

Pedestrian Detection Using Compact-CNN

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

Ch Srujana,Deepika (38110095,38110309)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

**(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

MARCH - 2020



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Ch Srujana, Deepika (38110095, 38110309)** who carried out the project entitled "**PEDESTRIAN DETECTION USING COMPACT-CNN**" under my supervision from November 2019 to March 2020.

Internal Guide

Ms. Brumancia E. M.E.,

Head of the Department

Submitted for Viva-Voce Examination held on:

Internal Examiner

External Examiner

DECLARATION

I **BATHALA MARUTHI PRASAD (36110151)**, hereby declare that the Project Report entitled “**PEDESTRIAN DETECTION USING COMPACT-CNN**” done by me under the guidance of **Ms.Brumancia E. M.**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge our sincere thanks to **BOARD OF MANAGEMENT** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA M.E., Ph.D.**, Dean, School Of Computing, **Dr. S. VIGNESHWARI M.E., Ph.D.**, and **Dr. L. LAKSHMANAN M.E., Ph.D.**, Heads Of the Department, Department of Computer Science and Engineering for providing us the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.Brumancia E. M.E.**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express our thanks to all Teaching and Non-teaching staff of the **DEPARTMENT of COMPUTER SCIENCE & ENGINEERING** who were helpful in many ways for the completion of the project.

ABSTRACT

Image Classification and detecting objects through machine learning is one of the advancements in modern technology. Pedestrian Detection is also one of those extended applications. However, there are many flaws in the existing systems of pedestrian detection. An algorithm we proposed will benefit the flaws in existing systems. Where we can detect the pedestrians using our system without a single flaw.

In our system we use convolutional neural network (CNN) for image classification and R-CNN (Regional CNN) for object detection and IOU helps to measure the accuracy of algorithms used for object detection using bounding boxes. Thus output is validated by detecting the pedestrians exactly. This training cascade classifier improves the machine to overcome the false detections near pedestrians. Experimental results on two widely used pedestrian datasets demonstrate that the proposed training strategy and the CNN based detector can effectively improve the detection rate and the localization accuracy using fewer parameters.

TABLE OF CONTENTS

CHAPTER NO.	CHAPTER NAME	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	iv
1	INTRODUCTION	
	1.1 Aim of the Project	03
	1.2 Outline of Project	03
	1.3 Project Scope	04
	1.4 Types Of Recommender Systems	04
	1.5 System Features	07
2	LITERATURE SURVEY	
	2.1 Literature Survey	03
	2.2 Existing System	04
	2.3 Proposed System	05
	2.4 Drawbacks Of Existing System	05
	2.5 Advantages Of Proposed System	05
3	METHODOLOGY	
	3.1 Introduction To Python	06
	3.1.1 What Can Python Do	06
	3.1.2 Why Python	06
	3.1.3 Good to Know	07
	3.1.4 Python Syntax Compared To Other Language	08
	3.1.5 Python Is Interpreted	08
	3.2 Regional CNN	09
	R-CNN Architecture	10
	Procedure For Object Detection	11

	And Localization In R-CNN	
Convolutional Neural Network	11	
The Problem Space	12	
	Inputs And Outputs	13
	Biological Connection	13
	Structure	14
	First Layer-Math Part	14
	First Layer-High Level	16
	Prespective	
	Disclaimer	18
	Going Deepder Through The	18
	Network	
	3.3.9 Fully Connected Layer	19
	3.3.10 Training	21
	3.3.11 Testing	23
	3.3.12 How Companies Use CNNs	23
4	RESULTS AND DISCUSSION	
Design And Implementation	27	
	Constraints	
	Constraints In Analysics	27
	Constraints In Design	27
	Constraints In Implementation	27
	Other Non Functional Requirements	28
	Performance Requirements	28
	Safety Requirements	28
5	CONCLUSION AND FUTURE WORK	
	Conclusion	29

Future Work	29
REFERENCES	30
APPENDIX	31

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
3.1	R-CNN Architecture	10
3.2	CNN Classification	11
3.3	Internal Processing	12
3.4	Activation Map	15
3.5	Pixel To Visualization	16
3.6	Visuaization Of The Filter On Image	17
3.7	Summation Of Obtained Matrices	17
3.8	Working Of CNN	20
3.9	Recurrent Neural Network	24
3.10	Unrolled Recurrent Neural Network	24
3.11	System Architecture	25
4.1	Actual Output	26

CHAPTER 1

INTRODUCTION

Pedestrian detection has made significant advances benefiting from the region-based convolutional neural networks. The project introduces a strict matching metric, which is beneficial to selecting well-aligned positive samples. A positive sample matches a ground truth only if all its region-IoUs are bigger than a threshold. Secondly, an improved negative example selection strategy using both the classification and localization information is proposed to mine hard negative examples, which can further suppress the false positive detections near the pedestrians. Recently, pedestrian detection has made significant advances benefiting from the region-based convolutional neural networks.

However, training R-CNN with a holistic intersection over union always brings many flawed positive samples. This paper introduces a strict matching metric, which is beneficial to selecting well-aligned positive samples. Specifically, this matching metric is defined on a set of region-IoUs instead of a holistic IoU, which considers the alignments of different part regions in a whole bounding box simultaneously.

A positive sample matches a ground truth only if all its region-IoUs are bigger than a threshold. Secondly, an improved negative example selection strategy using both the classification and localization information is proposed to mine hard negative examples, which can further suppress the false positive detections near the pedestrians. Based on the proposed sample selection strategy, a cascade compact convolutional neural network is proposed for accurate pedestrian detection. Each stage of the CC-CNN is constructed with a compact network that only consists of a small number of parameters, thus making the detector suitable to be implemented on onboard embedded systems.

To get well-aligned positive samples, this paper proposes a novel positive example selection strategy based on the region wise metric, i.e., region-IoU which considers the alignments of different part regions in a whole bounding box simultaneously. First, this strategy defines a pedestrian template and a group of regions based on the importance of pedestrian parts. Then a proposal bounding box is treated as a positive sample if all its regions match the corresponding ones of any ground truth

with IoUs higher than a threshold. By this way, the well-aligned positive samples with the existence of important parts are collected and used for training.

These unsatisfied performances are mainly due to two reasons. First, the training methodologies, especially the positive example selection strategy, are designed for general object detection, which are not appropriate for a specific category, such as pedestrian detection. During the training process of single-stage/two-stages CNN-based detectors most researchers select the positive examples whose IoUs between anchors/proposals and ground truths are higher than a given threshold. Traditional pedestrian detectors treat the ground truth bounding boxes as positive examples during training. Different from them, the single-stage/two-stages CNN based detectors select positive examples based on the overlap between anchor/proposal and ground truth

To further suppress the false positives near the pedestrians, we also proposes an improved negative example selection strategy which selects negative samples using the fused score that combines both the classification score and the overlap value with the ground truth.

THE MAIN CONTRIBUTIONS

(1) We are here to investigate the errors of the state-of-the art R-CNN based pedestrian detector. The localization errors and the background errors are two main failures for current R-CNN based pedestrian detectors. These false detections are mainly caused by the flawed training samples extracted by the holistic IoU.

(2) A region-IoU based positive example selection strategy is introduced to get well-aligned positive samples. Besides, the hard negative example mining method is improved by combining both classification and localization information.

(3) A cascade compact network is proposed for pedestrian detection, in which each stage is constructed by a compact network. It gains a high detection rate and localization accuracy.

(4) The proposed detector achieves leading performances among the state-of-the-art detectors on different data sets. Especially, it gains a significant improvement on detection performance under the stricter localization criterion.

CHAPTER 2

LITERATURE SURVEY

LITERATURE SURVEY

As a method to achieve real-time detection with high accuracy, cascade has been widely used in pedestrian detection. They use a soft-cascade approach to transform a classifier into cascaded ones to gain a detector with high detection performance and efficiency. Despite improvements in detection performance, it brings too many parameters when multiple stages are included.

The proposed detector is similarly constructed in the form of cascade R-CNN framework. Differently, benefiting from the modified example selection strategy, a compact network is designed for each attached stage, which gains a similar detection performance but saves the memory costs. As a method to achieve real-time detection with high accuracy, cascade has been widely used in pedestrian detection. propose of cascade face detection architecture built on CNNs operating at multiple resolutions. A chained cascade network for object detection, where the early cascade stages are used to reject obvious negative detections and the attached contextual stages are used to learn more discriminative features for handling more difficult examples.

Both of them need extra proposal generation modules. propose of cascade R-CNN architecture for high quality object detection. Despite improvements in detection performance it brings too many parameters when multiple stages are included. The proposed detector is similarly constructed in the form of cascade R-CNN framework.

As a method to achieve real-time detection with high accuracy, cascade has been widely used in pedestrian detection. They use a soft-cascade approach to transform a classifier into cascaded ones to gain a detector with high detection performance and efficiency. Most of the cascaded pedestrian detectors use hand-crafted features, such as Haar-like features and HOG. Recently, the combination of the cascade framework and CNNs has been studied in face detection and general object detection. The following content has been studied in order to develop the accurate detection using cnn, which is used here to develop the accurate pedestrian detection.

TITLE: STRENGTHENING THE EFFECTIVENESS OF PEDESTARIAN DETECTION WITH SPATIAL POOLED FEATURES

AUTHORS: Sakrapee Paisitkriangkrai, Chunhua Shen, and Anton van den Hengel

YEAR: 2017

DESCRIPTION: We propose a simple yet effective approach to the problem of pedestrian detection which outperforms the current state-of-the-art. Our new features are built on the basis of low-level visual features and spatial pooling. Incorporating spatial pooling improves the translational invariance and thus the robustness of the detection process. We then directly optimise the partial area under the ROC curve (pAUC) measure, which concentrates detection performance in the range of most practical importance. The combination of these factors leads to a pedestrian detector which outperforms all competitors on all of the standard benchmark datasets. We advance state-of-the-art results by lowering the average miss rate from 13% to 11% on the INRIA benchmark, 41% to 37% on the ETH benchmark, 51% to 42% on the TUD-Brussels benchmark and 36% to 29% on the Caltech-USA benchmark.

TITLE: AN EMPIRICAL STUDY OF PEDESTRIAN DETECTION TECHNIQUES WITH DIFFERENT IMAGE RESOLUTION

AUTHORS: Govardhan.S.D, Vasuki.A

YEAR: 2018 International Journal of Engineering and Advanced Technology (IJEAT)

DESCRIPTION: The designed technique identifies the pedestrian when the whole body of pedestrians appears in system. Pedestrian detection is an essential issue in many applications like robotics, video surveillance and computerized driver assistance. Pedestrian detection techniques are enhanced one, but the detection results of current systems are not adequate to be used in practice. The key issue is to identify the pedestrians, despite large inconsistency of shape and appearance caused by variations in pose, clothing, occlusion and lighting. The detection rate reduces when vehicle or surrounding facilities cover person body parts. For addressing these issues, many pedestrian detection techniques are employed for detecting part of an object.

EXISTING SYSTEM

R-CNN analysis shows that localization error and background error are still the two main errors for Faster R-CNN based detectors. In the following sections, the proposed approach will deal with these two types of errors by introducing a new example selection strategy and a cascade compact CNN. Besides, the annotations of city person's validation data set are manually re-annotated for fairer comparison of different pedestrian detectors. Two types of errors are investigated in this section: false positives and false negatives. In this system there are lots of problems regarding to positive selection and negative selection, they are been mistakenly overlapping each other to overcome this method more efficient way has to be developed.

PROPOSED SYSTEM

A cascade compact convolutional neural network (CC-CNN) is proposed for accurate pedestrian detection. CC-CNN based detector can effectively improve the detection rate and the localization accuracy using fewer parameters. The experiment is conducted on the City persons reasonable validation subset. All the false positives are clustered into two categories: background errors and localization errors, To simultaneously eliminate both localization and background errors, a new measurement is introduced to select the negative samples. Specifically, the score of each negative.

DRAWBACKS OF EXISTING SYSTEM:

1. It provides plenty of distractions
2. It is an time effective method
3. Most of the pedestrians are not detected
4. Positives are also recognised as negatives

ADVANTAGES OF PROPOSED SYSTEM:

1. Improved Image Classification
2. Improved Image Interpretation
3. Improved video classification Performance
4. Efficient at Delivering High-quality Results
5. Reduces the need for feature engineering
6. Strength to pure differences in the data is automatically learned

CHAPTER 3

METHODOLOGY

INTRODUCTION TO PYTHON

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- System scripting.

WHAT CAN PYTHON DO?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

WHY PYTHON?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python is a language that we can use in integrated systems efficiently.

- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

GOOD TO KNOW

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- Python 2.0 was released in 2000, and the 2.x versions were the prevalent releases until December 2008. At that time, the development team made the decision to release version 3.0, which contained a few relatively small but significant changes that were not backward compatible with the 2.x versions. Python 2 and 3 are very similar, and some features of Python 3 have been backported to Python 2. But in general, they remain not quite compatible.
- Both Python 2 and 3 have continued to be maintained and developed, with periodic release updates for both. As of this writing, the most recent versions available are 2.7.15 and 3.6.5. However, an official End Of Life date of January 1, 2020 has been established for Python 2, after which time it will no longer be maintained.
- Python is still maintained by a core development team at the Institute, and Guido is still in charge, having been given the title of BDFL (Benevolent Dictator For Life) by the Python community. The name Python, by the way, derives not from the snake, but from the British comedy troupe Monty Python's Flying Circus, of which Guido was, and presumably still is, a fan. It is common to find references to Monty Python sketches and movies scattered throughout the Python documentation.
- It is possible to write Python in an Integrated Development Environment, such

as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

PYTHON SYNTAX COMPARED TO OTHER LANGUAGES

- Python was designed to for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

PYTHON IS INTERPRETED

- Many languages are compiled, meaning the source code you create needs to be translated into machine code, the language of your computer's processor, before it can be run. Programs written in an interpreted language are passed straight to an interpreter that runs them directly.
- This makes for a quicker development cycle because you just type in your code and run it, without the intermediate compilation step.
- One potential downside to interpreted languages is execution speed. Programs that are compiled into the native language of the computer processor tend to run more quickly than interpreted programs. For some applications that are particularly computationally intensive, like graphics processing or intense number crunching, this can be limiting.
- In practice, however, for most programs, the difference in execution speed is measured in milliseconds, or seconds at most, and not appreciably noticeable to a human user. The expediency of coding in an interpreted language is typically worth it for most applications.
- For all its syntactical simplicity, Python supports most constructs that would be expected in a very high-level language, including complex dynamic data types,

structured and functional programming, and object-oriented programming.

- Additionally, a very extensive library of classes and functions is available that provides capability well beyond what is built into the language, such as database manipulation or GUI programming.
- Python accomplishes what many programming languages don't: the language itself is simply designed, but it is very versatile in terms of what you can accomplish with it.

REGIONAL CNN

In this chapter we will talk about the Region based methods for object detection. In particular, R-CNN (Regional CNN), the original application of CNNs to this problem, along with its descendants Fast R-CNN, and Faster R-CNN. In classification, there's generally an image with a single object as the focus and the task is to say what that image is. But when we look at the world around us, we carry out far more complex tasks. We see complicated sights with multiple overlapping objects, and different backgrounds and we not only classify these different objects but also identify their boundaries, differences, and relations to one another. The goal of a Region based CNN is to take an image input to detected and localize the object in the image. All three above mentioned architectures were used for object detection and localization for a driving scenario to detect cars, pedestrians and traffic signs etc. and performed a time study to conclude which of the three architectures can be used as a detector for a real-world driving scenario for detection of objects present on road. We will also discuss their architecture and describe how they perform the task of object detection and localization given the input image.

R-CNN was first introduced by Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik in 2014. Region based Convolutional Network(R-CNN) combines two ideas:

- 1) one can apply high-capacity Convolutional Networks (CNN'S) to bottom-up region proposals in order to localize and segment objects.
- 2) When labelling data is scarce, supervised pre-training for an auxiliary task, followed

by domain-specific-fine-tuning, boosts performance significantly. They combined region proposals with CNNs and called the resulting model an R-CNN or Region-based Convolutional Network.

R-CNN ARCHITECTURE.

In R-CNN we input an image and then using the basic feature detection techniques such as edge detection etc. we get Region Proposals Also referred as Regions of interest. This process is also known as selective search.

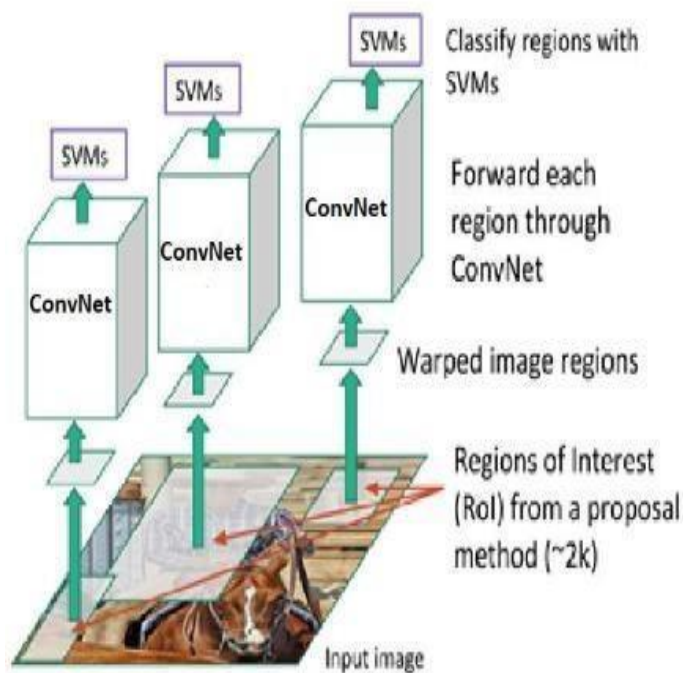


Fig:3.1 R-CNN Architecture

In the next step after these region proposals are extracted these wrapped image regions go through a trained CNN in our implementation AlexNet and then on the final layer the classification is done Using a Support Vector Machine(SVM) classifier which classifies whether this is an object and if so what. Pictorial representation of above process is shown in Figure.3.2

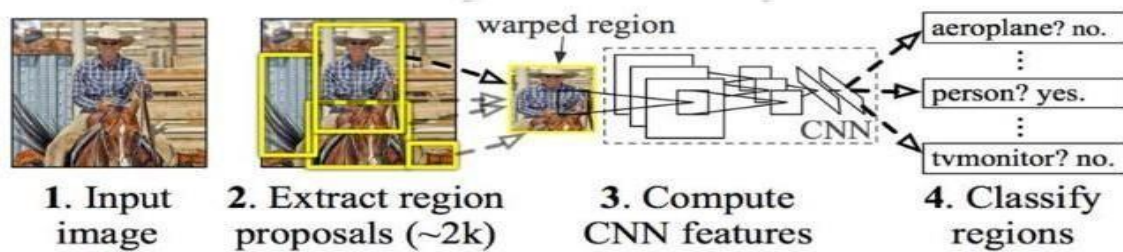


FIG:3.2 CNN CLASSIFICATION

PROCEDURE FOR OBJECT DETECTION AND LOCALIZATION IN R-CNN

R-CNN works really well but is really quite slow for implementation in real time object detection. The reasons being

1. It requires a forward pass of the CNN (AlexNet) for every single region proposal for every single image that's around 2000 forward passes per image.
2. It must train three different models separately -the CNN to generate image features, the classifier that predicts the class and the regression model to tighten the bounding boxes. This makes the pipeline extremely hard to train.

CONVOLUTIONAL NEURAL NETWORK (CNN)

Convolutional neural networks (CNN) sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure. However, the classic, and arguably most popular, use case of these networks is for image

processing. Within image processing, let's take a look at how to use these CNNs for image classification.

THE PROBLEM SPACE

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.



What We See

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 33 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 24 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

What Computers See

FIG:3.3 INTERNAL PROCESSING

INPUTS AND OUTPUTS

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The

representative array will be $480 \times 480 \times 3$. Each of these numbers is given a value from 0 to 255 which describe the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for cat, .15 for dog, .05 for bird, etc).

Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat. This is the process that goes on in our minds subconsciously as well. When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts through a series of convolutional layers. This is a general overview of what a CNN does. Let's get into the specifics.

BIOLOGICAL CONNECTION

But first, a little background. When you first heard of the term convolutional neural networks, you may have thought of something related to neuroscience or biology, and you would be right. Sort of. CNNs do take a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 ([Video](#)) where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. For example, some neurons fired when exposed to vertical edges and some when shown horizontal or diagonal edges. Hubel and Wiesel found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception. This idea of specialized components inside of a system having specific tasks(the neuronal cells in the visual cortex looking for specific

characteristics) is one that machines use as well, and is the basis behind CNNs.

STRUCTURE

Back to the specifics. A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (down sampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part understands what each of these layers do. So let's get into the most important one.

FIRST LAYER-MATH PART

The first layer in a CNN is always a Convolutional Layer. First thing to make sure you remember is what the input to this conv (I'll be using that abbreviation a lot) layer is. Like we mentioned before, the input is a $32 \times 32 \times 3$ array of pixel values. Now, the best way to explain a conv layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5×5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a filter (or sometimes referred to as a neuron or a kernel) and the region that it is shining over is called the receptive field. Now this filter is also an array of numbers (the numbers are called weights or parameters). A very important note is that the depth of this filter has to be the same as the depth of the input (this makes sure that the math works out), so the dimension of this filter is $5 \times 5 \times 3$. Now, let's take the first position the filter is in for example. It would be the top left corner. As the filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing element wise multiplications). These multiplications are all summed up (mathematically speaking, this would be 75 multiplications in total). So now you have a single number. Remember, this number is just representative of when the filter is at the top left of the image. Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right

again by 1, and so on). Every unique location on the input volume produces a number. After sliding the filter over all the locations, you will find out that what you're left with is a $28 \times 28 \times 1$ array of numbers, which we call an activation map or feature map. The reason you get a 28×28 array is that there are 784 different locations that a 5×5 filter can fit on a 32×32 input image. These 784 numbers are mapped to a 28×28 array.

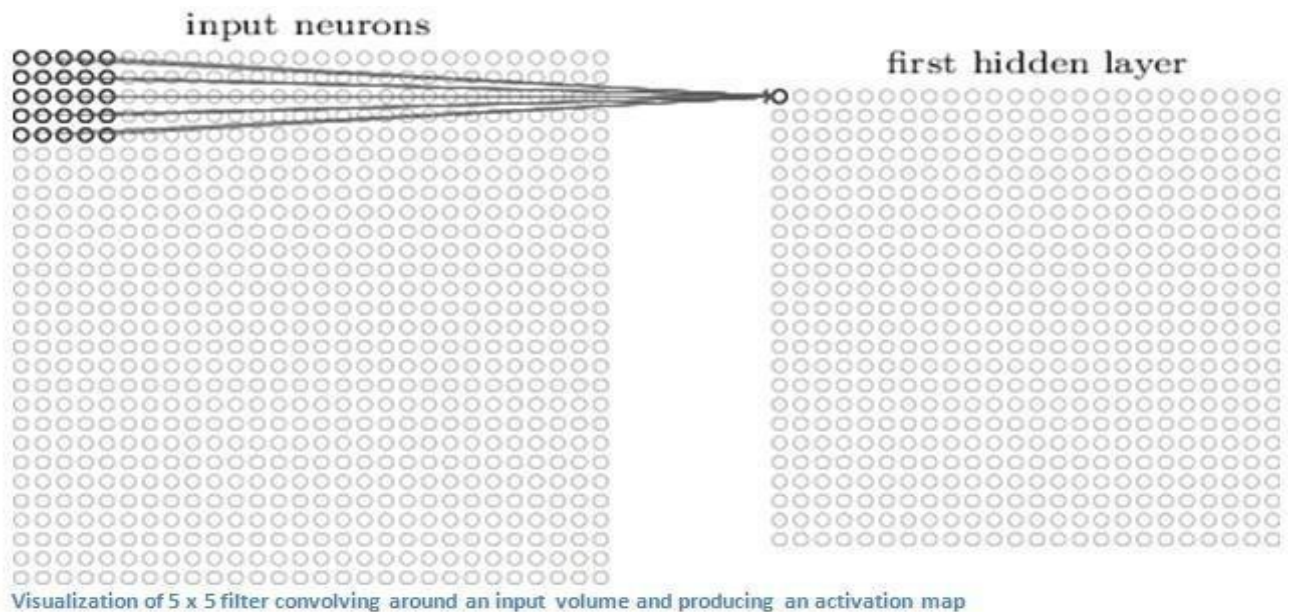


FIG: 3.4 ACTIVATION MAP

Let's say now we use two $5 \times 5 \times 3$ filters instead of one. Then our output volume would be $28 \times 28 \times 2$. By using more filters, we are able to preserve the spatial dimensions better. Mathematically, this is what's going on in a convolutional layer.

FIRST LAYER – HIGH LEVEL PRESPECTIVE

However, let's talk about what this convolution is actually doing from a high level. Each of these filters can be thought of as feature identifiers. When I say features, I'm talking about things like straight edges, simple colors, and curves. Think about the simplest characteristics that all images have in common with each other. Let's say our first filter is $7 \times 7 \times 3$ and is going to be a curve detector. As a curve detector, the filter will have a pixel structure in which there will be higher numerical values along the area

that is a shape of a curve (Remember, these filters that we're talking about as just numbers!).

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

FIG 3.5 PIXEL TO VISUALIZATION

Now, let's go back to visualizing this mathematically. When we have this filter at the top left corner of the input volume, it is computing multiplications between the filter and pixel values at that region. Now let's take an example of an image that we want to classify, and let's put our filter at the top left corner.

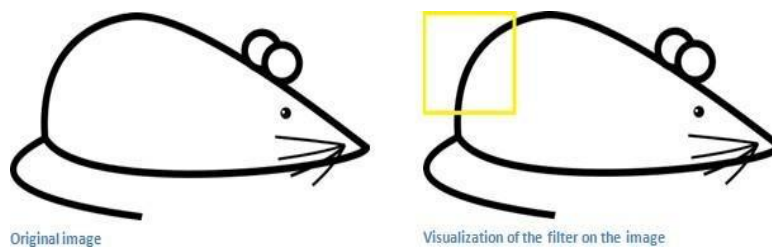


FIG: 3.6 VISUALIZATION OF THE FILTER ON IMAGE

Remember, what we have to do is multiply the values in the filter with the original pixel values of the image.

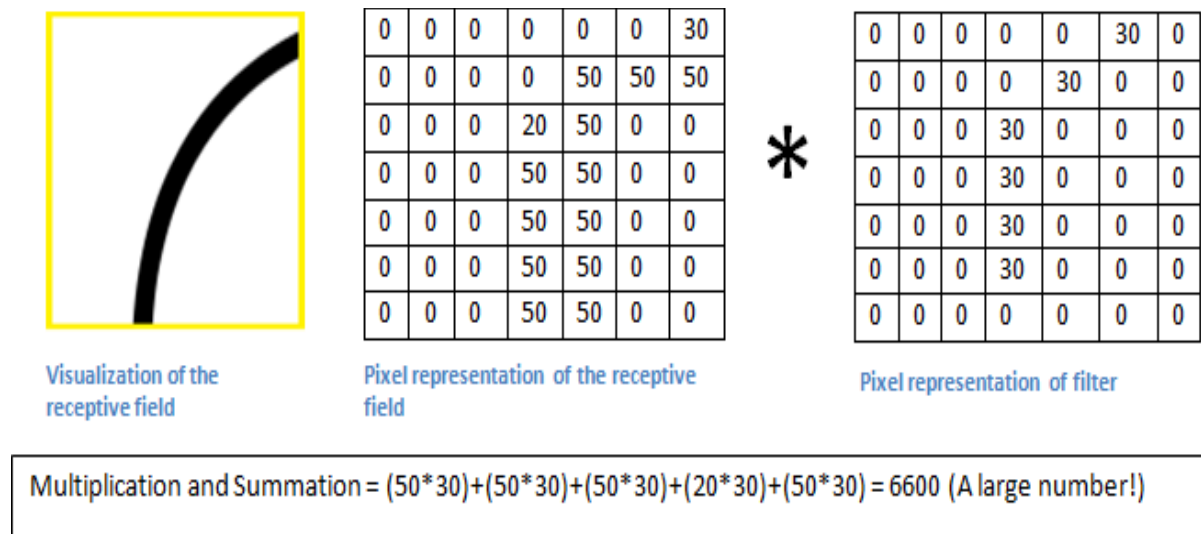


FIG: 3.7 SUMMATION OF OBTAINED MATRICES

Basically, in the input image, if there is a shape that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value! Now let's see what happens when we move our filter.

The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map. So, in the simple case of a one filter convolution (and if that filter is a curve detector), the activation map will show the areas in which there are most likely to be curves in the picture. In this example, the top left value of our 26 x 26 x 1 activation map (26 because of the 7x7 filter instead of 5x5) will be 6600. This high value means that it is likely that there is some sort of curve in the input volume that caused the filter to activate. The top right value in our activation map will be 0 because there wasn't anything in the input volume that caused the filter to activate (or more simply said, there wasn't a curve in that region of the original image). Remember, this is just for one filter. This is just a filter that is going to detect lines that curve outward and to the right. We can have other filters for lines that curve to the left or for straight edges. The more filters, the greater the depth of the activation map, and the more information we have about the input volume.

DISCLAIMER:

The filter I described in this section was simplistic for the main purpose of describing the math that goes on during a convolution. In the picture below, you'll see some examples of actual visualizations of the filters of the first conv layer of a trained network. Nonetheless, the main argument remains the same. The filters on the first layer convolve around the input image and "activate" (or compute high values) when the specific feature it is looking for is in the input volume.

GOING DEEPER THROUGH THE NETWORKS

Now in traditional convolutional neural network architecture, there are other layers that are interspersed between these conv layers. I'd strongly encourage those interested to read up on them and understand their function and effects, but in a general sense, they provide nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting. A classic CNN architecture would look like this.

The last layer, however, is an important one and one that we will go into later on. Let's just take a step back and review what we've learned so far. We talked about what the filters in the first conv layer are designed to detect. They detect low level features such as edges and curves. As one would imagine, in order predicting whether an image is a type of object, we need the network to be able to recognize higher level features such as hands or paws or ears. So let's think about what the output of the network is after the first conv layer. It would be a $28 \times 28 \times 3$ volume (assuming we use three $5 \times 5 \times 3$ filters). When we go through another conv layer, the output of the first conv layer becomes the input of the 2nd conv layer. Now, this is a little bit harder to visualize. When we were talking about the first layer, the input was just the original image. However, when we're talking about the 2nd conv layer, the input is the activation map(s) that result from the first layer. So each layer of the input is basically describing the locations in the original image for where certain low level features appear. Now when you apply a set of filters on top of that (pass it through the 2nd conv layer), the

output will be activations that represent higher level features. Types of these features could be semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As you go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, you may have some filters that activate when there is handwriting in the image, filters that activate when they see pink objects, etc. Another interesting thing to note is that as you go deeper into the network, the filters begin to have a larger and larger receptive field, which means that they are able to consider information from a larger area of the original input volume (another way of putting it is that they are more responsive to a larger region of pixel space).

FULLY CONNECTED LAYER

Now that we can detect these high level features, the icing on the cake is attaching a **fully connected layer** to the end of the network. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. For example, if the resulting vector for a digit classification program is [0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is a 1, a 10% probability that the image is a 2, a 75% probability that the image is a 3, and a 5% probability that the image is a 9 (Side note: There are other ways that you can represent the output, but I am just showing the softmax approach). The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class. For example, if the program is predicting that some image is a dog, it will have high values in the activation maps that represent high level features like a paw or 4 legs, etc. Similarly, if the program is predicting that some image is a bird, it will have high values in the activation maps that represent high level features like wings or a beak, etc. Basically, a FC layer looks at what high level features most strongly correlate to a

particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.

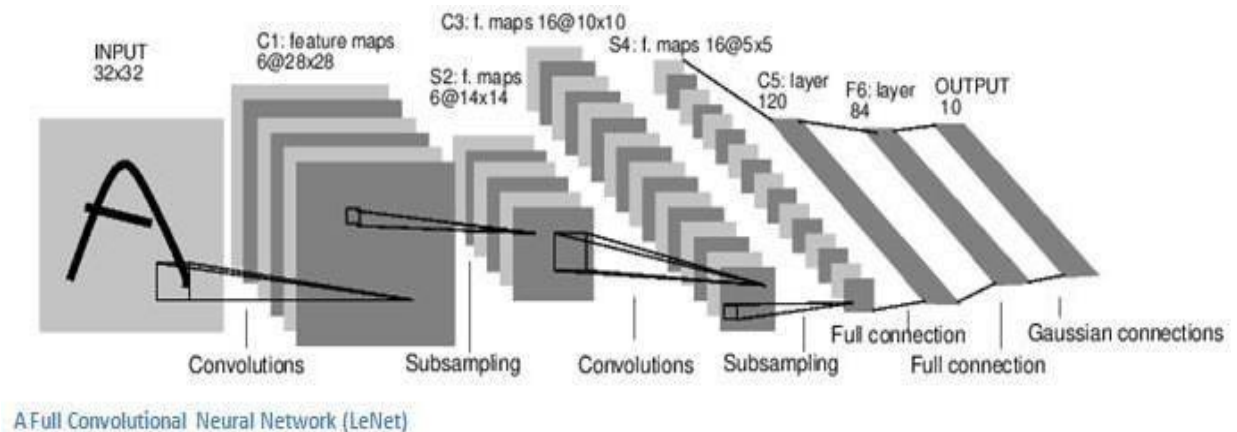


FIG: 3.8 WORKING OF CONVOLUTIONAL NEURAL NETWORK

TRAINING

Now, this is the one aspect of neural networks that I purposely haven't mentioned yet and it is probably the most important part. There may be a lot of questions you had while reading. How do the filters in the first conv layer know to look for edges and curves? How does the fully connected layer know what activation maps to look at? How do the filters in each layer know what values to have? The way the computer is able to adjust its filter values (or weights) is through a training process called **backpropagation**.

Before we get into backpropagation, we must first take a step back and talk about what a neural network needs in order to work. At the moment we all were born, our minds were fresh. We didn't know what a cat or dog or bird was. In a similar sort of way, before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. The filters in the higher layers don't know to look for paws and beaks. As we grew older however, our parents and teachers showed us different pictures and images and gave us a corresponding label. This idea of being

given an image and a label is the training process that CNNs go through. Before getting too into it, let's just say that we have a training set that has thousands of images of dogs, cats, and birds and each of the images has a label of what animal that picture is. Back to backprop.

So backpropagation can be separated into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the **forward pass**, you take a training image which as we remember is a 32 x 32 x 3 array of numbers and pass it through the whole network. On our first training example, since all of the weights or filter values were randomly initialized, the output will probably be something like [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1], basically an output that doesn't give preference to any number in particular. The network, with its current weights, isn't able to look for those low level features or thus isn't able to make any reasonable conclusion about what the classification might be. This goes to the **loss function** part of backpropagation. Remember that what we are using right now is training data. This data has both an image and a label. Let's say for example that the first training image inputted was a 3. The label for the image would be [0 0 0 1 0 0 0 0 0]. A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is $\frac{1}{2}$ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2} (target - output)^2$$

Let's say the variable L is equal to that value. As you can imagine, the loss will be extremely high for the first couple of training images. Now, let's just think about this intuitively. We want to get to a point where the predicted label (output of the ConvNet) is the same as the training label (This means that our network got its prediction right). In order to get there, we want to minimize the amount of loss we have. Visualizing this as just an optimization problem in calculus, we want to find out which inputs (weights in our case) most directly contributed to the loss (or error) of the network.

This is the mathematical equivalent of a dL/dW where W are the weights at a particular layer. Now, what we want to do is perform a **backward pass** through the network, which is determining which weights contributed most to the loss and finding ways to adjust them so that the loss decreases. Once we compute this derivative, we then go to the last step which is the **weight update**. This is where we take all the weights of the filters and update them so that they change in the opposite direction of the gradient.

$$W = W_i - \eta \frac{dL}{dW}$$

w = Weight
w_i = Initial Weight
η = Learning Rate

The **learning rate** is a parameter that is chosen by the programmer. A high learning rate means that bigger steps are taken in the weight updates and thus, it may take less time for the model to converge on an optimal set of weights. However, a learning rate that is too high could result in jumps that are too large and not precise enough to reach the optimal point.

The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program will repeat this process for a fixed number of iterations for each set of training images (commonly called a batch). Once you finish the parameter update on the last training example, hopefully the network should be trained well enough so that the weights of the layers are tuned correctly.

TESTING

Finally, to see whether or not our CNN works, we have a different set of images and labels (can't double dip between training and test!) and pass the images through the CNN. We compare the outputs to the ground truth and see if our network works!

HOW COMPANIES USE CNN's

Data, data, data. The companies that have lots of this magic 4 letter word are the ones that have an inherent advantage over the rest of the competition. The more training data that you can give to a network, the more training iterations you can make, the more weight updates you can make, and the better tuned to the network is when it goes to production. Facebook (and Instagram) can use all the photos of the billion users it currently has, Pinterest can use information of the 50 billion pins that are on its site, Google can use search data, and Amazon can use data from the millions of products that are bought every day. And now you know the magic behind how they use it.

RECURRENT NEURAL NETWORK

Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

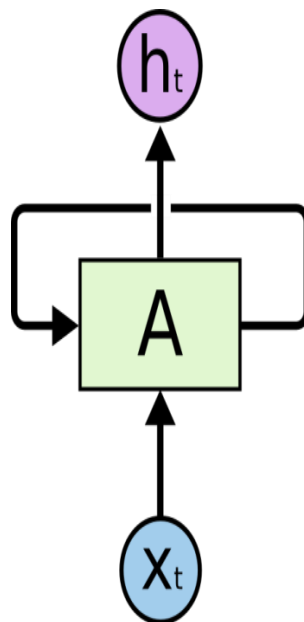


Fig: 3.9 RECURRENT NEURAL NETWORK.

In the above diagram, a chunk of neural network, $\mathcal{A}$, looks at some input x_t and outputs a value h_t . A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

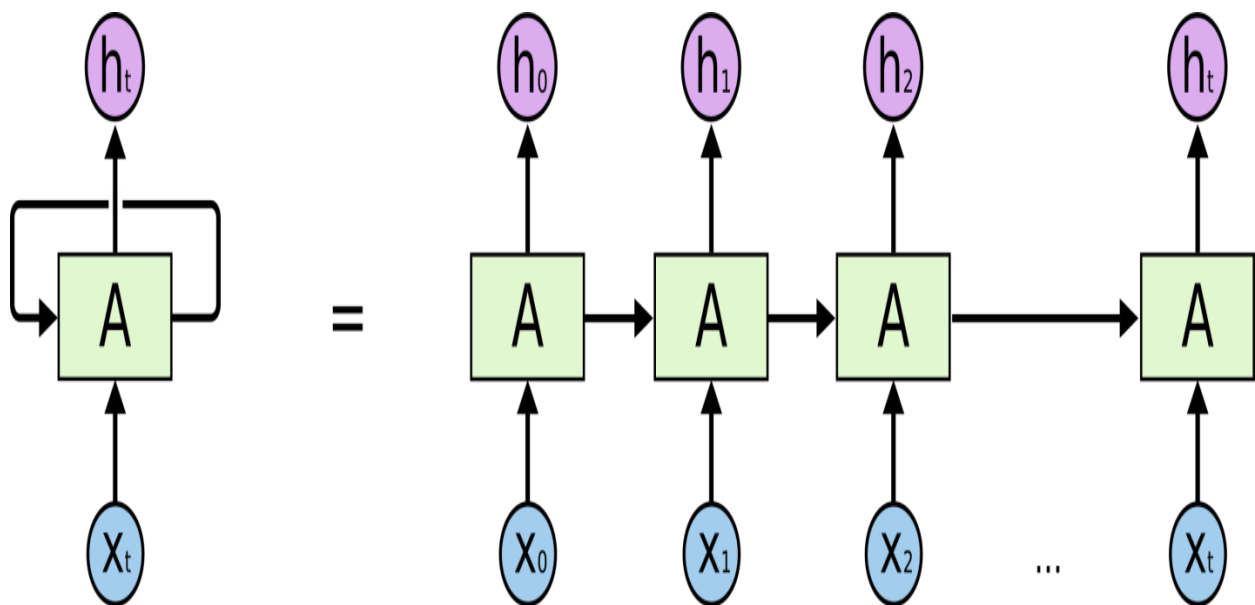


Fig: 3.10 UNROLLED RECURRENT NEURAL NETWORK

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data. And they certainly are used! In the last few years, there has been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning... The list goes on. But they really are pretty amazing. Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

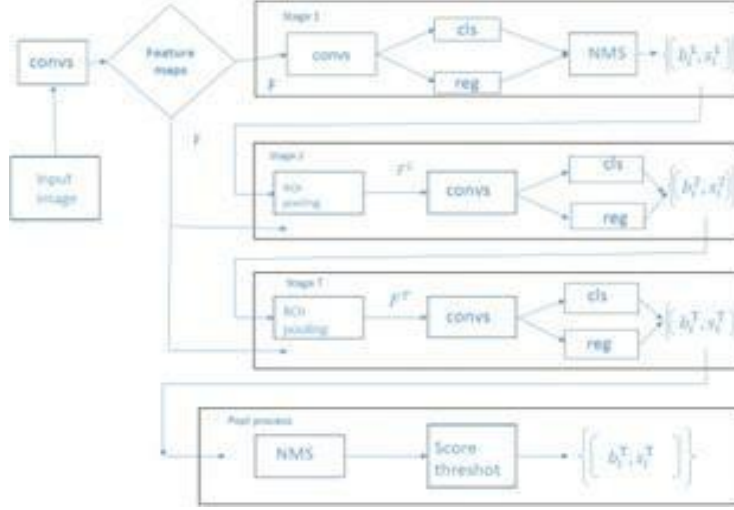


FIG: 3.11 SYSTEM ARCHITECTURE

The cascade CNN detector with a few no of parameters has more advantages for applications on embedded systems. Extensive experiments demonstrate that the compact network can achieve competitive performances Finally, the proposed approach achieves superior performances on two widely used datasets compared to other state-of-the-art methods, which proves that the proposed example selection methodology and the cascade compact CNN can effectively improve both the detection rate and the localization accuracy.

A cascade compact CNN framework is proposed to improve the detection rate and the localization accuracy of the pedestrian detection. The structure is shown in Fig 3.11 It consists of multiple cascaded networks, where each stage is fed with the shared feature maps F and the bounding boxes from the previous stage. Due to the good alignment of the positive samples selected based on the region-IoU measurement, each stage is constructed using compact networks.

The box in each stage is estimated based on the previous detection. With the increase of the stage number, the localization of detected box becomes more precise. A pragmatic multi-step training strategy is employed to train the network with multiple stages. The networks are initialized using the parameters from the pre-trained model in different stages, and the parameters of all the layers are jointly optimized.

CHAPTER 4

RESULTS AND DISCUSSION

The experiments are performed on python compiler jupyter version. The computations are performed using the video Classification and by comparing to the given inputs which are positives and negative data set. All of the experiments are conducted on the reasonable subsets (height ≥ 50 pixels and overlap $\leq 35\%$) for training and testing. The log miss rate averaged is used to evaluate the performance of detectors, which is computed by averaging miss rate along FPPI evenly log-spaced in the range of $[10^{-2}, 10^0]$.

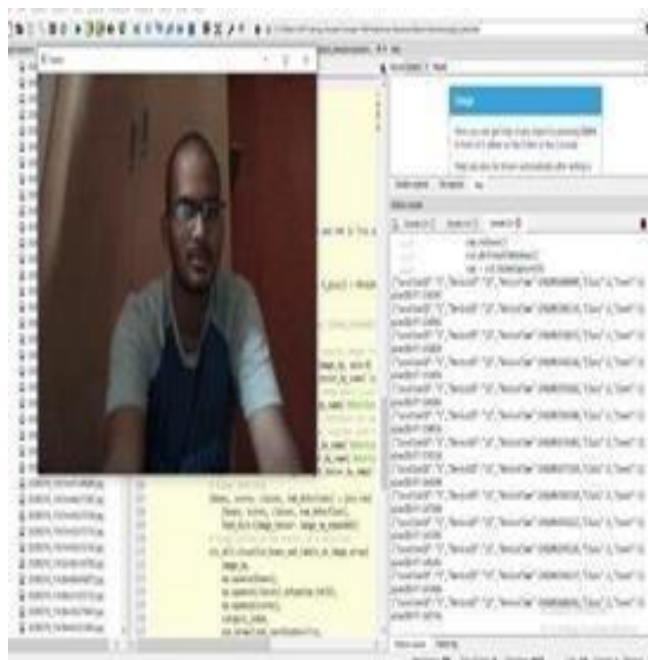


FIG:4.1 ACTUAL OUTPUT

First, the input video from the camera is recorded and it is compared to already existing positives and negatives and increases the count value in the above experiment as only one visual human is existing and the count value is one When this image is passed then the obtained data can be used in various segments such as autonomous cars, public traffic system etc.,

DESIGN AND IMPLEMENTATION CONSTRAINTS

CONSTRAINTS IN ANALYSICS

- Constraints as Informal Text
- Constraints as Operational Restrictions
- Constraints Integrated in Existing Model Concepts
- Constraints as a Separate Concept
- Constraints Implied by the Model Structure
- Constraints involved samples

CONSTRAINTS IN DESIGN

- Determination of the Involved Classes
- Determination of the Involved Objects
- Determination of the Involved Actions
- Determination of the Required Classes
- Global actions and Constraint Realization
- Determination of involved crowd

CONSTRAINTS IN IMPLEMENTATION

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one.

It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease.

There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods. More simpler examples has to be given at start in order to make it

work more efficiently by giving more no of images the mechanism gets adapted in order to pick more positive outputs greater the no of inputs it becomes more complex to get positive outputs.

OTHER NONFUNCTIONAL REQUIREMENTS

PERFORMANCE REQUIREMENTS

The application at this side controls and communicates with the following three main general components.

- embedded browser in charge of the navigation and accessing to the web service;
- Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

SAFETY REQUIREMENTS

1. The software may be safety-critical. If so, there are issues associated with its integrity level
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.
6. Systems with different requirements for safety levels must be separated.
7. Otherwise, the highest level of integrity required must be applied to all systems in the same environment.

CHAPTER 5

CONCLUSION AND FUTURE WORK

CONCLUSION

This paper has presented a novel example selection strategy for training the cascade CNN detector. Instead of the conventional IoU, the strategy exploits a set of region-IoUs to select positive examples, which can effectively get rid of the flawed positive samples that are poorly aligned or missing essential parts.

The strategy also suppresses the false positives near the pedestrians by considering both the classification and localization information during mining the hard negative examples. Benefiting from the high quality examples, each stage of the cascade CNN detector is constructed with shallow convolution layers. The cascade CNN detector with a few number of parameters has more advantages for applications on embedded systems.

Extensive experiments demonstrate that the compact network can achieve competitive performances. Finally, the proposed approach achieves superior performances on two widely used datasets compared to other state-of-the-art methods, which proves that the proposed example selection methodology and the cascade compact CNN can effectively improve both the detection rate and the localization accuracy.

FUTURE WORK

By clustering the false negatives, it can be found that the main sources are the small pedestrians and the occluded pedestrians. Taking a deeper look at the missing detection of occluded pedestrian, it can be found that most failures are caused by wrong annotations. They automatically annotate the visible bounding box for each instance as the tightest one fully covering the segment mask. However, it fails when two pedestrians overlap with each other. To overcome this type of examples we can use more no of cameras in different angles and by using the pedestrian moments we can more accurately detect the pedestrian.

REFERENCES:

- [1] Y. Tian, P. Luo, X. Wang, and X. Tang, "Deep learning strong parts for pedestrian detection," Dec. 2015, in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), pp. 1904-1912.
- [2] Y. Tian, P. Luo, X. Wang, and X. Tang, "Pedestrian detection aided by deep learning semantic tasks," Jun. 2015, in Proc. IEEE Conf. Comput. Vis. Pattern Recognition., pp. 5079-5087.
- [3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," Jun. 2014, in Proc. IEEE Conf. Comput. Vis. Pattern Recognition., pp. 580-587.
- [4] Z. Cai, M. Saberian, and N. Vasconcelos, "Learning complexity-aware cascades for deep pedestrian detection," Dec. 2015, in Proc. IEEE Int. Conf. Comput. Vis., pp. 3361-3369.
- [5] W. Nam, P. Dollár, and J. H. Han, "Local decorrelation for improved pedestrian detection," Dec. 2014, in Proc. Adv. Neural Inf. Process. Syst., pp. 424-432.
- [6] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in Proc. Eur. Conf. Comput. Vis., 2016, pp. 354-370.
- [7] M. Cordts et al., "The cityscapes dataset for semantic urban scene understanding," 2016, in Proc. IEEE Conf. Comput. Vis. Pattern Recognition., pp. 3213-3223.
- [8] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards realtime object detection with region proposal networks," 2015 in Proc. Adv. Neural Inf. Process. Syst. , pp. 91-99.
- [9] W. Liu et al., "SSD: Single shot multi box detector," 2016 in Proc. Eur. Conf. Comput. Vis., pp. 21-37.

APPENDIX

Sample Code

```
import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile

import tensorflow.compat.v1 as tf

import zipfile

from datetime import datetime

import json

import requests

import os

from requests.auth import HTTPBasicAuth


from collections import defaultdict

from io import StringIO

from matplotlib import pyplot as plt

from PIL import Image

# import ujson as json

import pandas as pd
```

```
import numpy as np

import time

import sys

#from firebase import firebase


config = tf.ConfigProto()

config.gpu_options.allow_growth = True

session = tf.Session(config=config)


import cv2


# cam = 0 # Use local webcam.

# cam = "http://192.168.0.101:8081/video"

# cap = cv2.VideoCapture(cam)


# cap = cv2.VideoCapture(1)


sys.path.append("..")


#fixefixed_interval = 3
```

```
#firebase = firebase.FirebaseApplication('https://iot-guard-770dc.firebaseio.com',  
None)
```

```
count=1
```

```
# In[3]:
```

```
from utils import label_map_util
```

```
from utils import visualization_utils as vis_util
```

```
def diffImg(t0, t1, t2):
```

```
    d1 = cv2.absdiff(t2, t1)
```

```
    d2 = cv2.absdiff(t1, t0)
```

```
    return cv2.bitwise_and(d1, d2)
```

```
#url ='rtsp://admin:888888@192.168.0.164:10554/tcp/av0_0'
```

```
threshold = 100000 # Threshold for triggering "motion detection"
```

```
cap = cv2.VideoCapture(0) # Lets initialize capture on webcam
```

```
# Read three images first:
```

```

t_minus = cv2.cvtColor(cap.read()[1], cv2.COLOR_RGB2GRAY)

t = cv2.cvtColor(cap.read()[1], cv2.COLOR_RGB2GRAY)

t_plus = cv2.cvtColor(cap.read()[1], cv2.COLOR_RGB2GRAY)

# Lets use a time check so we only take 1 pic per sec

timeCheck = datetime.now().strftime('%Ss')

# In[4]:


# What model to download.

MODEL_NAME = 'ssd_mobilenet_v2_coco_2018_03_29'

MODEL_FILE = MODEL_NAME + '.tar.gz'

DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the object
detection.

PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'


# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = os.path.join('data', 'mscoco_complete_label_map.pbtxt')


NUM_CLASSES = 90


# ## Download Model

```

```

# In[5]: opener = urllib.request.URLopener()

opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar_file = tarfile.open(MODEL_FILE)

for file in tar_file.getmembers():

    file_name = os.path.basename(file.name)

    if 'frozen_inference_graph.pb' in file_name:

timeCheck = datetime.now().strftime('%Ss')

    # Read next image

    t_minus = t

    t = t_plus

    t_plus = cv2.cvtColor(image2, cv2.COLOR_RGB2GRAY)


if ret is False:

    cap.release()

    cv2.destroyAllWindows()

    cap = cv2.VideoCapture(0)

```

SCREENSHOTS

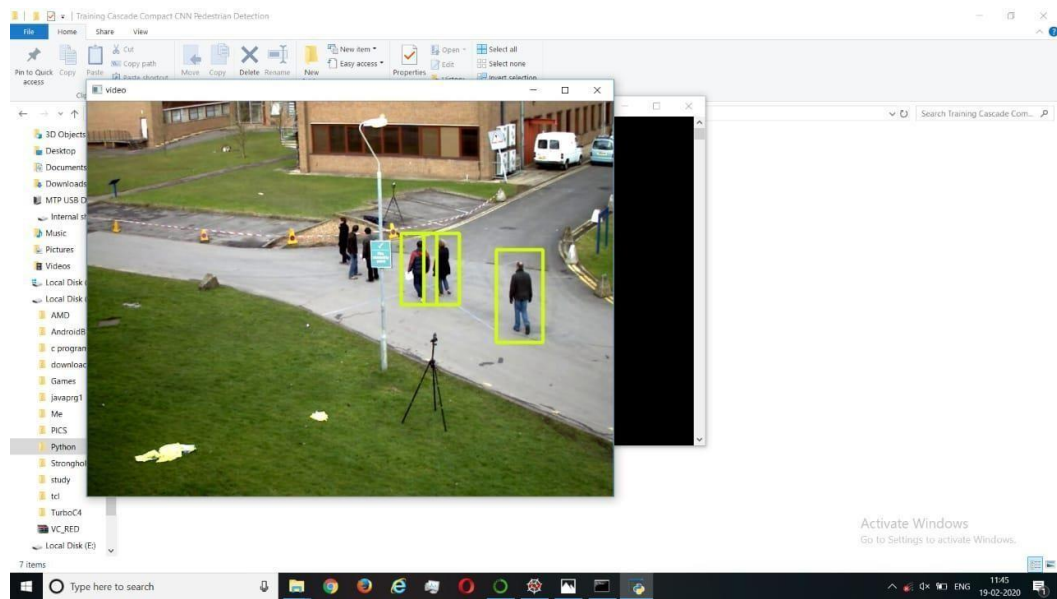


FIG:5.1 FALSE DETECTION

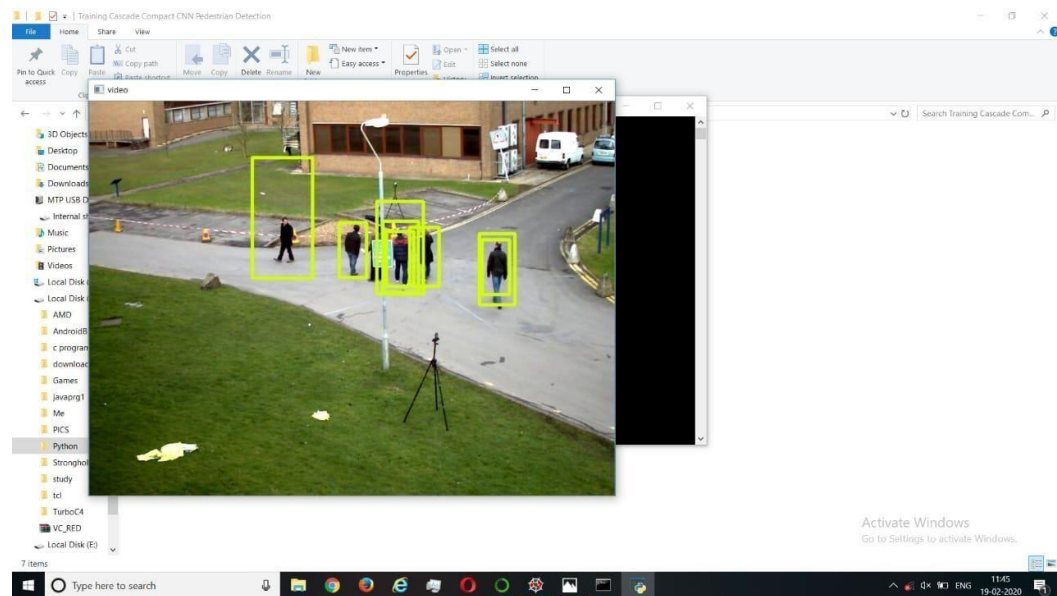


FIG:5.2 OVERCOMING FALSE DETECTION

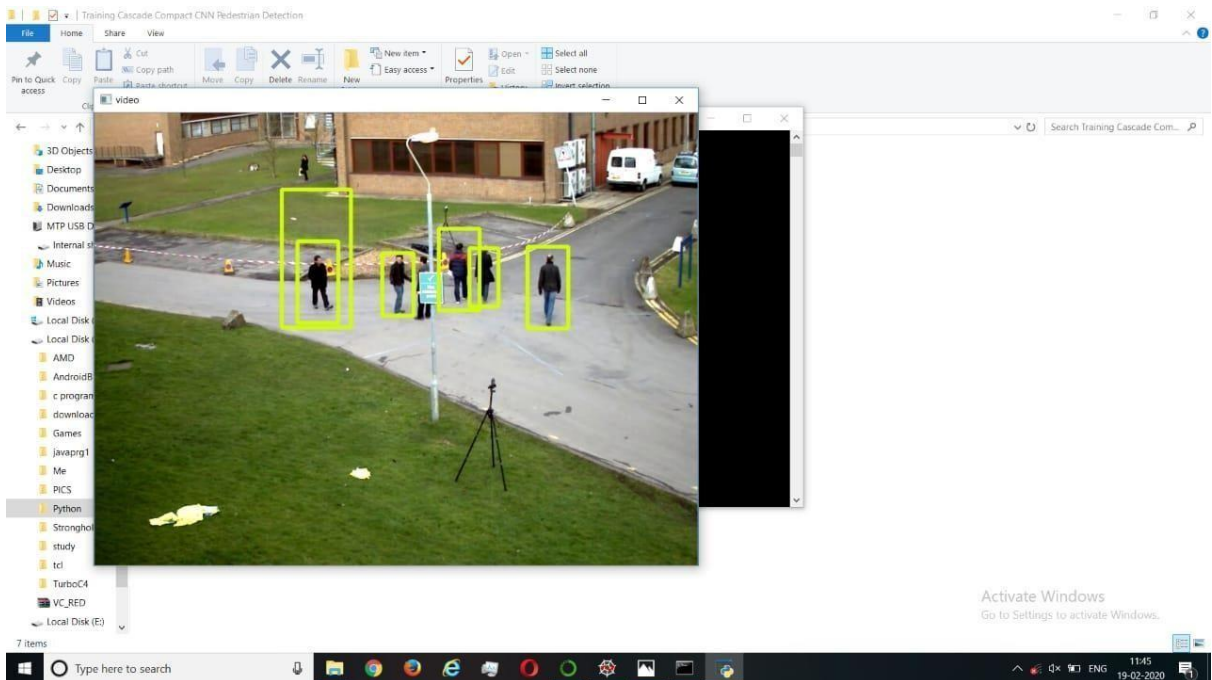


FIG:5.3 MORE ACCURATE DETECTION

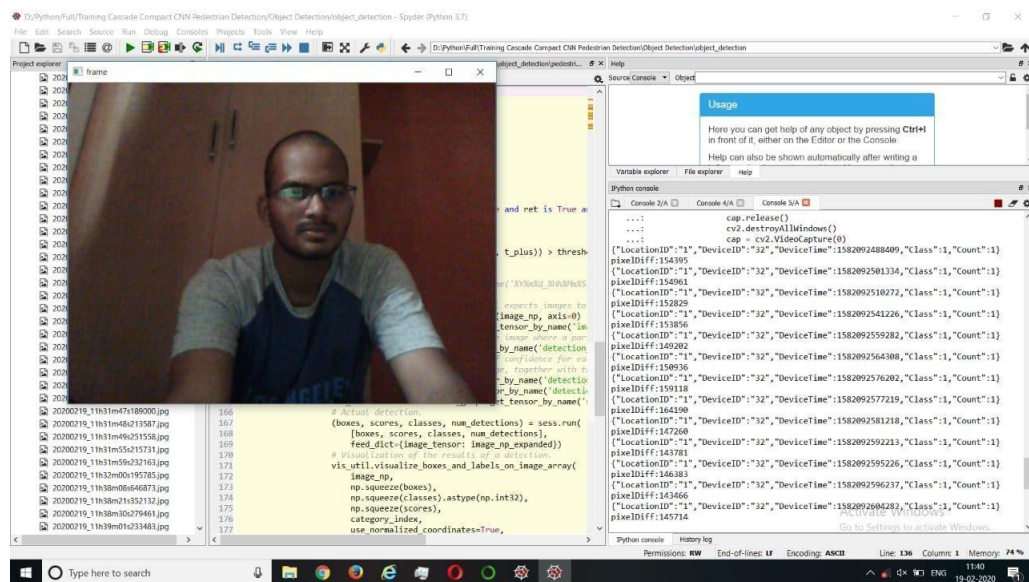


FIG:5.4 OUTPUT