A Major Project Report

On

"POSE DETECTION USING R-CNN"

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

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

By

CH.PRABHU CHAITANYA (15H61A04D3)

Under the guidance of

Mrs. M. KUSUMA SRI
Assistant professor
Department of ECE

Department of Electronics and Communication Engineering

ANURAG GROUP OF INSTITUTIONS
(Formerly CVSR College of Engineering)

An Autonomous Institution

Permanently Affiliated to Jawaharlal Nehru Technological University,
Hyderabad Venkatapur (v) Ghatkesar (M) Medchal Dist-500088

2019

ANURAG GROUP OF INSTITUTIONS

(An Autonomous Institution)

Venkatapur (V) Ghatkesar (M) Medchal (Dist.), Telangana-500088.

Department of Electronics & Communication Engineering

CERTIFICATE

This is to certify that the work embodies in this dissertation entitled **'POSE DETECTION USING R-CNN)'** being submitted by

CH.PRABHU CHAITANYA (15H61A04D3)

for partial fulfilment of the requirement for the award of Bachelor of Technology in Electronics & Communication Engineering to Anurag Group of Institutions, Venkatapur (V), Ghatkesar (M), Medchal (Dist.), Telangana State. During the academic year 2018 – 19 is a record of bonafide piece of work, undertaken by him/her the supervision of the undersigned.

INTERNAL GUIDE	**HOD**
Mrs. M. KUSUMA SRI	**DR. S. SATHEESKUMARAN**
Assistant Professor	HOD,Professor
Dept. of ECE	Dept. of ECE
Anurag group of Institutions	Anurag group of Institutions

EXTERNAL EXAMINNER

ANURAG GROUP OF INSTITUTIONS

(An Autonomous Institution)

Venkatapur(V) Ghatkesar (M) Medchal (Dist.), Telangana 500088.

Department of Electronics & Communication Engineering

DECLARATION

I, **CH.PRABHU CHAITANYA(15H61A04D3**) the student of 'Bachelor of Technology in Electronics & Communication Engineering', pertaining to 2015 – 19 batch, Anurag Group of Institutions, Venkatapur (V), Ghatkesar (M), Medchal (Dist.), Telangana State, hereby declare that the work presented in this Major Project Work entitled '**POSE DETECTION USING R-CNN**' is the outcome of my own bonafide work and is correct to the best of my knowledge and this work has been undertaken by taking care of Engineering Ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

CH.PRABHU CHAITANYA - 15H61A04D8

DATE:

ACKNOWLEDGEMENT

This project is an acknowledgement to the inspiration, drive and technical assistance contributed by many individuals. This project would have never seen light of this day without the help and guidance we have received. We would like to express our gratitude to all the people behind the screen who helped us to transform an idea into a real application.

It’s our privilege and pleasure to express our profound sense of gratitude to **Mrs. M. KUSUMA SRI,** Assistant Professor Department of ECE for his guidance throughout this dissertation work.

We express our sincere gratitude to **Dr. S. SATHEES KUMARAN,** Head of the Department of Electronics and communication Engineering for his precious suggestions for the successful completion of this project. He is also a great source of inspiration to our work.

We would like to express my deep sense of gratitude to **Dr. K. S. RAO**, Director of Anurag group of Institutions for his tremendous support, encouragement and inspiration.

Lastly, we thank almighty, our parents, friends for their constant encouragement without which this assignment would not be possible. We would like to thank all the other staff members, teaching and non- teaching, which have extended their timely help and eased our work.

By

CH.PRABHU CHAITANYA **(15H61A04D3)**

ABSTRACT

Human Movement detection is vital in Tele-presence Robots, Animations, Games and Robotic movements. By using Traditional methods with the help of sensor suits it is difficult to find and interpret the movements. As it includes so much sensor data which is difficult to interpret, find the action and send to long distances.
It is also very expensive and bulky too.

Image processing and computer vision provides a solution to detect and interpret Human movement based on R-CNN approach. It is cheap , easy and light weight algorithm. It takes the video input and divides it in to frames ,then it is Human body is separated for the background image.

This work is focused on skeleton and its major points and its relative positions in successive picture frames. A set of frames (Video) is given as a input to the model, so that the model compares the coordinates of the successive frames and estimates the movement.

These dynamic objects are then processed. First, the human is identified and separated from the rest of the image by drawing a bounding box around the human by using CNN(Convolution neural networks) , then by applying R-CNN human is segmented and converted to skeleton. . From this shape it is inferred whether the skeleton is that of a human or not. Comparing the relative coordinates of skeletons so extracted from frames photographed over time gives the movement of the human and its direction.

CONTENTS

	Pg No
CHAPTER 1: INTRODUCTION	
1.1 PURPOSE OF IMPLEMENTATION	1
1.2 COMPUTER VISION	3
1.2.1 COMPUTER VISION APPLICATIONS	4
1.2.2 COMPUTER VISION TYPICAL TASKS	6
1.2.3 RECOGNITION	7
1.3 NEURAL NETWORKS	10
1.3.1 NEURAL NETWORK ELEMENTS	12
1.3.2 KEY CONCEPTS OF DEEP NEURAL NETWORKS	13
CHAPTER 2: DESIGN ASPECTS AND APPROACH	
2.1 BLOCK DIAGRAM	28
2.2 INTERNAL BLOCK DIAGRAM	29
2.3 FLOW DIAGRAM	30
2.3.1 PADDING	31
2.3.2 POOLING	31
CHAPTER 3: SOFTWARE AND HARDWARE REQUIREMENTS	37
3.1 SOFTWARE AND HARDWARE REQUIREMENTS	37
3.1.1 PYTHON	38
3.1.2 TENSORFLOW	41
3.1.3 ANACONDA	43
CHAPTER 4: TESTING	
4.1 TESTING	46
4.2 RUNTUME ANALYSIS	46
CHAPTER 5: RESULTS & DISCUSSIONS	47
CHAPTER 6: CONCLUSION	54
CHAPTER 7: FUTURE SCOPE	55

CHAPTER 8: REFERENCES 56

LIST OF FIGURE

Sl. No	FIGURE NAME	Page No.
1.3.1	Diagram of a node	12
1.3.2	Layers of a neural network	13
1.3.3	Underlying factors and concepts	14
2.1	Block Diagram of machine learning framework	28
2.2	Internal Block Diagram	29
2.3	Algorithmic flowchart of R-CNN	30
2.3	Convolution operation	31
2.3.1	Padding operation	31
2.3.2	After pooling layer	32
2.3.3	Flow chart of identifying human	33
2.3.4	Inception Res-net flow	33
2.3.5	Pooling replaced by residual net	34
2.3.6	Relu activation	35
2.3.7	Architecture of R-CNN	36
5.1	Input image	47
5.1.1	Skeletonised human of input frame	47
5.2	Segmented and skeletonised plotted image	47
5.3	Vector plots	48
5.4	Coordinates of human	49
5.5	Input image(Multi human)	50
5.5.1	Segmented and skeletonised plotted image	50
5.5.2	Individually 3D plotted human skeletons	51
5.5.3	Coordinates of human	52
5.4	Vector plots of human	53

LIST OF TABLES

SL.N	TABLE NAME	PG.NO
3.1	Software requirements	37
3.2	Hardware requirements	38

CHAPTER 1

INTRODUCTION

1.1 PURPOSE OF IMPLEMENTATION

Human Movement detection is vital in Tele-presence Robots, Animations, Games and Robotic movements . By using Traditional methods with the help of sensor suits it is difficult to find and interpret the movements. As it includes so much sensor data which is difficult to interpret, find the action and send to long distances.

The experiments have been conducted under different illumination levels, at different times of day and at different locations. Promising results were obtained in all cases. This algorithm is capable of tracking human beings in different environments, under different lighting conditions. It is deformity tolerant to a significant extent in the sense that bending or twisting does not affect its ability.

This project is mainly useful for robo vision and robo walking dynamics.

This work is focused on skeleton and its major points and its relative positions in successive picture frames. A static camera takes continuous pictures from a fixed angle with a static background in a controlled environment. Subtracting this background from successive images generates what can be called as the foreground. It consists of only dynamic objects coming into view.

These dynamic objects are then processed. First, the human is identified and separated from the rest of the image by drawing a bounding box around the human by using CNN(Convolution neural networks) , then by applying R-CNN human is segmented and converted to skeleton. . From this shape it is inferred whether the skeleton is that of a human or not. Comparing the relative coordinates of skeletons so extracted from frames photographed over time gives the movement of the human and its direction.

PROBLEM

- Human Movement detection is vital in Telepresence Robots, Animations, Games and Robotic movements.
- By using Traditional methods with the help of sensor suits it is difficult to find and interpret the movements. As it includes so much sensor data which is difficult to interpret, find the action and send to long distances.
- It is also very expensive and bulky too.

SOLUTION

- Image processing provides a solution to detect and interpret Human movement based on skeleton algorithm based approach.
- It is cheap , easy and light weight algorithm.
- It takes the video input and divides it in to frames ,then it is Human body is separated for the background image .
- To this Background image skeleton algorithm is applied and body part movements are identified.

APPLICATIONS

- 1. Used for surveillance: It analyses CC cam feed and predicts the action of man in the image and alerts.
- 2. Used in Gaming to create Animations that move with our actions.
- 3. Used in Tele presence Robots: To perform actions in surgeries, Demonstrations etc.., remotely from distant places.
- 4. Used to train AI to study body movements ,so it can more quickly learn to walk like Human.

1.2 COMPUTER VISION

Computer vision is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, and image restoration.

- Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do. "Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual understanding." As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

1.2.1 COMPUTER VISION APPLICATIONS

Applications range from tasks such as industrial machine vision systems which, say, inspect bottles speeding by on a production line, to research into artificial intelligence and computers or robots that can comprehend the world around them. The computer vision and machine vision fields have significant overlap. Computer vision covers the core technology of automated image analysis which is used in many fields. Machine vision usually refers to a process of combining automated image analysis with other methods and technologies to provide automated inspection and robot guidance in industrial applications. In many computer-vision applications, the computers are pre-programmed to solve a particular task, but methods based on learning are now becoming increasingly common. Examples of applications of computer vision include systems for:

Learning 3D shapes has been a challenging task in computer vision. Recent advances in deep learning has enabled researchers to build models that are able to generate and reconstruct 3D shapes from single or multi-view depth maps or silhouettes seamlessly and efficiently [20]

- Automatic inspection, *e.g.*, in manufacturing applications;
- Assisting humans in identification tasks, e.g., a species identification system;[23]
- Controlling processes, *e.g.*, an industrial robot;
- Detecting events, *e.g.*, for visual surveillance or people counting;
- Interaction, *e.g.*, as the input to a device for computer-human interaction;

- Modeling objects or environments, *e.g.*, medical image analysis or topographical modeling;
- Navigation, *e.g.*, by an autonomous vehicle or mobile robot; and
- Organizing information, *e.g.*, for indexing databases of images and image sequences.

One of the most prominent application fields is medical computer vision, or medical image processing, characterized by the extraction of information from image data to diagnose a patient. An example of this is detection of tumours, arteriosclerosis or other malign changes; measurements of organ dimensions, blood flow, etc. are another example. It also supports medical research by providing new information: *e.g.*, about the structure of the brain, or about the quality of medical treatments. Applications of computer vision in the medical area also includes enhancement of images interpreted by humans—ultrasonic images or X-ray images for example—to reduce the influence of noise.

A second application area in computer vision is in industry, sometimes called machine vision, where information is extracted for the purpose of supporting a manufacturing process. One example is quality control where details or final products are being automatically inspected in order to find defects. Another example is measurement of position and orientation of details to be picked up by a robot arm. Machine vision is also heavily used in agricultural process to remove undesirable food stuff from bulk material, a process called optical sorting.[24]

Military applications are probably one of the largest areas for computer vision. The obvious examples are detection of enemy soldiers or vehicles and missile guidance. More advanced systems for missile guidance send the missile to an area rather than a specific target, and target selection is made when the missile reaches the area based on locally acquired image data. Modern military concepts, such as "battlefield awareness", imply that various sensors, including image sensors, provide a rich set of information about a combat scene which can be used to support strategic decisions. In this case, automatic processing of the data is used to reduce complexity and to fuse information from multiple sensors to increase reliability.

One of the newer application areas is autonomous vehicles, which include submersibles, land-based vehicles (small robots with wheels, cars or trucks), aerial vehicles, and unmanned aerial vehicles (UAV). The level of autonomy ranges from fully autonomous (unmanned) vehicles to vehicles where computer-vision-based systems support a driver or a pilot in various situations. Fully autonomous vehicles typically use computer vision for navigation, *e.g.* for knowing where it is, or for producing a map of its environment (SLAM) and for detecting obstacles. It can also be used for detecting certain task specific events, *e.g.*, a UAV looking for forest fires. Examples of supporting systems are obstacle warning systems in cars, and systems for autonomous landing of aircraft. Several car manufacturers have demonstrated systems for autonomous driving of cars, but this technology has still not reached a level where it can be put on the market. There are ample examples of military autonomous vehicles ranging from advanced missiles to UAVs for recon missions or missile guidance. Space exploration is already being made with autonomous vehicles using computer vision, *e.g.*, NASA's Mars Exploration Rover and ESA's ExoMars Rover.

Other application areas include:

- Support of visual effects creation for cinema and broadcast, *e.g.*, camera tracking (matchmoving).
- Surveillance.
- Tracking and counting organisms in the biological sciences.

1.2.2 COMPUTER VISION TYPICAL TASKS

Each of the application areas described above employ a range of computer vision tasks; more or less well-defined measurement problems or processing problems, which can be solved using a variety of methods. Some examples of typical computer vision tasks are presented below.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

1.2.3 RECOGNITION

The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity.

- **Object recognition** (also called **object classification**) – one or several pre-specified or learned objects or object classes can be recognized, usually together with their 2D positions in the image or 3D poses in the scene. Blippar, Google Goggles and LikeThat provide stand-alone programs that illustrate this functionality.
- **Identification** – an individual instance of an object is recognized. Examples include identification of a specific person's face or fingerprint, identification of handwritten digits, or identification of a specific vehicle.
- **Detection** – the image data are scanned for a specific condition. Examples include detection of possible abnormal cells or tissues in medical images or detection of a vehicle in an automatic road toll system. Detection based on relatively simple and fast computations is sometimes used for finding smaller regions of interesting image data which can be further analyzed by more computationally demanding techniques to produce a correct interpretation.

Currently, the best algorithms for such tasks are based on convolutional neural networks. An illustration of their capabilities is given by the ImageNet Large Scale Visual Recognition Challenge; this is a benchmark in object classification and detection, with millions of images and hundreds of object classes. Performance of convolutional neural networks, on the ImageNet tests, is now close to that of humans. The best algorithms still struggle with objects that are small or thin, such as a small ant on a stem of a flower or a person holding a quill in their hand. They also have trouble with images that have been distorted with filters (an increasingly common phenomenon with modern digital cameras). By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues. For example, they are not good at classifying objects into fine-grained classes, such as the particular breed of dog or species of bird, whereas convolutional neural networks handle this with ease.

Several specialized tasks based on recognition exist, such as:

- **Content-based image retrieval** – finding all images in a larger set of images which have a specific content. The content can be specified in different ways, for example in terms of similarity relative a target image (give me all images similar to image X), or in terms of high-level search criteria given as text input (give me all images which contains many houses, are taken during winter, and have no cars in them).
- **Pose estimation** – estimating the position or orientation of a specific object relative to the camera. An example application for this technique would be assisting a robot arm in retrieving objects from a conveyor belt in an assembly line situation or picking parts from a bin.
- **Optical character recognition** (OCR) – identifying characters in images of printed or handwritten text, usually with a view to encoding the text in a format more amenable to editing or indexing (*e.g.* ASCII).
- **2D code reading** – reading of 2D codes such as data matrix and QR codes.
- **Facial recognition**
- **Shape Recognition Technology** (SRT) in people counter systems differentiating human beings (head and shoulder patterns) from objects
- Motion analysis

Several tasks relate to motion estimation where an image sequence is processed to produce an estimate of the velocity either at each points in the image or in the 3D scene, or even of the camera that produces the images . Examples of such tasks are:

- **Egomotion** – determining the 3D rigid motion (rotation and translation) of the camera from an image sequence produced by the camera.
- **Tracking** – following the movements of a (usually) smaller set of interest points or objects (*e.g.*, vehicles, humans or other organisms) in the image sequence.
- **Optical flow** – to determine, for each point in the image, how that point is moving relative to the image plane, *i.e.*, its apparent motion. This motion is a result both of how the corresponding 3D point is moving in the scene and how the camera is moving relative to the scene.

SCENE RECONSTRUCTION

Given one or (typically) more images of a scene, or a video, scene reconstruction aims at computing a 3D model of the scene. In the simplest case the model can be a set of 3D points. More sophisticated methods produce a complete 3D surface model. The advent of 3D imaging not requiring motion or scanning, and related processing algorithms is enabling rapid advances in this field. Grid-based 3D sensing can be used to acquire 3D images from multiple angles. Algorithms are now available to stitch multiple 3D images together into point clouds and 3D models .

IMAGE RESTORATION

The aim of image restoration is the removal of noise (sensor noise, motion blur, etc.) from images. The simplest possible approach for noise removal is various types of filters such as low-pass filters or median filters. More sophisticated methods assume a model of how the local image structures look, to distinguish them from noise. By first analyzing the image data in terms of the local image structures, such as lines or edges, and then controlling the filtering based on local information from the analysis step, a better level of noise removal is usually obtained compared to the simpler approaches.

An example in this field is in painting.

1.3 NEURAL NETWORKS

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.

Neural networks help us cluster and classify. You can think of them as a clustering and classification layer on top of the data you store and manage. They help to group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. (Neural networks can also extract features that are fed to other algorithms for clustering and classification; so you can think of deep neural networks as components of larger machine-learning applications involving algorithms for reinforcement learning, classification and regression.)

CLASSIFICATION

All classification tasks depend upon labeled datasets; that is, humans must transfer their knowledge to the dataset in order for a neural network to learn the correlation between labels and data. This is known as *supervised learning*.

- Detect faces, identify people in images, recognize facial expressions (angry, joyful)
- Identify objects in images (stop signs, pedestrians, lane markers…)
- Recognize gestures in video
- Detect voices, identify speakers, transcribe speech to text, recognize sentiment in voices
- Classify text as spam (in emails), or fraudulent (in insurance claims); recognize sentiment in text (customer feedback)

Any labels that humans can generate, any outcomes that you care about and which correlate to data, can be used to train a neural network.

Clustering

Clustering or grouping is the detection of similarities. Deep learning does not require labels to detect similarities. Learning without labels is called *unsupervised learning*. Unlabeled data is the majority of data in the world. One law of machine learning is: the more data an algorithm can train on, the more accurate it will be. Therefore, unsupervised learning has the potential to produce highly accurate models.

- Search: Comparing documents, images or sounds to surface similar items.
- Anomaly detection: The flipside of detecting similarities is detecting anomalies, or unusual behavior. In many cases, unusual behavior correlates highly with things you want to detect and prevent, such as fraud.

PREDICTIVE ANALYTICS: REGRESSIONS

With classification, deep learning is able to establish correlations between, say, pixels in an image and the name of a person. You might call this a static prediction. By the same token, exposed to enough of the right data, deep learning is able to establish correlations between present events and future events. It can run regression between the past and the future. The future event is like the label in a sense. Deep learning doesn't necessarily care about time, or the fact that something hasn't happened yet. Given a time series, deep learning may read a string of number and predict the number most likely to occur next.

- Hardware breakdowns (data centers, manufacturing, transport)
- Health breakdowns (strokes, heart attacks based on vital stats and data from wearables)
- Customer churn (predicting the likelihood that a customer will leave, based on web activity and metadata)
- Employee turnover (ditto, but for employees)

The better we can predict, the better we can prevent and pre-empt. As you can see, with neural networks, we're moving towards a world of fewer surprises. Not zero surprises, just marginally fewer. We're also moving toward a world of smarter agents that combine neural networks with other algorithms like reinforcement learning to attain goals.

With that brief overview of deep learning use cases, let's look at what neural nets are made of.

1.3.1 NEURAL NETWORK ELEMENTS

Deep learning is the name we use for "stacked neural networks"; that is, networks composed of several layers.

The layers are made of *nodes*. A node is just a place where computation happens, loosely patterned on a neuron in the human brain, which fires when it encounters sufficient stimuli. A node combines input from the data with a set of coefficients, or weights, that either amplify or dampen that input, thereby assigning significance to inputs with regard to the task the algorithm is trying to learn; e.g. which input is most helpful is classifying data without error? These input-weight products are summed and then the sum is passed through a node's so-called activation function, to determine whether and to what extent that signal should progress further through the network to affect the ultimate outcome, say, an act of classification. If the signals passes through, the neuron has been "activated."

Here's a diagram of what one node might look like.

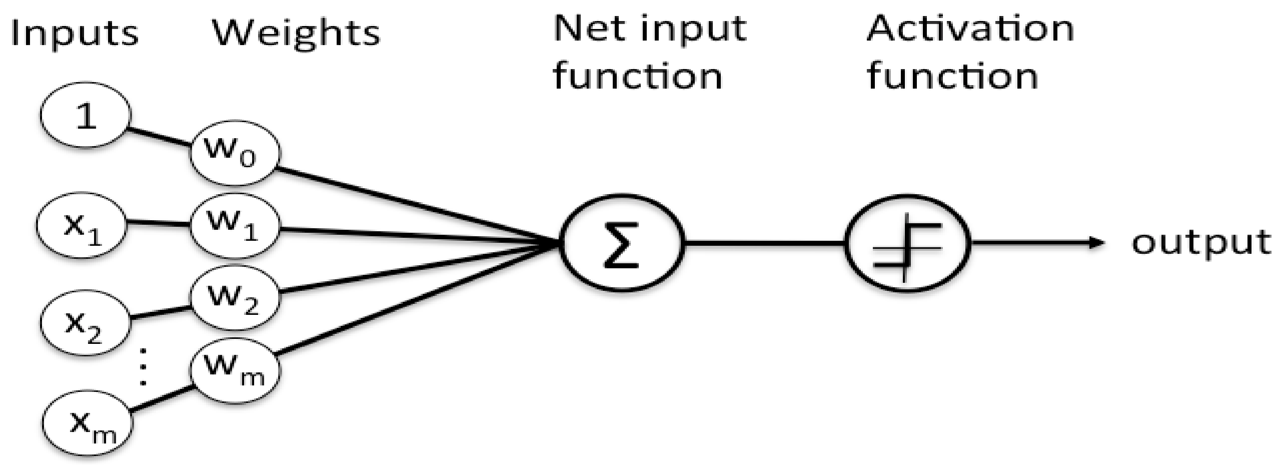


Fig 1.3.1 Diagram of a node

A node layer is a row of those neuron-like switches that turn on or off as the input is fed through the net. Each layer's output is simultaneously the subsequent layer's input, starting from an initial input layer receiving your data.

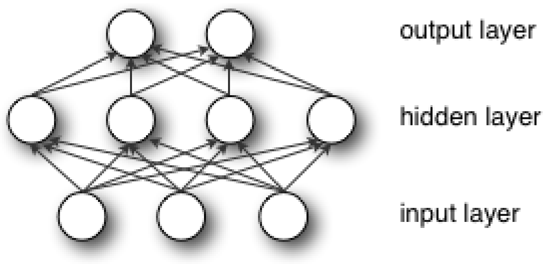


Fig 1.3.2 Layers of a neural network

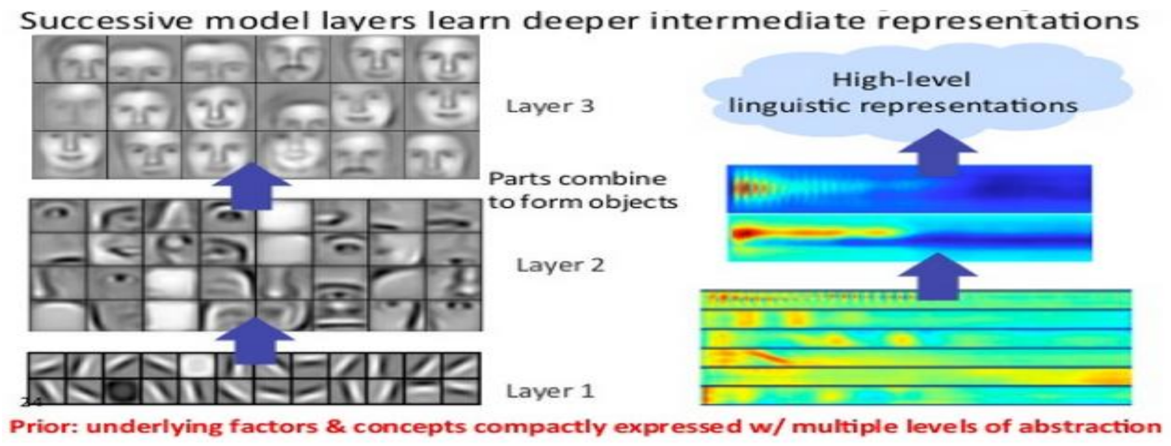
Pairing the model's adjustable weights with input features is how we assign significance to those features with regard to how the neural network classifies and clusters input.

1.3.2 KEY CONCEPTS OF DEEP NEURAL NETWORKS

Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their **depth**; that is, the number of node layers through which data must pass in a multistep process of pattern recognition.

Earlier versions of neural networks such as the first perceptrons were shallow, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as "deep" learning. So *deep* is not just a buzzword to make algorithms seem like they read Sartre and listen to bands you haven't heard of yet. It is a strictly defined term that means more than one hidden layer.

In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer's output. The further you advance into the neural net, the more complex the features your nodes can recognize, since they aggregate and recombine features from the previous layer.


Successive model layers learn deeper intermediate representations
Layer 3
Parts combine to form objects
Layer 2
Layer 1
High-level linguistic representations
Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction

This is known as **feature hierarchy**, and it is a hierarchy of increasing complexity and abstraction. It makes deep-learning networks capable of handling very large, high-dimensional data sets with billions of parameters that pass through nonlinear functions.

Above all, these neural nets are capable of discovering latent structures within **unlabeled, unstructured data**, which is the vast majority of data in the world. Another word for unstructured data is *raw media*; i.e. pictures, texts, video and audio recordings. Therefore, one of the problems deep learning solves best is in processing and clustering the world's raw, unlabeled media, discerning similarities and anomalies in data that no human has organized in a relational database or ever put a name to.

For example, deep learning can take a million images, and cluster them according to their similarities: cats in one corner, ice breakers in another, and in a third all the photos of your grandmother. This is the basis of so-called smart photo albums.

Now apply that same idea to other data types: Deep learning might cluster raw text such as emails or news articles. Emails full of angry complaints might cluster in one corner of the vector space, while satisfied customers, or spambot messages, might cluster in others. This is the basis of various messaging filters, and can be used in customer-relationship management (CRM). The same applies to voice messages.

With time series, data might cluster around normal/healthy behavior and anomalous/dangerous behavior. If the time series data is being generated by a smart phone, it will provide insight into users' health and habits; if it is being generated by an autopart, it might be used to prevent catastrophic breakdowns.

Deep-learning networks perform **automatic feature extraction** without human intervention, unlike most traditional machine-learning algorithms. Given that feature extraction is a task that can take teams of data scientists years to accomplish, deep learning is a way to circumvent the chokepoint of limited experts. It augments the powers of small data science teams, which by their nature do not scale.

When training on unlabeled data, each node layer in a deep network learns features automatically by repeatedly trying to reconstruct the input from which it draws its samples, attempting to minimize the difference between the network's guesses and the probability distribution of the input data itself. Restricted Boltzmann machines, for examples, create so-called reconstructions in this manner.

In the process, these neural networks learn to recognize correlations between certain relevant features and optimal results – they draw connections between feature signals and what those features represent, whether it be a full reconstruction, or with labeled data.

A deep-learning network trained on labeled data can then be applied to unstructured data, giving it access to much more input than machine-learning nets. This is a recipe for higher performance: the more data a net can train on, the more accurate it is likely to be. (Bad algorithms trained on lots of data can outperform good algorithms trained on very little.) Deep learning's ability to process and learn from huge quantities of unlabeled data give it a distinct advantage over previous algorithms.

Deep-learning networks end in an output layer: a logistic, or softmax, classifier that assigns a likelihood to a particular outcome or label. We call that predictive, but it is predictive in a broad sense. Given raw data in the form of an image, a deep-learning network may decide, for example, that the input data is 90 percent likely to represent a person.

FEEDFORWARD NETWORKS

Our goal in using a neural net is to arrive at the point of least error as fast as possible. We are running a race, and the race is around a track, so we pass the same points repeatedly in a loop. The starting line for the race is the state in which our weights are initialized, and the finish line is the state of those parameters when they are capable of producing sufficiently accurate classifications and predictions.

The race itself involves many steps, and each of those steps resembles the steps before and after. Just like a runner, we will engage in a repetitive act over and over to arrive at the finish. Each step for a neural network involves a guess, an error measurement and a slight update in its weights, an incremental adjustment to the coefficients, as it slowly learns to pay attention to the most important features.

A collection of weights, whether they are in their start or end state, is also called a model, because it is an attempt to model data's relationship to ground-truth labels, to grasp the data's structure. Models normally start out bad and end up less bad, changing over time as the neural network updates its parameters.

This is because a neural network is born in ignorance. It does not know which weights and biases will translate the input best to make the correct guesses. It has to start out with a guess, and then try to make better guesses sequentially as it learns from its mistakes. (You can think of a neural network as a miniature enactment of the scientific method, testing hypotheses and trying again – only it is the scientific method with a blindfold on. Or like a child: they are born not knowing much, and through exposure to life experience, they slowly learn to solve problems in the world. For neural networks, data is the only experience.)

Here is a simple explanation of what happens during learning with a feedforward neural network, the simplest architecture to explain.

Input enters the network. The coefficients, or weights, map that input to a set of guesses the network makes at the end.

input * weight = guess

Weighted input results in a guess about what that input is. The neural then takes its guess and compares it to a ground-truth about the data, effectively asking an expert "Did I get this right?"

ground truth - guess = error

The difference between the network's guess and the ground truth is its *error*. The network measures that error, and walks the error back over its model, adjusting weights to the extent that they contributed to the error.

error * weight's contribution to error = adjustment

The three pseudo-mathematical formulas above account for the three key functions of neural networks: scoring input, calculating loss and applying an update to the model – to begin the three-step process over again. A neural network is a corrective feedback loop, rewarding weights that support its correct guesses, and punishing weights that lead it to err.

Let's linger on the first step above.

MULTIPLE LINEAR REGRESSION

Despite their biologically inspired name, artificial neural networks are nothing more than math and code, like any other machine-learning algorithm. In fact, anyone who understands *linear regression*, one of first methods you learn in statistics, can understand how a neural net works. In its simplest form, linear regression is expressed as

Y_hat = bX + a

where Y_hat is the estimated output, X is the input, b is the slope and a is the intercept of a line on the vertical axis of a two-dimensional graph. (To make this more concrete: X could be radiation exposure and Y could be the cancer risk; X could be daily pushups and Y_hat could be the total weight you can benchpress; Xthe amount of fertilizer and Y_hat the size of the crop.) You can imagine that every time you add a unit to X, the dependent variable Y_hat increases proportionally, no matter how far along you are on the X axis. That simple relation between two variables moving up or down together is a starting point.

The next step is to imagine multiple linear regression, where you have many input variables producing an output variable. It's typically expressed like this:

Y_hat = b_1*X_1 + b_2*X_2 + b_3*X_3 + a

(To extend the crop example above, you might add the amount of sunlight and rainfall in a growing season to the fertilizer variable, with all three affecting Y_hat.)

Now, that form of multiple linear regression is happening at every node of a neural network. For each node of a single layer, input from each node of the previous layer is recombined with input from every other node. That is, the inputs are mixed in different proportions, according to their coefficients, which are different leading into each node of the subsequent layer. In this way, a net tests which combination of input is significant as it tries to reduce error.

Once you sum your node inputs to arrive at Y_hat, it's passed through a non-linear function. Here's why: If every node merely performed multiple linear regression, Y_hat would increase linearly and without limit as the X's increase, but that doesn't suit our purposes.

What we are trying to build at each node is a switch (like a neuron…) that turns on and off, depending on whether or not it should let the signal of the input pass through to affect the ultimate decisions of the network.

When you have a switch, you have a classification problem. Does the input's signal indicate the node should classify it as enough, or not_enough, on or off? A binary decision can be expressed by 1 and 0, and logistic regression is a non-linear function that squashes input to translate it to a space between 0 and 1.

The nonlinear transforms at each node are usually s-shaped functions similar to logistic regression. They go by the names of sigmoid (the Greek word for "S"), tanh, hard tanh, etc., and they shaping the output of each node. The output of all nodes, each squashed into an s-shaped space between 0 and 1, is then passed as input to the next layer in a feed forward neural network, and so on until the signal reaches the final layer of the net, where decisions are made.

GRADIENT DESCENT

The name for one commonly used optimization function that adjusts weights according to the error they caused is called “gradient descent.”

Gradient is another word for slope, and slope, in its typical form on an x-y graph, represents how two variables relate to each other: rise over run, the change in money over the change in time, etc. In this particular case, the slope we care about describes the relationship between the network’s error and a single weight; i.e. that is, how does the error vary as the weight is adjusted.

To put a finer point on it, which weight will produce the least error? Which one correctly represents the signals contained in the input data, and translates them to a correct classification? Which one can hear “nose” in an input image, and know that should be labeled as a face and not a frying pan?

As a neural network learns, it slowly adjusts many weights so that they can map signal to meaning correctly. The relationship between network *Error* and each of those *weights* is a derivative, *dE/dw*, that measures the degree to which a slight change in a weight causes a slight change in the error.

Each weight is just one factor in a deep network that involves many transforms; the signal of the weight passes through activations and sums over several layers, so we use the chain rule of calculus to march back through the networks activations and outputs and finally arrive at the weight in question, and its relationship to overall error.

The chain rule in calculus states that

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}.$$

In a feedforward network, the relationship between the net’s error and a single weight will look something like this:

$$\frac{dError}{dweight} = \frac{dError}{dactivation} * \frac{dactivation}{dweight}$$

That is, given two variables, *Error* and *weight*, that are mediated by a third variable, *activation*, through which the weight is passed, you can calculate how a change in *weight* affects a change in *Error* by first calculating how a change in *activation*affects a change in *Error*, and how a change in *weight* affects a change in *activation*.

The essence of learning in deep learning is nothing more than that: adjusting a model's weights in response to the error it produces, until you can't reduce the error any more.

OPTIMIZATION ALGORITHMS

Some examples of optimization algorithms include:

- ADADELTA
- ADAGRAD
- ADAM
- NESTEROVS
- NONE
- RMSPROP
- SGD
- CONJUGATE GRADIENT
- HESSIAN FREE
- LBFGS
- LINE GRADIENT DESCENT

ACTIVATION FUNCTIONS

The activation function determines the output a node will generate, based upon its input. In Deeplearning4j, the activation function is set at the layer level and applies to all neurons in that layer.

Some examples include:

- CUBE
- ELU
- HARDSIGMOID
- HARDTANH
- IDENTITY
- LEAKYRELU
- RATIONALTANH
- RELU
- RRELU
- SIGMOID
- SOFTMAX
- SOFTPLUS
- SOFTSIGN
- TANH

Custom layers, activation functions and loss functions

Deeplearning4j, one of the major AI frameworks Skymind supports alongside Keras, includes custom layers, activations and loss functions.

LOGISTIC REGRESSION

On a deep neural network of many layers, the final layer has a particular role. When dealing with labeled input, the output layer classifies each example, applying the most likely label. Each node on the output layer represents one label, and that node turns on or off according to the strength of the signal it receives from the previous layer's input and parameters.

Each output node produces two possible outcomes, the binary output values 0 or 1, because an input variable either deserves a label or it does not. After all, there is no such thing as a little pregnant.

While neural networks working with labeled data produce binary output, the input they receive is often continuous. That is, the signals that the network receives as input will span a range of values and include any number of metrics, depending on the problem it seeks to solve.

For example, a recommendation engine has to make a binary decision about whether to serve an ad or not. But the input it bases its decision on could include how much a customer has spent on Amazon in the last week, or how often that customer visits the site.

So the output layer has to condense signals such as $67.59 spent on diapers, and 15 visits to a website, into a range between 0 and 1; i.e. a probability that a given input should be labeled or not.

The mechanism we use to convert continuous signals into binary output is called logistic regression. The name is unfortunate, since logistic regression is used for classification rather than regression in the linear sense that most people are familiar with. It calculates the probability that a set of inputs match the label.

$$F(x) = \frac{1}{1 + e^{-x}}$$

Let's examine this little formula.

For continuous inputs to be expressed as probabilities, they must output positive results, since there is no such thing as a negative probability. That's why you see input as the exponent of *e* in the denominator – because exponents force our results to be greater than zero. Now consider the relationship of *e*'s exponent to the fraction 1/1. One, as we know, is the ceiling of a probability, beyond which our results can't go without being absurd. (We're 120% sure of that.)

As the input *x* that triggers a label grows, the expression *e to the x* shrinks toward zero, leaving us with the fraction 1/1, or 100%, which means we approach (without ever quite reaching) absolute certainty that the label applies. Input that correlates negatively with your output will have its value flipped by the negative sign on *e*'s exponent, and as that negative signal grows, the quantity *e to the x* becomes larger, pushing the entire fraction ever closer to zero.

Now imagine that, rather than having *x* as the exponent, you have the sum of the products of all the weights and their corresponding inputs – the total signal passing through your net. That's what you're feeding into the logistic regression layer at the output layer of a neural network classifier.

With this layer, we can set a decision threshold above which an example is labeled 1, and below which it is not. You can set different thresholds as you prefer – a low threshold will increase the number of false positives, and a higher one will increase the number of false negatives – depending on which side you would like to err. **1.5 Literature survey**

Human 2D pose estimation—the problem of localizing anatomical keypoints or "parts"—has largely focused on finding body parts of individuals [18, 14, 23, 31, 33], Inferring the pose of multiple people in images, especially socially engaged individuals, presents a unique set of challenges. First, each image may contain an unknown number of people that can occur at any position or scale. Second, interactions between people induce complex spatial interference, due to contact, occlusion, and limb articulations, making association of parts difficult. Third, runtime complexity tends to grow with the number of people in the image, making realtime performance a challenge. A common approach [33, 19] is to employ a person detector and perform single-person pose estimation for each detection. These top-down approaches directly leverage existing techniques for single-person pose estimation, but suffer from early commitment: if the person detector fails–as it is prone to do when people are in close proximity–there is no recourse to recovery. Furthermore, the runtime of these, body parts belonging to the same person are linked. Bottom left: Part Affinity Fields (PAFs) corresponding to the limb connecting right elbow and right wrist. The color encodes orientation. Bottom right: A zoomed in view of the predicted PAFs. At each pixel in the field, a 2D vector encodes the position and orientation of the limbs. top-down approaches is proportional to the number of people: for each detection, a single-person pose estimator is run, and the more people there are, the greater the computational cost. In contrast, bottom-up approaches are attractive as they offer robustness to early commitment and have the potential to decouple runtime complexity from the number of people in the image. Yet, bottom-up approaches do not directly use global contextual cues from other body parts and other people. In practice, previous bottom-up methods [21] do not retain the gains in efficiency as the final parse requires costly global inference. For example, the seminal work of Pishchulin et al. [32] proposed a bottom-up approach that jointly labeled part detection candidates and associated them to individual people. However, solving the integer linear programming problem over a fully connected graph is an NP-hard problem and the average processing time is on the order of hours. Insafutdinov et al. [31] built on [32] with stronger part detectors based on ResNet [30] and image-dependent pairwise scores, and vastly improved the runtime, but the method still takes several minutes per image, with a limit on the number of part proposals. The pairwise representations used in [31], are difficult to regress precisely and thus a separate logistic regression is required.

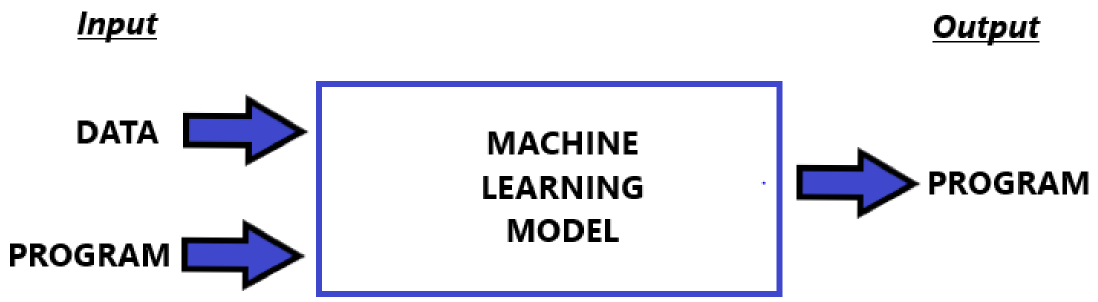
CHAPTER 2

DESIGN ASPECTS AND APPROACH

PRE-PROCESSING

The major step in designing the processing is the Pre-Processing task. It is very time consuming and most important than anything else in the project. The pre-processing cleanses the data well and sends it to the model for main high-end computation. The dataset is collected from various open-source databases like COCO. The images are renamed and formatted to a single .png format for a consistent computation. The images can be of any format, but we went ahead with .png format. Re-Sizing it, allocating images to specific folders by name, and the given to the model for the training.

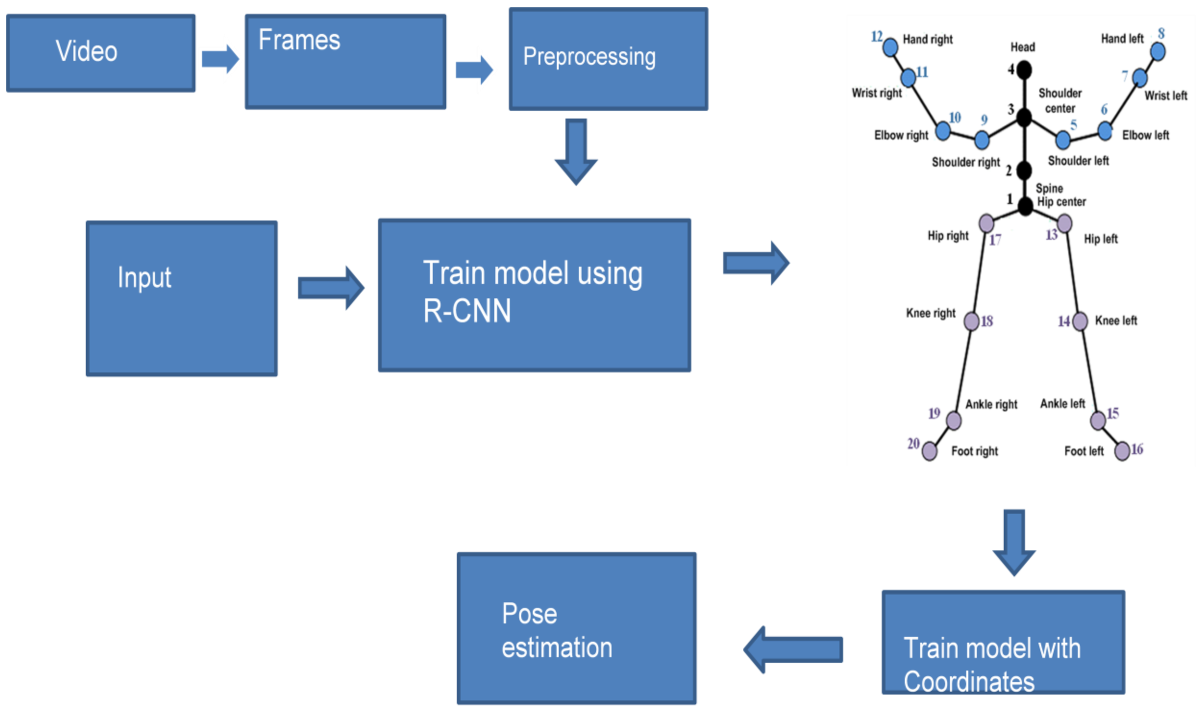
2.1 BLOCK DIAGRAM



2.1 Fig. Block Diagram of Machine Learning Framework

The overall block diagram of the proposed project is shown in the figure. The input given is the image dataset and Model building program, which in return generates a program that can be deployable in any hardware and software development kit or environment for making it into an end-to-end product. So, the output program is the program where test input image and trained weight file must be given to get the output.

2.2 INTERNAL BLOCK DIAGRAM:

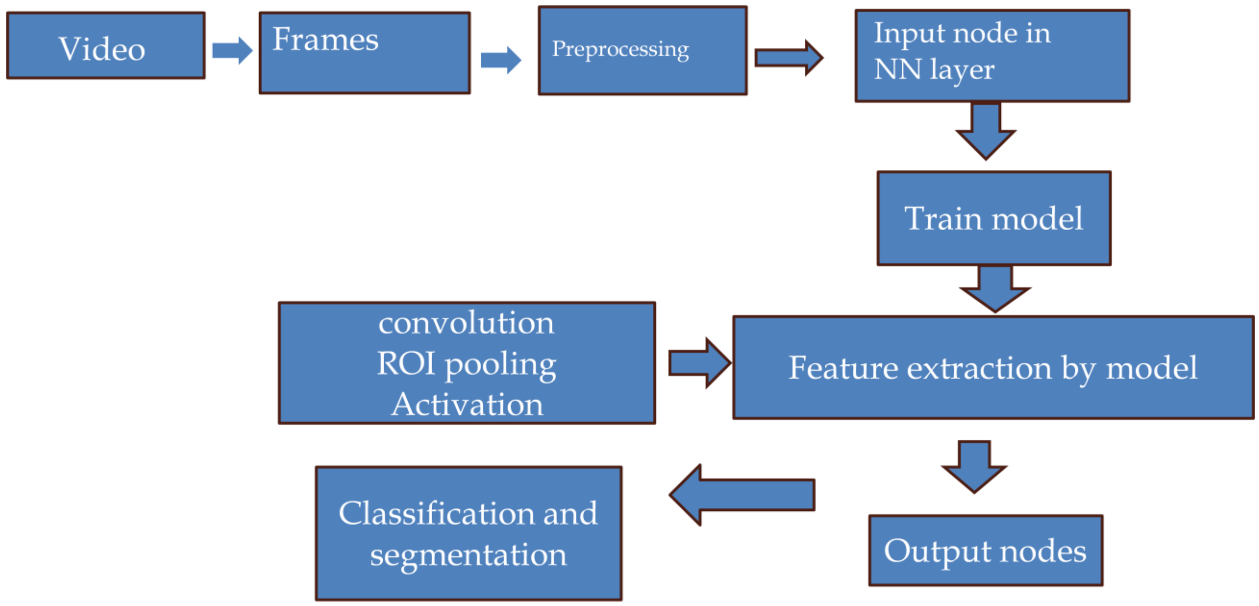


2.2 Internal Block Diagram

The image dataset is given to the model after pre-processing irrespective of the Deep Learning framework. Whether it is CNN or Transfer Learning, the pre-processing will be the same and the outer block diagram will be the same. The features extracted will be the same in each and every model. Both CNN and Transfer Learning uses Global View parameters.

Here in this project we used pre trained coco dataset model weights, so as to speed up the process of training.

2.3 FLOW DIAGRAM



2.3 Algorithmic flowchart of R-CNN

Nearly 2 million images from open sourced dataset COCO is taken and a model is trained using Convolution neural networks to identify different objects in a image and to classify them. Objects from the test input are first identified and classified in to different objects. From these objects human (required object is selected) and a bounding box is drawn in order to separate human from the image which is called Region of Interest(ROI). To this ROI, RCN(Region convolved networks) is applied to segment the human from the background.

The first step is the Convolution operation which extracts the features from the image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It's a mathematical operation that takes two inputs such as image matrix and a filter, this is called "Feature Map".

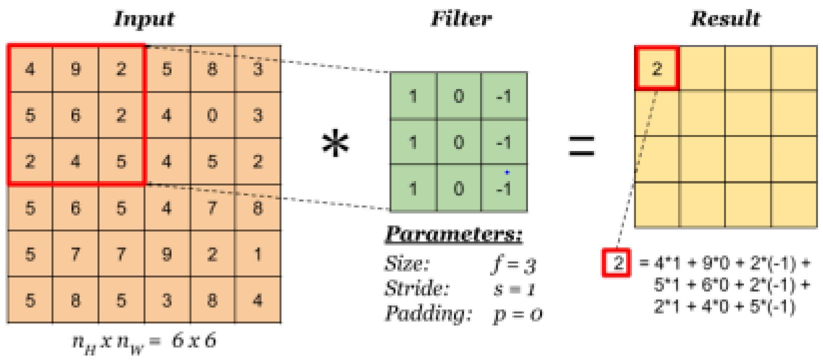


Fig. Convolution Operation

PADDING

Padding operation is followed after Convolution. It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers. 2. It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

A 2-Stride operation is used in order to avoid the overlapping and reduce the computation.

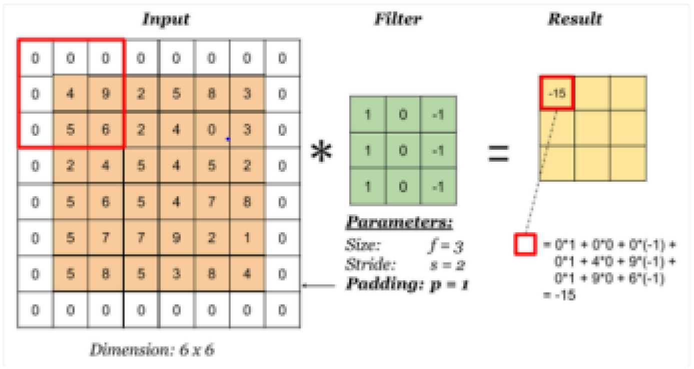


Fig.2.3.1 Padding Operation

Pooling Layers section would reduce the number of parameters when the images are too large. Pooling simplifies the output by performing nonlinear down-sampling, reducing the number of parameters that the network needs to learn. Spatial pooling also called subsampling or down-sampling which reduces the dimensionality of each map but retains the important information. **Max Pooling** operation is performed which takes largest element from the rectified feature map. This avoids the data corruption and hence, guarantees all features are rightly chosen and taken into account.

Fully Connected Layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network. Almost **10 billion+** parameters are extracted from the whole training set which gives highest accuracy possible in classifying the human and parameters as well from a test image.

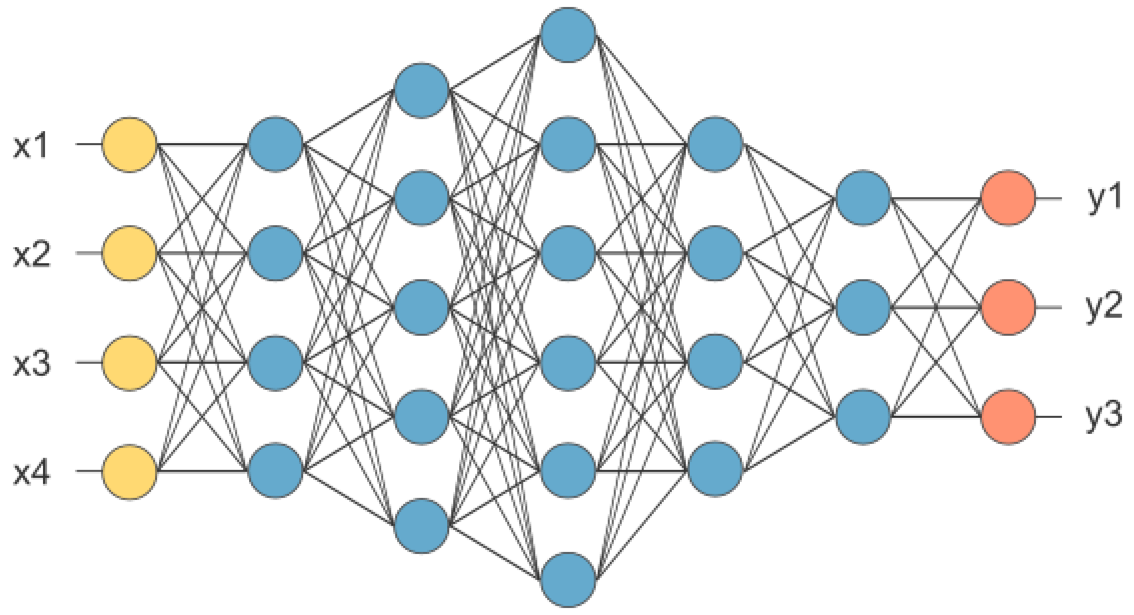


Fig.2.3.2 After pooling layer, flattened as Fully Connected layer

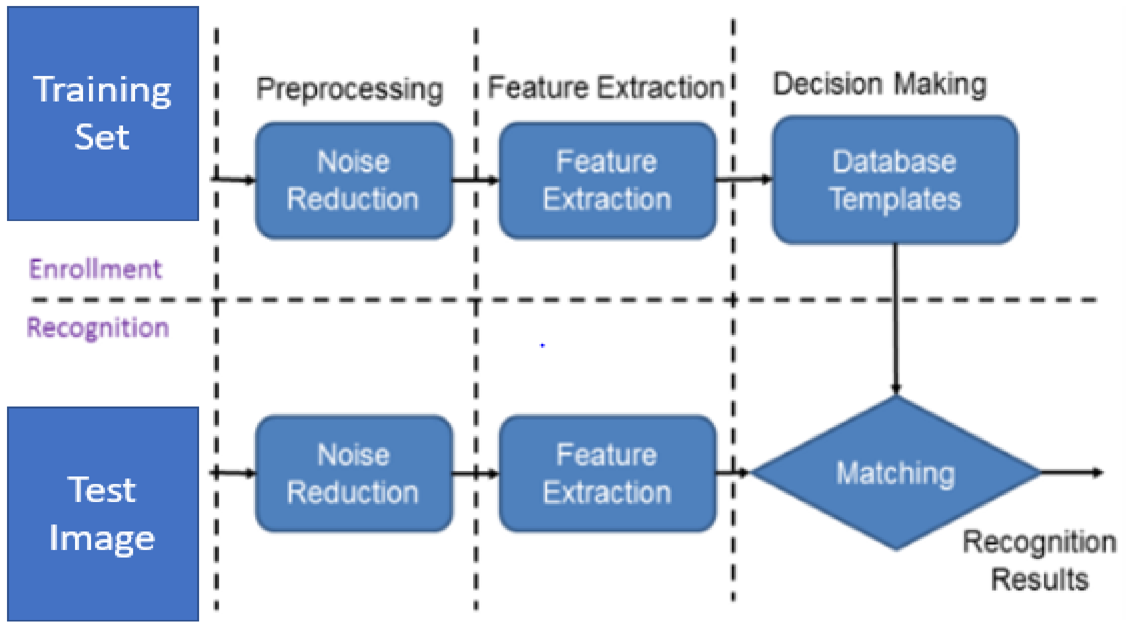
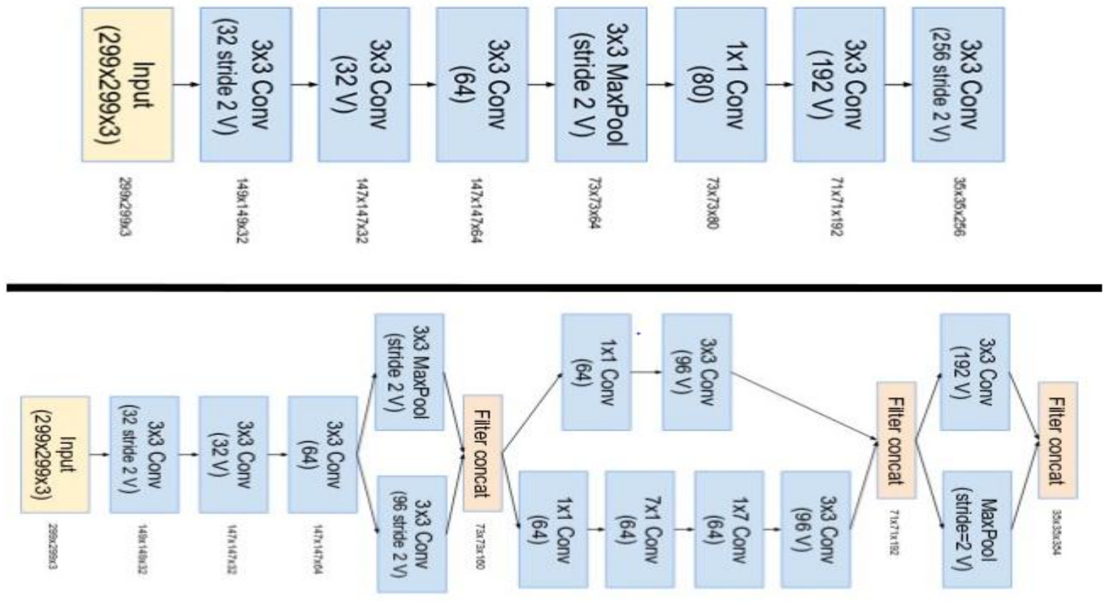


Fig.2.3.3 Flowchart of identifying Human



The top image is the stem of Inception-ResNet v1. The bottom image is the stem of Inception v4 and Inception-ResNet v2. (Source: Inception v4)

Fig.2.3.4 Inception and Inception-ResNet operation flow

- Inception v4 has specialized “**Reduction Blocks**” which are used to change the width and height of the grid.
- In Inception-ResNet, for residual addition to work, the input and output after convolution must have the same dimensions. Hence, we use 1x1 convolutions after the original convolutions, to match the depth sizes (Depth is increased after convolution).
- The pooling operation inside the main inception modules were replaced in favor of the residual connections. However, you can still find those operations in the reduction blocks. Reduction block A is same as that of Inception v4.

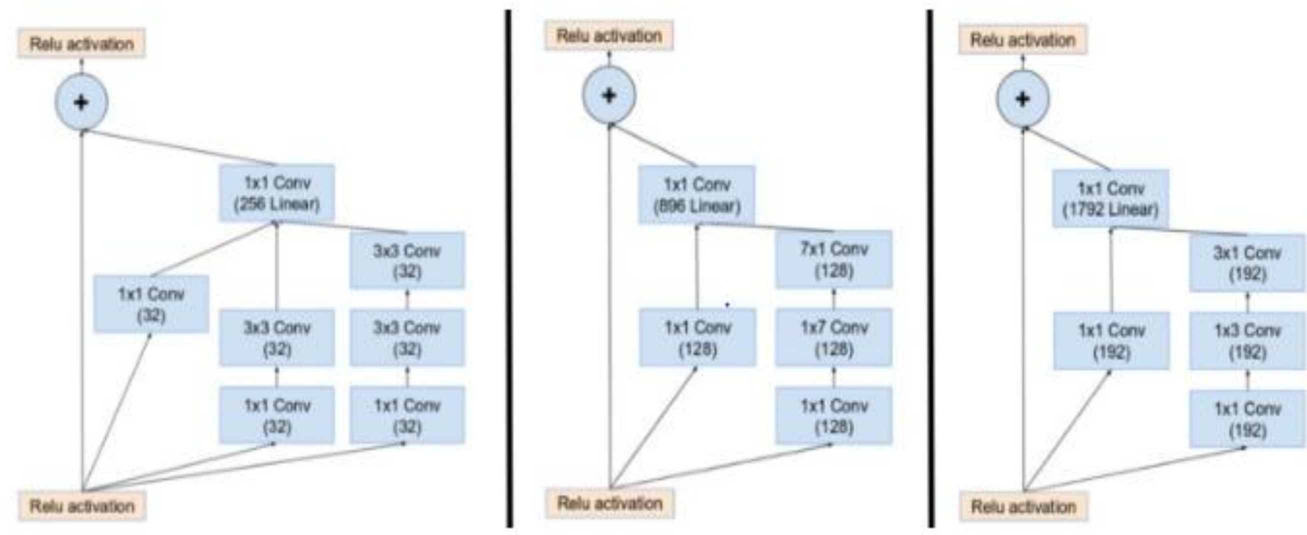


Fig.2.3.5 (From left) Inception modules A,B,C in an Inception ResNet. Note how the pooling layer was replaced by the residual connection, and also the additional 1x1 convolution before addition

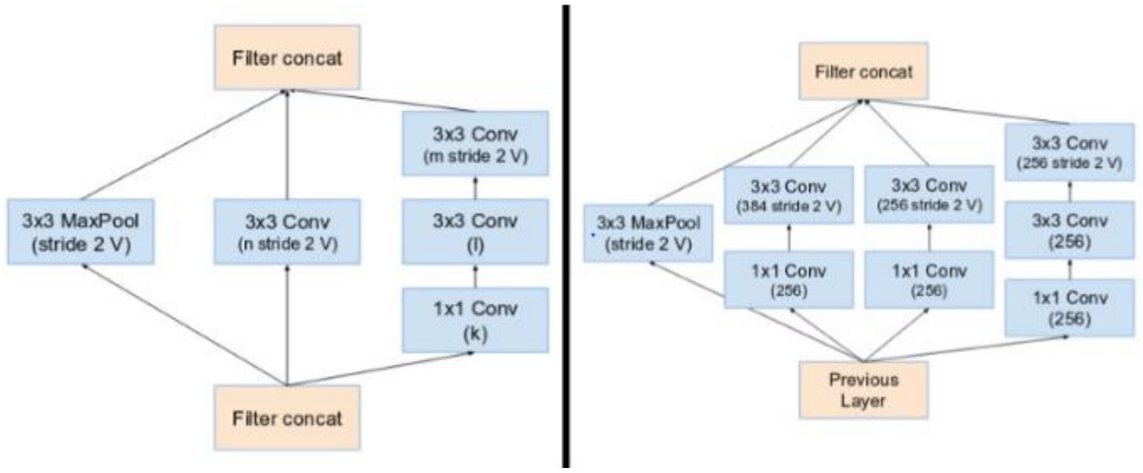


Fig. (From Left) Reduction Block A (35x35 to 17x17 size reduction) and Reduction Block B (17x17 to 8x8 size reduction)

- Networks with residual units deeper in the architecture caused the network to “die” if the number of filters exceeded 1000. Hence, to increase stability, the authors scaled the residual activations by a value around 0.1 to 0.3.

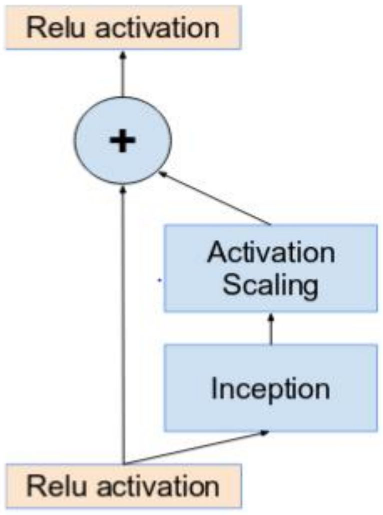


Fig.2.3.6 Activations are scaled by a constant to prevent the network from dying.

ARCHITECTURE

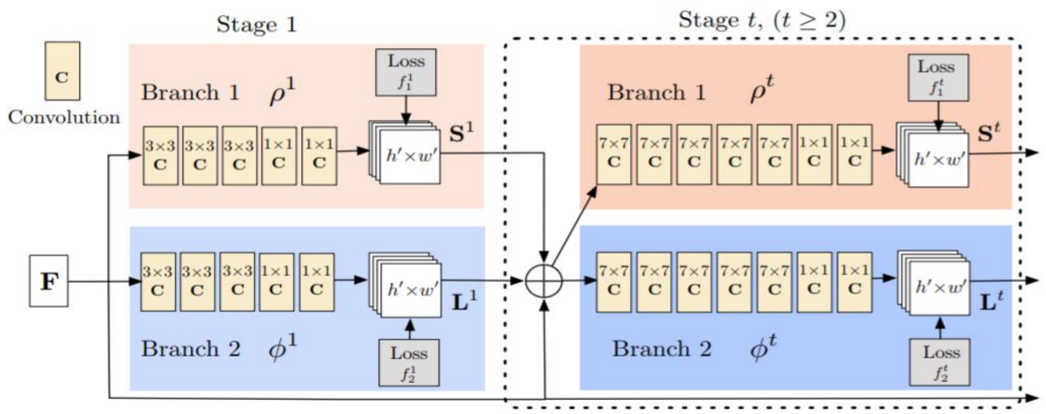


Fig:2.3.7 Architecture of R-CNN

CHAPTER-3

SOFTWARE AND HARDWARE REQUIREMENTS

3.1 SOFTWARE AND HARDWARE REQUIREMENTS

This project requires python of version 3.6 or above and libraries such as OpenCv, Tensorflow and Numpy. We use Anaconda IDE for programming in python because of its good U.I and efficient management of data.

S.No.	**Software Package**	**Version**
1	Python	3.6
2	Anaconda	4.5.12
3	Ubuntu	16.4
4	Windows	10
5	Keras	2.2.4
6	Tensorflow	CPU and GPU
7	OpenCV	4.0
8	Numpy	1.16.1
9	Pandas	0.23.4
10	ImageAI	-
11	Sklearn	0.20.3

12	Matplotlib	3.0.3
13	Cuda	10.1
14	SciPy	0.19.1
15	Pillow	6.0
16	h5py	2.9.0

Table 3.1: Software requirements

HARDWARE REQUIREMENTS

S.No.	**Hardware Package**	**Version**
1	NVIDIA GPU	1080x
2	Intel CPU	i7

Table 3.2: Hardware requirements

3.1.1 PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It also has a comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

LIBRARIES:

- Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals,[95] manipulating regular expressions, and unit testing.
- Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgiref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.
- As of March 2018, the Python Package Index (PyPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:
- Graphical user interfaces
- Web frameworks
- Multimedia
- Databases
- Networking
- Test frameworks

- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

OpenCV (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

Applications

- Egomotion estimation
- 2D and 3D feature toolkits
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking
- Augmented reality

TENSORFLOW

Tensorflow is a open source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0open-source license on November 9, 2015.

Tensorflow is Google Brain's second-generation system. Version 1.0.0 was released on Februrary 11,2017. While the reference implementation runs on single devices, Tensorflow can run on multiple CPUs and GPUs(with optional CUDA and SYCL extensions for general –purpose computing on graphics processing units). Tensorflow is available on 64-bit Linux, macos, Windows, and mobile computing platforms including Android and ios.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as *tensors*. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHubmentioned TensorFlow, of which only 5 were from Google.[12]

TENSOR PROCESSING UNIT (TPU)

In May 2016, Google announced its Tensor Processing Unit (TPU), an application-specific integrated circuit (a hardware chip) built specifically for machine learning and tailored for TensorFlow. TPU is a programmable AI accelerator designed to provide high throughput of low-precision arithmetic (e.g., 8-bit), and oriented toward using or running models rather than training them. Google announced they had been running TPUs inside their data centers for more than a year, and had found them to deliver an order of magnitude better-optimized performance per watt for machine learning.

In May 2017, Google announced the second-generation, as well as the availability of the TPUs in Google Compute Engine.The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs, provide up to 11.5 petaflops.

In February 2018, Google announced that they were making TPUs available in beta on the Google Cloud Platform.

EDGE TPU

In July 2018 the Edge TPU was announced. Edge TPU is Google's purpose-built ASIC chip designed to run TensorFlow Lite machine learning (ML) models on small client computing devices such as smartphones known as edge computing.

TENSORFLOW LITE

In May 2017, Google announced a software stack specifically for mobile development, TensorFlow Lite. In January 2019, TensorFlow team released a developer preview of the mobile GPU inference engine with OpenGL ES 3.1 Compute Shaders on Android devices and Metal Compute Shaders on iOS devices.

PIXEL VISUAL CORE (PVC)

In October 2017, Google released the Google Pixel 2 which featured their Pixel Visual Core (PVC), a fully programmable Image, Vision and AI processor for mobile devices. The PVC supports TensorFlow for machine learning (and Halide for image processing).

APPLICATIONS

Google officially released RankBrain on October 26, 2015, backed by TensorFlow.

Google also released Colaboratory, which is a TensorFlow Jupyter notebook environment that requires no setup to use.

ANACONDA

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing(data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system *conda*. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS.

Overview:

Anaconda distribution comes with more than 1,400 packages as well as the Conda package and virtual environment manager, called Anaconda Navigator , so it eliminates the need to learn to install each library independently.

The open source packages can be individually installed from the Anaconda repository with the `conda install` command or using the `pip install` command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases they can work together.

Custom packages can be made using the `conda build` command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, you can create new environments that include any version of Python packaged with conda .

ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange
- Rstudio
- Visual Studio Code

CONDA

Main article: Conda (package manager)

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The Conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

ANACONDA CLOUD

Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages. Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications. You do not need to log in or to have a Cloud account, to search for public packages, download and install them.

You can build new packages using the Anaconda Client command line interface (CLI), then manually or automatically upload the packages to Cloud.

CHAPTER-4

TESTING

4.1 TESTING

- Python with Tensorflow and Opencv library is used to achieve the results.
- We pass the input such as an image or an video to the program to perform the analysis.
- We can also process the live webcam feed.
- I used Anaconda IDE environment to test the code

4.2RUNTIME ANALYSIS

To analyze the runtime performance of our method, we collect videos with a varying number of people. The original frame size is 1080×1920, which we resize to 368×654 during testing to fit in GPU memory. The runtime analysis is performed on a laptop with one NVIDIA GeForce GTX-1080 GPU. In Fig. 8d, we use person detection and single-person CPM as a top-down comparison, where the runtime is roughly proportional to the number of people in the image. In contrast, the runtime of our bottom-up approach increases relatively slowly with the increasing number of people. The runtime consists of two major parts: (1) CNN processing time whose runtime complexity is O(1), constant with varying number of people; (2) Multi-person parsing time whose runtime complexity is O(n 2), where n represents the number of people. However, the parsing time does not significantly influence the overall runtime because it is two orders of magnitude less than the CNN processing time, e.g., for 9 people, the parsing takes 0.58 ms while CNN takes 99.6 ms. Our method has achieved the speed of 8.8 fps for a video with 19 people.

CHAPTER-5

OUTPUT SCREENS

OUTPUT FOR ONE SUBJECT

INPUT IMAGE

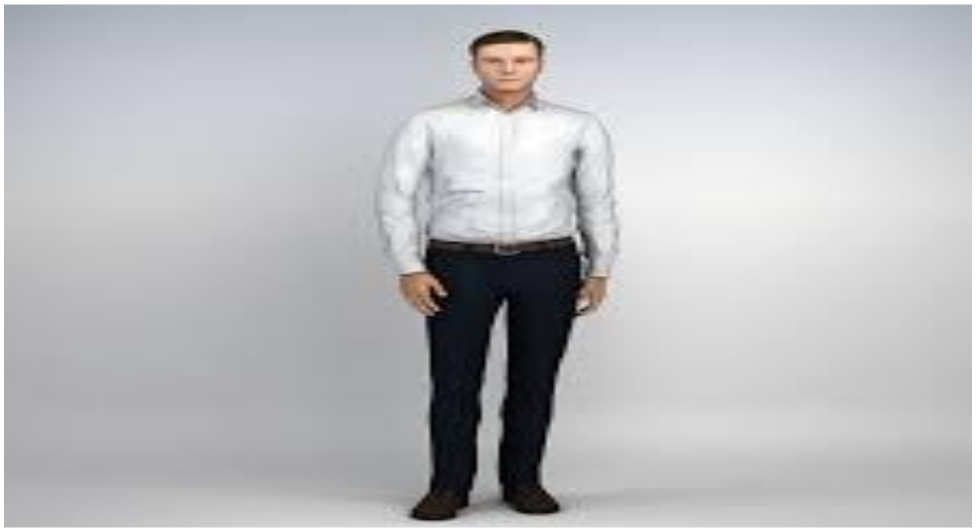


Fig 5.1 Input image

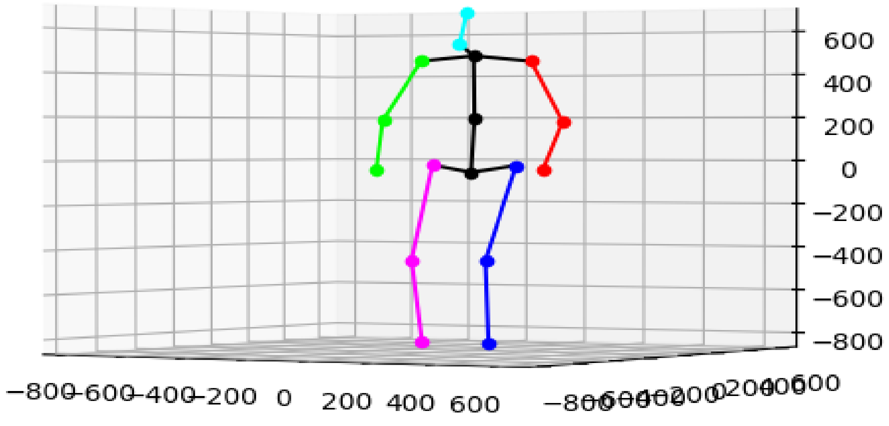


Fig 5.2 Segmented and skeletonised plotted image

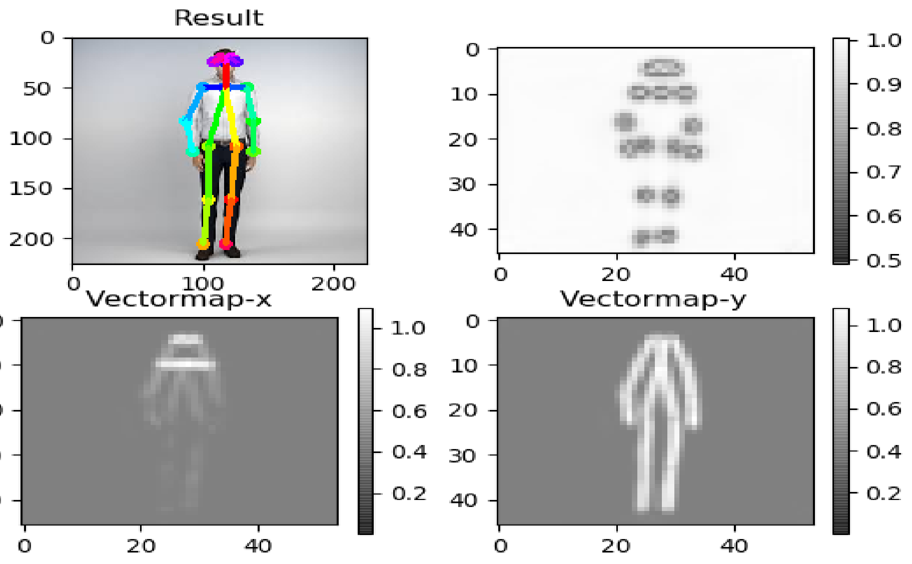
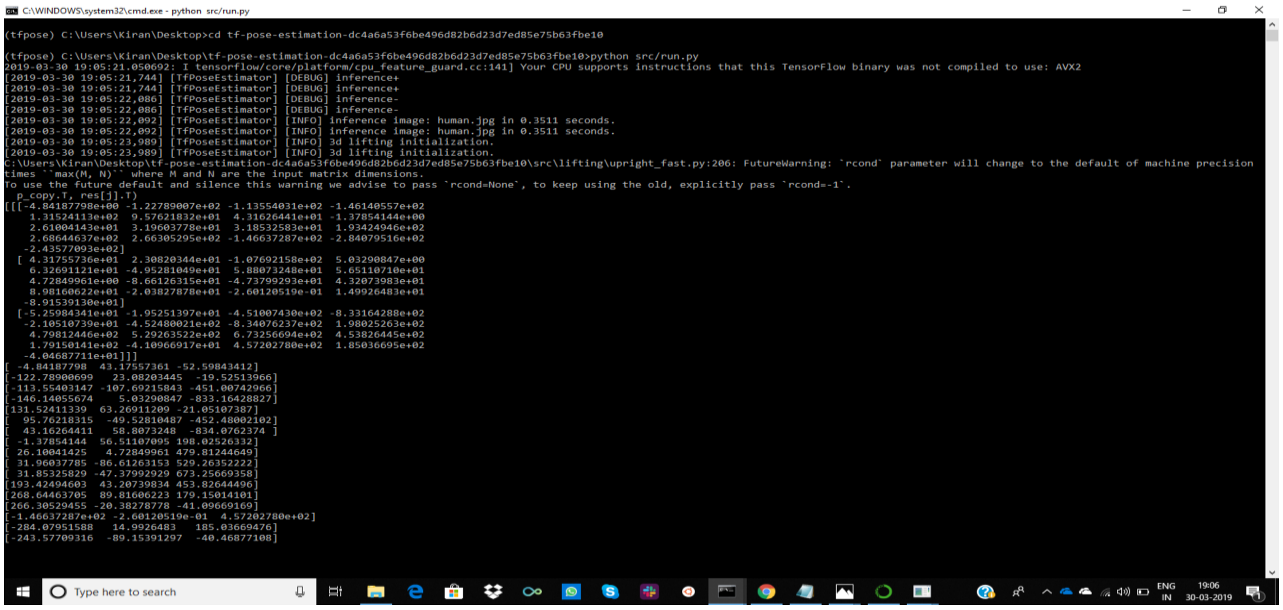


Fig 5.3 vector plots

Coordinates of human



```
C:\WINDOWS\system32\cmd.exe - python  src/run.py

(tfpose) C:\Users\Kiran\Desktop>cd tf-pose-estimation-dc4a6a53f6be496d82b6d23d7ed85e75b63fbe10

(tfpose) C:\Users\Kiran\Desktop\tf-pose-estimation-dc4a6a53f6be496d82b6d23d7ed85e75b63fbe10>python src/run.py
2019-03-30 19:05:21.050692: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
[2019-03-30 19:05:21,744] [TfPoseEstimator] [DEBUG] inference+
[2019-03-30 19:05:21,744] [TfPoseEstimator] [DEBUG] inference+
[2019-03-30 19:05:22,086] [TfPoseEstimator] [DEBUG] inference-
[2019-03-30 19:05:22,086] [TfPoseEstimator] [DEBUG] inference-
[2019-03-30 19:05:22,092] [TfPoseEstimator] [INFO] inference image: human.jpg in 0.3511 seconds.
[2019-03-30 19:05:22,092] [TfPoseEstimator] [INFO] inference image: human.jpg in 0.3511 seconds.
[2019-03-30 19:05:23,989] [TfPoseEstimator] [INFO] 3d lifting initialization.
[2019-03-30 19:05:23,989] [TfPoseEstimator] [INFO] 3d lifting initialization.
C:\Users\Kiran\Desktop\tf-pose-estimation-dc4a6a53f6be496d82b6d23d7ed85e75b63fbe10\src\lifting\upright_fast.py:206: FutureWarning: `rcond` parameter will change to the default of machine precision
times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  p_copy.T, res[j].T)
[[[-4.84187798e+00 -1.22789007e+02 -1.13554031e+02 -1.46140557e+02
    1.31524113e+02  9.57621832e+01  4.31626441e+01 -1.37854144e+00
    2.61004143e+01  3.19603778e+01  3.18532583e+01  1.93424946e+02
    2.68644637e+02  2.66305295e+02 -1.46637287e+02 -2.84079516e+02
   -2.43577093e+02]
  [ 4.31755736e+01  2.30820344e+01 -1.07692158e+02  5.03290847e+00
    6.32691121e+01 -4.95281049e+01  5.88073248e+01  5.65110710e+01
    4.72849961e+00 -8.66126315e+01 -4.73799293e+01  4.32073983e+01
    8.98160622e+01 -2.03827878e+01 -2.60120519e-01  1.49926483e+01
   -8.91539130e+01]
  [-5.25984341e+01 -1.95251397e+01 -4.51007430e+02 -8.33164288e+02
   -2.10510739e+01 -4.52480021e+02 -8.34076237e+02  1.98025263e+02
    4.79812446e+02  5.29263522e+02  6.73256694e+02  4.53826445e+02
    1.79150141e+02 -4.10966917e+01  4.57202780e+02  1.85036695e+02
   -4.04687711e+01]]]
[  -4.84187798  43.17557361 -52.59843412]
[-122.78900699   23.08203445  -19.52513966]
[-113.55403147 -107.69215843 -451.00742966]
[-146.14055674    5.03290847 -833.16428827]
[131.52411339  63.26911209 -21.05107387]
[  95.76218315  -49.52810487 -452.48002102]
[  43.16264411   58.8073248  -834.0762374 ]
[  -1.37854144  56.51107095 198.02526332]
[  26.10041425   4.72849961 479.81244649]
[  31.96037785 -86.61263153 529.26352222]
[  31.85325829 -47.37992929 673.25669358]
[193.42494603  43.20739834 453.82644496]
[268.64463705  89.81606223 179.15014101]
[266.30529455 -20.38278778 -41.09669169]
[-1.46637287e+02 -2.60120519e-01  4.57202780e+02]
[-284.07951588   14.9926483   185.03669476]
[-243.57709316  -89.15391297  -40.46877108]
```

Fig 5.4 Coordinates of human

OUTPUT FOR MULTI SUBJECTS

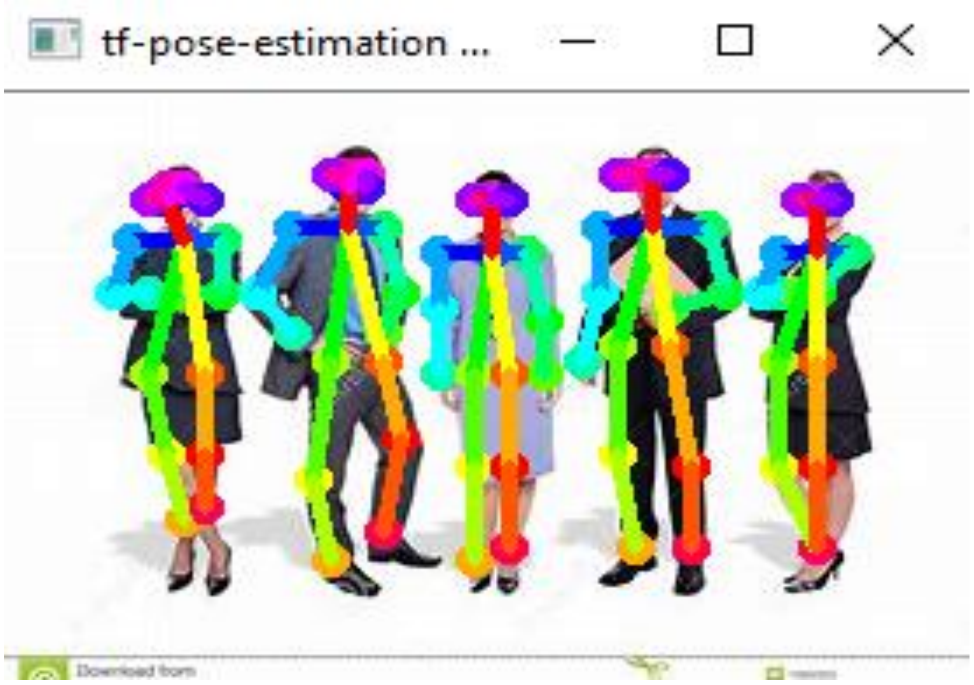
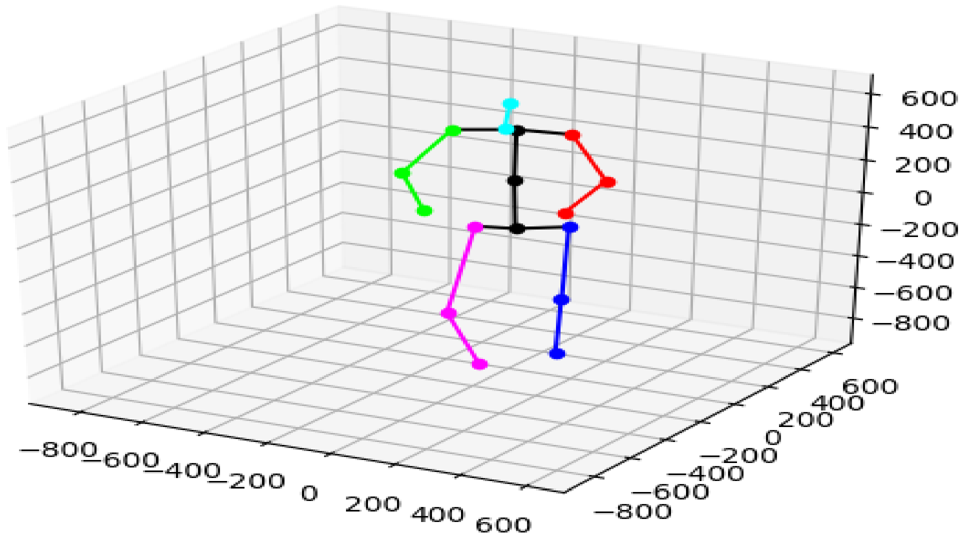
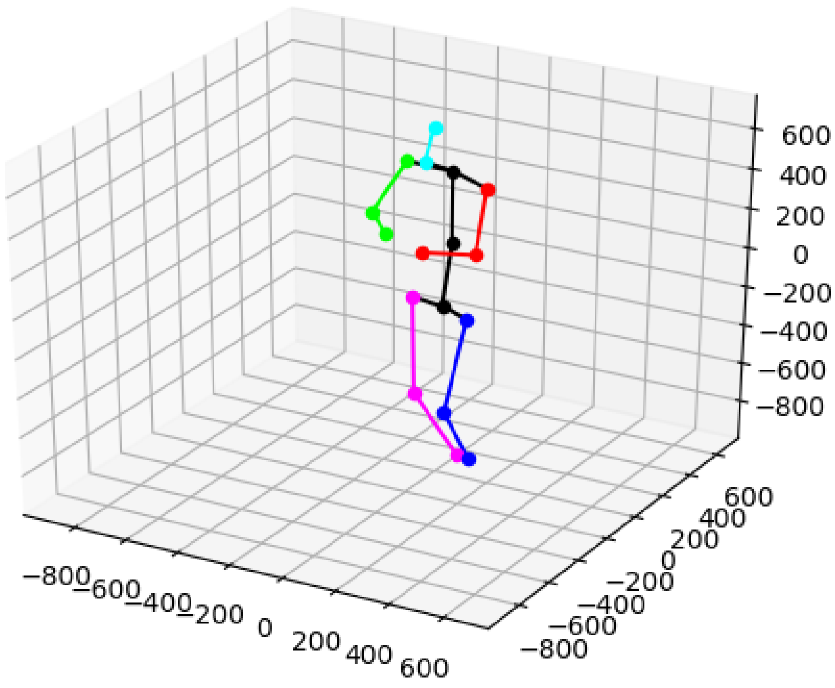
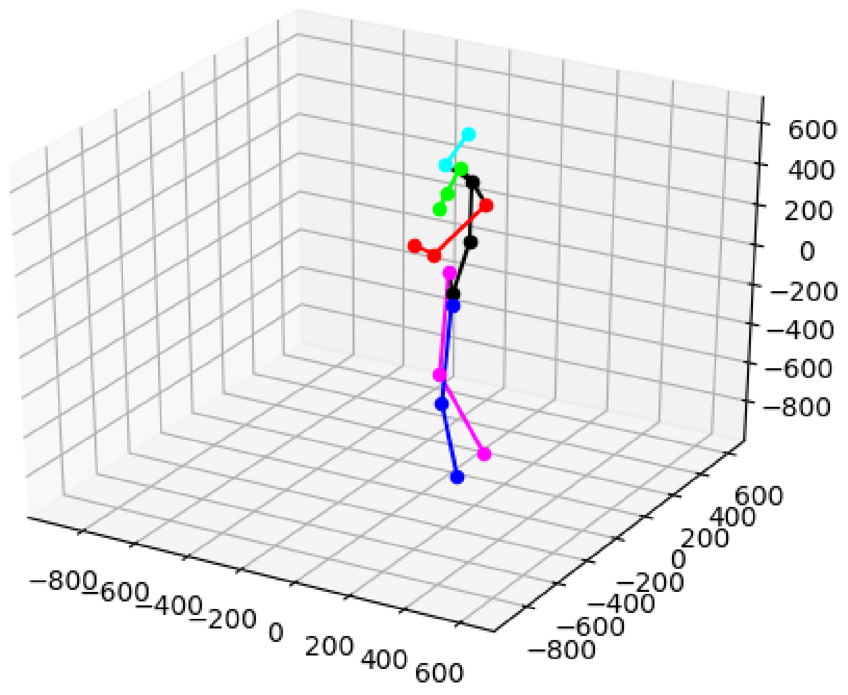


Fig 5.5 Input image

INDIVIDUAL 3D PLOTS OF HUMANS IN THE FRAME







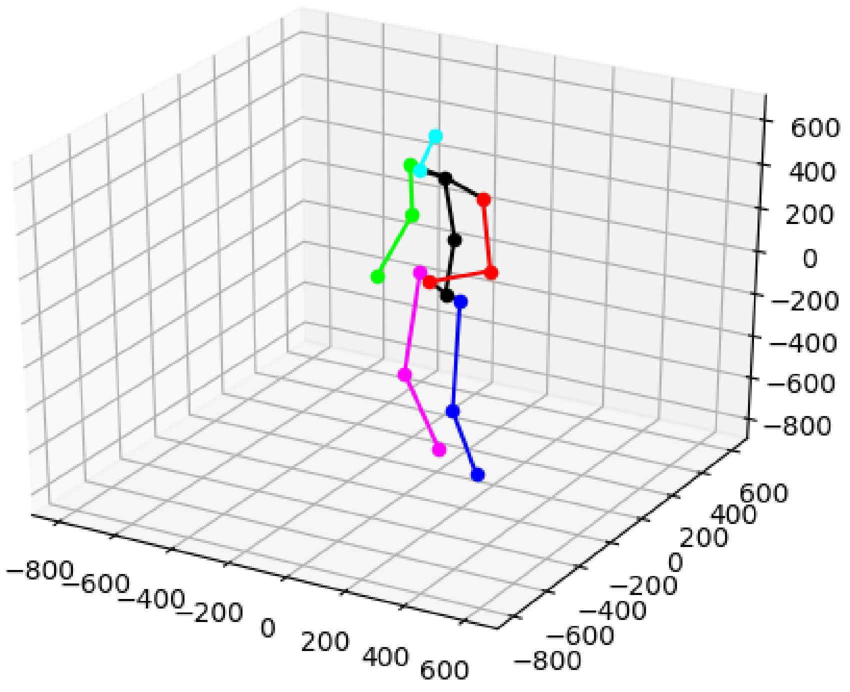
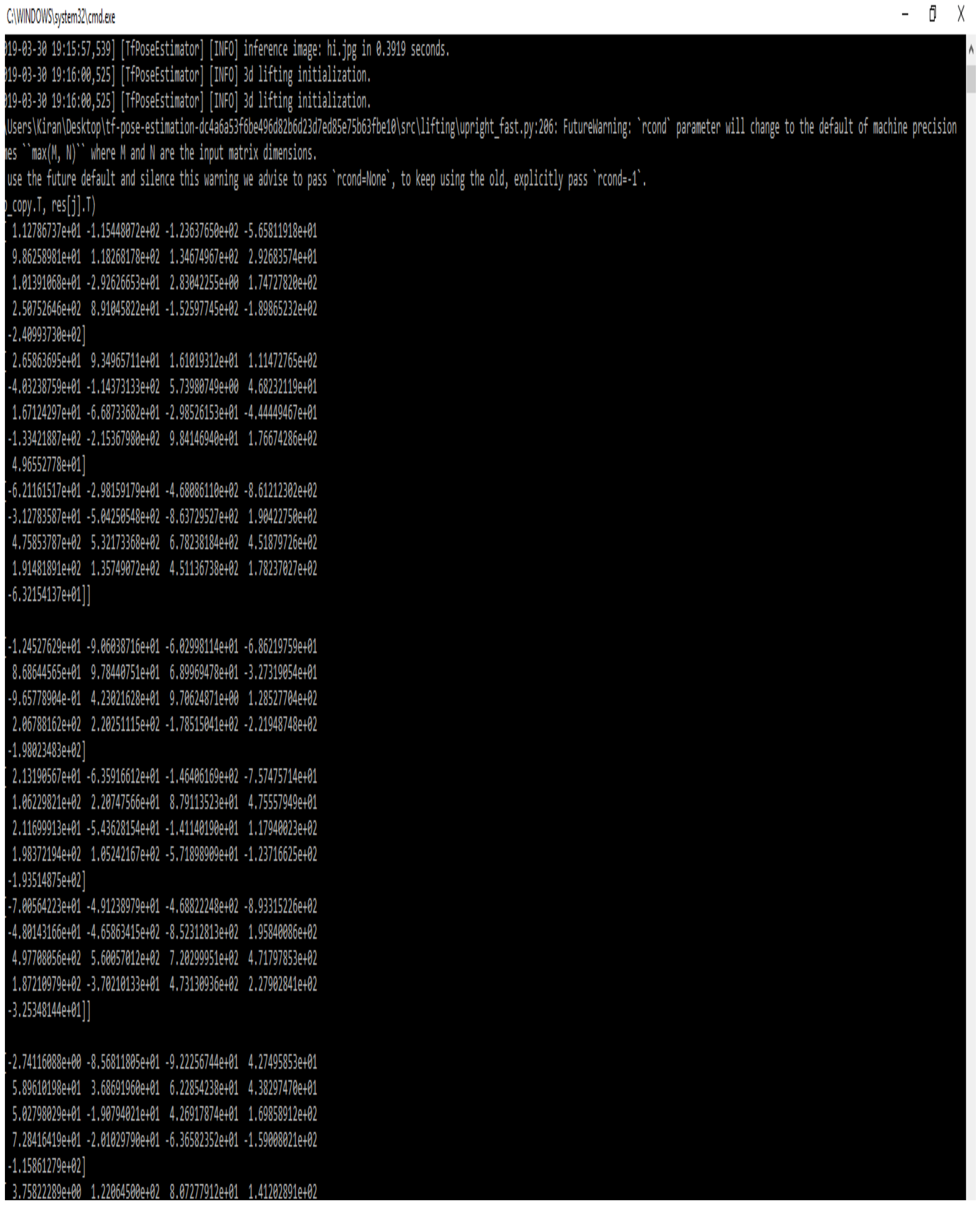


Fig 5.6 Individual 3d plots of humans in the frame



```
C:\WINDOWS\system32\cmd.exe
019-03-30 19:15:57,539] [TfPoseEstimator] [INFO] inference image: hi.jpg in 0.3919 seconds.
019-03-30 19:16:00,525] [TfPoseEstimator] [INFO] 3d lifting initialization.
019-03-30 19:16:00,525] [TfPoseEstimator] [INFO] 3d lifting initialization.
\Users\Kiran\Desktop\tf-pose-estimation-dc4a6a53f6be496d82b6d23d7ed85e75b63fbe10\src\lifting\upright_fast.py:206: FutureWarning: `rcond` parameter will change to the default of machine precision
mes ``max(M, N)`` where M and N are the input matrix dimensions.
 use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
p_copy.T, res[j].T)
[ 1.12786737e+01 -1.15448072e+02 -1.23637650e+02 -5.65811918e+01
  9.86258981e+01  1.18268178e+02  1.34674967e+02  2.92683574e+01
  1.01391068e+01 -2.92626653e+01  2.83042255e+00  1.74727820e+02
  2.50752646e+02  8.91045822e+01 -1.52597745e+02 -1.89865232e+02
 -2.40993730e+02]
[ 2.65863695e+01  9.34965711e+01  1.61019312e+01  1.11472765e+02
 -4.03238759e+01 -1.14373133e+02  5.73980749e+00  4.68232119e+01
  1.67124297e+01 -6.68733682e+01 -2.98526153e+01 -4.44449467e+01
 -1.33421887e+02 -2.15367980e+02  9.84146940e+01  1.76674286e+02
  4.96552778e+01]
[-6.21161517e+01 -2.98159179e+01 -4.68086110e+02 -8.61212302e+02
 -3.12783587e+01 -5.04250548e+02 -8.63729527e+02  1.90422750e+02
  4.75853787e+02  5.32173368e+02  6.78238184e+02  4.51879726e+02
  1.91481891e+02  1.35749072e+02  4.51136738e+02  1.78237027e+02
 -6.32154137e+01]]

[-1.24527629e+01 -9.06038716e+01 -6.02998114e+01 -6.86219759e+01
  8.68644565e+01  9.78440751e+01  6.89969478e+01 -3.27319054e+01
 -9.65778904e-01  4.23021628e+01  9.70624871e+00  1.28527704e+02
  2.06788162e+02  2.20251115e+02 -1.78515041e+02 -2.21948748e+02
 -1.98023483e+02]
[ 2.13190567e+01 -6.35916612e+01 -1.46406169e+02 -7.57475714e+01
  1.06229821e+02  2.20747566e+01  8.79113523e+01  4.75557949e+01
  2.11699913e+01 -5.43628154e+01 -1.41140190e+01  1.17940023e+02
  1.98372194e+02  1.05242167e+02 -5.71898909e+01 -1.23716625e+02
 -1.93514875e+02]
[-7.00564223e+01 -4.91238979e+01 -4.68822248e+02 -8.93315226e+02
 -4.80143166e+01 -4.65863415e+02 -8.52312813e+02  1.95840086e+02
  4.97708056e+02  5.60057012e+02  7.20299951e+02  4.71797853e+02
  1.87210979e+02 -3.70210133e+01  4.73130936e+02  2.27902841e+02
 -3.25348144e+01]]

[-2.74116088e+00 -8.56811805e+01 -9.22256744e+01  4.27495853e+01
  5.89610198e+01  3.68691960e+01  6.22854238e+01  4.38297470e+01
  5.02798029e+01 -1.90794021e+01  4.26917874e+01  1.69858912e+02
  7.28416419e+01 -2.01029790e+01 -6.36582352e+01 -1.59008021e+02
 -1.15861279e+02]
[ 3.75822289e+00  1.22064500e+02  8.07277912e+01  1.41202891e+02
```

Fig 5.7 coordinates of human individually

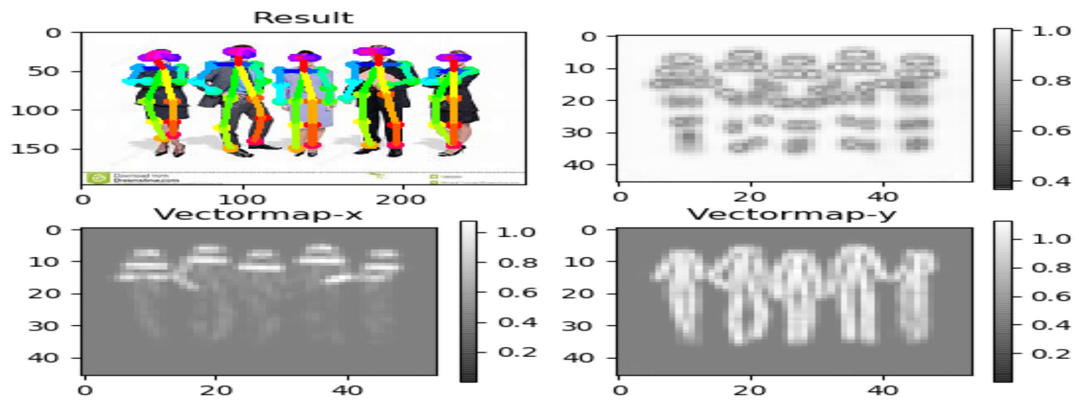


Fig 5.8 Vector plots of human

CHAPTER-6

CONCLUSION

CONCLUSION

- Video frames are analysed by applying R-CNN algorithm and transformed in to 3d space by which coordinates are obtained.
- These co-ordinates should be given to the trained model of data sets having people performing different actions and movements there by actions or movements of human are identified.
- This project is made by using pre-trained weights of coco dataset, as training a neural network requires heavy use of computational power by GPUs.
- The project is 80 percent completed up to obtaining of coordinates of humans in a frame, for remaining 20 percent to complete, high end GPUs and human movement data sets are required.

CHAPTER-7

FUTURE SCOPE

FUTURE SCOPE

- Human detection, Segmentation, Skelenatisation is performed and coordinates are obtained.
- A model can be trained with the coordinates obtained by the frames of different poses and movements of human.
- To this Model, a video can be given, so that it can identify the pose and movement of the human in the video.
- As the human body is segmented from the background and the skeleton, human coordinates are obtained, these coordinates can be calibrated and used to program in multimedia to make animated characters to move like the human.
- The model can be effectively trained and can be employed in Automatic surveillance of CC T.V feeds to detect any suspicious activities.
- The coordinates obtained as a result can be calibrated to move a Tele-presence Robot.
- With the advent of increasing of computing power day by day it is possible to increase speeds algorithm.

CHAPTER-8

REFERENCES

REFERENCES

1. https://arxiv.org/pdf/1611.08050.pdf
2. https://pdfs.semanticscholar.org/63b3/fef3eb52c7d34ad1deb1bd62ea670771121b.pdf>.
3. https://towardsdatascience.com/what-even-is-computer-vision-531e4f07d7d0
4. Dhriti Sengupta, Merina Kundu, Jayati Ghosh Dastidar, 2014. “Human Shape Variation - Efficient Implementation using Skeleton”, IJACR, Vol-4, No-1, Issue-14, pg-145-150, March-2014.
5. Blum, H. 1973. "Biological shape and Visual Science", J. Theoretical Biology, Vol 38, pp 205-287.
6. Comaniciu, D. and Ramesh, V. 2000. “Robust detection and tracking of human faces with an active camera.”, IEEE International Workshop on Visual Surveillance.
7. Darrell, T., Gordon, G., Harwille, M. and Woodfill, J. 1998. “Integrated person tracking using stereo, color, and pattern recognition.” In CVPR, pages 601–609.
8. Haritaoglu, I., Harwood, D. and Davis, L. 2000. “Realtime surveillance of people and their activities” PAMI, 22(8):809–830.
9. August, J., Siddiqi, K. and Zucker, S. 1999. "Ligature Instabilities and the Perceptual Organization of Shape", Comp. Vision and Image Understanding, Vol 76, No. 3, pp 231--243.
10. https://in.mathworks.com/matlabcentral/answers/39545-detect-a-human-figure-in-a-live-video

11. http://www.ijrsset.org/pdfs/v1-i5/3.pdf
12. https://www.theengineeringprojects.com/2016/07/motion-detection-matlab.html
13. Bai, X., Latecki L. J. and Liu, W.-Y. 2007. "Skeleton Pruning by Contour Partitioning with Discrete Curve Evolution", IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI), 29(3), pp. 449-462.
14. Zhu, S. C. and Yullie, A. L. 1996. "Forms: A Flexible Object Recognition and Modeling System", IJCV, 20(3):187–212.
15. Siddiqi, K., Shokoufandeh, A., Dickinson, S. and Zucker, S. 1999. "Shock graphs and shape matching", IJCV, 35(1):13–32. [11] Pizer, S. et al. 2003. "Deformab
16. M. Andriluka, S. Roth, and B. Schiele. Monocular 3D pose estimation and tracking by detection. In CVPR, 2010. 1
17. V. Belagiannis and A. Zisserman. Recurrent human pose estimation. In 12th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), 2017.
18. A. Bulat and G. Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In ECCV, 2016.
19. X. Chen and A. Yuille. Articulated pose estimation by a graphical model with image dependent pairwise relations. In NIPS, 2014.
20. P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. In IJCV, 2005.
21. G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. Using k-poselets for detecting people and localizing their keypoints. In CVPR, 2014.
22. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.
23. E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele. Deepercut: A deeper, stronger, and faster multiperson pose estimation model. In ECCV, 2016. 1, 5, 6

24. U. Iqbal and J. Gall. Multi-person pose estimation with local joint-to-person associations. In ECCV Workshops, Crowd Understanding, 2016.
25. S. Johnson and M. Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In BMVC, 2010.
26. H. W. Kuhn. The hungarian method for the assignment problem. In Naval research logistics quarterly. Wiley Online Library, 1955.
27. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft COCO: com- ´ mon objects in context. In ECCV, 2014.
28. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. Ssd: Single shot multibox detector. In ECCV, 2016.
29. A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In ECCV, 2016.
30. W. Ouyang, X. Chu, and X. Wang. Multi-source deep learning for human pose estimation. In CVPR, 2014.
31. G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, and K. Murphy. Towards accurate multi-person pose estimation in the wild. arXiv preprint arXiv:1701.01779, 2017.
32. T. Pfister, J. Charles, and A. Zisserman. Flowing convnets for human pose estimation in videos. In ICCV, 2015.
33. L. Pishchulin, M. Andriluka, P. Gehler, and B. Schiele. Poselet conditioned pictorial structures. In CVPR, 2013.
34. L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. Gehler, and B. Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In CVPR, 2016.
35. L. Pishchulin, A. Jain, M. Andriluka, T. Thormahlen, and ¨ B. Schiele. Articulated people detection and pose estimation: Reshaping the future. In CVPR, 2012.
36. V. Ramakrishna, D. Munoz, M. Hebert, J. A. Bagnell, and Y. Sheikh. Pose machines: Articulated pose estimation via inference machines. In ECCV, 2014. 1