UNIVERSITY OF OTTAWA

MASTERS THESIS

Shoulder Keypoint-Detection From Object Detection

Author:
Prince KAPOOR

Supervisor:
Dr. Robert LAGANIERE

uOttawa

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Applied Science

in the

School of Electrical Engineering and Computer Science
Faculty of Engineering, University of Ottawa

Declaration of Authorship

I, Prince KAPOOR, declare that this thesis titled, "Shoulder Keypoint-Detection From Object Detection" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date:22nd August, 2018

UNIVERSITY OF OTTAWA

Abstract

Faculty of Engineering
School of Electrical Engineering and Computer Science

Masters of Applied Science

Shoulder Keypoint-Detection From Object Detection

by Prince KAPOOR

This thesis presents detailed observation of different Convolutional Neural Network (CNN) architecture which had assisted Computer Vision researchers to achieve state-of-the-art performance on classification, detection, segmentation and much more to name image analysis challenges. Due to the advent of deep learning, CNN had been used in almost all the computer vision applications and that is why there is utter need to understand the miniature details of these feature extractors and find out their pros and cons of each feature extractor meticulously. In order to perform our experimentation, we decided to explore object detection task using a particular model architecture which maintains a sweet spot between computational cost and accuracy. The model architecture which we had used is LSTM-Decoder [45]. The model had been experimented with different CNN feature extractor and found their pros and cons in variant scenarios. The results which we had obtained on different datasets elucidates that CNN plays a major role in obtaining higher accuracy and we had also achieved a comparable state-of-the-art accuracy on Pedestrian Detection Dataset.

In extension to object detection, we also implemented two different model architectures which find shoulder keypoints. So, One of our idea can be explicated as follows: using the detected annotation from object detection, a small cropped image is generated which would be feed into a small cascade network which was trained for detection of shoulder keypoints. Second strategy is to use the same object detection model and fine tune their weights to predict shoulder keypoints. Currently, we had generated our results for shoulder keypoint detection. However, this idea could be extended to full-body pose Estimation by modifying the cascaded network for pose estimation purpose and this had become an important topic of discussion for future work of this thesis.

Acknowledgements

First of all, I would like to thank my supervisor, Professor Robert Laganiere who had trusted my abilities and provided clear path to complete this thesis. Without his guidance and experience, this work cannot be achieved with ground breaking results. His role in this research work cannot be expressed in words as his feedback always proves to be an important asset and makes this thesis to complete in timely fashion.

Apart from this, it is my radiant sentiment to place on records my best regards to my Uncle, Aunt and my friend named, Sukhdeep Kaur Dhindsa. My deepest sense of gratitude for their unfailing support. I came by myself from my home country and they had supported me financially and emotionally as my family in this country.

Also, I would like to thank my co-author of the paper named Md Atiqur Rahman, that we both had presented in 15^{th} conference in Computer and Robot Vision.

Finally, I express my deepest thanks to my parents who resides in my home country but their continuous encouragement throughout my years of study had given me the ability to complete this thesis. Without their encouragement and motivational speeches of my father, I might have dropped from this program and will reckon myself as a failure.

Contents

List of Figures

List of Tables

List of Abbreviations

ADAS	**A**dvanced **D**riving **A**ssistance **S**ystems
CNN	**C**onvolutional **N**eural **N**etwork
DNN	**D**eep **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
ILSVRC	**I**mageNet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge
NMS	**N**on **M**aximum **S**uppression
IOU	**I**ntersection **O**ver **U**nion
ReLU	**Re**ctified **L**inear **U**nit
BN	**B**atch **N**ormalization
RPN	**R**egion **P**roposal **N**etwork
ROI	**R**egion **O**f **I**nterest
MTCNN	**M**ulti **T**ask **C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphical **P**rocessing **U**nit
GHz	**G**iga **H**ertz
FPPI	**F**alse **P**ositives **P**er **I**mage
LAMR	**L**og **A**verage **M**iss **R**ate
AP	**A**verage **P**recision
EER	**E**qual **E**rror **R**ate

Chapter 1

Introduction

Our world is changing in many ways and one of the things which are going to have a huge impact on our future is *deep learning* and *machine learning*. These two technological fields have the potential to bring an AI(Artificial Intelligence) revolution. *Deep learning* is currently reaching new heights of exploration for the researchers and nowadays, a large number of computer vision researchers are trying to put their hands-on this fascinating field. Deep learning has broken almost all the previous trademarks which had been set-up by various computer vision researchers in regards to image classification, object detection, image segmentation and many more to name computer vision challenges.

The success story of deep learning started from late 90's when a computer vision researcher Yann LeCun came up with his CNN architecture [31] which showed state-of-the-art performance on various image and speech recognition dataset (in particular, he showed interesting and marvelous results on a dataset consisting of hand-written digits, popularly known as "MNIST Database"). However, peers were perplexed by the concept of CNN architectures at that time and deep CNN model architectures became popular only when a computer vision researcher *Alex Krizhevsky* along with *Geoffrey Hinton* and *Ilya Sutskever* showed state-of-the-art performance on a huge dataset (ImageNet [6]) with their deep CNN architecture model [30]. They had beaten up all the previous model architecture's performances by an impressive margins. Since then, many computer vision researchers had played around with different deep CNN architecture in order to achieve state-of-the-art performance on various datasets.

Moving further, till now, Deep CNN model architectures had achieved state-of-the-art performance on various computer vision challenges such as image classification, object detection, image segmentation and object landmark detection. In this thesis, we had experimented different CNN model architecture and made a detailed analysis of these model architectures in regards with speed, accuracy, computational cost, training speed and reaching best optimization point. For our experimentation, we had used single object detection as an application and LSTM-Decoder [45] as a model architecture. We used different CNN feature extractor with LSTM-Decoder [45] in order to perform our experimentation. In our experimentation, we had also achieved state-of-the-art performance on various head detection and people detection datasets.

In extension to object detection, we had also made a small cascade network overlaid on the output of above mentioned object detector model which serves as shoulder key-point detection. The details are explained in Chapter 6 of this thesis.

In this Chapter, a brief description and working of CNN feature extractor is explained in section 1.1. Section 1.2 explains the object detection algorithms which are generally used for object detection along with our choice of algorithm that we had used in this thesis. Our extended work for detection of shoulder key-points is described in section 1.3 followed by problem statement and overall structural orientation of thesis which is written in section 1.4

1.1 Convolutional Neural Network Overview

A CNN is a network architecture for deep learning which have the capability to learn directly from images. It is made to process an image to produce an output in form of classes, bounding boxes around objects or region proposals. A standard example of convolutional neural network is shown in figure 1.1 in which the input image is feed into the series of convolutional layers. This is a typical example of classification task in which the input image is classified into a particular class. Here, as we can see; the input image is Car then, the probability of classification for "Car" class is highest as compared other classes. In CNN feature extractors, the initial layer is the "Convolutional layer" followed by "Relu"activation unit and finally, the "Pooling" layer for dimensional reduction. In standard CNN feature extractor, all these units are stacked multiple times in order to make deep CNN feature extractor. The details of each layer shown in the figure 1.1 are out of the scope of this chapter and is discussed in Chapter 2

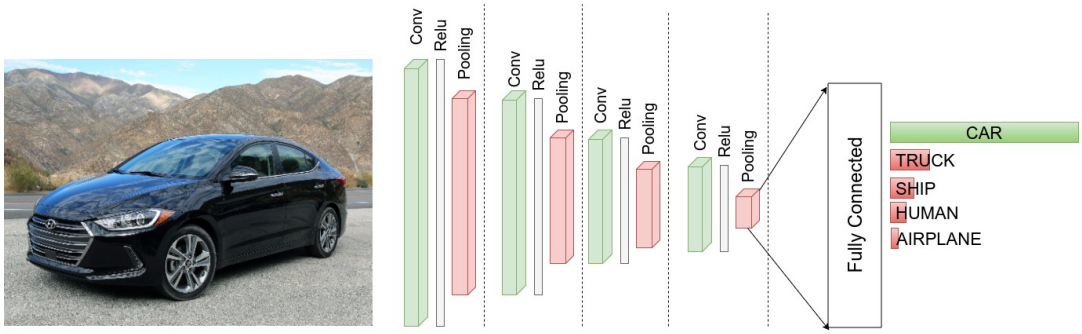


FIGURE 1.1: Basic CNN Structure for Image-Classification

1.2 Object Detection Algorithm

Object detection is a fundamental application of computer vision. Nowadays, specifically for object detection task, researchers had reached an accuracy level which is beyond the level of human and this is the reason that we had chosen object detection task for comprehensive analysis of different CNN feature extractors.

There are different algorithms which are available for object detection tasks depending upon the requirement of the user. In general, achieving higher accuracy for object detection leads to an increase in computational cost of model architecture which substantially decrease the speed of model architecture. So, a sweet spot is always tried to be maintained between accuracy and speed which leads to the choice

of model architecture.

There are various object detection algorithms. Few of the basic and important object detection algorithms are explained as follows:

1.2.1 Overview of Different object detection Algorithms

Traditional object detection algorithms were entirely based on 2 techniques: either by using sliding window approach or by using region proposals. In sliding window approach, a window of predefined size is made to slide over the whole image and model tries to predict a particular object at each location of the sliding window as shown in figure 1.2. For the case of pedestrian detection, Figure 1.2 represents a binary classifier which represents class probability of pedestrian and background for each window. Based on class probability, the model retains a number of bounding boxes and finally, all the redundant overlapping bounding boxes are suppressed using some suppression technique.

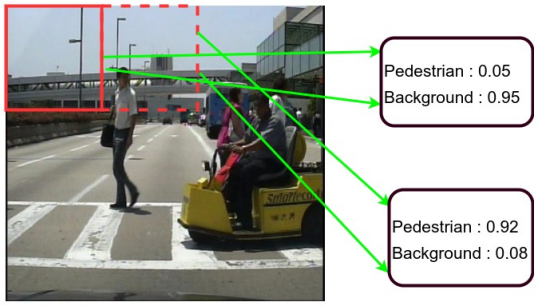


FIGURE 1.2: Object Detection using Sliding Window(class Probability shown for each window)

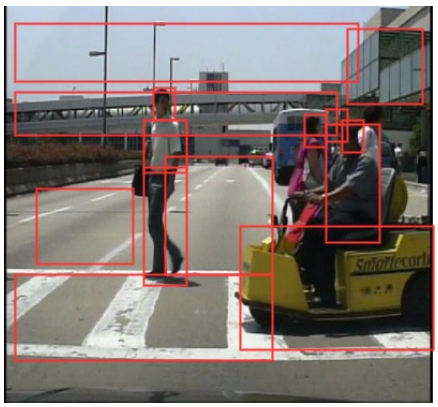


FIGURE 1.3: Object Detection using Region Proposals

This process is done at multiple scales in order to achieve scale invariance. This can be achieved by using CNN feature extractors and it was done by OverFeat [41] which achieved state-of-the-art performance in *2013 ImageNet Challenge* (ILSVRC2013) [6] for localization task. However, this process is highly computationally expensive and it is not appropriate for real-time applications.

On the other hand, in region proposals algorithm, the input image is converted into region proposals which means it is transformed into multiple blobs or patches which looks similar in an input image as shown in Figure 1.3. For simplicity, only few region proposal are shown in Figure 1.3. However, in practice, this number is much higher than shown in the figure 1.3. The model is responsible to find object and regress the exact bounding box coordinates within those multiple region proposals. This technique can also be performed using CNN feature extractors and [14], [13], [38] and [4] had adopted this region proposal technique, achieving state-of-the-art performances on object detection task.

Recently, with the advent of CNN feature extractors, the object detection task can be fully based on convolutional feature extractors which could be trained end-to-end. In this, using CNN feature extractors, the input image is converted into a

small grid (let say the size of grid is 7x7) and the model tries to find object at each cell of the grid as shown in figure 1.4.

1.2.2 Abstract Explanation of LSTM-Decoder

The different object detection algorithms are not just limited to the models as explained in Section 1.2.1. There are various approaches which could be useful for detecting the objects. One of the recent approaches which we had used in this thesis is the method proposed by [45]. This model architecture is trained end-to-end using CNN feature extractors and the input image is converted into a grid of cells (here, 15x20 grid cells). Afterwards, for regressing the bounding boxes from the grid cells, a decoder block is used which decodes the high-level representation of an image into box-coordinates and class confidence for each cell of the grid. The overview of this model algorithm is shown in figure 1.5 and the details of the model is explained in Chapter 3 of this thesis.

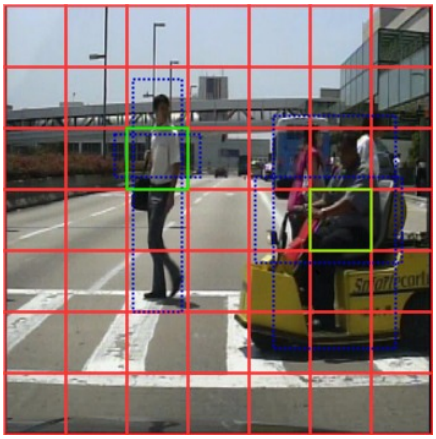


FIGURE 1.4: Object Detection with end-to-end CNN

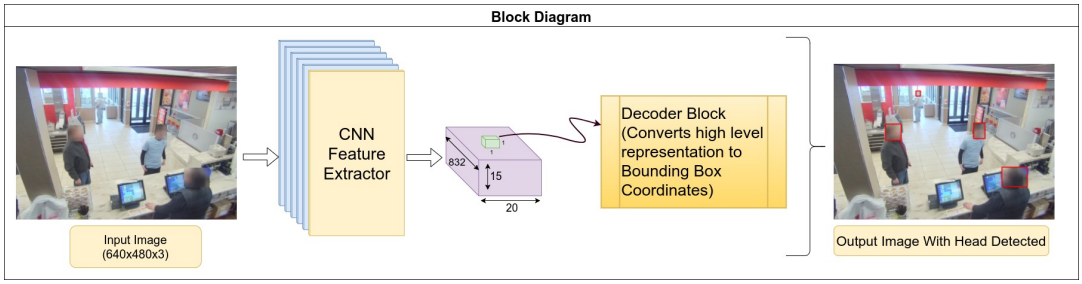


FIGURE 1.5: Block Diagram for LSTM-Decoder

1.3 Extension of Object Detection for Keypoint Detection

Keypoint detection can be categorized into multiple categories depending upon the detection of object. Facial landmark includes keypoints from faces like eyes, nose, ears, lips and to name a few. Human pose estimation requires keypoints from body parts such as shoulders, knees, arms, and many more to name. There are number of algorithms which are available for facial landmark detection and human pose estimation. For human pose estimation, there are basically three datasets which are available and publically used by researchers. These are MPII[1], COCO Keypoints[32] and FLIC dataset[40].

The keypoint detection algorithm presented in this thesis is an assemblage of previous single object detection and few more convolutional layers attached after the output of the object detection model. An abstract details for shoulder keypoint detection is explained as follows:

1.3.1 Detecting Shoulder Keypoints

The idea can be well explained with the help of Figure 1.6. From Figure 1.6, it can seen that the output of object detection model is bounding boxes around the objects of interest. A cropped image is created using these regression points and this cropped image is feed into convolutional layers trained for shoulder keypoint detection. The final result from this cascade network are 2 shoulder points along class score of shoulder detection. Afterwards, these coordinates are back-traced into original image and represented on this image along with detection results.

The details of our shoulder keypoint detection module are explicated in Chapter 6 of this thesis. We had trained our model on variety of datasets and evaluated our results with state-of-the-art model architectures on MPII dataset[1]. We successfully achieved a comparable state-of-the-art results by just using small cascade network for shoulder keypoint detection.

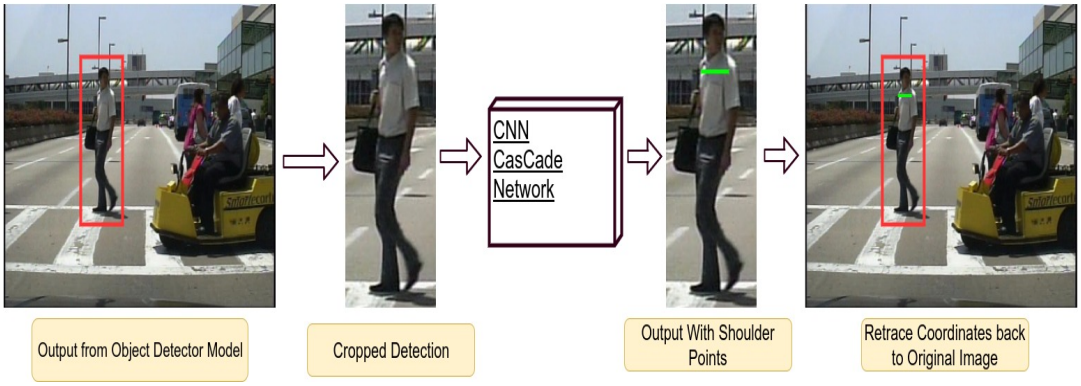


FIGURE 1.6: Block Diagram for Shoulder Keypoint Detection

1.4 Problem Statement

The prime motive of this thesis is to provide a detailed analysis of different convolutional feature extractors and and their effects on object detection application in terms with accuracy and speed. We further developed a small cascaded network which proved helpful for detecting shoulder Keypoints.

By just replacing the Feature extractors with different CNN architectures, we found a substantial increase in the accuracy of the model. However, this notable increase in accuracy comes with corresponding increase in computational cost of the model which leads to decrease in speed. If we consider accuracy, then we had achieved state-of-art performance on head detection datasets and a comparable state-of-the-art performance on pedestrian detection dataset.

1.4.1 Structure of the Thesis

For detailed and comprehensive writing, this thesis is composed of 7 Chapters and the overview of each chapter is described as follows:

- **Chapter 1** "Introduction:" This chapter includes the introduction to some important concepts on which this thesis is based, that is, CNN, object detection and Keypoint Detection.
- **Chapter 2** "Previously Related Work:" This chapter presents a literature review of different CNN models along with available object detection models and Keypoint detection models for face and human pose estimation.
- **Chapter 3** "Object Detection Model:" In this chapter, we discussed the details of object detection model architecture[45] with different feature extractor and method of training this neural network with different loss function and suppression technique.
- **Chapter 4** "Pedestrian Detector:" The 8 different object detection models that we have implemented had been tested on pedestrian detection dataset and their corresponding results are reported in this chapter.
- **Chapter 5** "Head Detector:" Similarly, the 8 different object detection models had been tested on different head detection datasets and evaluated results are being reported. In this chapter, we consider three different head detection datasets and made our experimentation with the combination of these 3 different datasets.
- **Chapter 6** "Shoulder-Keypoint detector": This chapter consists of an extension to object detection for shoulder Keypoints. It includes the algorithm, datasets used and evaluated results on different datasets for single as well as multiple person detection.
- **Chapter 7** "Conclusion:" Lastly, we concluded the thesis with the summary of all our experimentation along with the future work which we can perform in order to further analyse the details of CNN and shoulder keypoint detection.

Chapter 2

Related Work

Our work is based on different CNN feature extractors and the implementation of these feature extractors for object detection task using LSTM-Decoder model architecture [45]. This chapter presents different feature extractors along with popular object detection algorithms. Along with this, we had also described different keypoint detection algorithms and mentioned the algorithm from which we got inspired to extend object detection algorithm for detecting shoulder keypoints.

The structure of this chapter is as follows: Section 2.1 gives an overview of different layers used within CNN feature extractors. Section 2.2 exposes different CNN feature extractors starting from AlexNet [30] till Inception-Resnet-V2 [46]. The overview of different object detection algorithms are explained in Section 2.3. Lastly, the related work regarding our extension to keypoint detection from object detection is explicated in Section 2.4.

2.1 Explanation of different layers of CNN Feature extractors

Traditional neural networks work receives an input (that is, a single vector) that is converted into a set of hidden layers which are further fully connected to all neurons from the previous layer. However, the disadvantage of these kinds of neural networks is that it cannot be implemented on images as this will drastically increase the computational cost of model that would be difficult to train and parameters will definitely lead to over-fitting [44].

CNN feature extractors have an upper hand compared with traditional neural networks as the input to convolutional neural network consists of images. The layers of CNN feature extractors are arranged in 3 dimensions: *width, height and depth*. Here, depth refers to the number of feature maps assigned in each layer. A standard CNN feature extractor maily consists of 4 types of layers: **Convolutional layer**, **ReLU activation**, **Pooling layer** and **Fully-Connected layer**. These layers are stacked together multiple times in order to form convolutional neural network architecture. The details of each layer are explained as follows:

2.1.1 Convolutional Layer

The Convolutional Layer is a primordial building block of an CNN feature extractor which is responsible for most of the heavy computations for detecting different feature maps like edges, spatial orientation or some patterns in the input image. Basically, a convolutional layer consists of some filters which gets trained for specific problem. Each of these filters have some small spatial *height* and *width* and extends along the full depth of input volume. For instance, a typical filter for the first layer

of CNN feature extractor might have size 3x3x3 (that is, *height* x *width* x *depth*). Here, the depth of filter is assigned to be 3 because usually, input images have 3 color channels.

During convolution with the input image, each filter computes a dot product between values of the filter and the input volume at a particular position. We slide and compute the dot product over the entire input volume along height and width of input volume. This leads to formation of 2-dimensional feature maps which gives the activation of that particular filter at every spatial position. Intuitively, the activation are features in the input image which gets activated whenever there are some kind of visual features in the input images like pattern, objects or a simple blob of same color. In CNN, we have multiple filters for each Convolutional layer which could generate separate 2-dimensional feature maps. So, all these 2-dimensional feature maps are stacked together along depth and produce the output volume for the Convolutional layer.

A simple example of convolutional feature maps is shown in Figure 2.1 but in order to explain this simple diagram, we need to understand few terminologies.

Local Connections

When dealing with images, it is impractical to connect all the neurons from previous layer with all the neurons of next layer. So, convolutional filters plays an important role as they are responsible to connect each neuron with a local region of input volume. The size of local region is determined by the filter size. One important thing to be noted is that the connectivity along depth depends upon the depth of input volume. This means that the convolutional filters are locally connected along space (that is, *height* and *width*) but connected fully along the depth of the input volume. As shown in Figure 2.1, the size of input volume is 7x7x3 and filter size is 3x3. Due to full connections along depth, the filter size should be 3x3x3. For simplicity of the diagram, only 1 dimension along depth is shown here with local connections.

Spatial Arrangement

We discussed that convolutional filter are based on the dot product with some locally spaced portion of the input volume and that filter is glided along the whole spatial arrangement of the input volume. In this section, we explain the computation of the output volume by applying convolutional filters. The size of output volume depends upon three hyper-parameters: **depth**, **stride** and **padding**.

- **Depth:** It refers to the number of filter which we would like to use for learning. For simplicity, in Figure 2.1, we used just one filter. However, multiple filter outputs can be stacked along depth of the output volume so that each filter is responsible for different features in the input image.
- **Stride:** This is the amount by which filter slides along the whole spatial arrangement of the input volume. Stride = 1 means that we move filter by one pixel at a time and Stride = 2 means movement by 2 pixels. Increasing stride will eventually decrease the spatial size of the output volume.
- **Padding:** Sometimes, we need to control the spatial size of the output volume and at that instant, padding plays an important role as it will pad the input volume with zeros along the border.

Figure 2.1 is a simple representation showing the first dimension along width and height is shown where the size of input volume is 7x7x3 and filter size is 3x3x3. The hyper-parameters are assigned as: Stride = 1, Depth = 1 and Padding = 1. After convolution, the size of output volume is 5x5x1 as we used just 1 filter.

Figure 2.1 shows a **"bias"** term in parallel with the filter weights. A *Bias* unit is an extra neuron added to each output of convolutional filter. These units allows an activation function to shift to the left or right, which is critical for successful learning.

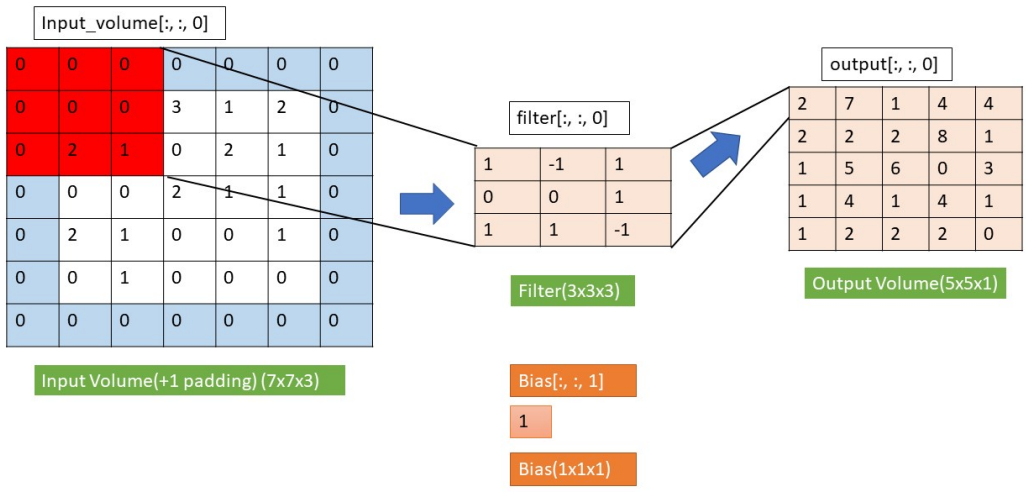


FIGURE 2.1: Convolutional layer explained with 7x7x3 input volume and filter size 3x3x3

The output volume for each convolutional layer can also be calculated as follows:

- Consider the input volume of size W1xH1xD1
- required hyper-parameters:
  - K = number of filters used
  - F = Filter size spatially
  - S = Stride
  - P = Padding
- Output Volume of size W2xH2xD2 computed as[1]
  - $W2 = (W1 - F + 2P)/S + 1$
  - $H2 = (H1 - F + 2P)/S + 1$
  - $D2 = K$

[1]In implementation, if the value of W2xH2xD2 is a fraction then, absolute value of the fraction is used for calculating the output volume. Same applies to the calculation of output volume after Pooling layer.

2.1.2 ReLU Activation

Every single neuron has a capacity to get activated with respect to particular feature as required by an application. So, higher values of cells in a feature map means that those features getting more accurate results for a particular application and weights need to be trained accordingly.

Here, activation function is responsible for taking an input value and perform some kind of mathematical operation so that the value of feature maps for predicting the features of an input image gets higher after the convolutional layer. There are several Activation functions that can used in practice. Few of them are Sigmoid, Tanh and ReLU.

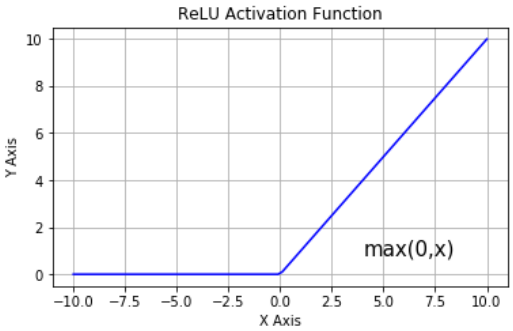


FIGURE 2.2: ReLU Activation Function

ReLU activation function has been found to lead to better performance and reach convergence faster during training.

It is defined as: $F(x) = max(0, x)$, which means that the value is zero when x < 0 and then follows a linear curve with slope 1 when x > 0 as shown in figure 2.2.

There are various merits and demerits of using ReLU activation which are described below:

- Firstly, it had been proved that it accelerates the training by large factor and reaches convergence faster with comparison with other models which don't use ReLU activation.
- Easy to implement and don't involve expensive operations like exponential or multiplication. For convolutional matrix, it is just simply threshold the matrix at zero.
- One demerit is that sometimes, ReLU unit can easily *die* during training. This means that a large gradient which back-propagated through ReLU neuron updates the weight in such a ways that the neuron will never get activated. This can happen if the learning rate is too high.

2.1.3 Pooling layer

This layer is used for reducing the spatial size of the output volume from convolutional and ReLU layer so that the amount of parameters gets reduced. Thus, reducing computational cost of the model architecture and eventually controls overfitting. There are various operations which are used for pooling like MAX, Average or L2-norm. The most common of them is the MAX pooling in which it corresponds to the maximum value in a input volume at a particular position of the filter size.

The most common pooling filter has a size of 2x2 with a Stride of 2. It reduces every feature by half around spatial dimensions (that is, along *width* and *height*).

The output volume for each pooling layer can also be calculated as follows:

- Consider the input volume of size W1xH1xD1
- with hyper-parameters:
  - F = Filter size
  - S = Stride
- Output Volume of size W2xH2xD2 computed as:
  - $W2 = (W1 - F)/S + 1$
  - $H2 = (H1 - F)/S + 1$
  - $D2 = D1$

An example of the pooling layer is shown in figure 2.3, illustrated by taking a simple example of a single depth slice from Figure 2.1. If we apply Max pooling to output of convolutional layer shown in figure 2.1 using a 3x3 filter size and stride of 2 then, the output will be as shown in Figure 2.3.

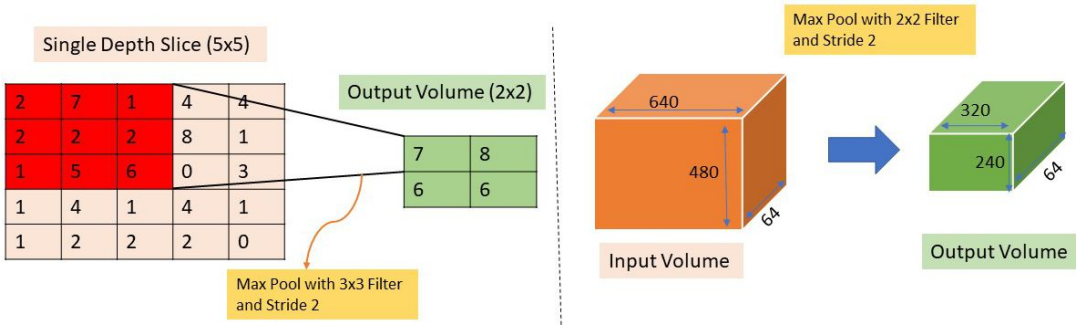


FIGURE 2.3: Pooling Layer shown with Single Depth slice (left) and Practical input volume (right)

2.1.4 Fully-Connected Layer

In this layer, each neuron from an input volume are connected to every neuron of the output as this is a traditional neural networks. An example of a fully-connected layer is shown in Figure 2.4 where the input volume consists of 4 neurons and output volume consists of 5 neurons. Each neuron in the input volume is connected to all neurons of the output volume. If we want to use fully-connected layers in a CNN feature extractor then we have to convert an output volume from convolutional or pooling into a single array of N rows and 1 column where N = number of activation used in the output volume defined by convolutional layer or pooling layer.

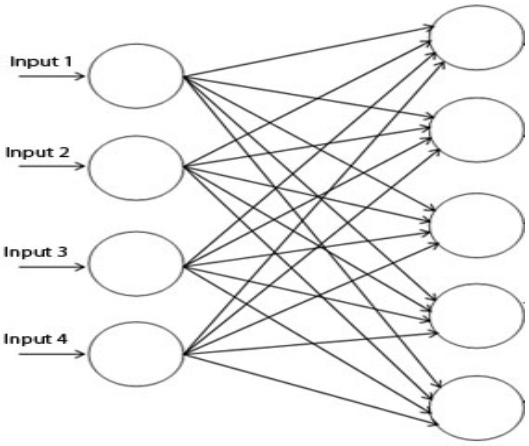


FIGURE 2.4: Fully-Connected Layer

2.2 Different CNN Feature Extractors

The different layers explained in section 2.1 constitute the base for discussing different CNN feature extractors used in our object detection task. The different layers which we had discussed in section 2.1 are stacked together multiple times in order to constitute a complete CNN feature extractor.

We had started the story of CNN feature extractors from Alex-Net [30] because this architecture had beaten previous state-of-the-art performance on ImageNet Challenge[6]. Since then, many CNN feature extractors came into existence and this becomes an important topic in computer vision challenges.

In this section, we discuss different CNN feature extractors starting from Alex-Net [30] till Inception-Resnet-V2 [46]. In between these networks, we describe VGG-Net [43], Squeeze-Net [23], Inception-V1 [47], Inception-V2 [48], Inception-V3[48], Residual Network [15] and MobileNet [21].

2.2.1 Alex-Net

The first successful application of CNN feature extractors was performed by Yann LeCun in 1990's in which he designed a small CNN feature extractor and able to perform state of the art performance on MNIST dataset at those times. However, CNN feature extractors became extremely popular after this CNN feature extractor designed by *Alex Krizhevsky*, *Geoffrey Hinton* and *Ilya Sutskever*.

They showed state-of-the-art performance in ImageNet challenge-2012 [6] and had beaten other competitive models by huge margins (Classfication task: AlexNet achieved Top-5 error rate = 16% whereas, first runner up have Top-5 error-rate = 26%). Table 2.1 shows the detailed architecture of AlexNet in which "conv+norm" refers to convolutional layer + normalization layer "WxHxD" refers to "Width X Height X Depth" of the input volume.

Input Layer	**Input-size (W x H x D)**	**Filter-size/Stride**	**number of Channels**	**Output-size**	**Parameters**
Input Image	640x480x3	-	-	-	-
Conv1+norm	640x480x3	11x11/4	96	160x120x96	35K
Pooling Layer	160x120x96	3x3/2	-	80x40x96	-
Conv2+norm	78*58*96	5x5/1	256	80x40x256	615K
Pooling Layer	78*58*256	3x3/2	-	40x20x256	-
Conv3	40x20x256	3x3/1	384	40x20x384	885K
Conv4	40x20x384	3x3/1	384	40x20x384	1.327M
Conv5	40x20x384	3x3/1	256	40x20x256	885K
Pooling Layer	40x20x256	3x3/2	-	20x15x256	-
Fully-Connected	20x15x256	-	4096	4096	314M
Fully-Connected	4096	-	4096	4096	16.78M
Fully-Connected	4096	-	1000	1000	4.097M
Total Parameters					**338.6M**

TABLE 2.1: Different Layers used in Alex-Net explained

2.2.2 VGG-Net

In ILSVRC-2014[6], the model architecture proposed by Karen Simonyan and Andrew Zisserman secured a first place in localisation challenge and second place in classification challenge. Their main contribuition was in the depth of network. They proved that increasing the depth of CNN feature extractor could lead to higher accuracy and this CNN-feature extractor popularly named as VGG-Net[43]. The layers used to built VGG-Net are shown in Table 2.2.

From table 2.2, it can be seen that they used only 3x3 sized convolutional filters and max-pooling with filter size of 2x2. This homogenous structure showed extremely good performance results on ImageNet dataset [6]. They also comes with different VGG-Net shown in the paper[43] with the concept of increasing depth of CNN feature extractor. The model architecture shown in Table 2.2 is a 16-layers network but in [43], they tested 11-layer, 13-layer and 19-layer network with just increasing or decreasing the additional convolutional layers stacked in the model architecture.

As we can see from Table 2.2, it uses a lot of parameters to train and eventually uses tremendous amount of memory space. However, this drawback can be avoided in object detection as we have to remove the fully connected layer and fine-tune the CNN feature extractor for detecting objects in an image.

Input Layer	**Input-size (WxHxD)**	**Filter-size /Stride**	**Number of Channels**	**Output-size**	**Parameters**
Input Image	640x480x3	-	-	-	-
Conv1	640x480x3	3x3/1	64	640x480x64	1.8K
Conv2	640x480x64	3x3/1	64	640x480x64	37K
Pooling Layer	640x480x64	2x2/2	-	320x240x64	-
Conv3	320x240x64	3x3/1	128	320x240x128	74K
Conv4	320x240x128	3x3/1	128	320x240x128	147K
Pooling Layer	320x240x128	2x2/2	-	160x120x128	-
Conv5	160x120x128	3x3/1	256	160x120x256	295K
Conv6	160x120x256	3x3/1	256	160x120x256	590K
Conv7	160x120x256	3x3/1	256	160x120x256	590K
Pooling Layer	160x120x256	2x2/2	-	80x60x256	-
Conv7	80x60x256	3x3/1	512	80x60x512	1.180M
Conv8	80x60x512	3x3/1	512	80x60x512	2.359M
Conv9	80x60x512	3x3/1	512	80x60x512	2.359M
Pooling Layer	80x60x512	2x2/2	-	40x30x512	-
Conv10	40x30x512	3x3/1	512	40x30x512	2.359M
Conv11	40x30x512	3x3/1	512	40x30x512	2.359M
Conv12	40x30x512	3x3/1	512	40x30x512	2.359M
Pooling Layer	40x30x512	2x2/2	-	20x15x512	-
Fully-Connected	20x15x512	-	4096	4096	2.097M
Fully-Connected	4096	-	4096	4096	16.77M
Fully-Connected	4096	-	1000	1000	4.096M
Total Parameters					**36.76M**

TABLE 2.2: Details of VGG-Net Explained

2.2.3 Squeeze-Net

Input Layer	Input-size (WxHxD)	Filter-size /Stride (# of channels)	S_1x1	E_1x1	E_3x3	Output-size	Original Parameters
Input Image	640x480x3	-	-	-	-	-	-
Conv1	640x480x3	7x7/2(x96)	-	-	-	320x240x96	14,208
Pooling Layer	320x480x96	3x3/2	-	-	-	160x120x96	-
Fire2	160x120x96	-	16	64	64	160x120x128	11,920
Fire3	160x120x128	-	16	64	64	160x120x128	12,432
Fire4	160x120x128	-	32	128	128	160x120x256	45,344
Pooling Layer	160x120x256	3x3/2	-	-	-	80x60x256	-
Fire5	80x60x256	-	32	128	128	80x60x256	49,440
Fire6	80x60x256	-	48	192	192	80x60x384	104,880
Fire7	80x60x384	-	48	192	192	80x60x384	111,024
Fire8	80x60x384	-	64	256	256	80x60x512	188,992
Pooling Layer	80x60x512	3x3/2	-	-	-	40x30x512	-
Fire9	40x30x512	-	64	256	256	40x30x512	197,184
Conv10	40x30x512	1x1/1(1000)	-	-	-	40x30x1000	513,000
Avgpool10	40x30x1000	40*30/1	-	-	-	-	-
Total Parameters							**1,248,424**

TABLE 2.3: Different Layers used in SqueezeNet explained

With the passage of time, researchers also focused on the computational and memory costs associated with the model architectures. Reduction int the model computational cost can be achieved by decreasing the number of parameters to be trained in CNN feature extractor. Squeezenet[23] is one of various approaches in which they achieved accuracy level of Alex-Net[30] with 50 times reduction in the number of parameters.

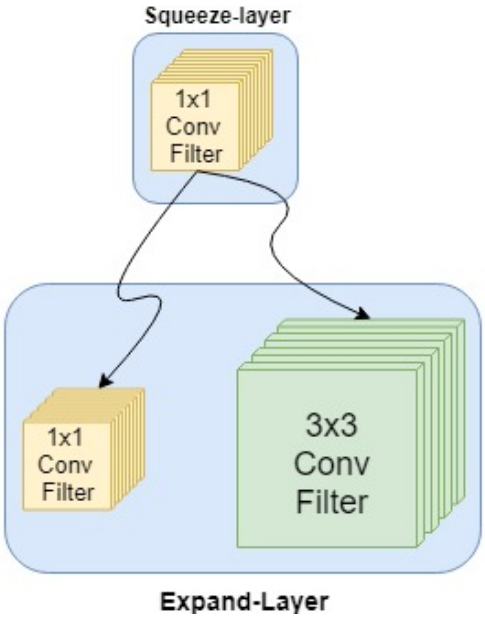


FIGURE 2.5: Fire-Module of SqueezeNet

They had introduced a small building block which they referred as *"Fire Module"* as shown in figure 2.5. A *Fire Module* consists of: a *squeeze layer* which is just 1x1 convolutional filters and *expand layer* which is combination of 1x1 and 3x3 filters concatenated together. This is done in order to reduce the quantity of parameters to be trained. For sustaining the accuracy, dimensional reduction (that is, pooling) is performed late in the network so that larger feature maps can be gathered. The different layers and fire module used in Squeeze-Net is shown in table 2.3.

In table 2.3, "S_1x1" represents the number of filters used in squeeze-layer."E_1x1" and "E_3x3" represents the number of filters used in expanded layer for 1x1 and 3x3 filter respectively.

2.2.4 Inception-V1

Input Layer	Input-size (WxHxD)	Filter-size /Stride (# of channels)	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	Pool Proj	Output-size	Parameters
Input Image	640x480x3	-	-	-	-	-	-	-	-	-
Conv1	640x480x3	7x7/2(x64)	-	-	-	-	-	-	320x240x64	2.7K
Pooling Layer	320x240x64	3x3/2	-	-	-	-	-	-	160x120x64	-
Conv2	160x120x64	3x3/1	-	64	192	-	-	-	160x120x192	112K
Pooling Layer	160x120x192	3x3/2	-	-	-	-	-	-	80x60x192	-
Inception_3a	80x60x192	-	64	96	128	16	32	32	80x60x256	159K
Inception_3b	80x60x256	-	128	128	192	32	96	64	80x60x480	380K
Pooling Layer	80x60x480	3x3/2	-	-	-	-	-	-	40x30x480	-
Inception_4a	40x30x480	-	192	96	208	16	48	64	40x30x512	364K
Inception_4b	40x30x512	-	160	112	224	24	64	64	40x30x512	437K
Inception_4c	40x30x512	-	128	128	256	24	64	64	40x30x512	463K
Inception_4d	40x30x512	-	112	144	288	32	64	64	40x30x532	580K
Inception_4e	40x30x528	-	256	160	320	32	128	128	40x30x832	840K
Pooling Layer	40x30x832	3x3/2	-	-	-	-	-	-	20x15x832	-
Inception_5a	20x15x832	-	256	160	320	32	128	128	20x15x832	1072K
Inception_5b	20x15x832	-	384	192	384	48	128	128	20x15x1024	1388K
Avg_pool	20x15x1024	20x15/1	-	-	-	-	-	-	1x1x1024	-
Dropout(40%)	1x1x1024	-	-	-	-	-	-	-	1x1x1024	-
Linear	1x1x1024	(x1000)	-	-	-	-	-	-	1x1x1000	1000K
Softmax	1x1x1000	-	-	-	-	-	-	-	1x1x1000	-
Total Parameters										**6.79M**

TABLE 2.4: Details of Inception-V1 explained

The winner of ILSVRC-2014 was GoogleNet or Inception-V1 by Szegedy et al.[47]. They secured their position by achieving 9.2% top-5 error on classification task and 43.9 mAP (mean average-precision) on detection task. Their main contritbuition was the introduction of *"Inception-modules"* which is shown in Figure 2.6. This reduces the number of parameters to be trained by a good margins as shown in Table 2.4. The details of the inception based architecture is shown in table 2.4.

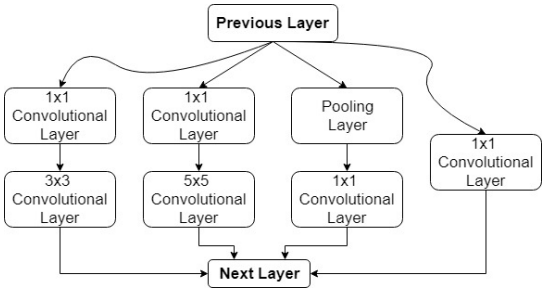


FIGURE 2.6: Inception-module of GoogleNet

In table 2.4, "#3x3 reduce" and "#5x5 reduce" refer to the number of 1x1 filters banks used for 3x3 and 5x5 convolutional layer respectively. "Pool Proj" refers to the number of 1x1 filter used after pooling layer in inception module.

2.2.5 Inception-V2 and Inception-V3

Input Layer	**Input-size (WxHxD)**	**Filter-size /Stride (# of channels)**	**Output-size**
Input Image	640x480x3	-	-
Conv1	640x480x3	7x7/2(x64)	320x240x64
Pooling Layer	320x240x64	3x3/2	160x120x64
Conv2	160x120x64	3x3/1	160x120x192
Pooling Layer	160x120x192	3x3/2	80x60x192
Inception_3a	80x60x192	As in figure 2.7	80x60x256
Inception_3b	80x60x256	As in figure 2.7	80x60x320
Inception_4a (Dimensional reduction)	80x60x320	As in figure 2.10	40x30x576
Inception_4b	40x30x576	As in figure 2.7	40x30x576
Inception_4c	40x30x576	As in figure 2.7	40x30x576
Inception_4d	40x30x576	As in figure 2.7	40x30x576
Inception_4e	40x30x576	As in figure 2.7	40x30x576
Inception_5a (dimensional reduction)	40x30x576	As in figure 2.10	20x15x1024
Inception_5b	20x15x1024	As in figure 2.7	20x15x1024
Inception_5c	20x15x1024	As in figure 2.7	20x15x1024
Avg_pool	20x15x1024	20x15/1	1x1x1024
Dropout(40%)	1x1x1024	-	1x1x1024
Linear	1x1x1024	(x1000)	1x1x1000
Softmax	1x1x1000	-	1x1x1000

TABLE 2.5: Details of Inception-V2

After the success of GoogleNet [47], inception based architecture became extremely popular. The inception-based modules were further enhanced by using the strategy of factorizing and asymmetric convolution. The idea of factorizing convolution using smaller filters is shown in figure 2.7. Here, large spatial filters can be factorized into smaller filters. For example, 5x5 filter can be represented as two 3x3 filter which reduces the computational cost by 28%.

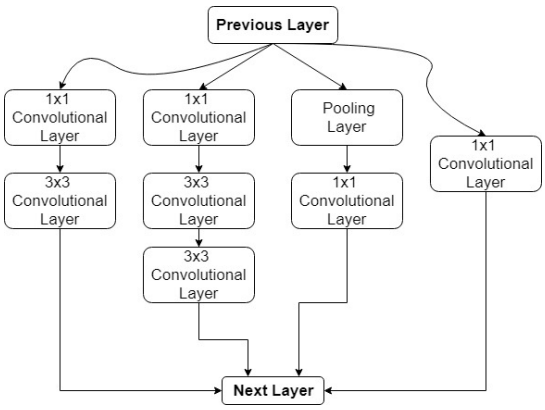


FIGURE 2.7: Inception-module using factorizing convolution using smaller filters

Moving further, the idea of asymmetric convolution is shown in figure 2.8. In this, a convolutional filter of shape "nxn" can be represented in two layer network, that is, nx1 followed by 1xn. This two-layer configuration reduces the computational cost by 33%.

Both of above mentioned ideas are combined together leading to another type of inception module which is used for expanding the filter bank outputs. The concept is shown in figure 2.9. The reason for building this inception module is that it should be used in higher dimensional representation or later coarsest grid. Increasing the number of activation in higher dimension leads to formation of more complicated feature maps.

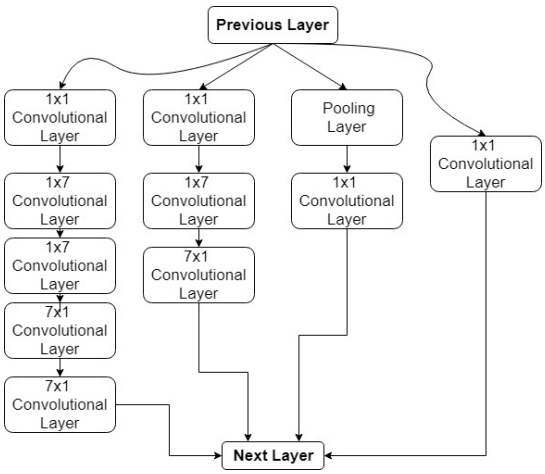


FIGURE 2.8: Inception-module using asymmetric convolution

Moreover, for spatial dimensional reduction, inception-based modules are used instead of using some kind of pooling layer. Generalized pooling layer like Max-pooling or average-pooling reduces the spatial dimension with some loss in representational feature maps. So, inception-based modules for dimensional reduction are used for maintaining the important feature of the input image as well as reducing the spatial dimensions of the input volume. The inception-based modules for dimensional reduction is shown in Figure 2.10.

In gist, connecting all the inception modules discussed in figure 2.7, 2.8, 2.9 and 2.10 in one CNN feature extractor, Szegedy et al. [48] comes up with a modified version of GoogleNet [47] which they referred as Inception-V2 and Inception-V3. The detailed network architecture of Inception-V2 is shown in Table 2.5. For Inception-V3, the different convolutional layers used along with inception modules are mentioned in Table 2.6. Here, one important thing to be noticed which is not mentioned in these tables is that every layer in Inception-V2 and Inception-V3 uses batch-normalization [26] and ReLU activations.

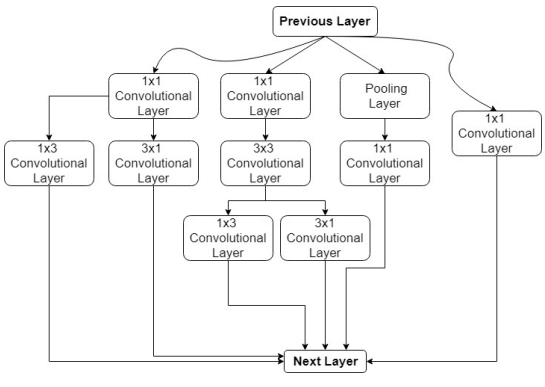


FIGURE 2.9: Inception-module using expanded filter bank outputs

This inception-based modification improved the accuracy on ImageNet dataset [6] as Inception-V2 achieved 6.3% Top-5 error rate and Inception-V3 achieved 5.6 Top-5 error in ILSVRC-2015 challenge.

Input Layer	**Input-size (WxHxD)**	**Filter-size /Stride (# of channels)**	**Number of Inception Modules**	**Output-size**
Input Image	640x480x3	-	-	-
Conv1	640x480x3	3x3/2(x32)	-	320x240x32
Conv2	320x240x32	3x3/1(x32)	-	320x240x32
Conv3	160x120x64	3x3/1(x64)	-	320x240x64
Pooling Layer	320x240x64	3x3/2	-	160x120x64
Conv4	160x120x64	3x3/1(x80)	-	160x120x80
Conv5	160x120x192	3x3/1(x192)	-	160x120x192
Pooling Layer	160x120x192	3x3/2	-	80x60x192
Inception_5	80x60x192	As in figure 2.7	3	80x60x288
Inception_6a (Dimensional reduction)	80x60x288	As in figure 2.10	1	40x30x768
Inception_6	40x30x768	As in figure 2.8	4	40x30x768
Inception_7a (dimensional reduction)	40x30x768	As in figure 2.10	1	20x15x1280
Inception_7	20x15x1280	As in figure 2.9	2	20x15x2048
Avg_pool	20x15x2048	20x15/1	-	1x1x2048
Dropout(40%)	1x1x2048	-	-	1x1x2048
Linear	1x1x2048	-	-	1x1x2048
Softmax	1x1x2048	-	-	1x1x1000

TABLE 2.6: Details of Inception-V3

2.2.6 ResNet

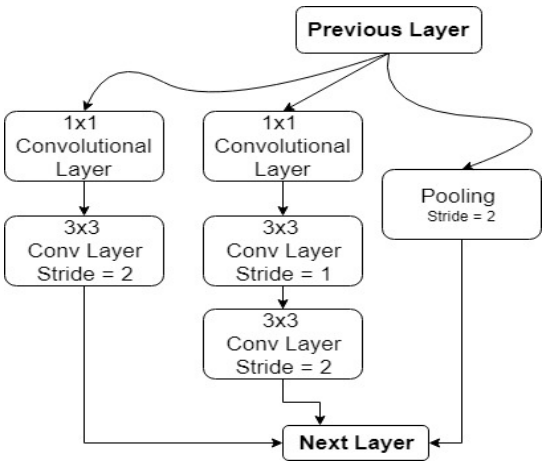


FIGURE 2.10: Inception-module for dimensional reduction

Residual Network [15] was the winner of ILSVRC-2015 by accomplishing Top-5 error rate of 4.49% with their ResNet model architecture which consists of 152 layers. They introduces the concept of short-cut connection because of the *degradation problem* which arises whenever the depth of CNN feature extractor increases to large extent. Indeed, training deeper convolutional networks comes with the problem of vanishing gradient. As a result, the gradient doesn't get back-propagated to deeper layers of the network. In order to resolve this issue, they come with idea of short-cut connection as shown in figure 2.11.

In this paper, the author [15] had resolved the problem of vanishing gradient [17] by introducing "*short-cut connection*" as shown in Figure 2.11 which skips one or more layers. The authors argued that they stacked the convolutional layer in residual mapping (as in Figure 2.11) and this would not decline the performance of the network. This had resolved the problem of vanishing gradient [17] as this residual mapping guarantees that the training error produced by deeper layers would not be larger than its shallower counterparts.

Input Layer	**Input-size (WxHxD)**	**Filter-size /Stride (# of channels)**	**#1x1**	**#3x3**	**#1x1**	**Number of Units**	
			Resdiual Unit			**ResNet-50**	**ResNet-101**
Input Image	640x480x3	-	-	-	-	-	-
Conv1	640x480x3	7x7/2(x64)	-	-	-	-	-
Pooling Layer	320x240x64	3x3/2(x32)	-	-	-	-	-
Res_conv_2	160x120x64	As in figure	64	64	256	3	3
Res_conv_3	80x60x256	As in figure	128	128	512	4	4
Res_conv_4	40x30x512	As in figure	256	256	1024	6	23
Res_conv_5	20x15x1024	As in figure	512	512	2048	3	3
Avg_pool	20x15x2048	20x15/1	-	-	-	-	-
Linear	1x1x2048	-	-	-	-	-	-
Softmax	1x1x2048	-	-	-	-	-	-

TABLE 2.7: Details of ResNet

Using the residual module shown in Figure 2.11, they come up with a new architecture in which they used the residual unit multiple times in order to form deep convolutional layers. The details of the ResNet are shown in Table 2.7. In their original paper [15], they proposed 50-layers, 101-layers and 152 layers residual network but in this table, we only discussed 50-layers and 101-layers residual network because in our experimentation, we only used these two CNN feature extractors.

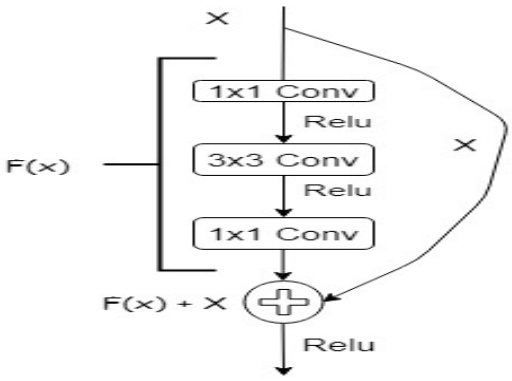


FIGURE 2.11: Residual-Connection: building block for ResNet

2.2.7 MobileNet

The total number of hyper-parameters that needs to be trained in CNN feature extractors is an important factor which determines the accuracy and speed of model architectures.Higher number of parameters will affect the latency of the model architecture but have ambidextrous effect on the accuracy. The CNN feature extractor named "MobileNet" [21] is able to maintain googlenet's [47] level of accuracy with 2.6 millions lesser number of parameters.

In this CNN feature extractor [21], Depthwise separable Convolutional layers are used rather than standard convolutional filters. In standard convolutional layer, the input volume is filtered with some weight matrix and stacked along the depth to produce output volume in a single step. In Depthwise separable convolutions, the input volume splits these two steps by using two different filters as shown in Figure 2.12. Firstly, depthwise convolutional filters are used which apply single filter per input channel. Afterwards, pointwise convolutional filters (that is, standard 1x1 convolutional filters) are used for linear combination of output volume of depthwise convolutional layer.

In comparison with standard convolutional layer, depthwise-separable convolutional filter uses 8-9 times less computational cost. The exact number of nultiply-add operations can be calculated by the formula:

$$\frac{1}{N} + \frac{1}{(D_w)^2}$$

where, "N" is the number of output channels for pointwise convolution and "D_w" is filter-size for depthwise convolution.

The detailed structure of MobileNet using depthwise-separable convolutional layers is shown in Table 2.8. In Table 2.8, , "Conv_dw" refers to the depthwise convolutional layer and "Conv_pw" refers to the pointwise convolutional layer.

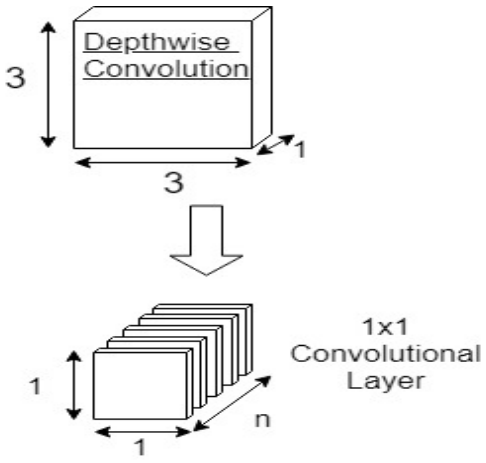


FIGURE 2.12: Depthwise Seperable Convolution

Input Layer Type	**Input-size (WxHxD)**	**Filter-size /Stride /number of channels**	**Number of Parameters**
Input Image	640x480x3	-	-
Conv1	640x480x3	3x3/2/32	864
Conv2_dw	320x240x32	3x3/1/dw	288
Conv2_pw	320x240x32	1x1/1/64	2048
Conv3_dw	320x240x64	3x3/2/dw	576
Con3_pw	160x120x64	1x1/1/128	8192
Conv3_dw	160x120x128	3x3/1/dw	1152
Conv3_pw	160x120x128	1x1/1/128	16,384
Conv4_dw	160x120x128	3x3/2/dw	1152
Conv4_pw	80x60x128	1x1/1/256	32,768
Conv4_dw	80x60x256	3x3/1/dw	2304
Conv4_pw	80x60x256	1x1/1/256	65,536
Conv5_dw	80x60x256	3x3/2/dw	2304
Conv5_pw	40x30x256	1x1/1/512	131,072
$5X\left\{\frac{Conv5\_dw}{Conv5\_pw}\right\}$	40x30x512	$\frac{3X3/1/dw}{1X1/1/512}$	1,333,760
Conv6_dw	40x30x512	3x3/2/dw	4608
Conv6_pw	20x15x512	1x1/1/1024	524,288
COnv6_dw	20x15x1024	3x3/1/dw	9216
Conv6_pw	20x15x1024	1x1/1/1024	1,048,576
Avg Pool	20x15x1024	20x15/1	-
Fully-Connected	1x1x1024	-/-/1000	1,024,000
Softmax	1x1x1000	Classifier	-
Total Parameters			**4,209,088**

TABLE 2.8: Details of MobileNet

2.2.8 Inception-ResNet-V2

Szedgedy et al.[46] combines the strategy of residual connection discussed in Section 2.2.6 with inception modules which are discussed in Section 2.2.5. The idea of this strategy is that it combines the optimization capability of residual connection as well as least-computational capability of inception modules.
In their paper[46], they came up with 2 new CNN feature extractors; one named as Inception-Resnet-V2 and the other named as Inception-V4. Inception-V4 didn't used residual connection. They had implemented more complex inception modules which are less computationally expensive and more optimized during training. They compared both CNN feature extractors (that is, Inception-Resnet-V2 and Inception-v4) and achieved similar level of accuracy on ILSVRC dataset [6].
We used Inception-Resnet-V2 as it combines the optimization and computational capacity of both feature extractors (that is, [15] and [48]).

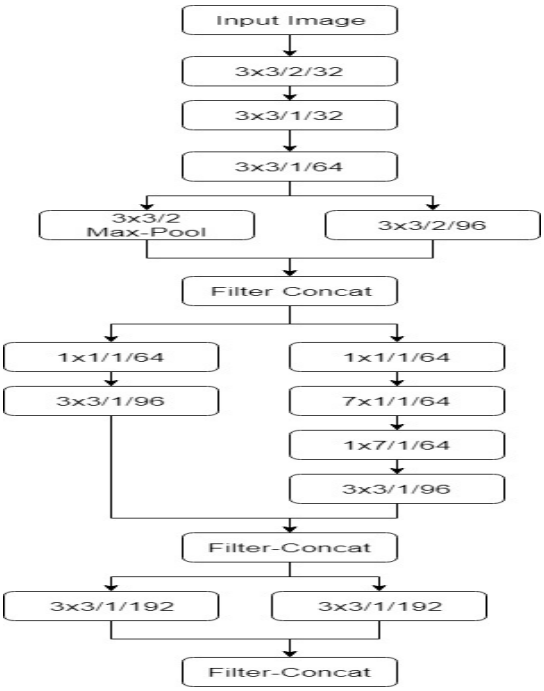


FIGURE 2.13: Stem module for Inception-ResNet-V2

As done in Inception-V3 [48], they optimized in the inception modules in order to reach faster convergence with less hyper-parameters for training. They combined the inception modules with residual connections. The full schematic of Inception-Resnet-V2 is shown in Table 2.9. Inception modules are shown in figure 2.14, 2.15 and 2.16 in the form of block diagrams.[2]

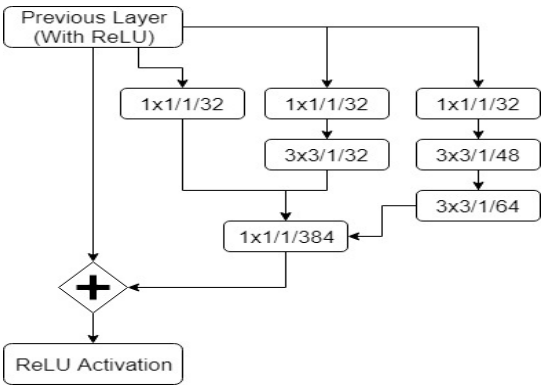


FIGURE 2.14: Inception-ResNet module numbered A

For dimensional reduction in Inception-Resnet-V2[46], they used approximately the same inception-module which we had discussed in section 2.2.5. This module is shown in figure 2.10. However, in this feature extractor (that is, Inception-ResNet-V2[46]), the number of layers had been increased tremendously which leads to a large number of parameters to be trained. However, it gives state-of-the-art performances on various datasets ([6], [32] and [10]).

[2]**Note:** In figure 2.13, 2.14, 2.15 and 2.16, there are some numbers shown in format WxW/S/N (for example: 3x3/2/32). This means that a convolutional layer had been used with filter size as W, Stride as S and number of output channels as N.

In their paper[46], along with Inception-V4 and Inception-Resnet-V2, they had introduced another CNN feature extractors named as Inception-ResNet-V1. It achieved Inception-V3 level of accuracy. The schematic diagram is almost similar to Inception-Resnet-V2. The difference resides in the modification of inception modules for each layer with lesser number of filters as compared to Inception-ResNet-V2.

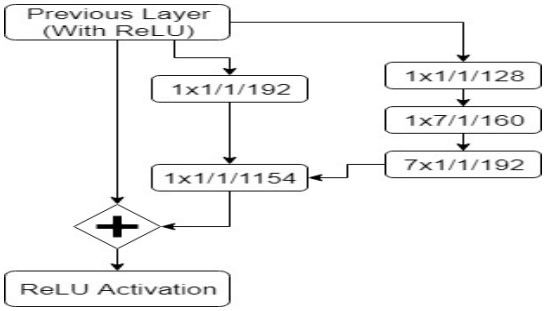


FIGURE 2.15: Inception-ResNet module numbered B

This CNN feature extractor is the current state-of-the-art model architecture for most of the popularly known datasets like Imagenet[6], MSCOCO[32] and PASCAL-VOC [10]. This models achieved Top-5 error rate of 4.9 % and Top-1 error rate of 19.9 % on Imagenet dataset[6].

Input Layer	**Reference Figure**	**Number of Modules**	**Output Size**
Input Image	-	-	640x480x3
Stem_module	As in figure 2.13	1	80x60x384
Inception-Resnet-A	As in figure 2.14	5	80x60x384
Reduction-module-A	As in figure 2.10	1	40x30x384
Inception-Resnet-B	As in figure 2.15	10	40x30x1154
Reduction-module-B	As in figure 2.10	1	20x15x1792
Inception-Resnet-C	As in figure 2.16	5	20x15x2048
Average pooling	-	1	1x1x2048
Dropout (keep-Prob = 0.8)	-	1	1x1x2048
Softmax Layer	-	1	1x1x1000

TABLE 2.9: Details of Inception-Resnet-V2

2.3 Object Detection Models

In computer vision, object detection task is considered to be an important challenge. Researchers had tried different strategies in order to achieve state-of-the-art performance on various datasets ([6], [32], [10]). Traditionally, object detection was applied using some linear classifiers which computes, for instance, histograms of oriented gradients at multiple scales and positions[5]. However, these kind of methods were not successful in detecting multiple objects having different labels in an image.

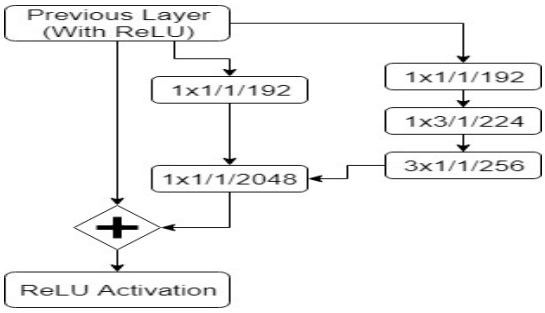


FIGURE 2.16: Inception-ResNet module numbered C

So, with of advent of deep learning, CNN feature extractors becomes extremely popular as they construct a very high-level representation of feature maps in an hierarchical manner which could be used for detecting objects using various end-to-end model architectures.

In this section, we discuss a particular example of those techniques that had achieved state-of-the-art performances on different datasets when this model architecture got published.
OverFeat model architecture[41] uses sliding window approach and had ranked 4^{th} in classification, 1^{st} in localization and detection task in ISLVRC-2013. Faster-RCNN uses region proposals method and it is current state-of-the-art in various datasets ([10], [32]). In order to achieve higher accuracy, Faster-RCNN is used and the accuracy can be improved by using very deep convolutional feature extractors like [46], [15] or [3]. SSD (Single Shot MultiBox Detector) is based on the approach of grid cells (discussed in section 1.2.1) and the motive of this approach is to develop real-time model architecture which could be used for autonomous and driving applications. These kind of model architecture are less computational expensive and achieved a comparable state-of-the-art accuracy with high frame-rate (FPS) as compared with other expensive architecture like [14], [4], [38] or [13].
The details of these architectures are discussed in this section and this description provides a base to our discussion on LSTM-decoder [45] presented in chapter 3 as LSTM-Decoder maintains a sweet-spot between speed and accuracy.

2.3.1 OverFeat Model

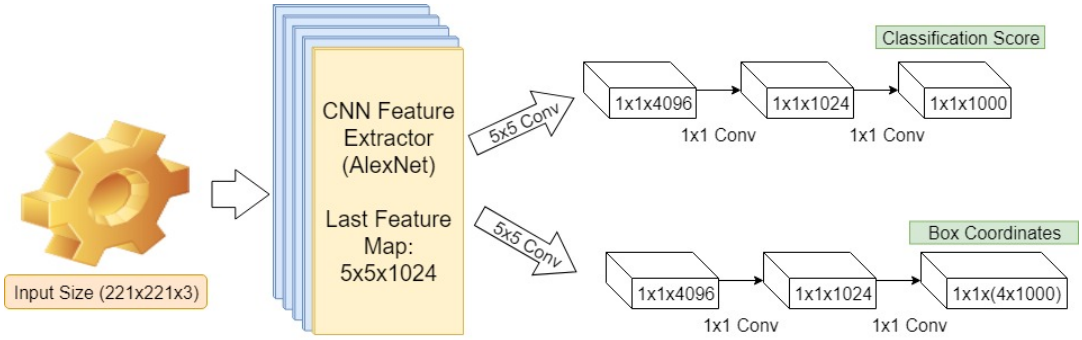


FIGURE 2.17: Block Diagram for OverFeat Model(Part-1)

OverFeat[41] had won classification plus localization challenge in ILSVRC-2013[6] by achieving Top-5 error-rate of 29.8 %. Their model architecture is based upon a sliding window approach. They also had come up with an fully-convolutional model architecture which could classify and localize objects in an input image; this model architecture is explained using Figure 2.17.
An input window of specific size is fed into a CNN feature extractor. The last layer of the CNN feature extractor had been removed and replaced by two different convolutional layers. One is responsible for classification purpose and other is responsible for regressing the bounding boxes around the object in the image.
A small window of a specific size is then chosen and it is made to slide across whole image for head detection as shown in Figure 2.18. According to the methodology shown in Figure 2.17, each window outputs a class score and bounding box coordinates for a particular object. Afterwards, some heuristic algorithm is developed for merging the bounding box coordinates for each prediction of window position

in order to obtain the final results[3].
This algorithm is extremely computational expensive but they tried to reduce the computational capacity by building fully convolutional feature extractor. Training this model at different scales increase the accuracy of the network but reduces the speed of the model and therefore, it is not applicable for real-time purposes.

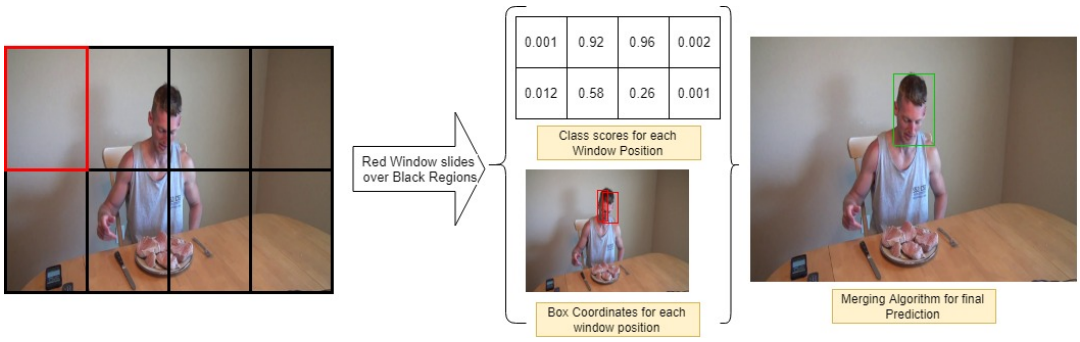


FIGURE 2.18: Block Diagram for OverFeat Model(Part-2)

2.3.2 SSD

The SSD architecture [33] uses a single feed-forward CNN that can directly generate a fixed-size collection of object bounding boxes along with the confidence values for each object class in each of those boxes. The core idea of SSD is to generate predictions from multiple feature maps from a single CNN where each feature map is targeted for detecting objects at a particular scale. The overall architecture of SSD is shown in Figure 2.19. As shown in this figure, SSD adds some additional feature maps (six in the original implementation [33]) on top of a feature extractor CNN called a base network.
An input image is first passed through the base network to produce a high-level CNN feature which is progressively down sampled by 3x3 convolutions with stride 2 at each feature map layer. Thus the feature maps have increasing receptive fields allowing them to capture objects at different scales. At feature map and for each location in the map, SSD employs some default anchor boxes with varying aspect ratios (e.g., 1, 2, 3, 1=2, and 1=3). Each anchor box actually looks for objects of the corresponding aspect ratio at a scale depending on the feature map. As a result , the first layer feature maps can detect the smallest scale of objects while the top layer captures the biggest scale.
To predict the box offsets and box scores at each location of a feature map, SSD uses 3x3 convolutional filters. With K default boxes at each location of a feature map of size MxN, SSD generates a total of (C +4)KMN predictions, where C denotes the total number of object classes (in this case C is 1, as we are only looking for people head). Finally, the predictions from all the feature maps are concatenated at the end of the network for the purpose of matching with ground-truths and optimization of the loss function which is a weighted sum of box location loss and classification loss. During inference, sufficiently overlapping predictions are canceled out following the Non-Maximum Suppression (NMS) procedure [19].

[3]For simplicity purpose, a particular window size is chosen and only two prediction for bounding box are shown in Figure 2.18. However, in actual practice, multiple window sizes are chosen in order to get higher accuracy

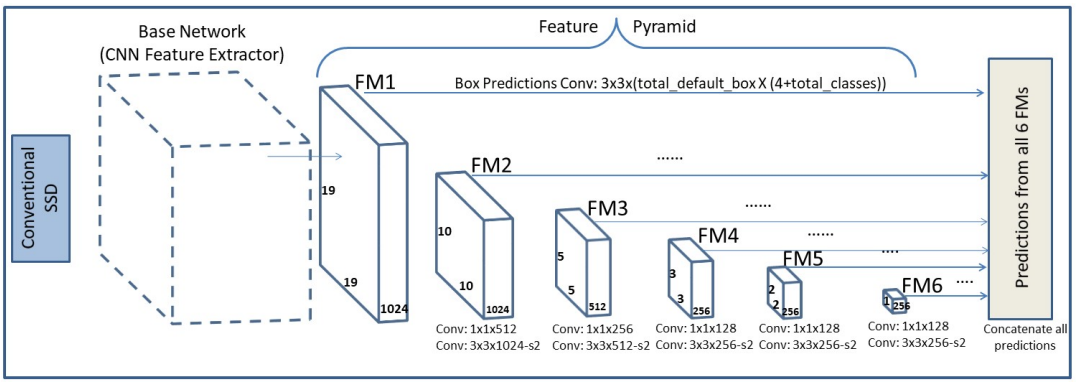


FIGURE 2.19: Block Diagram for SSD Model architecture

2.3.3 Faster-RCNN

This is a method based on region proposal network in which the input image is converted into some kind region proposals. There are many algorithms which can be used for the formation of region proposals but the most popular used algorithm is selective search [39]. Faster-RCNN [38] is not based on selective search for their region proposals. Instead, they used convolutional layers and default anchor boxes for forming region proposals.

The idea of Faster-RCNN comes from the drawbacks of previously proposed networks, that is, RCNN [14] and Fast-RCNN [13]. These aforementioned model architecture are very slow and expensive to train.

Faster-RCNN[38] was made to reduce the computational cost for formation of region proposals, exploiting shared deep features. The full model architecture can be trained end-to-end.

The model architecture of Faster-RCNN is explained using the figure 2.20. The input image is feed into a CNN feature extractor for extracting feature maps of the input image. Now, instead of using some external method to compute region proposals (like [39]), they added convolutional based network named as "Region Proposal Network (RPN)" which produces region proposals directly from CNN feature maps of the high resolution input image. Afterwards, they used *ROI Pooling* as proposed in [13] with fully-connected layer in order to get classification score and regression bounding box for predicted objects in an image.

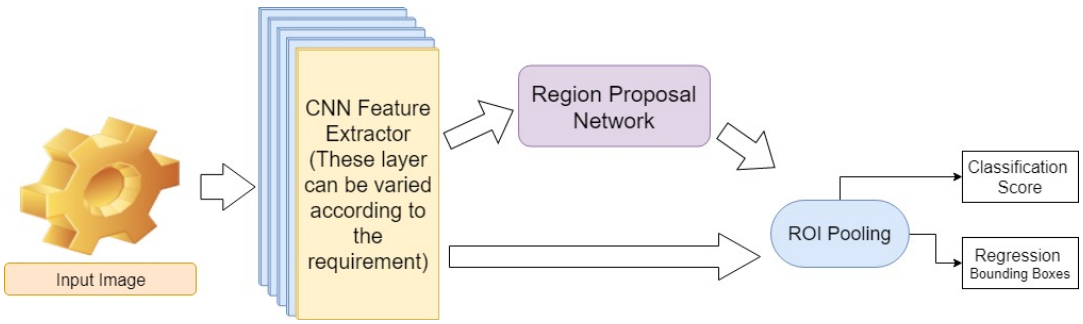


FIGURE 2.20: Block Diagram for Faster-RCNN model architecture

For RPN, the feature maps act as an input to this network as shown in Figure

2.21. Region Proposal network consists of 3x3 and 1x1 convolutional layers on top of the feature maps as shown in Figure 2.21. Their main idea is to project some default anchor boxes on the feature maps and predicts the object within those anchor boxes. So, the output from this RPN is a classification score for each region proposal to predict as an object along with offset of regression coordinates from default anchor boxes.

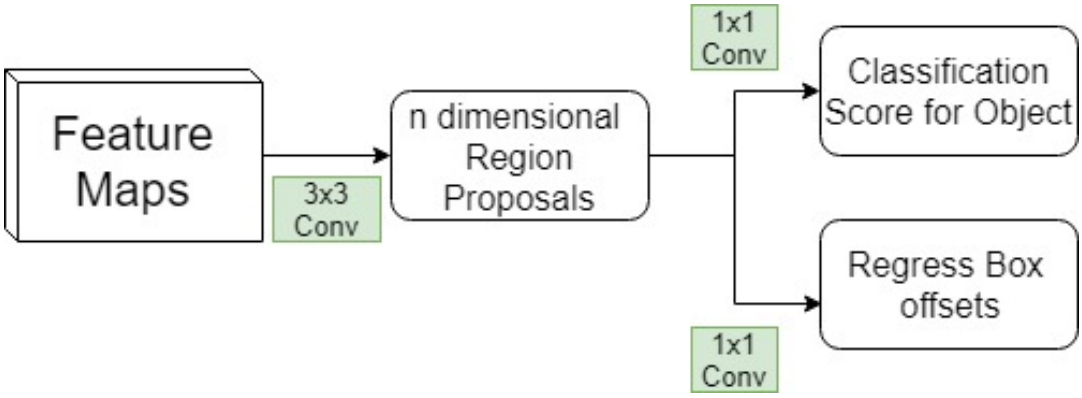


FIGURE 2.21: Block Diagram for Region Proposal Network

2.4 Keypoint Detection

Bounding boxes are used in computer vision in the detection of objects. Keypoint detection is used to detect specific distinct features within an object. Popular keypoint detection algorithms comes in two categories. One is facial keypoint detection and other is human pose estimation. The approach for resolving these challenges are different and that is why, different algorithms had been used for facial as well as human body keypoint detection.

For facial keypoint detection, a small convolutional cascaded network is enough for higher accuracy on benchmark datasets[51][28]. However, for human pose estimation, we need deep convolutional feature extractor for the localization of each body keypoints. So, the algorithms designed for resolving these two challenges have two major differences in general. One is depth of CNN feature extractor and the other is the variation in the datasets to be used for training. Many researchers had come up with different model architecture[37] [50] and they had shown state-of-the-art performances on benchmark datasets. For human pose estimation, every researcher had their own methodology for predicting body keypoints and they are successful in achieving state-of-the-art performances [25][24] on benchmark datasets.

For facial keypoints detection, the list of available datasets is long but the most popular datasets are [51] and [28]. One the other side, the popular benchmark datasets for human pose estimation are [1], [40] and [32].

However, we discussed two model architectures ([50] and [16]) from which we got inspired to build our model architecture for shoulder keypoint detection using object detection.

2.4.1 Facial Keypoint Detection from MTCNN [50]

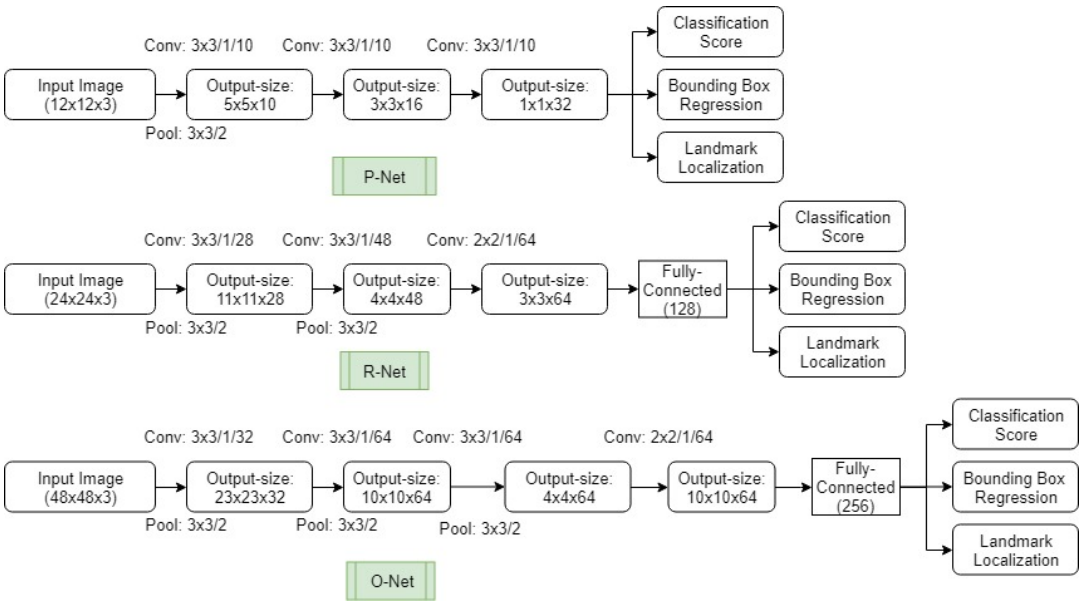


FIGURE 2.22: Details of 3-stage cascaded CNN used in MTCNN

This paper [50] comes up with an algorithm of face detection and landmark location using different cascaded convolutional feature extractors. They used three different types of cascaded CNN feature extractors. Firstly, from the input image, candidate windows are produced using a very small and fast cascaded convolutional network which detects faces and bounding box regression coordinates for each region. This stage is referred as "P-Net"

Afterwards, these candidate network are refined and suppressed using another cascaded CNN model and bounding box coordinates get suppressed in this stage. This stage is referred as "R-Net"
Finally, the final result of facial landmark detection along with face detection is represented by last cascaded convolutional layers. This stage is referred as "O-Net"
The details of each cascaded convolutional layers are shown in figure 2.22[4]. In figure 2.22, three stage cascaded network is used for face detection as well as landmark localization.
This methodology is computationally less expensive and fast as reported in the original paper [50] that the speed of this model architecture on 2.60GHz CPU is 16 FPS and 99 FPS on Nvidia Titan-X GPU. They also surpassed the state-of-the-art performance on FDDB dataset [28]
From this model architecture, we get our motivation and came up with a model architecture which could detect shoulder keypoints after detecting objects in an image. This means that a very minimal amount of parameters can be used for detecting human body keypoints and it could be used for real-time applications.

[4]In figure 2.22, "Conv: 3x3/1/10" means a convolutional filter with filter size "3x3", stride "1" and number of channels "10". Similarly for "Pool:3x3/2" means max-pooling with filter size "3x3" and stride "2"

2.4.2 Mask R-CNN[16]

This Paper[16] presented a simple and flexible approach for object instance segmentation. They also extended the same algorithm for Human Pose estimation and achieved state-of-the-art results on COCO-2016 Challenge[32]. This method is implemented on Faster-RCNN object detection model[38] by adding a layer for predicting segmentation masks on each Region of Interest(ROI). The extra added layer is in parallel with existing branch for classification and bounding box regression as shown in Figure 2.23[5].

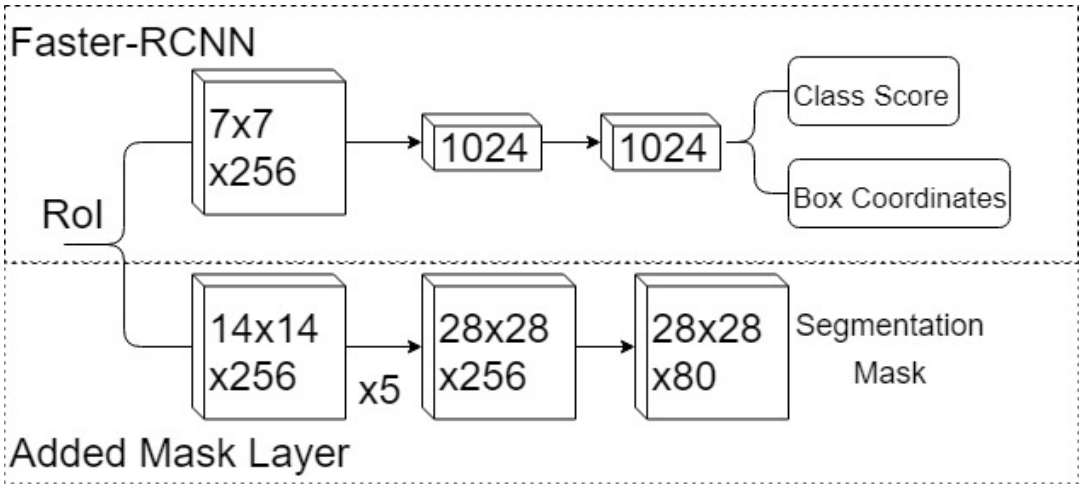


FIGURE 2.23: Details of Mask-RCNN

Originally, Faster-RCNN[38] was not designed for pixel-to-pixel alignment as it just generates the bounding boxes for distinct objects. This the reason that they used ROI-Pooling for feature extraction that performs coarse spatial quantization. For instance segmentation purpose, spatial locations of the input image needs to be preserved. They come up with new quantization-free layer, named as, *"ROIAlign"*. *ROIAlign* uses bilinear interpolation[27] to compute the exact values of the input features at four regularly sampled locations in each ROI. ROIAlign leads to large improvement in instance segmentation.
They used ResNet-101[15] as their feature extractor and feature maps are shared between RPN(Region Proposal Network) and Mask-RCNN stages. With this architecture, they achieved a speed of 5 FPS(Frames-Per-Second) on Nvidia Tesla M40 GPU.
Our shoulder keypoint detection is achieved by two different model architectures. One is based on external cascade model which is inspired from MTCNN[50]. Second is inspired from Mask-RCNN[16] that performs parallel processing with object detection model.

In this Chapter, we had discussed about different CNN feature extractors along with object detection and keypoint detection model architectures. In the next Chapter, we discuss about LSTM based model architecture for object detection in detail as all our experimentation are based upon this model architecture.

[5]The added extra layer is a small fully convolutional network applied to each Region of Interest (ROI). It predicts segmentation mask in a pixel-to-pixel manner.

Chapter 3

Object Detection Model

As discussed in the previous chapter, there are numerous object detection models which could be used depending upon the requirement of the application. Practitioners may require either high speed model architectures or models that achieve a better accuracy. So, the challenge is to find a sweet spot between speed and accuracy for their implementation in real-world examples.

LSTM-decoder[45] is one of the model architectures that maintains a good trade-off between speed and accuracy for object detection. LSTM-decoder always aims at achieving maximum accuracy with a minimal number of parameters to be trained.

In this chapter, we explain the detailed architecture of the LSTM-decoder[45] along with the technical contributions which the model had for improving the accuracy on object detection challenge. The overall details of the model architecture are explained in section 3.1. The original model uses Inception-V1[47] as their CNN feature extractor and presented their performance on a head detection dataset created by the authors. Section 3.2 gives the list of CNN feature extractors used for our experimentations. Section 3.3 provides the details of the network used for regressing the bounding box coordinates around the object to be detected. A theoretical explanation of Regression network is given in Section 3.1, but the working of LSTM along with the usage for regressing the bounding boxes is explained in section 3.3. The loss function for training and named "*Hungarian Loss function*" is presented in Section 3.4.1. Lastly, the suppression technique introduced by the authors of this paper (named "*Stitching*") is mentioned in Section 3.5.

3.1 Explanation of LSTM-Decoder

The work presented in this thesis is based on LSTM based architecture [45], and the loss function, as well as stitching module rely on the concept of LSTM-decoder for generating boxes from CNN encoding.

One of our contribuitions reside in testing the model architecture with different convolutional feature extractors as well as the inclusion of bootstrapping[49] during training. The model generates bounding boxes as a part of an integrated process, rather than an independent process as in most of object detection algorithms like OverFeat [41], Faster-RCNN [38] and R-FCN[4]. It results in predicted bounding boxes which directly corresponds to object detected in an image. The recourse to complex merging algorithms[41] or non-maximum suppression[19] is consequently avoided.

LSTM-decoder[45] uses a CNN feature extraction layer coupled with a LSTM module used to predict sequences of bounding boxes. [19] The detailed model architecture is explained in Figure 3.1.

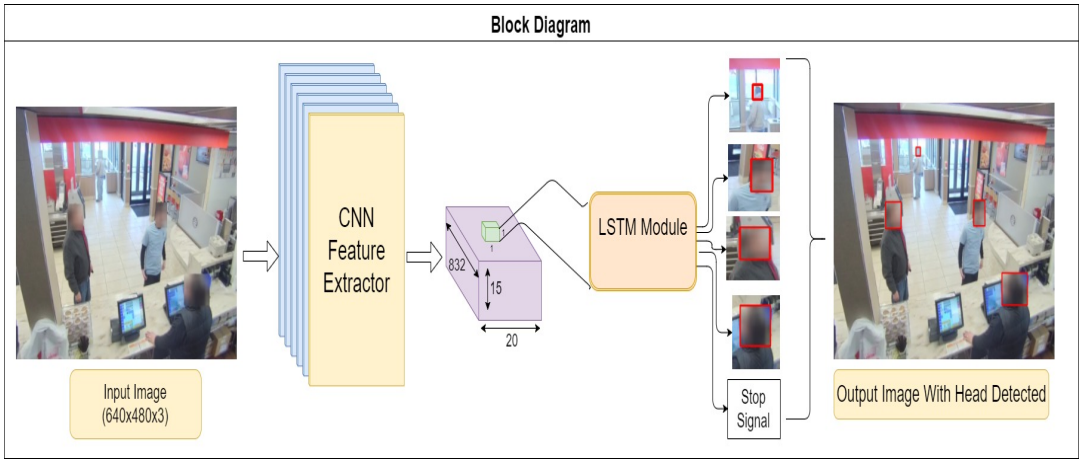


FIGURE 3.1: Network Diagram in a simplified way

In this model, the image is fed into a model that converts the image into a high-level descriptor via some Convolutional feature extractor along with decoding the representation into a set of bounding boxes. Here, the decoder is a LSTM unit which decodes the high level description of an image into a set of predicted bounding boxes[18]. The definition and working of LSTM architecture (Long Short-Term Memory) is given in section 3.3.2 of this chapter.

For predicting sequences, we build an LSTM unit as shown in Figure 3.1. The output of CNN the feature extractor is a grid of high dimensional vectors which summarize the contents of an image regions. LSTM draws the information from this grid and act as controller in decoding the highest features of each region[11]. The LSTM module is used as a regression network so that a coherent set of predictions could be made at the output. The model remembers the previously generated predictions; multiple predictions of the same target can thus be avoided.

At each iteration, a LSTM unit generates a new bounding box along with its corresponding confidence. The bounding boxes that are thus generated from the decoder are made to be produced in descending order of their prediction confidence. A STOP signal is then passed to the classifier whenever the LSTM cell becomes unable to find a new predicted objects in a particular region. The output image with all the predicted objects are gathered and presented as predicted bounding boxes as a final output.

During the testing or evaluation phase, bounding boxes from this model are suppressed using the suppression technique which they had used (named as "*Stitching*") as explained in section 3.5.

CNN Model Architecture	Reference Table from Chapter-2	Layer name used (as reference in table)	Number of Parameters Till this Layer	Classification Accuracy
VGG-Net	Table 2.2	Pooling Layer After Conv12	14.7M	71.1%
Squeeze-Net	Table 2.3	Fire9	0.73M	60.4%
Inception-V1	Table 2.4	Inception_5b	5.3M	69.8%
Inception-V2	Table 2.5	Inception_5c	19.3M	73.9%
Inception-V3	Table 2.6	Inception_7	23.9M	78.0%
Residual Network-50	Table 2.7	Res_conv_5	25.6M	75.2%
Residual Network-101	Table 2.7	Res_conv_5	44.5M	77.0%
MobileNet	Table 2.8	Conv6_pw	3.3M	70.9%
Inception-ResNet-V2	Table 2.9	Inception-Resnet-C	54.33M	80.4%

TABLE 3.1: List of CNN feature extractors experimented

3.2 Usage of Feature Classifier

Due to advent of deep learning and Convolutional neural networks, practitioners are implementing different CNN feature extractors in order to achieve better accuracy on distinct computer vision tasks such as classification, object detection and segmentation. In this research, we have investigated the use of different CNN feature extractors. The implementation of these CNN feature extractors on LSTM-decoder[45] serves dual purpose. Firstly, it gives a in-depth understanding of CNN feature extractors for researchers and makes it easier for practitioners to decide the type of convolutional feature extractors for their application. Secondly, it gives detailed understanding of the LSTM-decoder[45] framework and also list the different scenarios under which this model architecture should be adopted.

The list of all CNN feature extractors which we have analysed is presented in Table 3.1. Generally, for object detection task, large CNN feature extractors are trained on large amount of datasets (like ImageNet[6] and MS-COCO[32]) then, they are fined tuned on a small dataset for a particular type of application. In this thesis, all the CNN feature extractors had been pre-trained on ImageNet dataset [6] and we fine tuned these model architectures with different datasets. We had experimented with three different kind of application; one for pedestrian detector, second for head detector and third for shoulder keypoint detector.

In order to implement a particular LSTM-decoder module, the last layer of the considered CNN feature extractor is removed and it is connected to the regression network (here, LSTM). The feature maps are extracted up to particular layer of CNN; the names of each layer from which feature maps had been extracted are mentioned in Table 3.1 along with number of parameters pre-trained on ImageNet dataset. The Top-1 classification Accuracy for each CNN feature extractor is also mentioned in Table 3.1 which gives an idea of optimization ability of each CNN feature extractor.

For SqueezeNet[23], one important thing to be noticed is that the output feature maps after "*Fire-9*" layer comes out to be "40 X 30 X 512" whereas, all other model

architectures divide the input image into 15 X 20 grid cells. So, for SqueezeNet[23], we added 1 extra layer of dimensional reduction so that the input image can be divided into 15 X 20 grid cells. This extra layer is a convolutional layer with filter size 3 X 3, stride of 2 with the number of output channels to be 1024.

3.3 Regression Network

Generally, for object detection model, regression network is the model which is responsible for obtaining bounding box location around objects using the information from CNN feature extractor. The high-level representation of an image from given CNN feature extractor acts as an input to the regression network which outputs all the possible bounding box coordinates along with their prediction confidence (P_d).

An example of regression of network in shown in figure 3.1 in which high-level representation is transferred to a LSTM-module. LSTM modules act as regression network and are responsible for generating bounding box coordinates at each grid cells. The definition and working of LSTM is given in Section 3.3.2. Before that, let's first discuss the Recurrent Neural Networks (RNNs) which is the base of LSTM architecture. The Applications of RNNs in sequence predictions, image captioning, language translations and video classification are also mentioned.

3.3.1 Recurrent Neural Network

A Recurrent Neural Network(RNNs) is shown in Figure 3.2 which consists of initial state(h_t). At every time step 't', the network is fed with an input vector(X_t) and its initial state gets modified. Obviously, the RNN consists of weights which can be learned. Fine-tuned weights and modified state can be used to predict a vector(Y_t) which is finally used for different applications. Mathematically, the modified hidden state of RNN depends upon the previous state and current input vector at time step 't' as shown in equation 3.1:

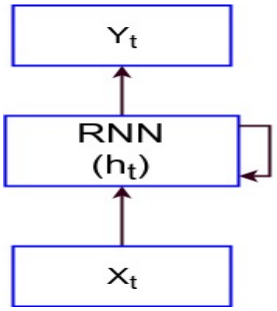


FIGURE 3.2: LSTM Module Explained

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}X_t) \tag{3.1}$$

Then, the vector predictions can be made by projecting the hidden state of RNN (h_t) with weight matrix (W_{hy}) as shown in equation 3.2:

$$Y_t = W_{hy}h_t \tag{3.2}$$

The different kind of RNN configurations for distinct applications are shown in Figure 3.3.

- Configuration 'A' is used for image captioning. It receives a fixed size image as an input. Using RNNs, it predicts a sequence of words which describes the content of the image.

- Configuration 'B' is used for sentiment classification in which a sequence of words is given as input to RNN. Then, RNN predicts the sequence to be positive or negative.
- Configuration 'C' is used for language translation in which a sequence of words in given as input. Then, RNNs translate the sequence to another form (Let's say, from English to French).
- Configuration 'D' is used for video classification in which RNN classify each frame of the video sequence and assign a particular class to each frame.

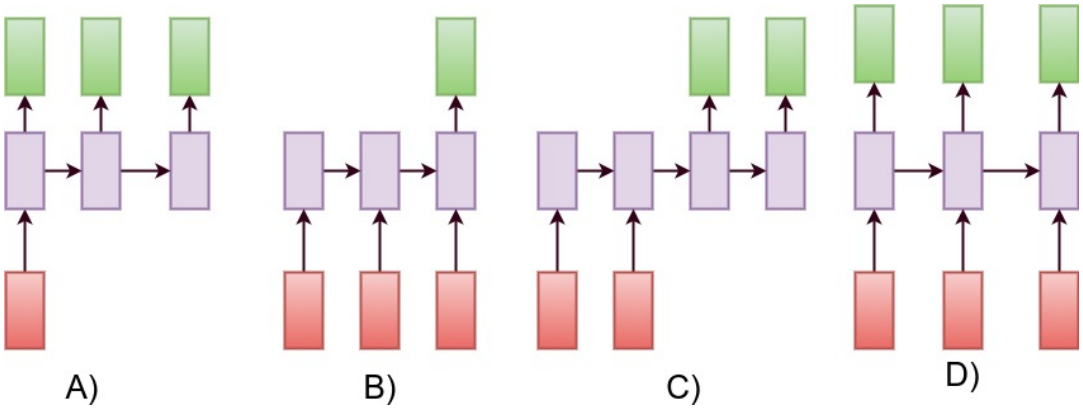


FIGURE 3.3: Different RNN architectures for different applications

To better understand the working and training of RNNs, let's take an example for character level sequence predictor, as shown in Figure 3.4. The idea is that at every time step, a character is fed into the RNN and The RNN is responsible to predict the next character in the sequence. In the example shown in Figure 3.4, the model tried to predicts the sequence "INDIA".

In this sequence, we have 4 characters:['I', 'N', 'D', 'A'] which are fed into the RNN using one-hot representation as shown in Figure 3.4. RNN with its current setting of weights predicts the next character in the sequence. As shown in Figure 3.4, there are few unnormalized log probability which are not correct for predicted sequence and needs to be trained with ground-truths.

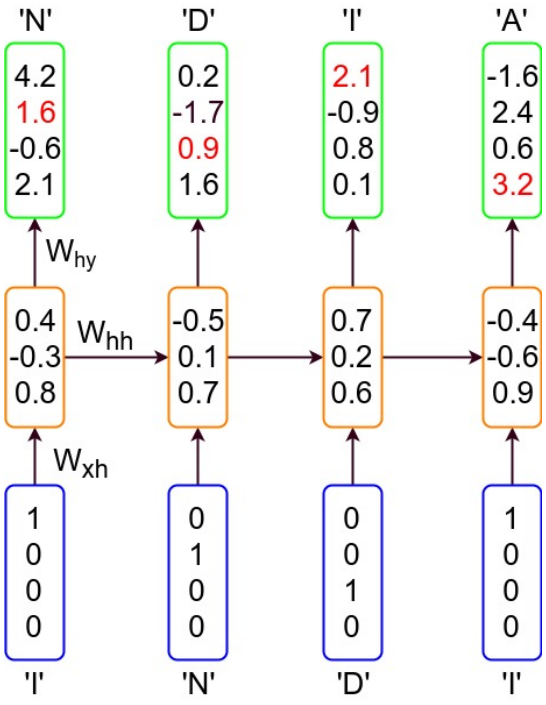


FIGURE 3.4: Character-level Sequence Predictor explained

So, the loss is calculated at each step with respect to ground-truth labels and the gradient from the loss function is back-propagated to change the weights of the RNN. Now, the RNN gets trained to predict the sequence : "INDIA". Similarly, variation of model configurations mentioned in Figure 3.3 can be implemented for different applications.

One limitation in RNN configurations is that as the size of RNN keeps on increasing, the gradient's value becomes smaller. This leads to negligible change in the weights of RNN and model doesn't get trained for specific application. This limitation had been rectified with the introduction of LSTM unit. So, practitioners nowadays replace the RNN units with LSTM units.

3.3.2 Explanation of LSTM

In this section, we described an approach to rectify the memory limitation of Recurrent Neural Networks (RNNs) which is known as Long Short Term Memory (LSTM)[18]. The difference comes in the way that the hidden state of LSTM is calculated. Previously, the hidden state of RNN is calculated by the equation 3.1 and 3.2. With LSTM, the procedure to calculate the hidden state becomes more complex by the addition of extra gates.

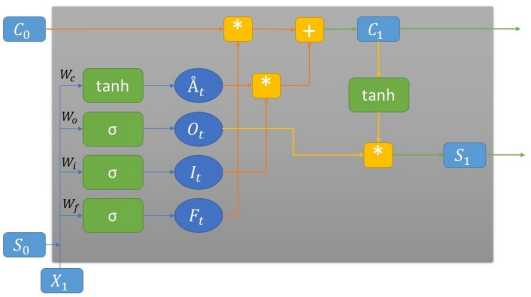


FIGURE 3.5: LSTM Module Explained

In LSTM[11], we can consider the dynamic state of a neural network to be a short term memory. The main idea is that we want to make the short term memory last for a longer period of time. This is done by designing a special module (LSTM) which allows information to be gated-in. Then, it allows information to be gated-out when needed. In the intermediate period, the gate is closed and the information which rises in the intermediate period doesn't interfere with the remembered state.

The architecture of LSTM is made up of 3 gates and 1 cell state. These gates called Forget Gate, Input Gate and Output Gate along with cell state resolved the problem of vanishing gradients. The design architecture of LSTM module is shown in Figure 3.5.

The three gates of LSTM module have individual weights to train which are referred as W_o, W_i and W_f for Output, Input and Forget Gate respectively. S_{t-1} is the previous state of LSTM. C_t refers to the cell state of LSTM.

Mathematically, each gate and cell state can be defined as:[1]

$$F_t = \sigma(W_f S_{t-1} + W_f X_t) - Forget\ Gate \tag{3.3}$$

$$I_t = \sigma(W_i S_{t-1} + W_i X_t) - Input\ Gate \tag{3.4}$$

$$O_t = \sigma(W_o S_{t-1} + W_o X_t) - Output\ Gate \tag{3.5}$$

[1]There are two activation functions which are used in LSTM module, that are, $\sigma(x)$ and $tanh(x)$. The details of these activation functions are mentioned in [42].

$$\dot{A}_t = tanh(W_c S_{t-1} + W_c X_t) - Intermediate\,Cell\,State \tag{3.6}$$

The output from each LSTM module is:

$$C_t = (I_t * \dot{A}_t) + (F_t * C_{t-1}) - Cell\,State \tag{3.7}$$

$$S_t = O_t * tanh(C_t) - Output\,State \tag{3.8}$$

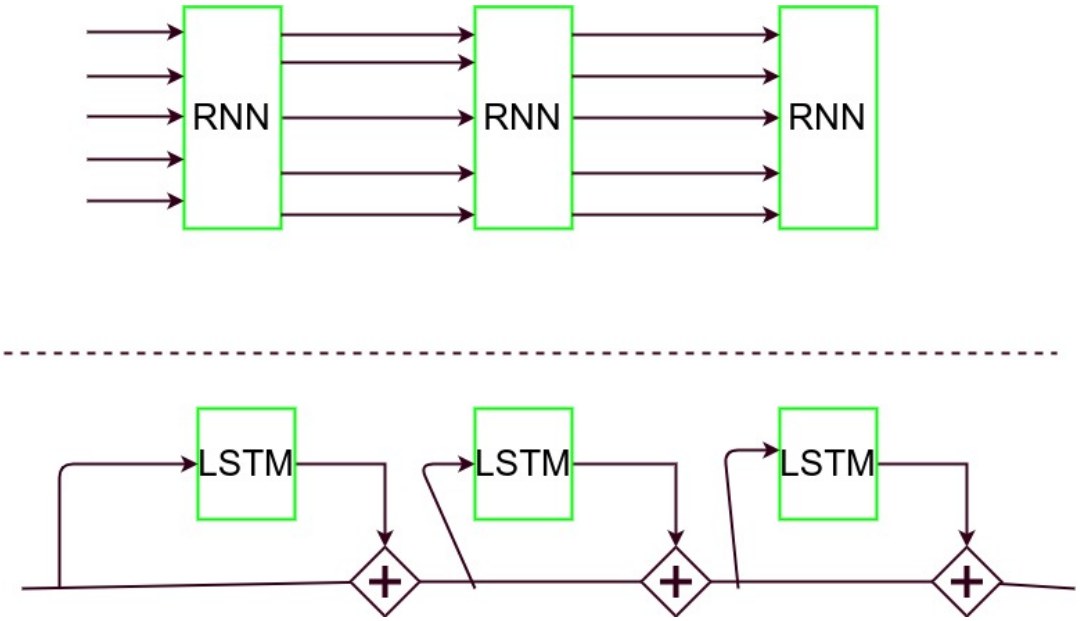


FIGURE 3.6: Difference between RNN and LSTM

Normally, the Recurrent Neural network (RNN) just consists of hidden state vectors but LSTM unit consists of cell state vector (C_t) along with hidden state (S_t) at every single time step. Now, let's assume all the gates of LSTM (that is; F_t, I_t and O_t) to be binary. Cell state equation either shut down the previous state of LSTM to be zero by using Forget Gate (F_t) or adds a value to the cell state. Output State equation squashed the cell state which means only some of the cell states leaked into the hidden state units modulated by output gate (O_t)

In comparison with RNN, LSTM has additive interaction with the model architecture as shown in Figure 3.6. In RNN, the hidden state vector gets completely transformed at every time step whereas, in LSTM, the cell state vectors gets leaked into the hidden states. So, the cell states had additive interactions which allows us to back-propagate more effectively. At some time-step, the gradient signal for RNN becomes smaller and perform negligible change in the weights. On the other hand, gradient flow in LSTM is additive and faster as compared to RNN. Also, LSTM's cell state will end up contributing their own gradients into the gradient flow and it will flow all the way back. Hence, the problem of vanishing gradient for RNN is resolved by LSTM.

3.3.3 Detection using LSTM-decoder

In [45], Stewart et. al. proposed to use LSTM to decode bounding boxes from the deep features computed over an image. This LSTM-decoder architecture is used as a regression network which generates sequence of bounding box prediction in a particular grid cell. This technique particularly effective in predicting small instances in an image. It also improved the accuracy in occlusion specific problems.

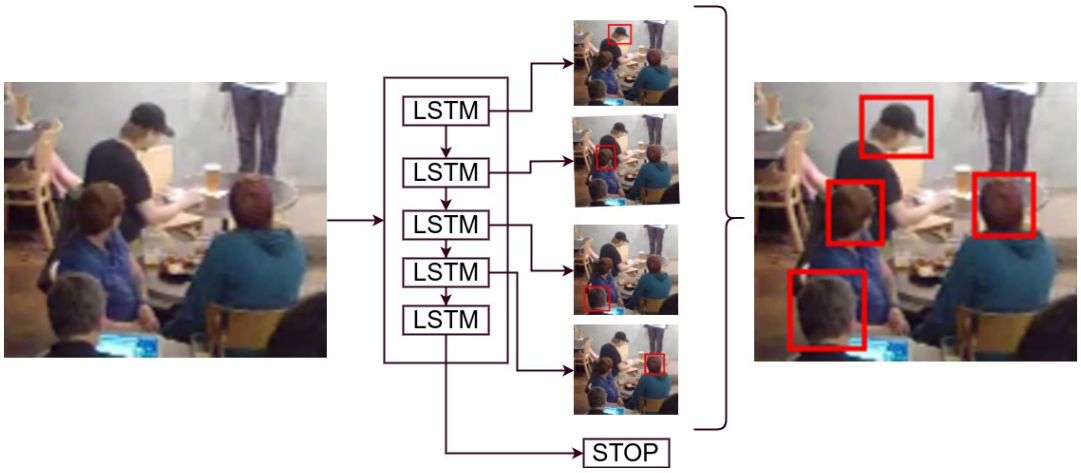


FIGURE 3.7: Working of LSTM in LSTM-decoder[45]

The complete model architecture is shown in Figure 3.1 which consists of feature extractor unit and regression unit. Revising from Figure 3.1, the input image if transformed into a 15x20 grid cells using CNN feature extractor. Each grid cell is transferred to LSTM unit to predict objects. An example of the LSTM unit predicting objects from grid cell is shown in Figure 3.7. Consider an example of a grid cell which fed as an input to a LSTM-unit.

- At each time step, the LSTM module draws information from the grid cell and outputs bounding box coordinates.
- After the first iteration, the bounding box coordinates with highest 'P_d' is presented as output.
- At second iteration, a new undetected head is predicted; its 'P_d' would be lower than first bounding box prediction.
- Finally, When the LSTM is unable to predict another head from the grid cell with a confidence above than pre-specified threshold, then, a STOP symbol is generated.
- The collection of all bounding box coordinates are presented as final predictions from LSTM-decoder.

This process is consequently executed for all cells of the image grid.

3.4 Loss Function

The accuracy of deep learning model architecture depends upon the loss function used for back-propagating the gradient through the convolutional layers. In object detection, the loss function is built from the classification probability (P_d) and the predicted bounding box coordinates, and how well they match with ground-truth annotations. Based upon the computed loss, the gradient is back-propagated to change the weights of model architecture.

The loss function therefore, plays an important role in determining the accuracy of the model architecture. This section explained the general loss function used for object detection models. It also explained the hungarian loss function that was proposed by [45].

3.4.1 General Loss Function

In object detection, the classic loss function is based on an algorithm that uniquely associate a proposed bounding box with a ground-truth box. The loss for the label is dealt with cross-entropy loss. The localization loss is calculated by the L1-Norm or L2-Norm. Mathematically, lets say, classification loss is denoted by $Loss_{pd}$ and localization loss is denoted by $Loss_{bb}$. These two loss function are shown in equation 3.9 and 3.10.

$$Loss_{pd} = -\sum_{i} P_g log(P_d) \tag{3.9}$$

$$Loss_{bb} = \sum_{i}^{n} \left| BB_{gt} - BB_{dt} \right| \tag{3.10}$$

The total loss is the summation of $Loss_{pd}$ and $Loss_{bb}$ shown in equation 3.11

$$.Loss_{total} = Loss_{pd} + Loss_{bb} \tag{3.11}$$

where,

- P_g is the ground truth probability which is 1 for those cells which contains ground truth annotations and 0 elsewhere.
- P_d is the probability of detection for all the object detected in an image.
- BB_{gt} is the ground truth annotations.
- BB_{dt} is the bounding box coordinates for detected objects.

In this, ground truth boxes are made to be sorted with respect to image position, that is, from left to right and top to bottom. Then, this algorithm sequentially assigns candidates proposal to sorted ground truth. This process is known as 'fixed order matching'.

3.4.2 Hungarian Loss function

The major drawback of above mentioned technique arises in the case when model architecture predicts false positives or false negatives. In this, the loss function also assigned weight to false predictions.

Figure 3.8 illustrates the possible detection scenarios. Box 1 and 2 are considered as the ground truth annotations. Box 5 is the false positive. The loss function should discard this box by assigning low probability to this box proposal. Box 3 is an imprecise localization and the loss function should correctly localize the coordinates of the box proposal. Similarly, box 4 is a duplicate assignment and it should also be discarded.

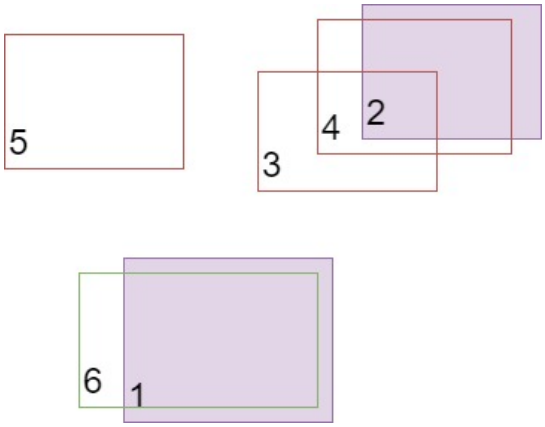


FIGURE 3.8: Different Cases for bounding box proposals. The shaded boxes are ground-truth annotations and other boxes are detection proposals

The loss function should therefore effectively predict the type of problem and act accordingly. The *Hungarian Loss function* performs bipartite matching using Hungarian algorithm[29]. This Hungarian loss function outputs 3 parameters, thus resolving limitation of fixed order matching as shown in equation 3.12

$$Loss_{hung} = \{O_{ij}, R_i, D_{ij}\} \tag{3.12}$$

where,

- D_{ij} is the L1 distance between detected bounding box and ground truth annotations.
- R_i is the rank of the sequence predicted by LSTM as the sequence is made to predict in decreasing order of detection probability.
- O_{ij} is the variable used for penalizing the bounding boxes that sufficiently donot overlap with ground truth annotations.

The output of Hungarian loss function is in three variables mentioned in equation 3.12. The output of these three variables can be used for resolving the problem of assigning weights to false predictions. The output of Hungarian loss function is in the form: (O_{ij}, R_i, D_{ij})

Using the above mentioned loss function, correctly matched boxes 1 and 6 would cost : (0, 2, 0.2)[2]. Matching 2 and 3 would cost: (1, 5, 2.8). Finally, matching 2 and 4 would cost : (0, 6, 0.3). Note that the first term (that is,O_{ij}) effectively handles the case where box predictions have lower rank but it is too far from the ground-truth in order to become a sensible match. This loss function improves the accuracy of the model architecture. It also reduces to false positives to be matched with ground-truths during training.

3.5 Suppression Techniques

The output from LSTM-decoder model architecture [45] is a set of bounding box regression points of objects in a particular image from each cell of the grid. Now, there

[2]values mentioned in output of loss function are just supposition. These are not the actual calculated values. These values are used for explanation of the concept of Hungarian loss function.

might be a possibility that multiple cells detect the same object with small variations in the bounding box coordinates. Figure 3.9 presents the output image from LSTM-decoder in which multiple cells get activated for particular object. The suppression technique is used to suppress the multiple bounding boxes and a single bounding box for a particular object is produced as shown in figure 3.9.

The most common suppression techniques used for eliminitating repetitive bounding box for same object is non-maximum suppression (NMS)[19] as explained in section 3.5.1. LSTM-decoder uses a different technique for suppressing the bounding box. They named that suppression technique "Stitching algorithm" and it is explained in section 3.5.2.

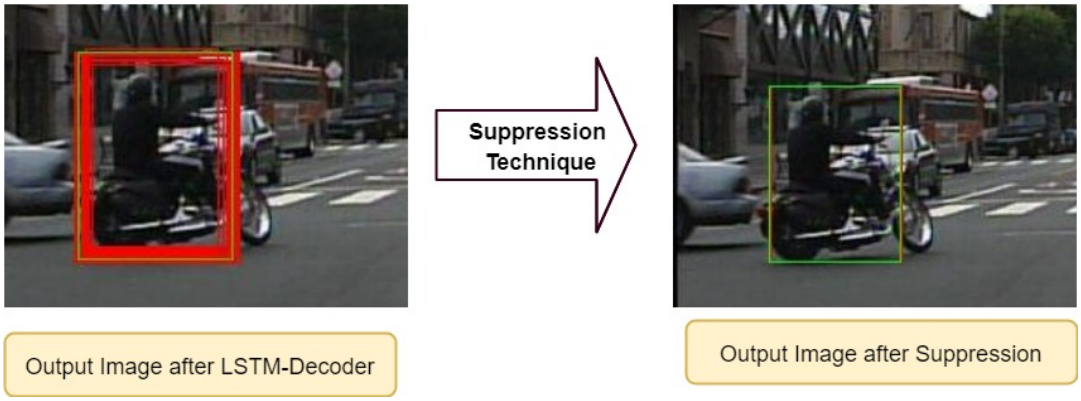


FIGURE 3.9: Simple Block diagram of Suppression Explained

3.5.1 Non-Maximum-Suppression

A general object detection algorithm, produces bounding box coordinates (x, y, w, h) along with detection probability (P_d). To better understand Non-maximum suppression, let's consider the example shown in figure 3.10 in which each cell outputs a bounding box points along with some detection probability. The NMS algorithm will pick the bounding box points with maximum P_d and suppress other bounding boxes which have Intersection-Over-Union (IOU) greater than pre-specified threshold with this bounding box points. This algorithms can be described as follows:

- Consider a detector in which output predictions are in the form: $\{x, y, w, h, P_d\}$
- Discard all boxes with $P_d \leq 0.7$
- While there are any remaining boxes:
  - Pick the box with largest P_d
  - Discard any remaining box coordinates which have $IoU \geq 0.5$ with the box output stated in previous step

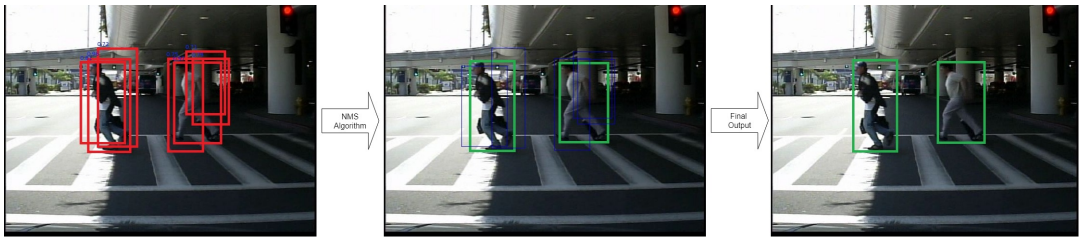


FIGURE 3.10: Block diagram of Non-Maximum Suppression Explained

3.5.2 Stitching Algorithm

As an alternative to NMS, a stitching operation can be applied during inference to merge the predictions from adjacent regions of the input image. In this, a newly proposed bounding box is killed by an already accepted boxes when two conditions are met. First, the new box must have non-zero overlap with some accepted bounding boxes. Second, a previously accepted boxes is allowed to kill at most one new box. Figure 3.11 shows the stitching operation.

FIGURE 3.11: Stitching Operation

In this Chapter, we had discussed about the details of LSTM-decoder[45] model architecture. We explained all the modules used in the model. In the next chapter, we discuss about the implementation details of LSTM-decoder model arhcitecture for pedestrian detection. We implemented the model architecture on pedestrian detection dataset using different CNN feature extractors. We also evaluated and compared our results with state-of-the-art model on Caltech-USA pedestrian detection dataset[7].

Chapter 4

Pedestrian Detection using LSTM-decoder

An important challenge in computer vision application is the pedestrian detection. It proves to be a valuable asset in autonomous driving and home security systems. Based on the LSTM-decoder architecture for object detection presented in the previous chapter, we present 8 pedestrian detector models using different CNN feature extractors in this chapter. We trained these object detection models on a popular pedestrian detection benchmark dataset. We had also evaluated their performances using the proposed benchmark evaluation technique. We compared our result with few state-of-the-art pedestrian detector model architectures and found that we achieved comparable state-of-the-art performance by using different CNN feature extractors.

In this chapter, we used Caltech-USA [7] Pedestrian dataset in order to evaluate our results whose details are mentioned in Section 4.1. Section 4.3 explicates about the implementation details or the hyper-parameters which we had assigned to each model architecture. Lastly, our evaluation results compared with state-of-the-art model architecture are shown in Section 4.4.

4.1 Dataset Used

Dataset is a catalyst for progress in any kind of computer vision applications.Most of the model architectures becomes successful when trained on larger dataset. The dataset must contain all scales of pedestrian annotated at different aspect ratio and even occlusion should be annotated so that those annotations could be used for better accuracy. In general, larger the dataset with above mentioned qualities, higher will be the accuracy of a particular model architecture.

For pedestrian detection, there are several benchmark datasets available. The popular ones are Caltech[7], INRIA[5], ETH[9] and to name a few which are efficient enough and can be used for evaluation purposes. In this chapter, we used a Caltech-USA[7] pedestrian dataset for training and testing the 8 different model architectures because our main motivation is to implement and analyse the best model architecture for ADAS applications.

In Caltech dataset,[7] approximately 10 hours of video sequence were collected from vehicle driving through regular traffic in an urban environment. The camera video resolution is 640x480 and they had annotated approximately 250,000 frames in a 137 minutes long segments. They had labelled a total of 350,000 bounding boxes

and 2300 unique pedestrian. The details of the dataset is summarized in the Table 4.1.

TABLE 4.1: Dataset details of Caltech

Parameters	**Number (in approximation)**
Total Frames	∼1000K
Labelled Frames	∼250K
Frames with pedestrians	132K
Bounding Boxes	350K
Occluded Bounding Boxes	126K
Unique Pedestrians	2300
Average Pedestrian Duration	5 seconds

The labels which are assigned to these bounding boxes are as follows: Individual Pedestrians were labelled as 'Person' (1900 instances). Large groups of pedestrian for which it would be very difficult to label individuals were labelled as 'People' (300). Lastly, a label 'Person?' was assigned to those bounding boxes in which there is no clear identification of pedestrian and network can be easily get confused.

For our pedestrian detection model, we used only the label 'Person' for training the model. We also manually extracted all the 'People' labels and annotated each pedestrian in that group which finally got added into the annotations of "*Person*". We didn't used the annotation named as "*Person?*" because we found that this tends to create a temporal and spatial ambiguity in the neural activation. The model architecture was unable to reach its expected convergence during training.

In that Caltech pedestrian dataset, set00 - set05 are used for training and set06 - set10 are used for testing and a part of training examples are used to build a validation dataset. In Caltech pedestrian dataset, the split between training, validation and testing is made in such a way that there is no spatial and temporal overlap with each other. Also, for testing dataset, one after every 30 frame of the video sequence is used for evaluation.

4.2 List of Model Architectures

For Pedestrian detection, we implemented 8 model architecture that are based on LSTM-decoder using different CNN feature extractors. The list of these implemented model architecture is mentioned as follows:

1. LSTM-decoder with "*VGG-19[43]*" CNN feature extractor
2. LSTM-decoder with "*SqueezeNet[23]*" CNN feature extractor
3. LSTM-decoder with "*Inception-V1[47]*" CNN feature extractor
4. LSTM-decoder with "*Inception-V3[48]*" CNN feature extractor
5. LSTM-decoder with "*ResNet-50[15]*" CNN feature extractor
6. LSTM-decoder with "*ResNet-101[15]*" CNN feature extractor

7. LSTM-decoder with "*MobileNet[21]*" CNN feature extractor
8. LSTM-decoder with "*Inception-Resnet-V2[46]*" CNN feature extractor

We had already discussed the details of these model architecture in Chapter 2 and the summary of these CNN feature extractors had been mentioned in table 3.1.

The list of selected CNN feature extractors is based upon depth, complexity and number of parameters. We studied the effect of increasing the depth of CNN feature extractors on accuracy; the effect of optimized convolutional filters and finally, the effect of increasing number of parameters for training. We started with basic CNN feature extractor, i.e. VGG-19 and evaluated their performance on pedestrian detection dataset. Afterwards, we amended the feature extraction layer with other CNN model architectures. These architecture are selected on the basis of their increased number of layers, optimization capability and number of parameters.

4.3 Implementation Details

There are different platforms available for training and evaluating different deep learning model architecture for various applications. Some of them are: Caffe, Torch, Theano, Keras and TensorFlow. Caffe is the most basic available platforms for training and testing of deep learning model architectures. However, we used open-source TensorFlow framework for training and evaluation of all 8 different model architectures. There are many reasons for choosing this platform for our experimentations. Firstly, the popularity of TensorFlow had inflated tremendously when Google made this platform open-source to everyone for research tasks. Secondly, building model architectures in TensorFlow is much easier than building models in other platforms like Caffe and Torch. Next, TensorFlow comes up with two commonly based wrappers, that is, Python or C++ which makes it easy for computer programmer to start with deep learning algorithms.

The hyper-parameters which we had selected for training our model architectures for pedestrian detection are mentioned in Table 4.2 and further details are mentioned in section 4.3.
The LSTM unit has 500 memory states with zero bias terms and also with no output non-linearities. After the generation of the sequence of detections, predicted bounding boxes are matched with ground truth labels by favouring the candidates boxes which had been shown earlier and closer to the ground-truth targets. When the optimal matching is determined, we back-propagate the loss through the full network.

4.3.1 Training Parameters

For training and evaluation, we used TensorFlow framework with GPU-support. We used a GeForce GTX 980 Ti GPU for training the models and a batch size of 4 images that could be trained in a single iteration. For Training, the optimization algorithm which we had used is RMSprop with initial learning rate "$\alpha = 0.001$" and learning rate decreases after 33,000 iteration according to equation 4.1:

$$\alpha_{new} = \alpha_{initial} * (0.5)^{\frac{i}{\beta}-1} \tag{4.1}$$

TABLE 4.2: Hyper-Parameters assigned to the network for Training and Modelling

Parameters	**Values (to be exact)**
Dataset Parameters	
Caltech dataset frames for Training	40,382
Caltech dataset Frames for testing	39,265
Caltech dataset Frames for Validation	5,104
Display Parameters	
Iterations after which training parameters are displayed	50
Iterations after which Checkpoint weights gets saved	20000
Training Parameters	
Training Algorithm	RMS-Prop
Jitter the training dataset	Yes
Value for epsilon in RMS-Prop	0.00001
Initial Learning Rate Value	0.001
Iteration after which Learning rate Decreases	33000
Initial Weights	SLIM-models trained weights (Trained on ImageNet)
IOU (Intersection over Union) value for Hungarian Loss	0.25
CNN parameters	
Image width	640
Image Height	480
Grid width	20
Grid height	15
Batch Size	4
Region Size (Window size)	32x32
Regression Parameters	
LSTM Size	500
Number of RNN cell	5
Number of Classes	2
Re-Zooming Parameters	
Re-zoom width coordinates	[-0.25, 0.25]
Re-zoom Height coordinates	[-0.25,0.25]

where,

- "α_{new}" is the new learning rate assigned to the model
- "$\alpha_{initial}$" is the initial learning rate which was assigned to model. Here it is 0.001
- "i" is the current step when the model is getting trained
- "β" is the learning rate step which is pre-assigned for the decrement of the learning rate.

4.3.2 Bootstrapping Pedestrian Detection with Hard-Negatives

For improving the average precision, one of the recent concept which was researched by [49] was training the model with hard-negatives which leads to fewer number of false positives in an detected image. So, during training, when the model reaches approximate converegence, that is, after 50,000 iterations, we tried to find false positive images in a validation dataset and these images are then added into the training dataset. Afterwards, the model is trained again and this step is repeated several times until we reach satisfaction on the validation dataset. Note that, we label the detected region as hard-negatives if its maximum IoU (Intersection over Union) with any ground truth annotation is less than 0.5. An example of hard-negative is shown in figure 4.1 by a triangle on the right side of the image.

FIGURE 4.1: Hard-negative image (Top) and Ground Truth(Bottom)

4.3.3 Regularization

In this network, dropout[44] had been used at the output layer of LSTM with a probability of 0.5. We experimentally noticed that if we remove dropout, then AP gets reduced by 0.11. In order to augment the dataset, images were made to jitter with a maximum pixel of 16 in both horizontal and vertical directions along with scaling is performed at random position by a factor which ranges in between 0.9 and 1.1. Using the above changes,we can remove L2 normalization entirely as well as we noticed that when we use L2 normalization with a multiplier of 2e-4 with any of our

implemented models, then the model doesn't get converged. Even if the multiplier is made small as say, 1e-6, then AP gets reduced by 0.03.

4.4 Evaluation Results

Using 8 CNN feature extractors, we implemented LSTM-decoder [45] on Caltech-USA pedestrian dataset[7] and evaluated the performances of each model architecture. The details and script for evaluation technique are available at [2]. We used this the same script for generating our evaluation results and compared our results with state-of-the-art model architecture[8].

Our implementation of LSTM-decoder with Inception-ResNet-V2 [46] achieved a comparable performance with state-of-the-art model architectures. Although, our model architecture and a state-of-the-art model architecture such as Fused-DNN [8] are quite different. It had been fine-tuned on various parameters of the CNN feature extractor and trained with multiple datasets. However, in our case, we only used Caltech-training dataset and achieved a comparable performance.

We evaluated our model architectures on various evaluation techniques listed in [7] and script provided in [2]. We performed our experimentation based on "Reasonable Evaluation" and "Evaluation based upon different scales" which gives a clear evaluation of each model architecture on Caltech-USA pedestrian dataset. The details of above mentioned evaluation are provided in the next sub-section along with results of our 8 model architecture compared with Fused-DNN[8].

4.4.1 Evaluation Technique

For a model architecture to be evaluated effectively, proper evaluation technique for a particular computer vision challenge is crucial. The evaluation protocol has been defined with the major goal of qualitative analysis in a realistic, informative and visually comprehensible manner.

The methodology of "Full-Image Evaluation" has been used in which detection made by a particular model architecture is evaluated for every single image of the test set. Some old techniques of evaluation is the "Per-Window Evaluation" in which evaluation is performed on cropped positive and negative image windows. This evaluation technique is restricted to particular kind of model architecture (like architectures that perform on automatic Region of Interest(ROI)[20][35]). Models is evaluated on test set and after applying stitching technique as discussed in section 3.5.2.

- Let final predictions from LSTM-decoder be denoted as BB_{dt} and ground truth annotations as BB_{gt}.
- The calculations of False positives and false negatives can be calculated by the Area of overlap (a_0) which is defined as:

$$a_0 = \frac{BB_{dt} \cap BB_{gt}}{BB_{dt} \cup BB_{gt}} > 0.5$$

- Detections with maximum confidence are matched first with ground truths.

- Unmatched detections (BB_{dt}) are counted as false positives and unmatched ground-truths(BB_{gt}) are counted at false negatives.
- Finally, detection results are plotted based upon miss-rate Vs False Positives Per Image (FPPI) with decreasing value of detection Probability (P_d). The overall performance of a model architecture is measured in *"log-average miss rate (LAMR)"* which is calculated by averaging the miss rate at 9 FPPI rates.

Furthermore, for better evaluation, some ignored bounding boxes (BB_{ig}) are introduced and matched detected bounding boxes with those ignored bounding boxes won't affect the evaluation performance of any model architecture. Crowded annotations as "People" and occluded annotations as "Person?" comes in this category of BB_{ig}. So, any BB_{dt} which found a match with BB_{ig} won't count as false positive or false negative.

4.4.2 Miss-Rate vs False-Positives-per-Image(FPPI)

Based on the above discussion, we computed our evaluated results of different models used and presented in form of a graph with respect to miss-rate and FPPI. Also, the log-average-miss-rate is computed and shown in their respective graphs.

Reasonable Evaluation

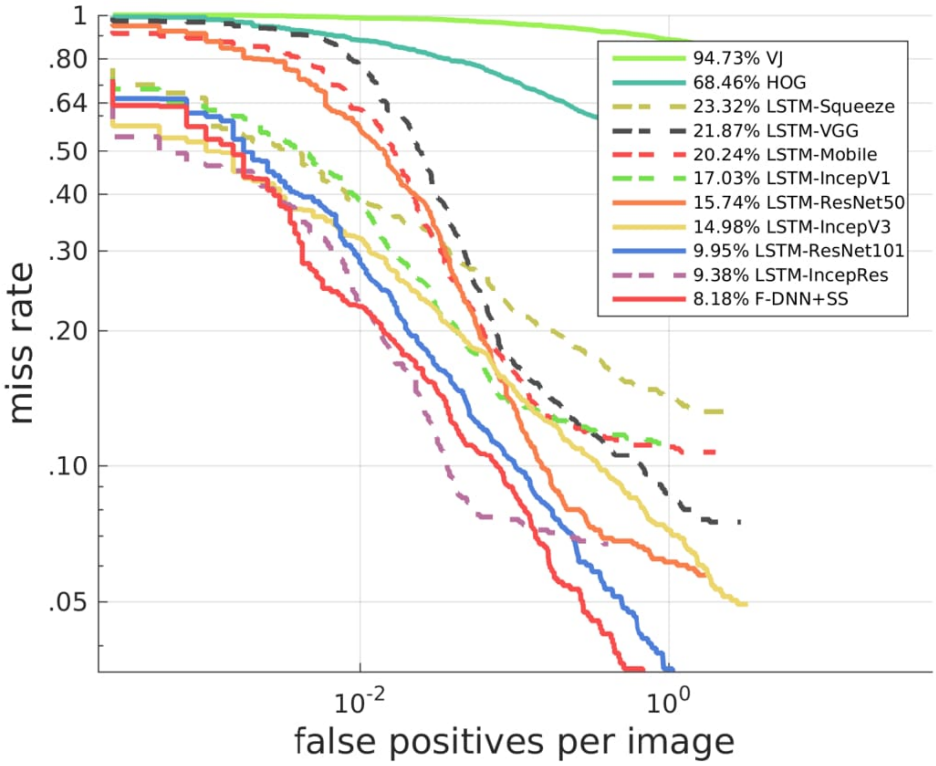


FIGURE 4.2: Miss-Rate Vs FPPI (Reasonable Evaluation)

There are various evaluation technique which are reported in [7] and one of the important evaluation technique is the "Reasonable Evaluation". This means that

the model is evaluated in particular constraints (like height range, visibility range, aspect ratio and many more) and detection made outside this particular constraints are considered as ignored bounding boxes and won't affect the evaluation measures. The height range for reasonable evaluation is between 50 pixel range and infinity. Other than this, the visibility range is also kept between 65% and 100%. Based on this constraints, we established our evaluation result which is shown in figure 4.2.

From figure 4.2, it is clear that our best model architecture, that is, LSTM-decoder with Inception-ResNet-V2[46] acheived 9.38% LAMR whereas state-of-the-art model architecture attained a LAMR of 8.18% which is relatively small difference from state-of-the-art performance. However, if we look closely, it is not fair to compare our model architecture with the state-of-the-art model because the state-of-the-art model architecture was trained on variety of datasets so that the model can be trained at different scales with variable aspect ratio whereas, our model architecture is being trained for Caltech-USA training dataset [7] and reached a comparable results.

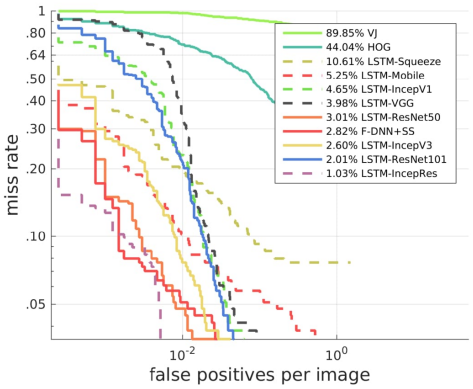


FIGURE 4.3: Miss-Rate Vs FPPI (Near-Scale Evaluation)

Other than this, the LAMR of remaining model architecture, that is, LSTM-decoder with ResNet-50, ResNet-101, Inception-V3, Inception-V1, mobilenet, VGG-Net and squeezenet are 15.74, 9.95, 14.98, 17.03, 20.24, 21.87 and 23.32 respectively. In gist, we came up with the conclusion that CNN feature extractors play a major role in deciding the accuracy of the model architecture.

Evaluation Based on Scale of Annotated Pedestrian

There is an another evaluation technique which is based on the scales of annotated pedestrians (which means the height of Ground Truths labels). In our experimentation, we used three types of scale evaluation, that is, near-scale, medium-scale and far-scale. Near-scale includes a height range for selected ground truths between 80 and infinity pixel range. Outside this pixel range, other detected results matched with ground truth bounding boxes are ignored. Similarly, for medium range, the height range is between 30 and 80 pixel range and for far scale, the height range is between 20 and 30 pixel range.

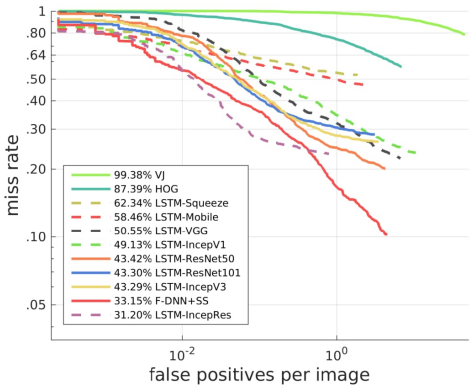


FIGURE 4.4: Miss-Rate Vs FPPI (Medium-Scale Evaluation)

Based on the above constraints, we evaluated our 8 model architectures and results are shown in figure 4.3, 4.4 and 4.5 for near, medium and far-scale respectively.

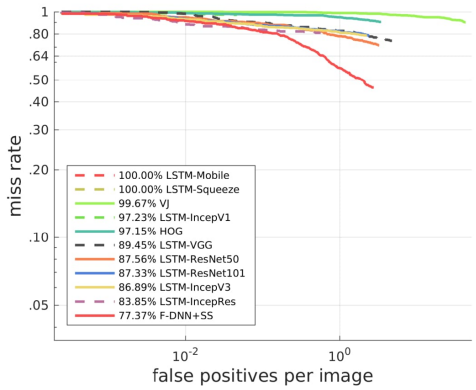


FIGURE 4.5: Miss-Rate Vs FPPI (Far-Scale Evaluation)

For highly computational expensive CNN architecture, that is, Inception-Resnet-V2 [46], we had beaten the state-of-the-art performance for near-scale and medium-scale evaluation as shown in figure 4.3 and 4.4 respectively. However, this model architecture cannot be implemented for real-time application which had been explained in Section 4.4.3. Due to higher number of parameters to be trained, the speed of the model architecture is lower than real-time value.

There are also some model architecture like LSTM-mobilenet and LSTM-squeezenet which could be implemented for real-time applications because of lower number of parameters to be trained. However, there is reduction in the accuracy but this is the sweet spot for speed and accuracy which can be implemented for real-time ADAS (Advanced Driver Assistance Systems) applications.

From figure 4.5, it had been clearly seen that almost all the model architectures are unable to perform under this tight constraint of far-scale evaluation. Even Fused-DNN[8], that is, state-of-the-art model architecture had achieved 77.37% of LAMR which is best as compared other evaluated models but it had also shown a debacle in its performance in this hard and strict evaluation technique.

4.4.3 Speed, Accuracy and Memory requirements

Along with evaluating the above mentioned models on Caltech Dataset, we also calculated the speed for each model architecture. The comparison of speed vs accuracy give more information to a practitioner regarding the implementation of different model architecture for real-world applications. A comparison of speed and accuracy had been made by [22]. Taking motivation from [22], we also reported the speed of all model architectures which we had implemented in Frames-per-Second(FPS).

The value of Frames-Per-Second gave a in-depth analysis to practitioner. Based upon this information, the practitioner can easily make a decision regarding the choice of CNN feature extractor for their own requirements. We used Tensorflow profiling tool[36] for calculation of Frame-Per-Second (FPS) on GPU as well CPU. Tensorflow Profiler also calculated the size of model architecture if the practitioner wants to deploy the model on a embedded architecture. All these details are mentioned in Table 4.3.

We used Intel Core I7-7700HQ Processor with 16GB RAM and a NVIDIA GeForce GTX-980 Titan-X. CPU configuration consists of 4 cores, 8 threads and 6 MB cache memory.Timings on GPU and CPU are reported with batch size of 1. The size of all images are 640x480x3. We also took the average over 500 images to report the timing

on GPU and CPU.

Model Archietecture	**FPS on GPU**	**FPS on CPU**	**Memory Requirement (MB)**	**Log-Average Miss Rate**
LSTM-squeezenet	25.52	10.68	47.4	23.32%
LSTM-mobilenet	21.94	8.03	60.3	20.24%
LSTM-Inception-V1	18.68	6.74	111.6	17.03%
LSTM-VGG	13.43	5.38	128.5	21.87%
LSTM-Inception-V3	12.56	4.25	189.4	14.98%
LSTM-Resnet-50	8.89	2.60	278.2	15.74%
LSTM-Resnet-101	5.12	1.06	476.7	9.95%
LSTM-Incep-resnet-V2	1.78	0.33	876.2	9.38%

TABLE 4.3: Speed, Accuracy and Memory requirement of different Model Architectures

In this Chapter, we had discussed about pedestrian detection using LSTM-decoder and presented our model architecture's results on Caltech-USA pedestrian detection dataset[7]. Our best model architecture had achieved comparable state-of-the-art performance. We also presented comparison between speed and accuracy for different model architectures.
In the next chapter, we discuss about head detection using LSTM-decoder. The implementation details are same but the difference comes in our proposed experimentations. We used 3 different variety of datasets and experiments are proposed based upon these datasets. This chapter is also a part of our achievement and it had been presented in 15^{th} Conference on Computer and Robot Vision.

Chapter 5

Head Detection using LSTM-decoder

Along with pedestrian detector, we implemented a LSTM-decoder network for the problem of people's head detection. Actually, LSTM-decoder for object detection as introduced in[45] was initially implemented for head detection and they had come up with their own dataset named as "*Brainwash*". Their main motivation was to design a model architecture which could resolve the problem of occlusion in a crowded environment. They implemented their model architecture using Inception-V1[47] feature extractor and shown their best results on Brainwash dataset.

As we did in the previous chapter, we implemented LSTM-decoder [45] using different CNN feature extractor so that we can compare the original model architecture with LSTM-decoder amended with different CNN feature extractor. In this chapter, we also introduced two new head detection dataset which we had annotated and these datasets are used for evaluating LSTM-decoder on different scenarios.

For our head detection experiments, we used 7 CNN feature extractors, that are, Squeezenet[23], MobileNet[21], GoogleNet (initially implemented by [45])[47], Inception-V3 [48], ResNet-50[15], ResNet-101[15] and Inception-ResNet-V2[46]. The hyper-parameters which we had assigned to each individual model architecture are the same as the ones presented in Section 4.3 of Chapter 4. The difference comes in the introduction of two new datasets and these new datasets gave us the opportunity to design 6 different experiments based upon the combination of all three head detection datasets.

Summarizing this chapter, Section 5.1 discusses about the three different head detection datasets along with variation of annotations with respect to scales in each dataset. Section 5.2 explains about the implementation details for 7 model architectures and it also includes the list of experiments which we had designed for comprehensive analysis of each model architecture. Section 5.3 shows the evaluation results for each experiment along with some detailed analysis.

This experimental study had been presented at the 15^{th} conference on Computer and Robot Vision which also include a comparative performance analysis of LSTM-decoder architecture with the SSD[33] object detector.

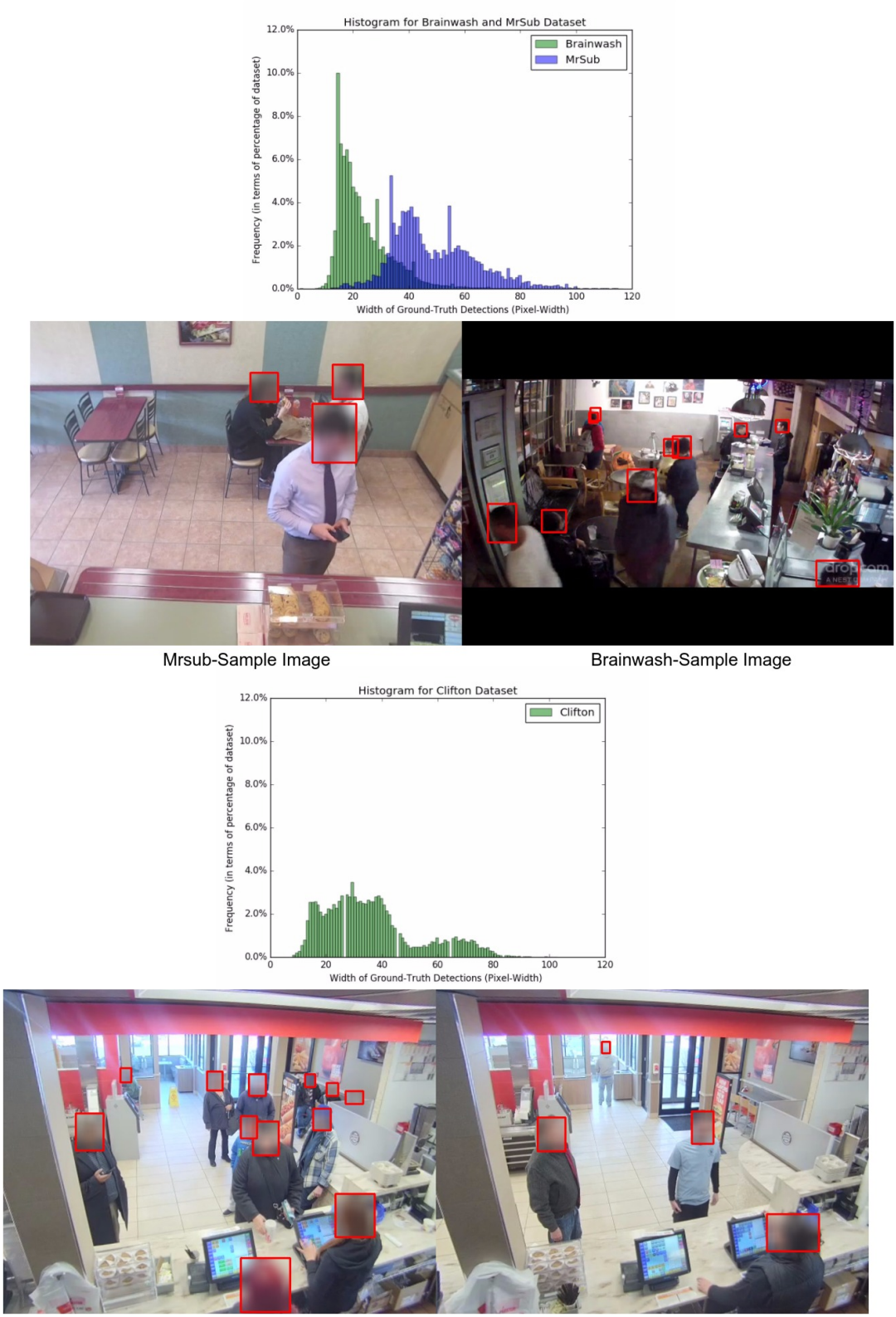


FIGURE 5.1: Description for size of annotations for Brainwash, MrSub and Clifton along with sample images

5.1 Dataset Used

As we had also discussed earlier, datasets played a vital role in determining the accuracy of the model architecture. However, in this chapter, we propose to use different datasets to explore the pros and cons of the LSTM-decoder architecture experimented at different scenarios.

To explore the weaknesses and strengths of the model architecture, we experimented on three different head detection datasets – Brainwash, MrSub and Clifton, all of which are collections of scenes from different restaurants. Brainwash is publicly available and was introduced by [45]. MrSub and Clifton are our own collections of images for head detection task.

Brainwash, MrSub and Clifton are the datasets designed for head detection but there is major difference between these datasets. Brainwash presents a scene from heavily crowded restaurant and includes a lot of occlusions in the images. The size of heads are not too large. On the other hand, Mrsub is a scene from a general fast food restaurant with little occlusion in the dataset. The size of heads are either medium size or large. Lastly, Clifton dataset can be categorized as a mixture of the above two mentioned datasets; this datasets consisting of small, medium and large size head annotations. Some of the images have heavy occlusions whereas, some just consists of large head annotations. This is the reason, why we only perform our testing on Clifton dataset and it has not been used for any kind of training in order to assess generalization of our head detectors.

Please note that, for our experiments, we denote head instances that are smaller than 20x20 pixel-area as small, between 20x20 and 40x40 pixel-area as medium and greater than 40x40 pixel-area as large. In order to show the variations for these datasets, we made histograms for ground-truth regions for all 3 datasets and the graphs are shown in figure 5.1 for Brainwash, MrSub and Clifton dataset. From figure 5.1, the variation in the ground-truths for 3 different datasets can be clearly testified and specifically, there is significant variation shown in terms of scales of head annotated for Brainwash and MrSub dataset. Figure 5.1 also shows some sample images from each of the three datasets which gives some indication of the variations of head instances for the 3 different datasets. Note that in our experiments, we ,made sure that no temporal overlap exist between the training, validation and test datasets. In particular, we use images from different days for each dataset (that is, brainwash, Mrsub and Clifton).

5.1.1 Brainwash dataset

Table 5.1 gives details about the Brainwash dataset, while Fig. 5.1 plots the distribution of the scales of the head instances in the dataset. As obvious, the dataset is mainly dominated by very small head instances with challenges such as, strong partial occlusions and large variability in clothing and appearance. This is why we used this dataset to explore the model architecture's capability to pick up smaller instances as well as robustness to occlusion as reported in Section 5.3.

5.1.2 MrSub dataset

MrSub includes frames captured from two front-facing surveillance cameras installed behind the left and right side of the counter of a general fast food restaurant. Images were captured at 2 frames/minute for a period of 10 hours per day over several days. Few human annotators labelled the images and images not having any people were discarded. The whole dataset is split into train, test and validation sets containing images captured in different days. Table 5.1 gives details of the dataset while Figure 5.1 plots the distribution of the scales of the head instances in the dataset. Compared to Brainwash, MrSub mainly includes larger head instances with little occlusions. We used this dataset to see the baseline performance of different model architecture.

5.1.3 Clifton Dataset

Lastly, Clifton dataset, which was collected in a similar fashion as MrSub, can be categorized as a mixture of the two datasets mentioned above, for the dataset includes a good balance of small, medium and large size head instances as shown in Figure 5.1. Also, the dataset includes images with moderate occlusions as shown in Table 5.1. This is the why we used Clifton dataset only to report the generalization performance of the two architectures and did not used for training any model.

TABLE 5.1: details about brainwash, MrSub and Clifton datasets

brainwash dataset	
number of training images	10,769
number of validation images	500
number of testing images	500
Total number of annotations	90,298
Annotations Per Image	7.67
Occlusion Percentage	17.99%
Mrsub dataset	
number of training images	2,511
number of validation images	500
number of testing images	500
Total number of annotations	7,842
Annotations Per Image	2.23
Occlusion Percentage	6.49%
Clifton Dataset	
number of training images	2,090
number of validation images	300
number of testing images	300
Total number of annotaions	13,065
Annotations Per Image	4.86
Occlusion Percentage	12.21%

5.2 Implementation Details

For implementing LSTM-decoder using 7 different CNN feature extractor, we used tensorflow Framework which we had already discussed in Chapter 4. However, the difference comes in training the network on a more advanced GPU. For training and

evaluation, we used NVIDIA TITAN X_p based on Pascal Architecture which have a frame buffer of 12 GB with memory speed of 11.4GBps and it is capable to boost the clock by 1582Mhz.

Using this GPU, we are to reach model convergence 3 times faster than previously available GPU (that is, NVIDIA TITAN-X) because for our experimentation, we had increase the batch size to 8 images which was previously assigned to 4 images. All the remaining hyper-parameters shown in table 4.2 are kept same for these experimentation of head detection. Please note that in this chapter, we are not aiming to achieve the state-of-the-art performances on a particular dataset rather, our main goal is to check that whether LSTM-decoder[45] is capable enough to perform in an environment different from the one it has been trained with. We didn't used bootstrapping[49] in these experiments just for avoiding over-rated performance of a particular model architecture. Also, this work is part of a comparative study in which SSD [33] model architecture was included. This is also the reason that we didn't used bootstrapping as we want evaluate both the model architectures on same grounds.

The major contribution of this chapter comes in the form of experiments which we designed using 3 different datasets and 7 different model architectures. Our main focus is to test the generalization of LSTM-decoder that is study how the model trained for particular setup is capable to perform at different scales of head instances, even for those that it had never been trained. Secondly, these experiments are made for verifying the domain adaptation capability which means the model trained on all scales should efficiently perform on any kind of environment.

Consequently, here is the list of experiments we performed[1]:

1. Models trained and tested on MrSub
2. Models trained and tested on Brainwash
3. Models trained on Brainwash and tested on MrSub
4. Models trained on MrSub and tested on Brainwash
5. Models trained on Brainwash and tested on Clifton
6. Models trained on Brainwash+MrSub and tested on Clifton

5.3 Evaluation Results

In this section, the results along with analytical discussion of all the experiments which are listed in previous section(5.2) are discussed. The evaluation measure which we had used for evaluating different model architecture is Average-Precision (AP) as explained in section 5.3.1. We designed 6 experiments which are based on 3 datasets and 7 different model architectures.

[1]Models trained and tested on same dataset means that the bifurcations shown in table 5.1 are used. Model trained a particular dataset and tested on different dataset means that the whole dataset is used for training and the models are tested only on the testing images of that particular dataset.

5.3.1 Evaluation Technique

For understanding the evaluation measure (Average-Precision), we need to understand the "*Confusion Matrix*" first. It is table which is often used for evaluating the performance of model architecture built for classification or object detection task. Consider the case of a binary classifier tested with 165 samples. So, the confusion matrix for this classifier could be as shown in table 5.2.

From Table 5.2, it is clear that False-Positive (FP) is defined as the total number predictions whose IoU didn't reached a pre-specified threshold with ground-truth annotations. False-Negative (FN) is defined as the number of ground-truth annotations whose IoU didn't reached a pre-specified threshold with any of the predictions of the model architecture. The same definition criteria follows for True-Positive(TP) and True-Negative(TN)

Number of Samples:165	**Not Predicted**	**Predicted**	
Ground-Truth=No	True-Negative = 50	False-Positive = 10	60
Ground-Truth=Yes	False-Negative = 5	True-Positive = 100	105
	55	110	

TABLE 5.2: Confusion Matrix explained using sample data

So, the Precision is defined as the ratio of True-Positives over the total number of True-Positives and False-Positives. Precision is a measure of exactness or quality. Higher precision value means model architecture's prediction are more reliable but this doesn't means that model architecture predicts all the ground-truths in an image.

$$Precision = \frac{TP}{TP + FP}$$

Similarly, Recall is defined as the ratio of True-Positives over the total number of True-Positives and False-Negatives. Recall is a measure of completeness or quantity. Higher recall value means model architecture's predictions consists of almost all the ground-truth predictions but this doesn't mean that the number of predictions doesn't include a high number of False-positives.

$$Recall = \frac{TP}{TP + FN}$$

So, a graph is plotted between Precision and Recall and Average Precision is calculated as the mean average values of precision over all the values of recall between 0 and 1. For each test case, we plotted Recall vs (1-Precision) curve and average precision is calculated for each model architecture. Higher the value of Average-Precision, more optimized the model is considered.

5.3.2 Models Trained on MrSub; Tested on MrSub

Firstly, in order to check the baseline performance of each model architecture, we trained all models on MrSub training dataset and tested their performances on MrSub testing dataset. Figure 5.2 plots the Recall vs. (1-Precision) curves for all seven model configurations trained and tested on MrSub. As mentioned earlier, MrSub being relatively less challenging, it is used to see the baseline performance of each model architectures. As shown in the figure, all models perform very well on this

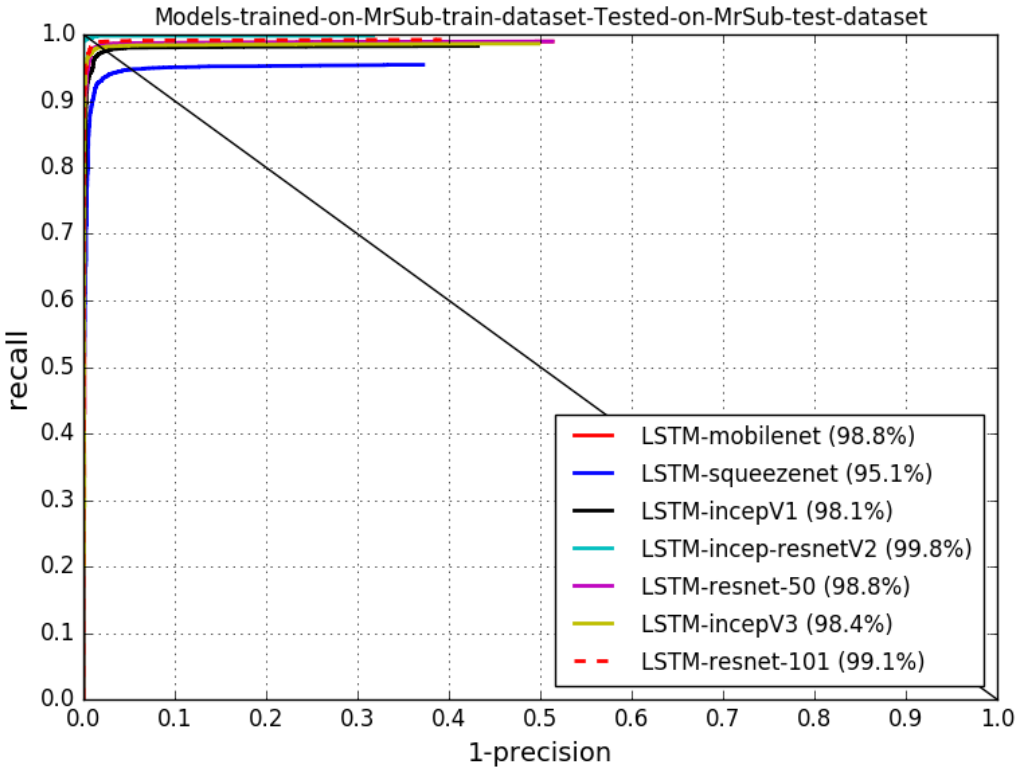


FIGURE 5.2: Recall Vs. (1 - Precision)[AP mentioned in Legends]

dataset achieving an AP in the ballpark of 99%. This clearly shows that the LSTM-decoder with any CNN feature extractor is equally good (in terms of accuracy) for a head detection scenario having low occlusions and without tiny scales of people.

As shown in Figure 5.2, even a LSTM-decoder with squeezenet achieved 95.1% of AP which means the model architecture can also be implemented for real-time applications which got satisfied with low occlusion rate and medium scale of people.

5.3.3 Models Trained on Brainwash; Tested on Brainwash

Figure 5.3 plots the Recall vs. (1-Precision) curves for the different models trained and tested on Brainwash. As mentioned earlier, Brainwash is a highly challenging dataset with strong partial occlusions and very tiny scales of head instances. On this dataset, the modification which we had performed by changing the CNN feature extractor showed better precision as compared to the original LSTM-decoder that used Inception-V1[47] as a CNN feature extractor. So, just by changing the CNN feature extractor, we had increased the Average-Precision by approximately 10% as shown in figure 5.3(referring to LSTM-incep-resnetV2). This can be attributed to the fact that the use of the LSTM modules on each cell of the feature map grid allows the LSTM-decoder to discover partial or even strongly occluded head instances based on the context from an already detected head in a previous LSTM step. Again, since the LSTM-decoder is designed to only look at a particular tiny scale (in this case, a 32x32 window of the input image), it can pick up smaller head instances. This

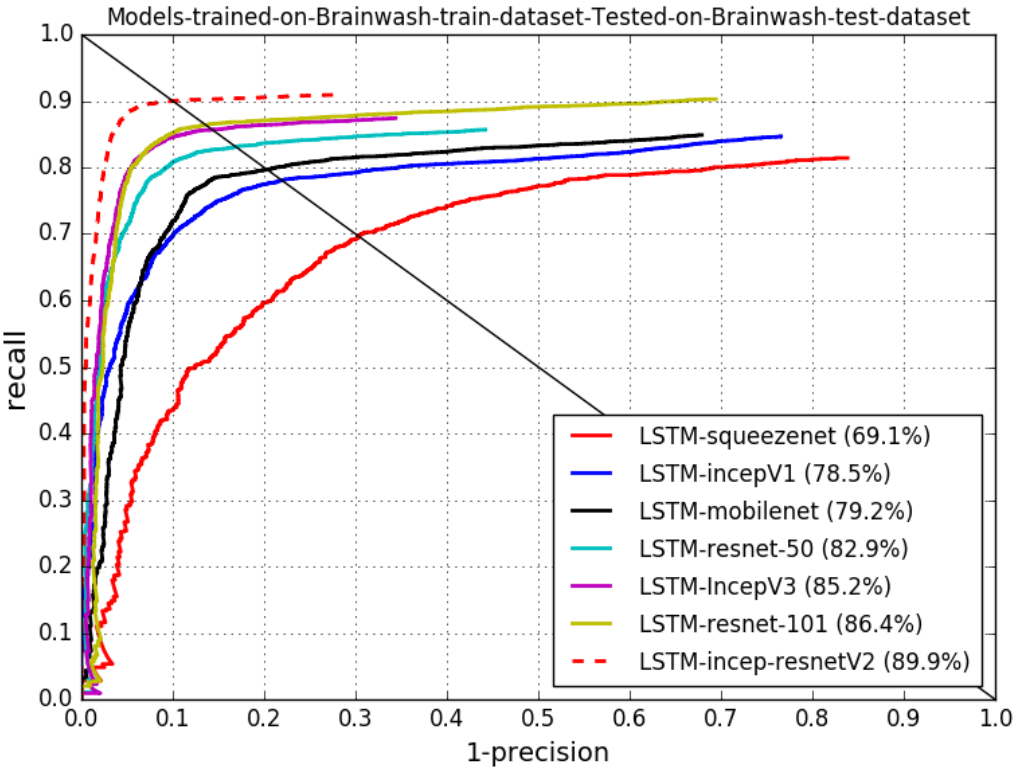


FIGURE 5.3: Recall Vs. (1 - Precision)[AP mentioned in Legends]

results show that for tiny scales, especially with moderate to strong occlusions, performance wise LSTM-decoder is a better choice.

Moreover, we saw that, within the same architecture, model performance varies depending on the feature extractor used. For example, for the LSTM-decoder, the Inception-ResNet-V2, being the most powerful one among the 7 CNN feature extractors, produces the highest AP, followed by ResNet-101, Inception-V3, Resnet-50, MobileNet, Inception-V1 and finally Squeezenet.

5.3.4 Models Trained on Brainwash; Tested on MrSub

In order to allow us to comment on the generalization ability of the these model architectures over new tasks, we trained models on Brainwash dataset and tested their performances on MrSub. Figure 5.4 shows the Recall vs. (1-Precision) plots for different models based on LSTM-decoder.

We see that there is sudden drop down in average precision of all model architectures when tested on Mrsub dataset. Even the best model architecture, that is, LSTM-decoder with Inception-ResNet-V2 achieved an average precision of just 75.4%. As mentioned earlier, Brainwash is a dataset dominated by tiny head instances, whereas, MrSub is mostly including large head instances. Since the LSTM-decoder is designed to look only at a particular scale, it fails to generalize over the new dataset having different distributions in scales.

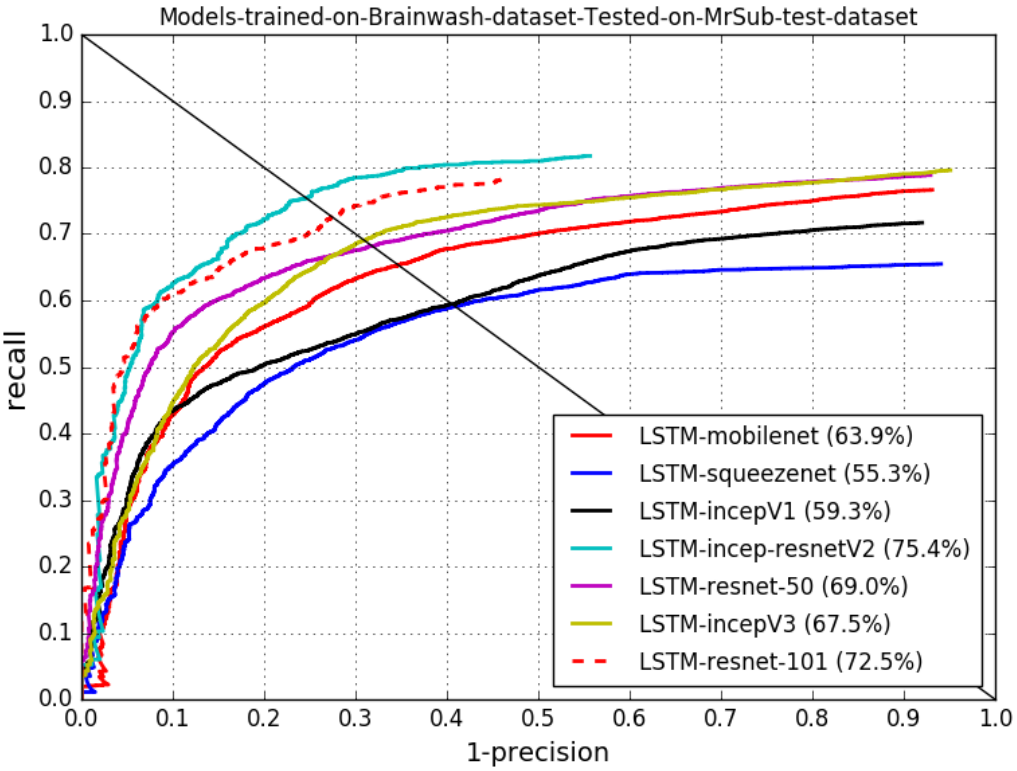


FIGURE 5.4: Recall Vs. (1 - Precision)[AP mentioned in Legends]

However, generalization is always a highly desirable property of any machine learning model, this findings could be crucial for practitioners in deciding the right architecture based on the application at hand. In gist, considering the implementation of LSTM-decoder for real-time applications, practitioners had to dig down some nails on the dataset and they have to train LSTM-decoder on larger dataset which consists of annotations at all scales.

5.3.5 Models Trained on Mrsub; Tested on Brainwash

We also tried the opposite way in which models are trained on MrSub and tested Brainwash. In this experiment, the results were extremely poor for all the model architectures generating AP values close to 0. To be specific, none of the model architecture is able to reach average precision of 2%. Even, LSTM-decoder with Inception-ResNet-V2 CNN feature extractor showed the average precision of 1.4%. From this debacle,we can conclude that Brainwash consists of too small people's head annotations and consists of a lot of occlusions which makes extremely difficult to generalize over from an easier dataset like MrSub.

5.3.6 Models Trained on Brainwash; Tested on Clifton

Performing training and testing on different datasets, this experiment was also designed in which models are trained on Brainwash and tested their performances on Clifton. Since, the idea of this experiment replicates the idea mentioned in section 5.3.4, the evaluation results are approximately the same as shown in figure 5.4. The

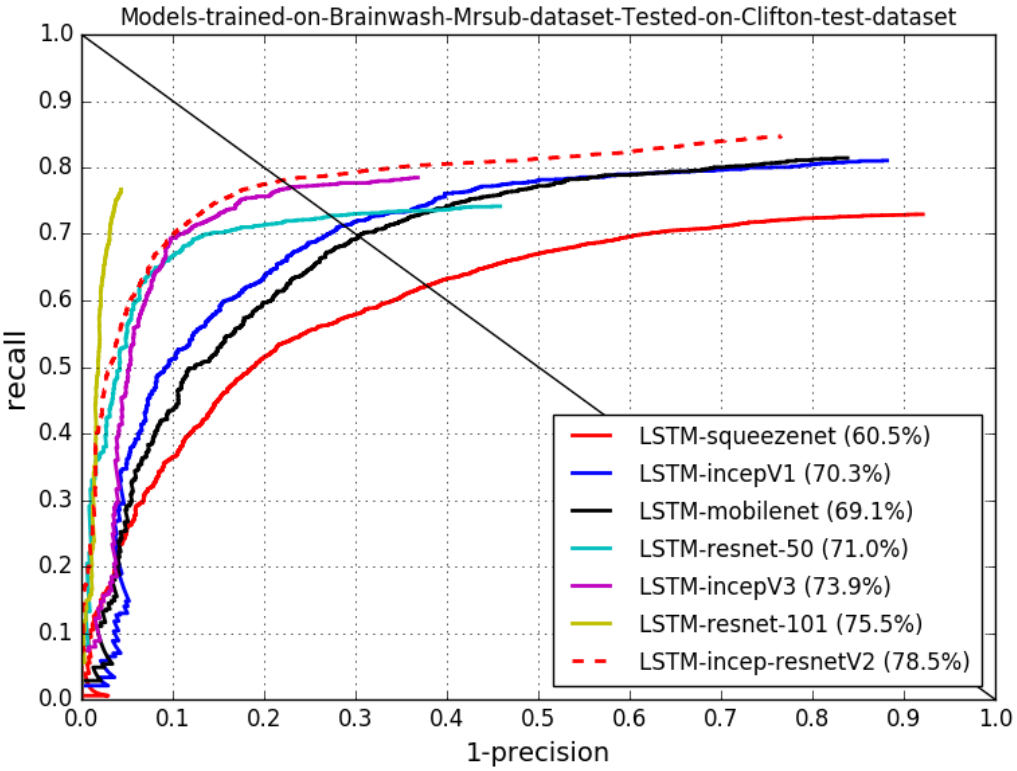


FIGURE 5.5: Precision Vs. (1 - Recall)

difference comes in the range Average-precision measure for all model architectures. The best model (that is, Inception-ResNet-V2) achieved average precision of 71.4% and the least performing model (that is, SqueezeNet) achieved the average precision of 49.8%. We didn't include the figure for this experiment as the analysis of this experiment is same as we had discussed in section 5.3.4.

5.3.7 Models Trained on Brainwash+MrSub; Tested on Clifton

We also trained models on a combination of Brainwash+MrSub datasets. Figure 5.5 shows the Recall vs. (1-Precision) curves. The purpose of this particular experiment was to explore the domain adaptation ability of LSTM-decoder[45]. Since Clifton has a good balance of tiny, medium and large heads, we used it for this particular test. While in the previous case when the LSTM-decoder models failed to generalize to MrSub after having trained on Brainwash, in this case, their performances are competitive with other model architecture. This result suggests that LSTM-decoder is able to adapt to a new domain when the training set covers a full range of scales.

In this Chapter, we had evaluated the performance of LSTM-decoder model architecture for occlusion specific problems, generalization ability and domain adaptation ability. We concluded that LSTM-decoder successfully achieved significant result for occlusion specific problems and domain adaptation ability. However, it fails to show significant results for generalization ability because LSTM-decoder model architecture is trained for a particular scale head instances. In the next Chapter, we discusses about our extension to object detection model architecture that could also be used for detecting shoulder keypoints.

Chapter 6

Shoulder-Keypoint Detector

Human pose estimation is also a challenging task in computer vision. With the advent of CNN feature extractors, the accuracy in human pose estimation had reached milestones on different datasets which are open-source for research work. Before the advent of CNN feature extractors, it was extremely difficult and computationally expensive to detect different keypoints of the human pose structure at variant challenges like the ambiguity of lighting, occlusions and other activities. Now, many researchers had created state-of-the-art end-to-end model architecture for human-pose estimation on different datasets. However, we will not discuss the success story of each model architecture. Instead, we had discussed the theoretical and implementation details of our shoulder-keypoint detector whose computational cost is much smaller than other state-of-the-art model architecture and successfully achieved comparable state-of-the-art performance on MPII dataset[1].

The model architecture which we had designed for shoulder keypoint detection uses information from object detection model and bounding box coordinates from object detection model is used for accurate detection of shoulder keypoints. Similar technique was used by [50] and details had already been explained in Section 2.4 of Chapter 2. Taking motivation from [50], we had designed our model architecture for shoulder keypoint detection and is explicated in this chapter.

Shoulder keypoints are not well defined as other body parts. However, they can be used to determine the position of the person with respect to the camera. These keypoints can be used to delineate torso in people images which can be useful for person re-identification.

So, this chapter is an extension to object detection algorithm in which information of bounding boxes from LSTM-decoder model had been used for detecting the shoulder keypoints. We designed two different strategies to build model architecture for shoulder keypoint detection. The details of the two different model architectures are explained in explained in section 6.1 of this chapter. We used 3 different datasets for training and evaluation of our model architecture and details of those datasets are explained in Section 6.2. Afterwards, implementation details are mentioned in section 6.3. Finally, our evaluated results on MPII dataset are presented in section 6.4.

6.1 Model Architecture

We designed two different strategies for detecting shoulder keypoints from object detection model. First is the use of external cascade network for detecting shoulder

keypoints. It means that the detection from object detection model act as input to a cascade network which had been trained for detecting shoulder keypoints. Second method is amending the object detection model. Using the same feature maps which are trained for object detection are made to fine-tuned to detect shoulder keypoints as well. This method reduces the number of parameters but we also found a significant reduction in the accuracy.Both of the model architectures are explained in this section.

6.1.1 Shoulder Keypoints Detection using External Cascade Network

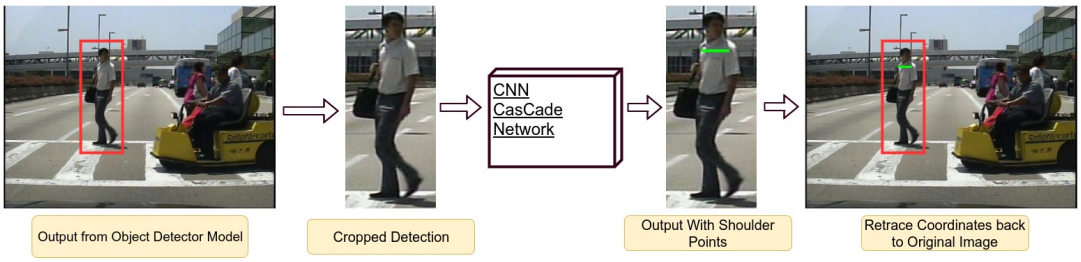


FIGURE 6.1: CNN Cascade Network for shoulder keypoints

In this, we used external cascade network for detecting shoulder keypoints. The model architecture can be easily explained using figure 6.1. The detection from LSTM-decoder[45] is used for cropping the image. The cropped image act as an input to the cascade network which outputs shoulder keypoints. Finally, the shoulder keypoints are back-tracked to the original image to represent object detected with shoulder keypoints.

From Chapter 4 and 5, we had experimented LSTM-decoder on pedestrian as well as heads using different CNN feature extractors. So, the model architecture for shoulder keypoint detector is different for different kind of objects. The main difference comes in data pre-processing technique for pedestrian and heads because the input image which goes to the CNN feature extractor should be a constant size (here, 64x64x3) and data needs to be pre-processed so that input image should precisely consists of shoulder keypoints.

Data Pre-processing

The input image for shoulder keypoint detection needs to be pre-processed before feeding it into the CNN feature extractor. As we had developed 2 different models for shoulder keypoint detection; One from pedestrian detector and second from head detector. The input images need to be pre-processed differently with respect to the kind of object detected at object detection model [45]. So, the data pre-processing from both detectors are different and are explicated as follows:

Pre-processing from Pedestrian Detector: The algorithm which we had used for data pre-processing can be easily explained from figure 6.2.

- Let's say, BB_{dt} is the set of all detection from LSTM-decoder where, $BB_{dt} \in \{x, y, w, h\}$ that are coordinates of detected bounding box.

- If the height (h) of the bounding box is greater than pre-specified threshold (H_t)[1]:
- Also, If the width(w) and height(h) is greater than 64 pixel value:
  - $BB_{cropped} \in \{x, y, w, h^1 = (h/2)\}$
- Else, ignore other detected bounding box coordinates.
- Afterwards, if width(w) is equal to height(h^1) of cropped bounding box:
  - Rescale the Cropped Image to (64x64x3) image size.
- Else, adjust the height(h^1) and width (w) of cropped box to become a squared box. Then, rescale to (64x64x3) image size

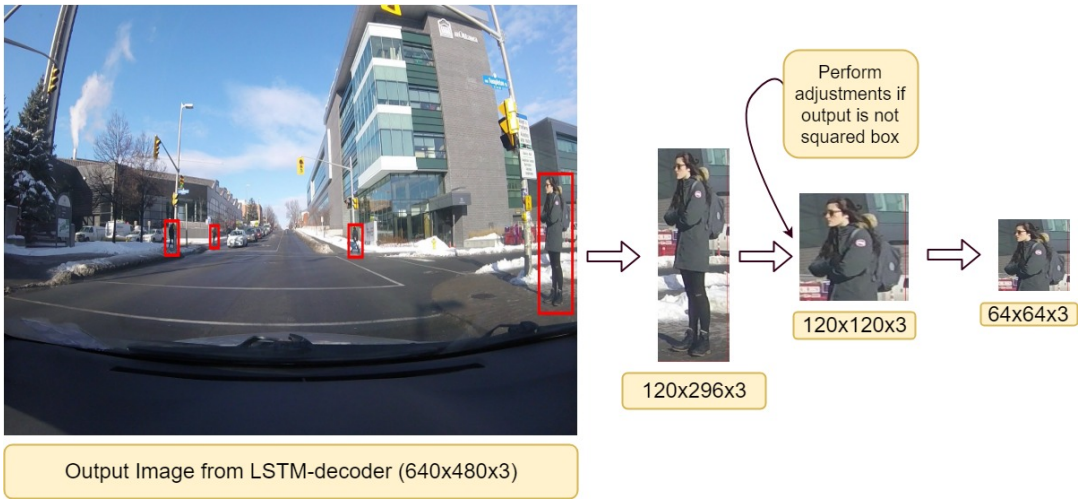


FIGURE 6.2: Data Pre-processing from Pedestrian detector

Pre-processing from Head Detector: The concept of pre-processing for head detector is slightly different from pre-processing from Pedestrian detector which is explicated in figure 6.3.

- Let's say, BB_{dt} is the set of all detection from LSTM-decoder where, $BB_{dt} \in \{x, y, w, h\}$ that are coordinates of detected bounding box for heads.
- If the width(w) and height(h) is greater than 20 pixel value:
  - $BB_{cropped} \in \{x^1 = x - w, y^1 = y - (0.3) * h, w^1 = 3w, h^1 = (2.3) * h\}$
- Afterwards, if new width(w^1) is equal to height(h^1) of cropped bounding box:
  - Rescale the Cropped Image to (64x64x3) image size.
- Else, adjust the height(h^1) and width (w^1) of cropped box to become a squared box. Then, rescale to (64x64x3) image size

[1]The pre-specified threshold in our experimentation is kept to be 50-pixel value because we designed our object detection model on the Caltech-Pedestrian dataset. If we lower the threshold value then, we found that the accuracy of the shoulder keypoints model got reduced. The model becomes unable to predict shoulder for small-scale bounding boxes.

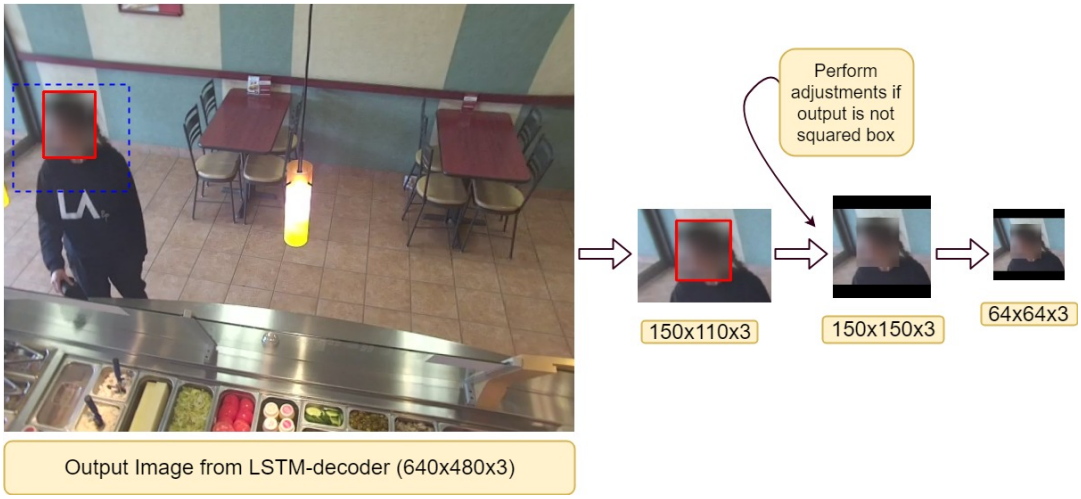


FIGURE 6.3: Data Pre-processing from Head Detector

CNN Cascade Network

Now, we got an image of size 64x64x3 and we want to detect shoulder keypoints in this image. We developed a small cascade network for detecting shoulder keypoints which is shown in figure 6.4[2]. The cascade network outputs class score with 4 regression coordinates which refer to left and right shoulder positions. Here, the class score is determined from Softmax classification function[12]. Introducing Softmax classification function in this cascade network improves accuracy of the model. It is helpful in determining whether the input image which had been extracted from object detection model contains head with shoulder keypoints or not. Thus, it reduces the number of false positives in object detection model and improves accuracy.

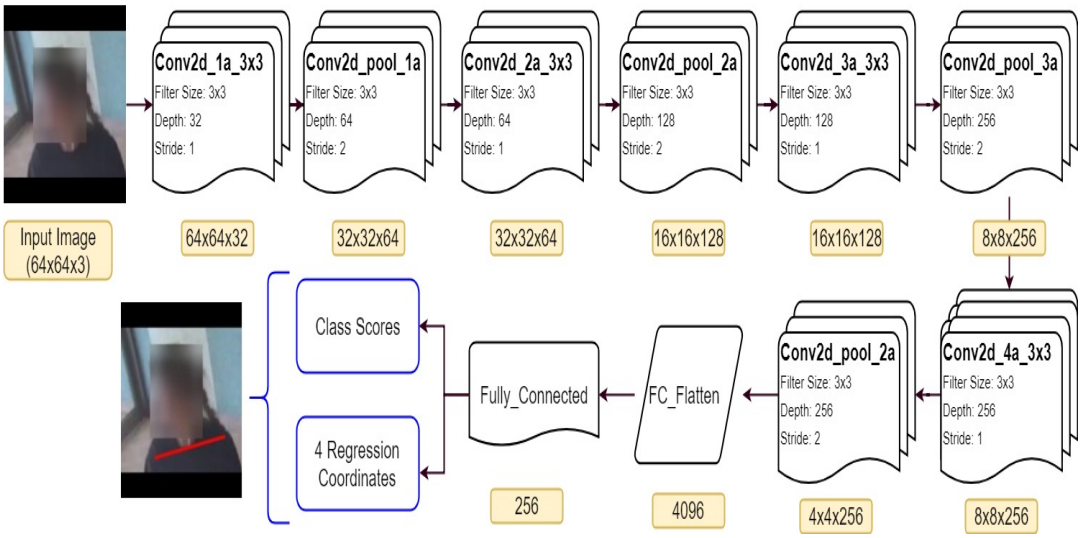


FIGURE 6.4: CNN Cascade Network for shoulder keypoints

[2]Here, the convolutional filters used in this cascade network are asymmetric convolution which we had discussed in Chapter-2. We found that asymmetric convolution maintained the same accuracy as standard convolutional filters. However, the benefit comes in the reduction in the total number of parameters for training.

Back-tracing the shoulder coordinates

The final step is to back-trace the shoulder coordinates which had been detected by CNN cascade network to the original image. For this, we know the the original coordinates of the detected object. We also stored the information for cropped image before re-scaling. Now, back-tracing can be possible by using the above mentioned information and resizing the shoulder coordinates to original image.

6.1.2 Shoulder Keypoints detection from LSTM-decoder

Another approach which we had designed for shoulder keypoints detection is using the same object detection model. The motive of this approach is to reduce external parameters which are used in cascade network. So, this approach uses the same feature maps which are fine-tuned for object detection in LSTM-decoder model. The overview of this architecture can be explained using figure 6.5 and 6.7.

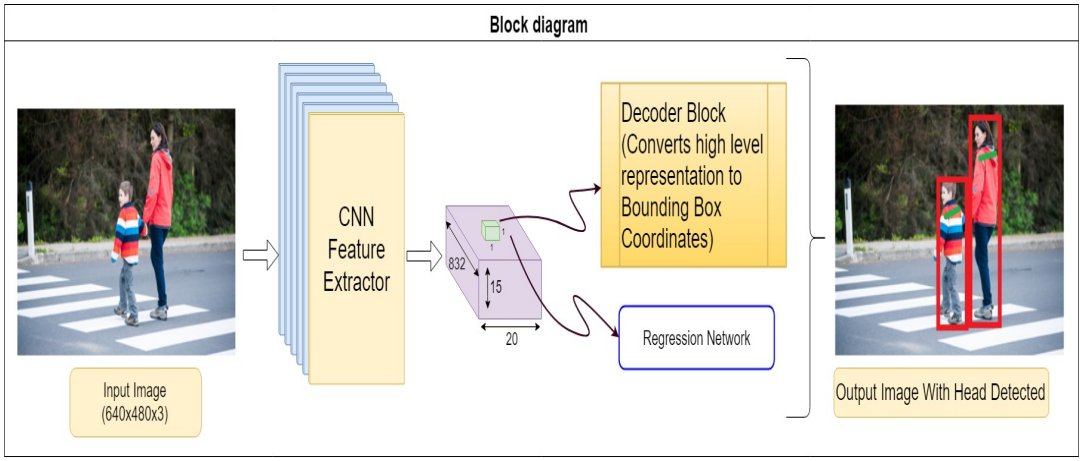
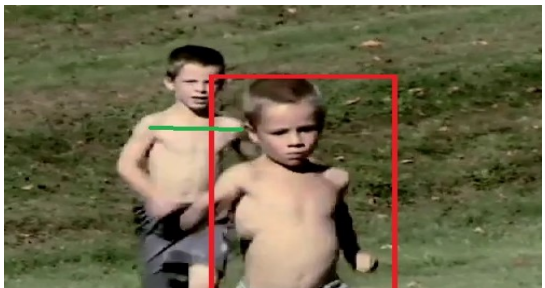


FIGURE 6.5: Model Architecture for shoulder keypoints detection using LSTM-decoder

Now, we had already discussed LSTM-decoder[45] model architecture in chapter 3. We had amended the same model architecture to detect shoulder keypoints. As shown in figure 6.5, the high-level representational feature maps are connected to a regression network which is responsible for detecting shoulder keypoints. In regression network, we simply used fully-connected layer which are connected to 300 neurons that represents the grid cells in the image. Then, these neurons are connected to 4 regression points for shoulder coordinates along with Softmax function[12] for classification.

In this approach, there might be the case in which the coordinates from object detection model are not aligned with shoulder coordinates for same object detected. An example of this particular case is shown in figure 6.6 in which it is clearly seen that the detection from LSTM-decoder are misaligned with shoulder keypoints detected. To resolve this issue, we implemented a constraint in our shoulder keypoints model architecture.



As shown in figure 6.5, decoder block tries to predict object within the grid cells of the feature maps(that is, output size for Softmax function is

(300,1,2)[3]). Similarly, the regression network also predicts shoulder coordinates within the same grid cells. We implemented a constraint that model architecture will output only those shoulder coordinates whose grid cells are aligned with object detected by LSTM-decoder as shown in figure 6.7. In figure 6.7, there can be other cases in which regression network had predicted shoulder keypoints other than object detected by the LSTM-decoder. However, those predictions got suppressed and only those shoulder coordinates would be shown in the output image whose grid cells are aligned with grid cells of the object detected by LSTM-decoder.

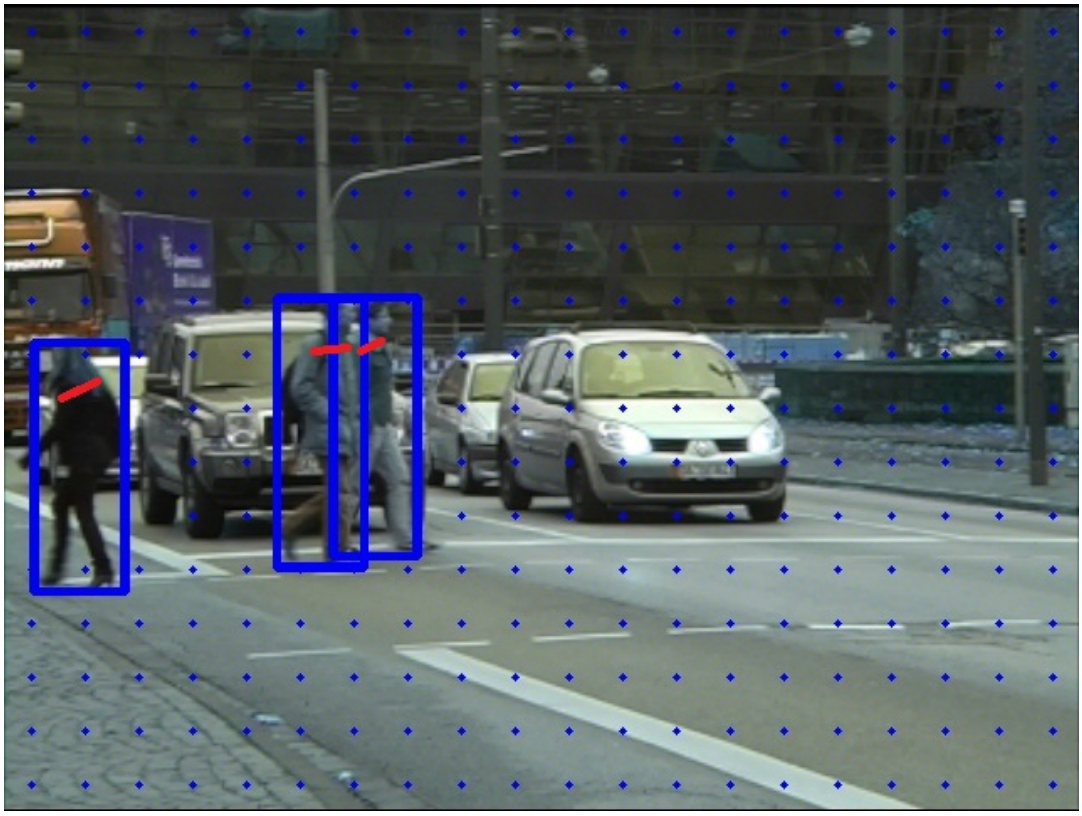


FIGURE 6.7: Alignment of region cells for object detection model with shoulder keypoints

With this approach, we didn't used any external feature maps for predicting shoulder coordinates as we did in the previous approach that used CNN cascaded model. The merits of this approach comes in reduction of extra feature maps which are replicating the computation for same image regions. However, there are also few demerits for this model architecture which we had discussed in evaluation results of

[3]The Notation (300,1,2) refers to (Number of grid cells, Batch-size, Classification Score)

this Chapter.

6.2 Dataset Used

For shoulder keypoints detection, we used three distinct datasets for training and evaluation of our 2 different model architecture. Two of them are publicly available(FLIC Dataset[40] and MPII Human Pose Dataset[1]) and one is our own contribution which we had already explained in Chapter 5 named as MrSub dataset. The two publicly available datasets are used for human Pose estimation. However, we had pre-processed the annotations so that it could be useful for detecting shoulder keypoints too. The abstract details of all three datasets are mentioned in table 6.1[4] and further details are as follows:

Dataset	Number of Images	Number of Annotations	Annotations Per Image
FLIC Dataset	20,928	20,928	1
MrSub Dataset	3,511	7,842	2.23
MPII Dataset	24,920	40,522	2.06

TABLE 6.1: Details of three distinct Shoulder keypoints dataset

6.2.1 FLIC Dataset

FLIC stands for **F**rames **L**abeled **I**n **C**inema is a popular dataset for upper body pose estimation. This dataset comes with conjugation with a research work[40]. They collected scenes from popular Hollywood movies and annotated upper body joints. They had collected 20,928 images. In each image, a particular person is annotated and therefore only 1 annotation per image is mentioned in table 6.1. This dataset includes many images whose shoulder keypoints are occluded and it proves to be helpful for training purpose.

As only 1 person is annotated in an image, we used this information for extracting the images so that it can be used for training the shoulder keypoints. We used this dataset only for training purpose in shoulder cascade model architecture. The second LSTM-shoulder model architecture cannot be trained on this dataset because the model gets trained on full image. The model architecture may not reach on its convergence if we train LSTM-shoulder on FLIC dataset.

6.2.2 MrSub Dataset

The details of MrSub dataset had already been explained in the previous chapter. In addition to the head annotations, we also annotated shoulder keypoints. We also used this dataset for training purpose because our main goal is to evaluate our model architecture with other state-of-the-art model architectures. MrSub is our own contribution for head detection dataset and it is yet to made public. So, we can't evaluate our model architecture with other state-of-the-art performances on MrSub dataset.

[4]In table 6.1, the number of annotation are mentioned in shoulder-pairs. It means that the left and right shoulder is considered as 1 annotation.

6.2.3 MPII Human Pose Dataset

MPII Human Pose dataset[1] is a state-of-the-art benchmark dataset for evaluation of articulated human pose estimation. This dataset included around 25K images containing approximately 41K annotations of shoulder keypoints. The images were systematically collected using an established taxonomy of every day human activities. Each image was extracted from a YouTube video and provided with preceding and following non-annotated frames.

The author had provided approximately 18,690 images for training and remaining images act as a test-set which is used for evaluation of different model architectures. Moreover, the author had also provided annotations for head and it is helpful for training our two shoulder keypoints model architectures on this dataset.

6.3 Implementation Details

In this section, we had discussed the miniature details of our shoulder keypoint model architecture; one is named as "*Shoulder Cascade*" which uses external cascade CNN feature extractor for detecting shoulder keypoints. Second is named as "*LSTM-Shoulder*" which uses the feature maps for LSTM-decoder to detect shoulder keypoints. For implementing both the model architectures, we used TensorFlow framework and trained on NVIDIA Titan-X_p GPU that we had used in previous chapter.

Other than this, there is nothing similar in both the model architectures. So, the implementation details are different from each other[5]. It had been discussed in the following sections:

6.3.1 Shoulder Cascade Model Architecture

The hyper-parameters which we had assigned for implementing shoulder cascade model architecture is shown in table 6.2. For LSTM-decoder[45], we used Inception-V3[48] as CNN feature extractor for detecting heads. We had implemented other CNN feature extractors for Shoulder Cascade model architecture. However, we found that precision of the shoulder keypoints mostly depends upon the optimization of cascade network. CNN feature extractors in this case didn't affect the accuracy of model architecture by huge margins.

Hyper-Parameters	**Values**
Dataset Used for Training	FLIC; MrSub, MPII Human Pose
Number of Images for Training	40,070
Number of shoulder annotations	57,591
Number of Images for Validation	3,059
Data Augmentation Rate	10%
Data Augmentation type used	Scaling and Random Rotation
Dataset used for Evalutation	MPII Human Pose
Number of Images for Testing	6,230
CNN feature extractor used for LSTM-decoder	Inception-V3

[5]For object detection, we had implemented head detector because we performed our evaluation on MPII Human Pose dataset. So, the author had provided head annotations which could be useful for training our model architectures.

Initial pre-trained weights for CNN cascade network	None
Weight Initializing Technique	Variance Scaling Initializer
Training Algorithm	RMS-Prop
Initial learning Rate	0.001
Learning rate decreased after	10,000
Input Image Size for Cascade Network	64x64x3
Convolutional Filter	Asymmetric Convolution

TABLE 6.2: Hyper-parameters for Shoulder Cascade Model Architecture

6.3.2 LSTM-Shoulder Model Architecture

FIGURE 6.8: Shoulder coordinates predicted from multiple Grid-cells

The hyper-parameters which we had assigned for implementing LSTM-Shoulder model architecture is shown in table 6.3. We had implemented 3 CNN feature extractors for LSTM-Shoulder model architecture as shown in table 6.3. The change in the performance of the model architecture based on different CNN feature extractors is discussed in the next section.

In this model architecture, the detection of shoulder keypoints depends upon the detection on grid cells. So, there could be possibility that multiple grid cells point out to a single shoulder points as shown in figure 6.8. In order to discard the multiple detection for same shoulder points, we implemented a simple post-processing technique.

We had focused on grid cells which are responsible for detecting shoulder keypoints and overlapping with grid cells for head detection. Out of the these grid cells, the grid cell with highest P_d for shoulder keypoints is selected. Other predicted grid cells are discarded. Using this post-processing technique, only single pair of shoulder keypoints is predicted for every head prediction.

Hyper-Parameters	**Values**
Dataset Used for Training	MrSub; MPII Human Pose
Number of Images for Training	20,145
Number of shoulder annotations	36,663
Number of Images for Validation	2,056
Data Augmentation Rate	10%
Data Augmentation type used	Scaling and Random Rotation

Dataset used for Evalutation	MPII Human Pose
Number of Images for Testing	6,230
CNN feature extractor used for LSTM-decoder	Inception-V3 ResNet-101 Inception-Resnet-V2
Weight Initializing Technique	Variance Scaling Initializer
Training Algorithm	RMS-Prop
Initial learning Rate	0.001
Learning rate decreased after	33,000
Input Image Size	640x480x3
Convolutional Filter	Standard 2-D Convolution

TABLE 6.3: Hyper-parameters for LSTM-Shoulder Model Architecture

6.4 Evaluation Results

We evaluated our two model architecture on MPII Human Pose test dataset. There are multiple evaluation technique[6] provided by the author but all these evaluation technique needs full body keypoints. Our model architectures predicts only shoulder keypoints on full image. So, we had only reported Multi-Person Evaluation and compared our results with state-of-the-art model architecture.

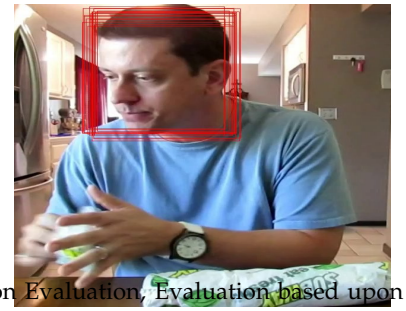
6.4.1 Evaluation Technique

We used the same evaluation protocol provided by author of MPII Human Pose Dataset. They defined "PCKh" as their evaluation measure which stand for "*Probability of Correct Keypoints*". However, we must discuss the "PCK" evaluation measure first. PCK is defined as a candidate keypoint to be correct if it falls within $\alpha * max(h, w)$ pixels of the ground-truth keypoint, where h and w are the height and width of the bounding box respectively. α controls the relative threshold for considering correctness.

In PCKh, the 'h' means that the matching threshold is considered with respect to head segment length. The threshold which they had taken in their evaluation is 50% of the head segment length. They choose head size because this makes metric articulation independent with other body joints.

6.4.2 Evaluation Results on MPII test dataset

We evaluated our shoulder keypoints model architectures on MPII Human Pose test dataset. The Mean Average Precision (mAP) is reported in table 6.4. From table 6.4, it had been concluded that our best model architecture(that is, Shoulder Cascade) lags behind from state-of-the-art model



[6]The list of Evaluation Protcols includes Single Person Evaluation, Evaluation based upon Complexity Measures, Evaluation based upon pose and Evaluation Based on viewpoint as well as activity

FIGURE 6.9: Failure Case-1: LSTM-Shoulder unable to predict shoulder for Large Scale Images

architecture's[34] performance by 5.2 mAP.

There is significant decline in the accuracy of LSTM-Shoulder based model architectures. We tried to explore the reason for this downtrend and it is found in LSTM-decoder model architecture itself. LSTM-decoder is made to perform prediction at a particular scale (here, each grid cell's size is 32x32 pixel value). So, when LSTM-decoder predict head on large-scale images then, LSTM-decoder successfully predict head but the grid cells doesn't get aligned with grid cells for shoulder keypoints. An example of this case is shown in figure 6.9.

Model	**Mean Average Precision**
Newell&Deng, arXiv'16[34]	89.3
Shoulder Cascade	84.1
LSTM-Shoulder-Inception-V3	74.6
LSTM-Shoulder-Resnet-101	74.8
LSTM-Shoulder-Inception-Resnet-V2	75.6

TABLE 6.4: Mean Average Precision Reported on MPII Human Pose test Dataset

6.4.3 Failure Cases for our model Architectures

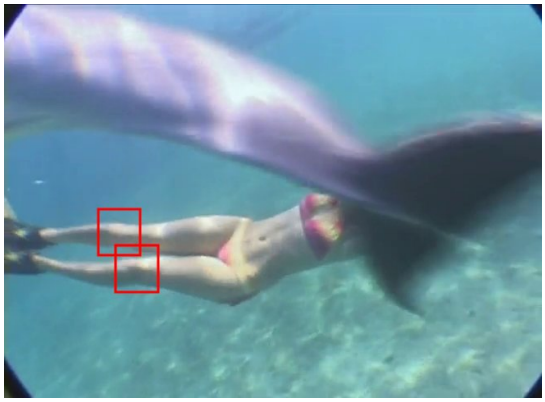


FIGURE 6.10: Failure Case-2: Both Model Architecture unable to predict shoulder for missing heads

After we analysed closely into the results, we found some significant drawbacks of our shoulder keypoint model architectures in which both our models failed to predict shoulder keypoints. No doubt, the two different model architectures had entirely different pipeline for predicting shoulder keypoints but there are some failure cases which are common for both model architectures. One of the case which we had already discussed in figure 6.9. The failure cases is not limited to an example shown in figure 6.9 but this case is less likely to be seen in Shoulder Cascade model architecture.

However, the frequent failure case is shown in figure 6.10 in which both of our model architectures unable to predict shoulder coordinates. As our model architecture is dependent upon head detections; if LSTM-decoder unable to predict head in an input image then, bit obviously, shoulder keypoints doesn't get detected.

As discussed in the implementation details, we had performed data augmentation for both of our model architectures by a factor from 0.9 to 1.1. However, there are

still very few images in which our model architectures are unable to predict shoulder keypoints. One of the failure case is shown in figure 6.11 in which our model architecture doesn't predict shoulder coordinates because of the dis-orientation of the human pose.

FIGURE 6.11: Failure Case-3: Both Model Architecture unable to predict shoulder for disoriented human pose

In this Chapter, we had discussed about the two model architecture which can be used for predicting shoulder keypoints and it is based upon LSTM-decoder[45] model architecture. We had also evaluated their performance on MPII Human Pose Estimation dataset and discussed few of the failure cases for both model architectures. In the next chapter, we presented a summary of the experimentation which we had performed in this thesis along with our future approach to improve the accuracy of our designed model architectures.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we had performed experiments for object detection task using different CNN feature extractors. The base model architecture which we had used is LSTM-decoder[45]. If we combine the head as well as pedestrian detection's model architectures then, we had implemented 9 CNN feature extractors using LSTM-decoder model architecture. The list of these CNN feature extractors are as follows: VGG-Net[43], Squeeze-Net[23], Inception-V1[47], Inception-V2[48], Inception-V3[48], Resnet-50[15], Resnet-101[15], Mobile-Net[21] and Inception-Resnet-V2[46].

Our main motivation is to explore the pros and cons of different CNN feature extractors. For this, we implemented 8 model architecture for pedestrian detection and results are shown in Chapter 4. We concluded that accuracy on object detection can be increased by increasing the depth of CNN feature extractors. However, increasing the depth of CNN feature extractors will increase the number of parameters which would eventually decrease the speed of the model architecture. The model architecture with higher number of parameters will definitely increase the accuracy but it would not be useful for real-time applications. Other than this, we found that optimized convolutional filters reduce the number of parameters and these kind of feature extractors (like MobileNet[21] and Squeezenet[23]) can be easily implemented for ADAS applications. There is a small reduction in the accuracy level but a sweet spot can be successfully achieved by these CNN feature extractors. Our findings can be generalized to other computer vision applications in which CNN feature extractors play a major role in attaining a sweet spot between speed and accuracy.

Our second motivation is to explore the LSTM-decoder model architecture[45] for different scenarios. This motive is achieved by our experimentation performed in Chapter 5. In this, LSTM-decoder is evaluated for occlusion specific problems, generalization ability and domain adaptation ability. LSTM-decoder achieved satisfactory results for occlusion specific problem and domain adaptation ability. However, LSTM-decoder lags behind in generalization ability as models trained for particular scale unable to perform detection on a dataset of different scale annotations.

We had also extended our work of object detection to perform shoulder keypoint detection with two methodologies. One is to use external cascade network for detecting shoulder keypoints. Second is to use the fine tune the feature maps of object detection model to detect shoulder keypoints. External cascade network achieved a comparable state-of-the-art results on MPII human pose estimation dataset[1]. For the second methodology, there is a gap of approximately 14% mAP(mean Average

Precision) with state-of-the-art model architecture. The reason for this failure is in LSTM-decoder model architecture as it predicts object at a particular scale.

7.2 Future Work

We had analysed significant parameters for CNN feature extractors. Still, there is a lot of scope to continue our research work and analyse these CNN feature extractors for other computer vision challenges like segmentation and human pose estimation. Moreover, the best CNN feature extractor cannot be implemented for real-time applications because of the large number of parameters to be trained. This number can be reduced by designing more optimized convolutional filters which is also an another platform of research for our future work.

The limitation of LSTM-decoder[45] to predict object at particular scale can be rectified by other object detection model architecture like Faster-RCNN[38] and SSD[33]. Although we had explored SSD model architecture for different scenarios and presented in 15^{th} Conference on Computer and Robot Vision but there is still a room for analysing these model architectures on other scenarios.

For shoulder keypoint detection, there is a alot of room for improvement of our shoulder keypoint detector which is based on LSTM-decoder feature maps. This model architecture can be improved if it tries to predict on multiple scales. This becomes an important pathway to our future work. Apart from this, these two shoulder keypoint model architectures can be designed to perform full human body pose estimation which is also considered as part of our future work for this thesis.

Appendix A

Link of Scripts for all our experimentations

In this Chapter, we had provided the link for the codes that are discussed in this thesis.

1. **LSTM-decoder model architecture:** We had implemented LSTM-decoder model architecture for pedestrian as well as head detection using different CNN feature extractors. Here is the link for the source-code: https://github.com/Prince-kapoor1991/LSTM-decoder

2. **Shoulder Cascade Model architecture:** After detection from LSTM-decoder, shoulder keypoints can be predicted using external cascade model. Here is the link for the source-code for external cascade model architecture: https://github.com/Prince-kapoor1991/LSTM-shoulder-cascade

3. **LSTM-shoulder model architecture:** Another strategy for detecting shoulder keypoints is to fine-tune the weights of LSTM-decoder object detection model for predicting shoulder keypoints. Here is the link for the source-code: https://github.com/Prince-kapoor1991/LSTM-shoulder

Bibliography

[1] M. Andriluka et al. "2D Human Pose Estimation: New Benchmark and State of the Art Analysis". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3686–3693. DOI: 10.1109/CVPR.2014.471.

[2] *Caltech Pedestrian Detection Benchmark*. 2012. URL: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/.

[3] François Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

[4] Jifeng Dai et al. "R-FCN: Object Detection via Region-based Fully Convolutional Networks". In: *CoRR* abs/1605.06409 (2016). arXiv: 1605.06409. URL: http://arxiv.org/abs/1605.06409.

[5] Navneet Dalal and Bill Triggs. "Histograms of Oriented Gradients for Human Detection". In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893. ISBN: 0-7695-2372-2. DOI: 10.1109/CVPR.2005.177. URL: http://dx.doi.org/10.1109/CVPR.2005.177.

[6] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: (2009).

[7] Piotr Dollar et al. "Pedestrian Detection: An Evaluation of the State of the Art". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 34.4 (Apr. 2012), pp. 743–761. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2011.155. URL: http://dx.doi.org/10.1109/TPAMI.2011.155.

[8] Xianzhi Du et al. "Fused DNN: A deep neural network fusion approach to fast and robust pedestrian detection". In: *CoRR* abs/1610.03466 (2016). arXiv: 1610.03466. URL: http://arxiv.org/abs/1610.03466.

[9] A. Ess, B. Leibe, and L. Van Gool. "Depth and Appearance for Mobile Scene Analysis". In: *2007 IEEE 11th International Conference on Computer Vision*. 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4409092.

[10] Mark Everingham et al. "The Pascal Visual Object Classes Challenge: A Retrospective". In: *Int. J. Comput. Vision* 111.1 (Jan. 2015), pp. 98–136. ISSN: 0920-5691. DOI: 10.1007/s11263-014-0733-5. URL: http://dx.doi.org/10.1007/s11263-014-0733-5.

[11] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Comput.* 12.10 (Oct. 2000), pp. 2451–2471. ISSN: 0899-7667. DOI: 10.1162/089976600300015015. URL: http://dx.doi.org/10.1162/089976600300015015.

[12] Kevin Gimpel and Noah A. Smith. "Softmax-margin CRFs: Training Log-linear Models with Cost Functions". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. HLT '10. Los Angeles, California: Association for Computational Linguistics, 2010, pp. 733–736. ISBN: 1-932432-65-5. URL: http://dl.acm.org/citation.cfm?id=1857999.1858111.

[13] Ross B. Girshick. "Fast R-CNN". In: *CoRR* abs/1504.08083 (2015). arXiv: 1504.08083. URL: http://arxiv.org/abs/1504.08083.

[14] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[15] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[16] Kaiming He et al. "Mask R-CNN". In: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[17] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6.2 (Apr. 1998), pp. 107–116. ISSN: 0218-4885. DOI: 10.1142/S0218488598000094. URL: http://dx.doi.org/10.1142/S0218488598000094.

[18] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[19] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. "Learning non-maximum suppression". In: *CoRR* abs/1705.02950 (2017). arXiv: 1705.02950. URL: http://arxiv.org/abs/1705.02950.

[20] Jan Hendrik Hosang et al. "Taking a Deeper Look at Pedestrians". In: *CoRR* abs/1501.05790 (2015). arXiv: 1501.05790. URL: http://arxiv.org/abs/1501.05790.

[21] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[22] Jonathan Huang et al. "Speed/accuracy trade-offs for modern convolutional object detectors". In: *CoRR* abs/1611.10012 (2016). arXiv: 1611.10012. URL: http://arxiv.org/abs/1611.10012.

[23] Forrest N. Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size". In: *CoRR* abs/1602.07360 (2016). arXiv: 1602.07360. URL: http://arxiv.org/abs/1602.07360.

[24] Eldar Insafutdinov et al. "Articulated Multi-person Tracking in the Wild". In: *CoRR* abs/1612.01465 (2016). arXiv: 1612.01465. URL: http://arxiv.org/abs/1612.01465.

[25] Eldar Insafutdinov et al. "DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model". In: *CoRR* abs/1605.03170 (2016). arXiv: 1605.03170. URL: http://arxiv.org/abs/1605.03170.

[26] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[27] Max Jaderberg et al. "Spatial Transformer Networks". In: *CoRR* abs/1506.02025 (2015). arXiv: 1506.02025. URL: http://arxiv.org/abs/1506.02025.

[28] Vidit Jain and Erik Learned-Miller. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Tech. rep. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[29] G. Ayorkor Korsah, Anthony (Tony) Stentz, and M Bernardine Dias. *The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs*. Tech. rep. CMU-RI-TR-07-27. Pittsburgh, PA: Carnegie Mellon University, 2007.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. URL: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[31] Yann LeCun and Yoshua Bengio. "The Handbook of Brain Theory and Neural Networks". In: ed. by Michael A. Arbib. Cambridge, MA, USA: MIT Press, 1998. Chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. ISBN: 0-262-51102-9. URL: http://dl.acm.org/citation.cfm?id=303568.303704.

[32] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[33] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[34] Alejandro Newell and Jia Deng. "Associative Embedding: End-to-End Learning for Joint Detection and Grouping". In: *CoRR* abs/1611.05424 (2016). arXiv: 1611.05424. URL: http://arxiv.org/abs/1611.05424.

[35] W. Ouyang and X. Wang. "Joint Deep Learning for Pedestrian Detection". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 2056–2063. DOI: 10.1109/ICCV.2013.257.

[36] Xin Pan et al. *Tensorflow Profiler and Advisor*. URL: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/profiler/README.md.

[37] Rajeev Ranjan, Vishal M. Patel, and Rama Chellappa. "HyperFace: A Deep Multi-task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition". In: *CoRR* abs/1603.01249 (2016). arXiv: 1603.01249. URL: http://arxiv.org/abs/1603.01249.

[38] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[39] K. E. A. van de Sande et al. "Segmentation as selective search for object recognition". In: *2011 International Conference on Computer Vision*. 2011, pp. 1879–1886. DOI: 10.1109/ICCV.2011.6126456.

[40] Benjamin Sapp and Ben Taskar. "MODEC: Multimodal Decomposable Models for Human Pose Estimation." In: *CVPR*. IEEE Computer Society, 2013, pp. 3674–3681. ISBN: 978-0-7695-4989-7. URL: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2013.html#SappT13.

[41] Pierre Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* abs/1312.6229 (2013). arXiv: 1312.6229. URL: http://arxiv.org/abs/1312.6229.

[42] Sagar sharma. *Activation Functions: Neural Networks Sigmoid, tanh, Softmax, ReLU, Leaky ReLU EXPLAINED !!!* URL: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[43] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[44] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

[45] Russell Stewart and Mykhaylo Andriluka. "End-to-end people detection in crowded scenes". In: *CoRR* abs/1506.04878 (2015). arXiv: 1506.04878. URL: http://arxiv.org/abs/1506.04878.

[46] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261 (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261.

[47] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[48] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[49] Shaohua Wan et al. "Bootstrapping Face Detection with Hard Negative Examples". In: *CoRR* abs/1608.02236 (2016). arXiv: 1608.02236. URL: http://arxiv.org/abs/1608.02236.

[50] Kaipeng Zhang et al. "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks". In: *CoRR* abs/1604.02878 (2016). arXiv: 1604.02878. URL: http://arxiv.org/abs/1604.02878.

[51] X. Zhu and D. Ramanan. "Face detection, pose estimation, and landmark localization in the wild". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 2879–2886. DOI: 10.1109/CVPR.2012.6248014.

•