real-life problems with computer vision.

ALGORITHM

Convolutional Neural Networks (CNNs) are a class of deep learning models designed for processing structured grid data, such as images and video frames. They have revolutionized computer vision tasks, achieving state-of-the-art results in image classification, object detection, segmentation, and more. In this detailed overview, we'll explore the fundamental concepts behind CNNs and provide an example of how they work.

The workflow of the algorithm follows these steps:-

- **Convolutional Layers:** A series of convolutional layers follow the input layer. Each layer applies convolution operations to the input using multiple filters, producing feature maps. For example, the first convolutional layer might have 32 filters, and the second layer might have 64 filters.
- **Pooling Layers:** After each convolutional layer, a pooling layer reduces the spatial dimensions of the feature maps. Common choices include Max Pooling, which retains the maximum value in each pool, and Average Pooling, which takes the average.
- **Fully Connected Layers:** The final feature maps are flattened into a vector and fed into one or more fully connected layers. These layers perform classification tasks, mapping the features to output classes. A soft max activation function is commonly used for multi-class classification.
- **Input Layer:** The network takes input images of a fixed size, typically represented as pixel values (e.g., 224x224x3 for a color image with three color channels).
- **Output Layer:** The output layer provides the final classification probabilities for each class in the dataset. The class with the highest probability is the predicted class.

TOOLS AND LIBRARIES USED

LIBRARY:

OpenCV is an open-source computer vision library that provides tools for image processing, object detection, and more. It's widely used for tasks like motion tracking.
Matplotlib is a Python plotting library that allows easy creation of charts and graphs. It's often used in combination with OpenCV to visualize and analyze computer vision results.

TOOLS:

Visual Studio Code (VS Code) is a lightweight and highly versatile code editor developed by Microsoft. It's known for its user-friendly interface, fast performance, and a vast collection of extensions that enhance its capabilities for various programming languages and development tasks.

SPLITTING THE DATA

The splitting demonstrates object detection using a pre-trained SSD model. To trainSSD on your dataset, prepare labelled data, split the training/validation sets using scikit-learn's train_test_split, implement SSD model in CNN , define weight functions, and train the model. After training, use it for Human pose estimation on new images/videos.

CODE

Import Packages:

```
In [1]: import cv2 as cv
        import matplotlib.pyplot as pt
```

Gathering Data:

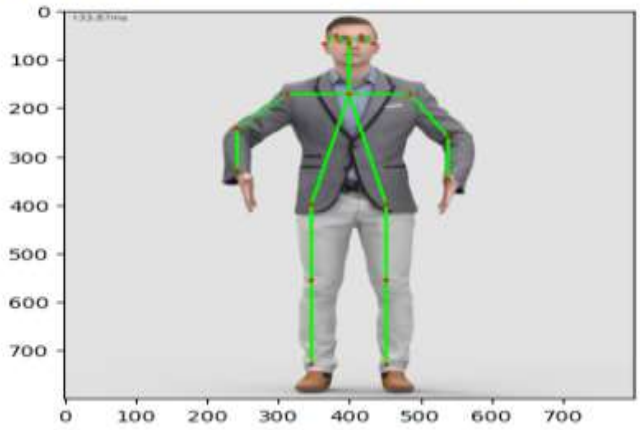
```
In [2]: n= cv.dnn.readNetFromTensorflow("graph_opt.pb")
```

```
In [3]: iw = 368
        ih= 368
        ct =0.2
```

```
In [4]: BODY_PARTS = { "Nose": 0, "Neck": 1, "RShoulder": 2, "RElbow": 3, "RWrist": 4,
                       "LShoulder": 5, "LElbow": 6, "LWrist": 7, "RHip": 8, "RKnee": 9,
                       "RAnkle": 10, "LHip": 11, "LKnee": 12, "LAnkle": 13, "REye": 14,
                       "LEye": 15, "REar": 16, "LEar": 17, "Background": 18 }

        POSE_PAIRS = [ ["Neck", "RShoulder"], ["Neck", "LShoulder"], ["RShoulder", "RElbow"],
                       ["RElbow", "RWrist"], ["LShoulder", "LElbow"], ["LElbow", "LWrist"],
                       ["Neck", "RHip"], ["RHip", "RKnee"], ["RKnee", "RAnkle"], ["Neck", "LHip"],
                       ["LHip", "LKnee"], ["LKnee", "LAnkle"], ["Neck", "Nose"], ["Nose", "REye"],
                       ["REye", "REar"], ["Nose", "LEye"], ["LEye", "LEar"] ]
```

Sample visualization

```
In [7]: ei = ps(image)
        pt.imshow(cv.cvtColor(image,cv.COLOR_BGR2RGB))
Out[7]: <matplotlib.image.AxesImage at 0x1f236867210>
```



Choose ML Model

```
In [6]: def ps(f):
            fw=f.shape[1]
            fh=f.shape[0]
            n.setInput(cv.dnn.blobFromImage(f,1.0,(iw,ih),(127.5,127.5,127.5),swapRB=True,crop=False))
            out = n.forward()
            out = out[:, :19, :, :]
            assert(len(BODY_PARTS) == out.shape[1])

            points = []
            for i in range(len(BODY_PARTS)):

                heatMap = out[0, i, :, :]

                .
                _, conf, _, point = cv.minMaxLoc(heatMap)
                x = (fw * point[0]) / out.shape[3]
                y = (fh * point[1]) / out.shape[2]

                points.append((int(x), int(y)) if conf > ct else None)

            for pair in POSE_PAIRS:
                partFrom = pair[0]
                partTo = pair[1]
                assert(partFrom in BODY_PARTS)
                assert(partTo in BODY_PARTS)

                idFrom = BODY_PARTS[partFrom]
                idTo = BODY_PARTS[partTo]

                if points[idFrom] and points[idTo]:
                    cv.line(f, points[idFrom], points[idTo], (0, 255, 0), 3)
                    cv.ellipse(f, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
                    cv.ellipse(f, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)

            t, _ = n.getPerfProfile()
            freq = cv.getTickFrequency() / 1000
            cv.putText(f, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
            return f
```

Predict Model or Result

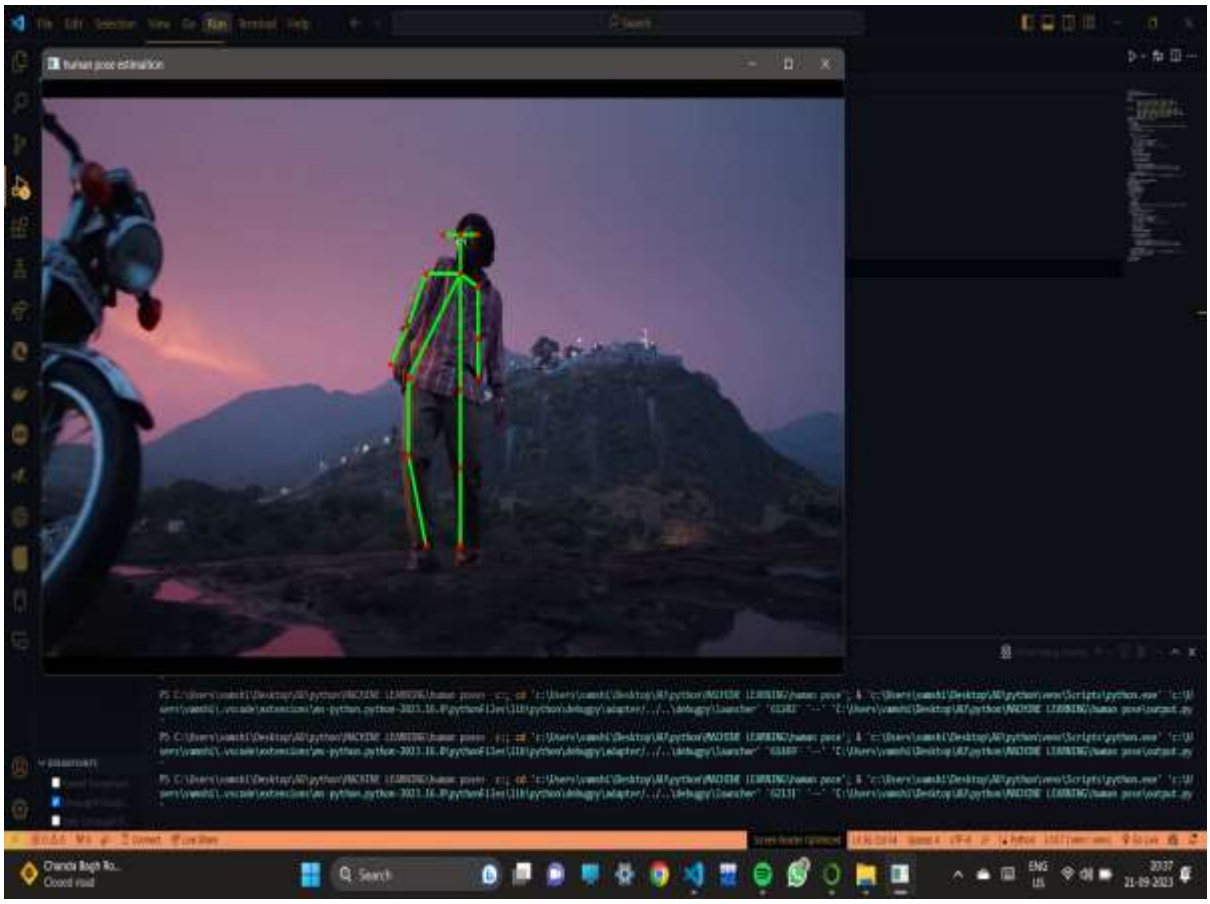
```
In [ ]: capture=cv.VideoCapture(0)
        capture.set(3,800)
        capture.set(4,800)
        if not capture.isOpened():
            cap=cv2.VideoCapture(0)
        if not capture.isOpened():
            raise IOError("hello")
        while True:
            k,f=capture.read()
            if not k:
                cv.waitKey()
                break
            fw=f.shape[1]
            fh=f.shape[0]
            n.setInput(cv.dnn.blobFromImage(f,1.0,(in_w,in_h),(127.5,127.5,127.5),swapRB=True,crop=False))
            out = n.forward()
            out = out[:, :19, :, :]
            assert(len(BODY_PARTS) == out.shape[1])

            points = []
            for i in range(len(BODY_PARTS)):
                # Slice heatmap of corresponging body's part.
                heatMap = out[0, i, :, :]

                # Originally, we try to find all the local maximums. To simplify a sample
                # we just find a global one. However only a single pose at the same time
                # could be detected this way.
                _, conf, _, point = cv.minMaxLoc(heatMap)
                x = (fw * point[0]) / out.shape[3]
                y = (fh * point[1]) / out.shape[2]
                # Add a point if it's confidence is higher than threshold.
                points.append((int(x), int(y)) if conf > ct else None)

            for pair in POSE_PAIRS:
                partFrom = pair[0]
                partTo = pair[1]
                assert(partFrom in BODY_PARTS)
                assert(partTo in BODY_PARTS)

                idFrom = BODY_PARTS[partFrom]
                idTo = BODY_PARTS[partTo]

                if points[idFrom] and points[idTo]:
```

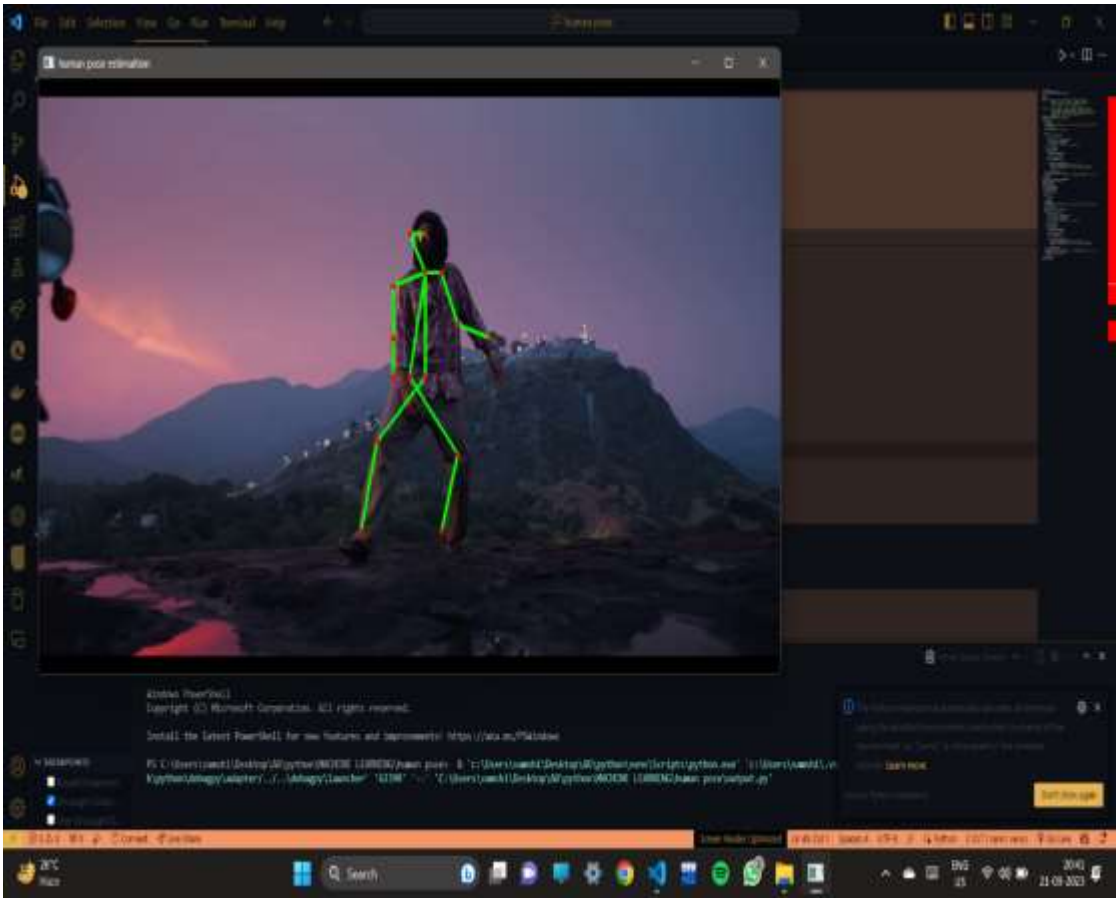
```
                # Slice heatmap of corresponging body's part.
                heatMap = out[0, i, :, :]

                # Originally, we try to find all the local maximums. To simplify a sample
                # we just find a global one. However only a single pose at the same time
                # could be detected this way.
                _, conf, _, point = cv.minMaxLoc(heatMap)
                x = (fw * point[0]) / out.shape[3]
                y = (fh * point[1]) / out.shape[2]
                # Add a point if it's confidence is higher than threshold.
                points.append((int(x), int(y)) if conf > ct else None)

            for pair in POSE_PAIRS:
                partFrom = pair[0]
                partTo = pair[1]
                assert(partFrom in BODY_PARTS)
                assert(partTo in BODY_PARTS)

                idFrom = BODY_PARTS[partFrom]
                idTo = BODY_PARTS[partTo]

                if points[idFrom] and points[idTo]:
                    cv.line(f, points[idFrom], points[idTo], (0, 255, 0), 3)
                    cv.ellipse(f, points[idFrom], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)
                    cv.ellipse(f, points[idTo], (3, 3), 0, 0, 360, (0, 0, 255), cv.FILLED)

            t, _ = n.getPerfProfile()
            freq = cv.getTickFrequency() / 1000
            cv.putText(f, '%.2fms' % (t / freq), (10, 20), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))
            cv.imshow('human pose estimation', f)
            if cv.waitKey(5) & 0xFF == ord('q'):
                break
        cv.destroyAllWindows()
        capture.release()
```

OUT COME

Accurate, efficient Human pose estimation using CNN, improves computer vision in surveillance, self-driving cars, and real- world applications.

OUTPUT





CONCLUSION

- In conclusion, the Human pose Estimation project using CNN showed that it could accurately and quickly estimate pose in pictures and live video and also recorded video.
- By using a diverse dataset and preparing the data carefully, we made the model work well in various situations.
- This project highlights how machine learning can help computers understand and interact with the world, making it useful in things like security cameras and self-driving cars.

REFERENCES

- https://nanonets.com/blog/human-pose-estimation-2d-guide/#deeppose