TRIBHUWAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

THESIS NO.: 072/MSCS/654

OBJECT DETECTION IN VIDEO USING REGION BASED CONVOLUTION NEURAL NETWORK

BY

DEEPESH LEKHAK

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINNEING

DEPARTMENT OF ELCTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

NOVEMBER, 2017

A

THESIS

ON

OBJECT DETECTION IN VIDEOS USING REGION BASED CONVOLUTION NEURAL NETWORK

BY

DEEPESH LEKHAK

072/MSCS/654

THESIS SUPERVIOR

DR. BASANTA JOSHI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SYSTEM AND KNOWLEDGE ENGINNEING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

INSTITUTE OF ENGINEERING, PULCHOWK CAMPUS

LALITPUR, NEPAL

NOVEMBER, 2017

Copyright

TRIBHUWAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

Recommendation

The undersigned certify that they have read and recommended to the Department of Electronics and Computer Engineering for acceptance, a thesis entitled **"Object Detection in Videos using Region Based CNN**", submitted by **Deepesh Lekhak** in partial fulfillment of the requirement for the award of the degree of "**Master of Science in Computer System and Knowledge Engineering**".

……

Supervisor: Dr. Basanta Joshi

Assistant Professor

Department of Electronics and Computer Engineering

……

External Examiner: Mr. Subhash Dhakal

Director, Department of Information and Technology

Government of Nepal

……

Committee Chairperson: Prof. Dr. Subarn Shakya

Professor

Department of Electronics and Computer Engineering

Date:

Departmental Acceptance

The thesis entitled **"Object Detection in Videos using Region Based CNN",** submitted by **Deepesh Lekhak** in partial fulfillment of the requirement for the award of the degree of "**Master of Science in Computer System and Knowledge Engineering**" has been accepted as a bonafide record of work independently carried out by him in the department.

Dr. Dibakar Raj Panta

Head of the Department

Department of Electronics and Computer Engineering,

Pulchowk Campus,

Institute of Engineering,

Tribhuvan University,

Nepal.

Acknowledgement

First and foremost, I would like to thank Dr. Basanta Joshi for providing guidance and useful information regarding this thesis. I am thankful to our Head of Department Dr. Dibakar Raj Pant for his support and suggestions. I would express a word of thankfulness to our Coordinator Dr. Aman Shakya for his motivation and support.

I would also like to thank Professor Dr. Shashidhar Ram Joshi, Professor Dr. Subarna Shakya, Dr. Sanjeeb Prasad Panday and all faculty of Department of Electronics and Computer Engineering for their support and suggestions. I would like to express my sincerest thanks to my classmates for the suggestions and encouragements.

I am thankful to my family and friends for their continuous support and encouragement for this research work.

Abstract

Object detection is the task of recognizing and localizing objects in an image. Object detection in images have many applications including object counting, Visual Search Engine, security, surveillance etc.

In this research work region based approach for object detection Faster R-CNN was refined to detect objects in videos more efficiently. The contextual information was incorporated by adding recurrent neural layer (LSTM) between consecutive frames in videos. The system was developed and evaluated using ImageNet VID Dataset for videos. The system provided mean average precision of 0.64 for detection of objects in that dataset which was higher than mean average precision of 0.56 for Faster R-CNN without LSTM. The system provided more than 15 % enhancement in F-measure of detection of objects in videos than Faster R-CNN method without LSTM layer.

Also performance Faster R-CNN method based on VGGNet architecture was compared with Residual Network (ResNet). The architecture of ResNET was modified to incorporate region proposal network to detect and classify objects and their boundary in an image. The results were compared with Faster R-CNN based on VGG-16 on PASCAL VOC dataset. It was found that Faster R-CNN based on ResNET provides mean average precision of 0.78 which is better performance on PASCAL VOC dataset than VGG-16 Net architecture with mean average precision of 0.699. It was also found that ResNet based system was faster than VGGNet based system for object detection using Faster R-CNN method.

Keywords: **Object Detection, Convolutional Neural Network, Region Proposals, Residual Network**

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
CPU	Central Processing Unit
FC	Fully Connected Layer
FN	False Negative
FP	False Positive
GPU	Graphical Processing Unit
HOG	Histogram of Gradient
IoU	Intersection-over-Union
LSTM	Long Short Term Memory
MAP	Mean Average Precision
R-CNN	Regions with CNN
RNN	Recurrent Neural Network
RoI	Region of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

CHAPTER ONE: INTRODUCTION

1.1 Background and Motivation

Humans glance at an image and instantly know what objects were in the image, where they were, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. The task of classification, when it relates to images, generally refers to assigning a label to the whole image, e.g. 'cat'. While localization refers to finding where the object is in said image, usually denoted by the output of some form of bounding box around the object. Object detection is a computer vision and image processing technology that deals with detecting instances of semantic objects of a certain class in digital images. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems. Object detection algorithms typically use extracted features and learning algorithms to recognize instances of an object category. It is commonly used in applications such as image retrieval, security, surveillance, and automated vehicle parking systems. An image classification problem is object detection problem, however, image classification is predicting the label of an image among the predefined labels. It assumes that there is single object of interest in the image and it covers a significant portion of image. Object Detection is about not only finding the class of object but also localizing the extent of an object in the image.

Object detection in images/videos have many applications including object counting, localization and classification of objects in cluttered images, Visual Search Engine, aerial image analysis, security, surveillance, automatic vehicle driving etc. Object detection is an essential task for object tracking and image captioning.

The problem of recognizing objects in images has been studied extensively over the decades but still remains a challenging task. In recent years, the study of deep learning has been a growing interest due to its superior performance in several recognition tasks,

for instance, activity recognition, object detection, and scene classification. Deep Convolutional networks based methods have become the state of the art in object detection in image. Object detection can be carried out with classification on different sub-windows or patches or regions extracted from the image. The patch with high probability had not only the class of that region but also implicitly gives its location too in the image. Most of the approaches vary on the type of methodology used for choosing the windows. Convolutional neural networks were used as feature extractor for the classification. Region based convolutional neural networks were popular method used to detect objects in images.

Although there have been many different types of methods for detection of images/videos, they can be broadly classified into two categories: classical approach and deep learning approach. Classical approach includes classifier based on features such as Haar-like Features, HOG features and classifiers such as SVM, Bayesian Classifier etc.

Deep Learning based approach deep learning models have crushed other classical models on the task of image classification, deep learning models are now state of the art in object detection as well. Deep learning based approaches are evolved using Convolutional Neural Networks which mimics the visual cortex of the animals. Methods such as Overfeat, Region Based CNN, Single Shot Multiplex Detector etc are some examples of deep learning approach.

1.2 Problem Statement

Object detection is the task of recognizing and localizing objects in an image. Traditionally, sliding windows or region proposals have been used to generate object hypotheses, which were then classified. Faster Region with Convolutional Neural Network (Faster R-CNN) [1] detects objects in images using region proposal network. By combining both region proposals and the classifier into one large network, it is able to automatically learn good feature representations for the task. Faster R-CNN is designed for object detection in a single independent frame. For object detection in videos, this method lacks temporal information due to processing of frames

independently. If temporal information from bounding boxes over time is taken into consideration, the objects in the given frame could be more accurately localized. In this thesis, it is to be analyzed whether incorporating temporal information of successive frames in region based convolutional neural network enhance the performance of the system and minimize the error locating objects in videos.

1.3 Research Objectives

The objectives of this research are:

i. To determine an efficient method to detect objects in images using Region Based Convolutional Neural Network.
ii. To refine and analyze the Convolutional Neural Network for object detection in videos with temporal information.

1.4 Scope of the Research Work

In this research, object detection in videos had been leveraged by using temporal information in region based convolutional neural network. Temporal information in consecutive frames had been incorporated in faster R-CNN [1] method to design an efficient technique for detecting objects in videos. By taking advantage of the temporal information of bounding boxes over time, system could accurately localize the object in any given frame. ImageNet VID [2] had been used for training, validating, and testing the system. The dataset has video snippets annotated with thirty object classes. The system had produced annotations consisting frame number, class labels, confidence scores, and bounding boxes for each test video clip. This thesis had incorporated refinement in Faster R-CNN algorithm with use of Recurrent Layer (LSTM) to leverage temporal information in consecutive video frames.

1.5 Organization of the Report

This report is divided into five chapters. Chapter one is introduction which consists of background this research topic, statement of the problem, research objectives, and scope of the research work. Chapter Two is Literature Review with basic theory and

related work to this research work. Chapter three explains about research methodology with research work flow and experimental setup and tools. Chapter four discusses the CNN architecture design, implementation of the design and results & analysis of the implementation. Chapter five is epilogue with conclusion of the research work and limitation & future enhancements. Chapter five is followed by references and appendix.

CHAPTER TWO: LITERATURE REVIEW

2.1 Related Theory

2.1.1 Convolution Neural Network

In machine learning, a convolutional neural network (CNN) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field. The receptive fields of different neurons partially overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation [3]. Convolutional networks were inspired by biological processes and were variations of multilayer perceptron designed to use minimal amounts of preprocessing [4].

LeNet is one of the very first convolutional neural networks which helped propel the field of Deep Learning [4]. This pioneering work by Yann LeCun was named LeNet5 was used mainly for character recognition tasks such as reading zip codes, digits, etc. Other famous CNN architecture were AlexNet [5], ZF Net [6] and VGG Net [7].

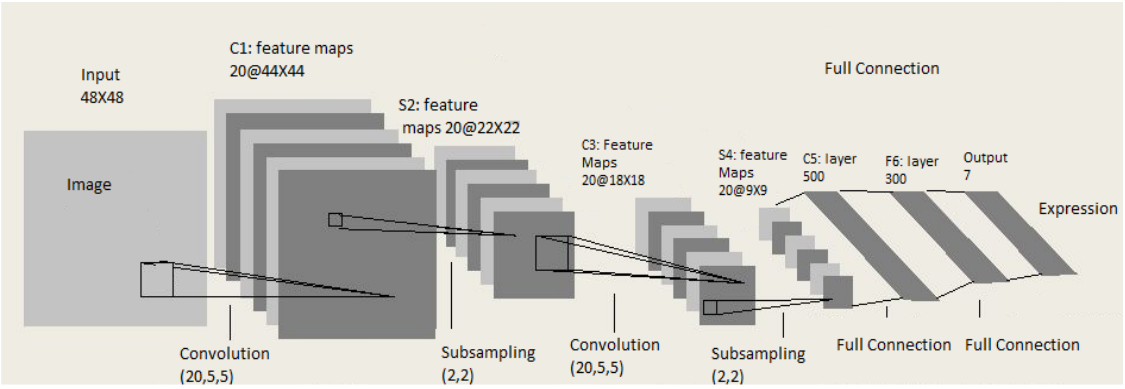


Figure 2.1: CNN LeNet architecture [4]

There were four main operations in the Convolution Neural Network were:

i. Convolution:

The primary purpose of Convolution in case of a CNN is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning

image features using small squares of input data. The convolution layer's parameters consist of a set of learnable filters. As the filter convolve over the width and height of the input volume it produces a 2dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network had learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

The 2-dimensional convolution between image A and Filter B can be given as:

$$(i, j) = \sum^{Ma}_{m=^{-}0^{1}} \sum^{Na}_{n=^{-}0^{1}} (m, n) * (i - m, j - n)$$

(2.1)

where size of A is (M_a x N_a), size of B is (M_b x N_b), $0 \leq i < Ma + Mb - 1 \wedge 0 \leq j < Na + Nb - 1$

A filter convolves with the input image to produce a feature map. The convolution of another filter over the same image gives a different feature map. Convolution operation captures the local dependencies in the original image. A CNN learns the values of these filters on its own during the training process (although parameters such as number of filters, filter size, architecture of the network etc. still needed to specify before the training process). The more number of filters, the more image features get extracted and the better network becomes at recognizing patterns in unseen images.

ii. Rectified Linear Unit:

An additional operation called ReLU has been used after every Convolution operation. A Rectified Linear Unit (ReLU) is a cell of a neural network which uses the following activation function to calculate its output given x:

$$R(x) = Max(0,x)$$

(2.2)

Using these cells is more efficient than sigmoid and still forwards more information compared to binary units. When initializing the weights uniformly, half of the weights were negative. This helps creating a sparse feature representation. Another positive aspect is the relatively cheap computation. No exponential function has to be calculated. This function also prevents the vanishing gradient error, since the gradients were linear functions or zero but in no case non-linear functions.

iii. Pooling (sub-sampling)

Spatial Pooling (also called subsampling or down-sampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, a spatial neighborhood (for example, a 2×2 window) is defined and the largest element is taken from the rectified feature map within that window. In case of average pooling the average or sum of all elements in that window is taken.

Max Pooling reduces the input by applying the maximum function over the input xi. Let m be the size of the filter, then the output calculates as follows:

$$M(x_i) = max\ \{xi+k,+l\ |k| \leq m/2\ ,\ |l| \leq m/2\ k,\ l\epsilon\ \square\}$$

(2.3)

The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling makes the input representations (feature dimension) smaller and more manageable, reduces the number of parameters and computations in the network, therefore, controlling over-fitting, makes the network invariant to small transformations, distortions and translations in the input image.

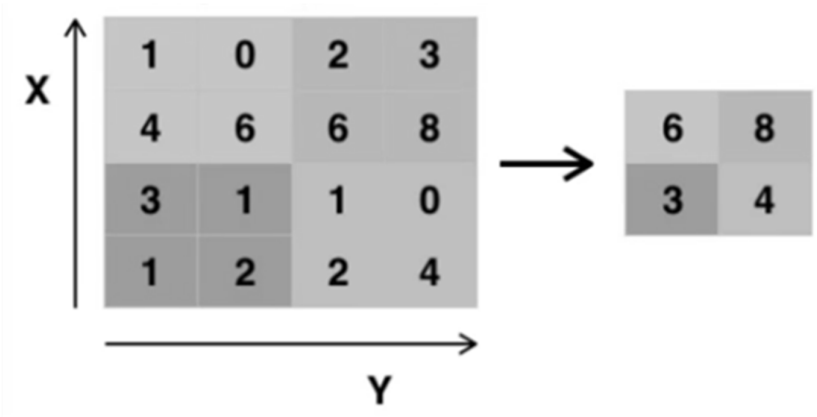


Figure 2.2: Max-Pooling Operation

iv. Multilayer Perceptron:

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses a softmax activation function in the output layer. The term Fully Connected implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represent high-level features of the input

image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

Softmax is used for activation function. It treats the outputs as scores for each class. In the Softmax, the function mapping stayed unchanged and these scores were interpreted as the un-normalized log probabilities for each class. Softmax is calculated as:

$$f(z)_j = \frac{exp(zj)}{\sum_{k=1}^{K} exp(zk)} \tag{2.4}$$

where j is index for image and K is number of total object class.

2.1.2 Deep Residual Network

Deep Residual Network [8] is state-of-the art architecture of CNN based on residual component. ResNet is a deeper architecture than VGG-16 Net and ZFNet, however model size for this architecture is smaller than those network due to global pooling after convolution. The architecture of ResNet with 34 layer is shown in figure 2.3.

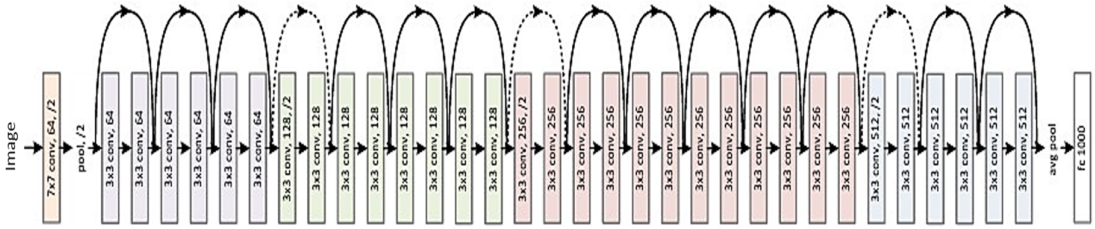


Figure 2.3: Architecture of ResNet [8]

Residual Network is composed of various residual Learing Block,

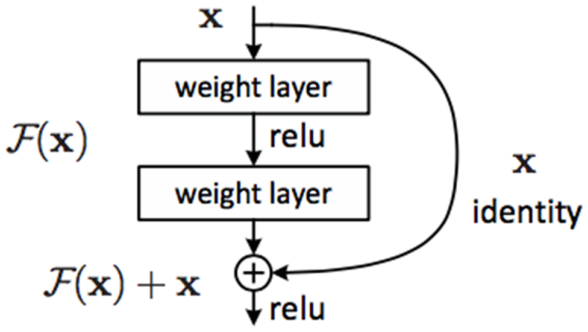


Figure 2.4: A Residual Learning Block [8]

In the above figure, one component from ResNet is shown. Each component batch normalization is performed to normalize the training samples in each batch, which enhance the training process, which is followed by ReLU non linearity activation function.

The final computation of a residual block is:

$$\sigma(F(x)+x)=\sigma([W_2\sigma(W_1x+b_1)+b_2]+x) \quad (2.5)$$

where σ is ReLU non-linearity activation function.

2.1.3 Faster R-CNN

Faster R-CNN [1] is an extension of the R-CNN [9] and Fast R-CNN [10] object detection techniques. All three of these techniques use convolutional neural networks. The difference between them is how they select regions to process and how those regions were classified. R-CNN and Fast R-CNN use a region proposal algorithm as a pre-processing step before running the CNN. The proposal algorithms were typically techniques such as EdgeBoxes [11] or Selective Search [12], which were independent of the CNN. In the case of Fast R-CNN, the use of these techniques becomes the processing bottleneck compared to running the CNN. Faster R-CNN addresses this issue by implementing the region proposal mechanism using the CNN and thereby making region proposal a part of the CNN training and prediction steps. Region Proposal Network (RPN) [1] is proposed to predict proposal regions. RPN slides over the last shared Convolution feature map to determine whether the region is an object or not. Another highlight of this network is shared Convolution layers for RPN and Object recognition.

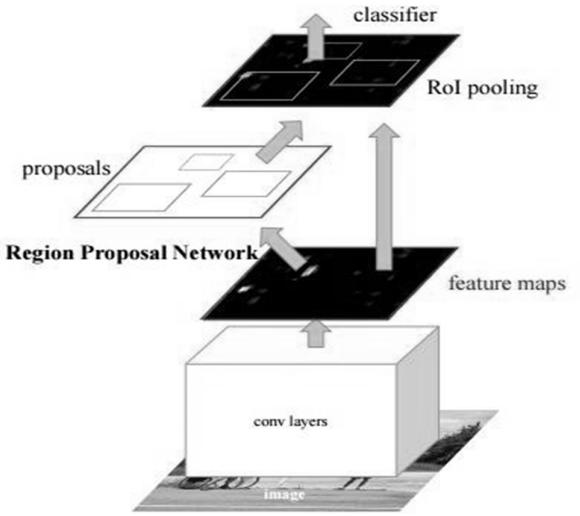


Figure 2.5: Architecture of Faster R-CNN [1]

For each image, it is fed forward to get a Convolution feature map from the last Convolution layer. Then RPN is used to determine if any object present or not in the image. On Convolution feature map of 256 channels, a 3x3 Convolution layer is sliding across the map. At each sliding region, a 256-d feature vector is produced and fully connected to classification layer and regression layer. Classification layer is a 2-class classification layer (object present or not) while regression layer is a boundary box regressor to adjust the position of bounding box. Anchor is introduced to give the network translation-invariant since each anchor represent a combination of different aspect ratio and scale.

After getting proposals, each proposed region on Convolution feature map is passed into a Region of Interest (RoI) pooling layer, the purpose is to get a fixed feature vector output to later Fully Connected layers. The layer size is varying according to the size of input feature map. Since the layer size is changing, it always outputs a fixed feature vector. The RoI feature vector is fully connected to a Fully Connected layer and performs classification of objects in RoI region.

2.1.4 Region Proposal Network

A Region Proposal Network (RPN) takes a video frame as input and outputs a set of rectangular object proposals, each with an objectness score [1]. RPN can be modelled with a fully convolutional network. Region proposals can be generated by sliding a

small network over the convolutional feature map output by the last shared convolutional layer. The input of this network is a spatial window of input convolutional feature map, which is then fed into two different fully-connected layers — a box-regression layer and a box-classification as shown in figure 2.2. RPN operates in a sliding-window fashion, the fully-connected layers were shared across all spatial locations. At each sliding-window location multiple region proposals were predicted. Let k be the maximum possible proposals for each location, regression layer had have 4k outputs encoding the co-ordinates of k boxes and classification layer had have 2k scores as output that estimate probability of objects or not object for each proposal. The k proposals were parameterized relative to k reference boxes or anchors. An anchor is centered at the sliding window and is associated with a scale and aspect ratio. For example 3 scales and 3 aspect ratios had yield k = 9 anchors at each sliding position. For a convolutional feature map of a size W x H, there had been WHk anchors in total. Each anchor has corresponding 2 scores for object is present or not and 4 co-ordinates of the bounding box.

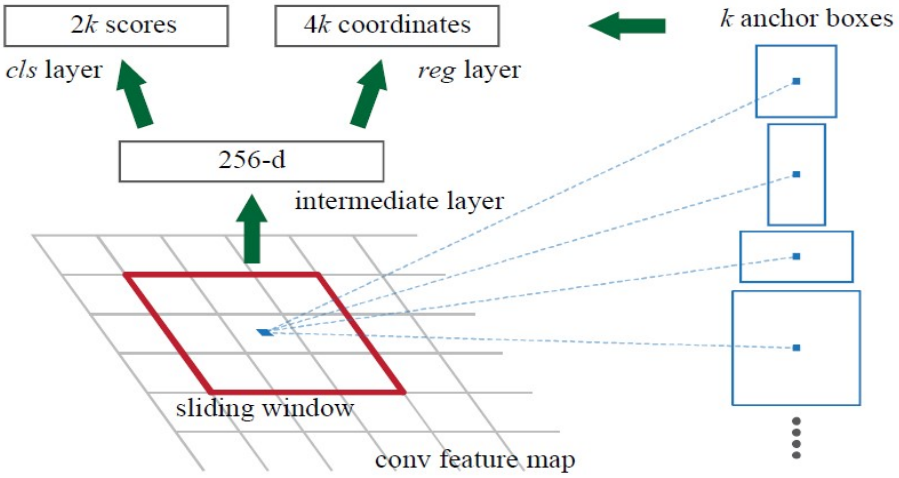


Figure 2.6: Region Proposal Network [1]

The RPN utilizes a sliding window approach in First, an n by n filter is convolved with last layer conv feature map. Then the result is projected to a lower dimensional space by convolving with a 1 by 1 filter (which just linearly combines the channels for each position independently), resulting in a fixed-size vector for each position. The vector is separately passed into a box-regression layer and a box-classification layer. In the box-regression layer, k bounding boxes are generated relative to the current position in the

conv feature map (the current anchor point). For example, if the vector for position (x_a, ya) is passed into the box-regression layer, then the layer will have 4k outputs. Each set of 4 outputs parameterizes a bounding box relative to (x_a, y_a). The four parameters are t_x, t_y, t_w, t_h, where,

$$t_x = (x - x_a)/w_a \,,\ \ t_y = (y - y_a)/h_a \,,\ t_w = \log (w/w_a),\ \ t_h = \log (h/h_a) \qquad (2.6)$$

Each of the k generated bounding boxes will have a different fixed w_a and h_a, which results in boxes with different areas and aspect ratios (because of different absolute and relative magnitudes of w_a and h_a).

The box-classification layer generates 2k outputs, where each pair of 2 outputs is the probability that the corresponding bounding box has an object in it or is just background. That is, the sum of each pair of outputs is 1, and is the probability distribution over whether the bounding box contains an object or not. To reduce the number of bounding box proposals, non-maxima suppression is used on proposals that have high intersection-over-union (IoU) with each other. The boxes are ranked based on the object probability score. Finally, the top N proposals are taken from the remaining proposals, again ranked by object probability score.

The methodology of this research is divided into two parts: first object detection in images and second object detection in videos.

2.1.5 Recurrent Neural Network

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. RNNs are able to handle long term dependencies in natural language processing task.

Long short-term memory (LSTM) is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called "forget" gates. LSTM prevents backpropagated errors from vanishing or exploding. Instead errors can flow backwards through unlimited numbers of virtual layers unfolded in

space. That is, LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier.

2.2. Related Work

In Computer Vision Object detection is one of the fundamental problems and has been studied for years to make it more efficient and faster. Most of the classical object recognition methods involve edge (or contour) and patch based feature extraction [13]. Convolutional neural network (CNN) has become dominating model due to its outstanding performance in object detection [1].

Girshick etal.[8] demonstrated outstanding performance in terms of detection accuracy for object detection in images using Region based Convolutional Neural Network (R-CNN).However, this approach has large computational complexity in order to classify a large number proposed regions. Selective search [12] is used commonly to generate object proposals. However, due to exhaustive search and large number of region proposals from an image, it is computationally expensive. R. Girshick [9] drastically reduced computational cost of R-CNN with the sharing convolutions across proposals by Fast R-CNN. Fast R-CNN achieves near real-time rates using very deep networks, when ignoring the time spent on region proposals. Proposals were the computational bottleneck in state-of-the-art detection systems using Fast R-CNN.

Shaoqing Ren et al. [1] developed Faster R-CNN method based on VGG-16 using Region Proposal Networks (RPNs) that share convolutional layers with state-of-the-art object detection that leads toan elegant and effective object detection solution for images. Ramachandran, Prajit [14] used Faster R-CNN technique based on VGG Net to detect objects in videos with 48.7% mean average precision. Objects detection in videos using Faster R-CNN lacks use of temporal information between consecutive video frames. Some method needs to be determined to incorporate temporal information to improve performance of object detection in videos.

CHAPTER THREE: RESEARCH METHODOLOGY

3.1 Research Work Flow

The research methodology involves the following steps:

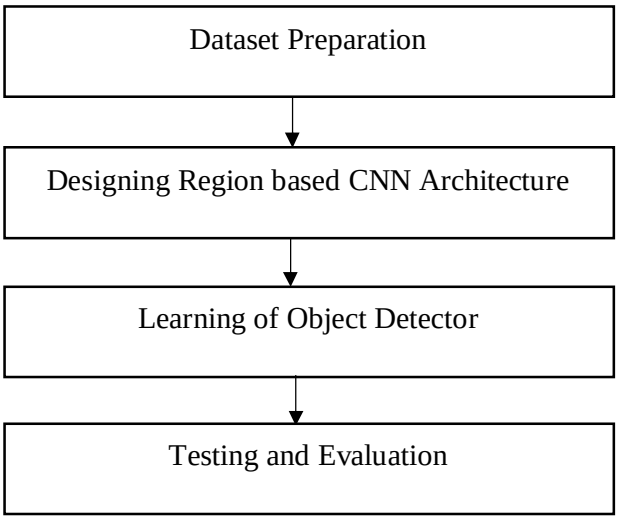


Figure 3.1: Steps for Development of Object Detection System

3.2 Dataset Preparation

PASCAL VOC Dataset is used for detection of objects in images while ImageNet VID dataset is used for detection of objects in video frames.

PASCAL VOC Dataset

PASCAL VOC 2007 detection benchmark [15] was used to train and evaluate object detection in images system. This dataset consists of about 5000 train/validation images and 5000 test images over 20 object categories. 10-fold validation is used for training from train/validation set.

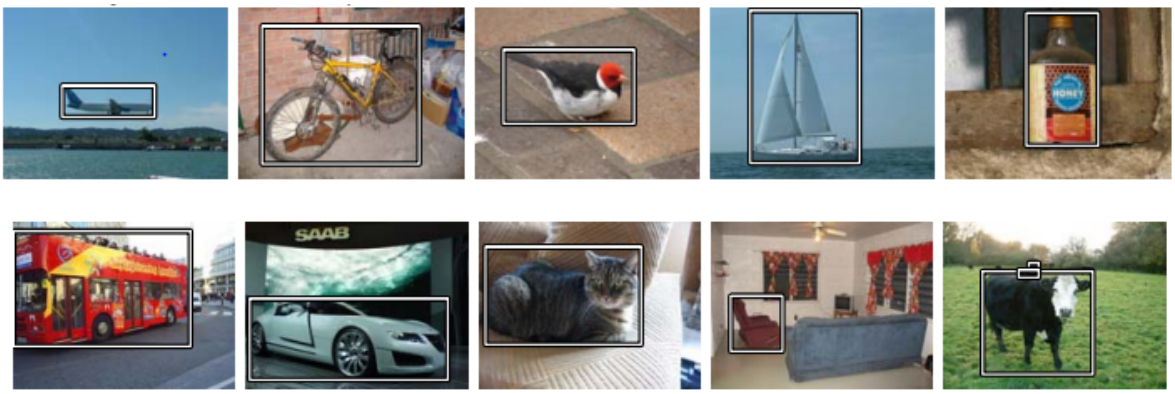


Figure 3.2: Visualization of Images from PASCAL VOC dataset [15]

The distribution of images in PASCAL VOC dataset is shown in table 3.1

Table 3.1: Distribution of PASCAL VOC Dataset

Class ID	Class	Train/val	Test
0	Aeroplane	238	236
1	Bicycle	243	241
2	Bird	330	328
3	Boat	181	179
4	Bottle	244	242
5	Bus	186	160
6	Car	713	711
7	Cat	337	335
8	Chair	445	443
9	Cow	141	139
10	Dining table	200	198
11	Dog	421	419
12	Horse	287	277
13	Motorbike	245	243
14	Person	2008	2006
15	Potted plant	245	243
16	Sheep	96	94
17	Sofa	229	227
18	Train	261	259
19	Tv monitor	256	254
Total		5011	4952

The dataset is fully annotated with object class and bounding box co-ordinates. The sample annotation of the dataset for one image is shown in figure 3.3.

```
<?xml version="1.0"?>
- <annotation>
    <folder>VOC2007</folder>
    <filename>000008.jpg</filename>
  - <source>
      <database>The VOC2007 Database</database>
      <annotation>PASCAL VOC2007</annotation>
      <image>flickr</image>
      <flickrid>332814424</flickrid>
    </source>
  - <owner>
      <flickrid>Artichoke_Soup</flickrid>
      <name>Kat Youwannaknow</name>
    </owner>
  - <size>
      <width>500</width>
      <height>375</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>chair</name>
      <pose>Frontal</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>192</xmin>
        <ymin>16</ymin>
        <xmax>364</xmax>
        <ymax>249</ymax>
      </bndbox>
    </object>
</annotation>
```

Figure 3.3: Annotation of objects in images on PASCAL VOC Dataset

ImageNet VID Dataset

ImageNet VID dataset for object detection in videos [2] is used for training, validating and testing the object detection system for videos. The dataset contains video snippets with bounded box annotations for thirty different object classes as airplane, antelope, bear, bicycle, bird, bus, car, cattle, dog, domestic cat, elephant, fox, giant panda, hamster, horse, lion, lizard, monkey, motorcycle, rabbit, red panda, sheep, snake, squirrel, tiger, train, turtle, watercraft, whale and zebra. All classes were fully labeled for each clip.

Some snippets from the training dataset were:

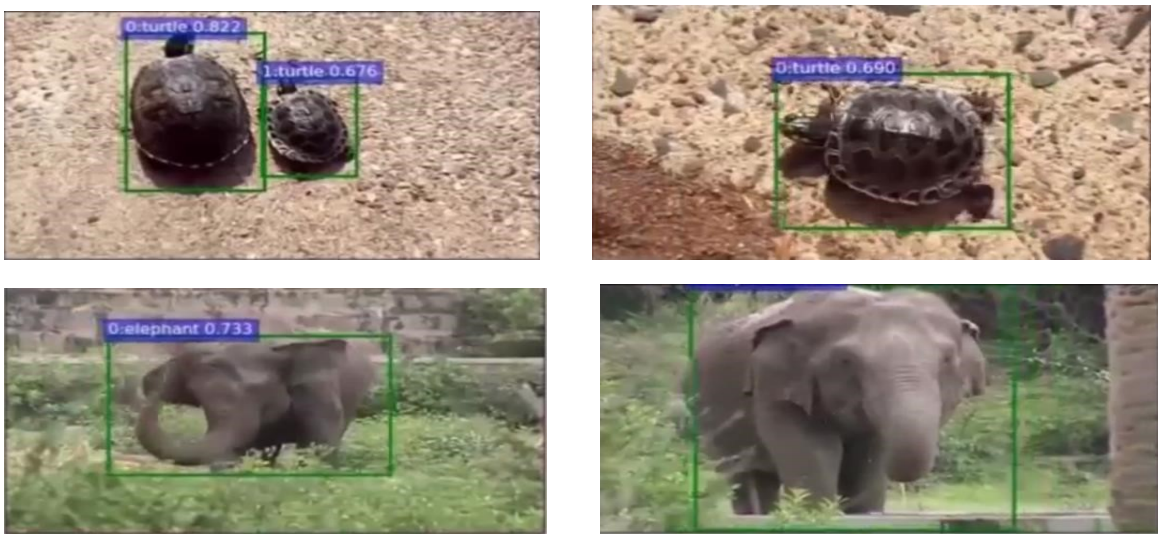


Figure 3.4: Object annotations in video snippets of ImageNet Dataset [2]

Validation dataset also has fully annotated snippets with bounded box annotations of the object categories.

The distribution of a part of ImageNet VID Dataset used in this research work is shown in figure 3.7.

Table 3.2.: Distribution of a part of ImageNet VID Dataset used

Class ID	Class	Train/Validation	Test
0	Airplane	1774	355
1	Antelope	257	51
2	Bear	191	38
3	Bicycle	954	191
4	Bird	393	79
5	Bus	45	12
6	Car	1128	226
7	cattle	813	163
8	dog	479	96
9	domestic_cat	221	44
10	elephant	26	10
11	Fox	107	21
12	giant_panda	445	89
13	hamster	175	35
14	horse	293	59

15	lion	187	37
16	lizard	73	15
17	monkey	214	43
18	motorcycle	429	86
19	Rabbit	45	20
20	red_panda	554	111
21	Sheep	552	110
22	Snake	266	75
23	squirrel	153	31
24	Tiger	148	30
25	Train	337	67
26	Turtle	187	37
27	watercraft	380	100
28	Whale	255	51
29	Zebra	40	10
	Total	5054	1300

The dataset has annotation of class name, and co-ordinates of boundary boxes of objects in the video frames. The sample file for annotation of ImageNet VID Dataset is shown in figure 3.7.

```
<?xml version="1.0"?>
- <annotation>
    <folder>ILSVRC2015_val_00003001</folder>
    <filename>000192</filename>
  - <source>
      <database>ILSVRC_2015</database>
    </source>
  - <size>
      <width>1280</width>
      <height>720</height>
    </size>
  - <object>
      <trackid>1</trackid>
      <name>n02084071</name>
    - <bndbox>
        <xmax>880</xmax>
        <xmin>490</xmin>
        <ymax>563</ymax>
        <ymin>327</ymin>
      </bndbox>
      <occluded>0</occluded>
      <generated>0</generated>
    </object>
</annotation>
```

Figure 3.5: Annotation in ImageNet VID Dataset

3.3 Designing Region based CNN Architecture

A Convolutional Neural Network consists of input layer, output layer and stack of hidden layers which consists of convolutional layer, pooling layers and fully connected (dense) layers. By stacking multiple layers of these hidden layers with different size, type, kernel numbers and activation functions complex architecture for CNN are designed.

Convolution Layer Design

Conv layer takes input as volume [W_1 x H_1 x D_1], and outputs another volume as [W_2 x H_2 x D_2]. It requires four hyper-parameters: number of filters (K), filter's spatial extent (F), amount of Stride (S), and amount of zero padding (P). Output Volume is determined as follows:

$$W_2 = \frac{W_1 - F + 2P}{S} \tag{3.1}$$

$$H_2 = \frac{H_1 - F + 2P}{S} \tag{3.2}$$

$$D_2 = K \tag{3.3}$$

Pooling/ Sub-Sampling Layer

Pooling layer takes input as volume [W_1 x H_1 x D_1], and outputs another volume as [W_2 x H_2 x D_2]. It requires two hyper-parameters: spatial extent (F), amount of Stride (S), and amount of zero padding (P). Output Volume is determined as follows:

$$W_2 = \frac{W_1 - F}{S} + 1 \tag{3.4}$$

$$H_2 = \frac{H_1 - F}{S} + 1 \tag{3.5}$$

$$D_2 = D_1 \tag{3.6}$$

Pooling is done independently on each depth dimension, therefore the depth of the image remains unchanged. Pooling may be min-pooling, max-pooling or averaging. The most common form of pooling layer generally applied is the max pooling.

Non-linear Layers

Non-linear layers are used in CNN to incorporate non-linear properties of the decision functions without affecting the receptive fields of the convolution layer. Rectified Linear Units (ReLUs) function is used after convolutional layers. A ReLU implements the function y = maz(x,0), so the input and out size of this layer are same.

Region Proposal Network

Region proposal network predicts possible region of interest in images that may contain objects in it. The output feature map from the convolution-pooling layers are fed to this network. The output from this network is a number of bounding boxes with their co-ordinates in feature map. The RPN is designed using fully convolutional layers without any fully connected layers.

RoI Pooling

Region of interest pooling (also known as RoI pooling) is an operation widely used in object detection tasks using convolutional neural networks. Its purpose is to perform max pooling on inputs of non-uniform sizes to obtain fixed-size feature maps. The output from RPN is different sized region proposal that may contain objects in them. RoI Pooling layer is designed to output fixed sized feature maps for different sized input region proposals.

Fully connected layers

These layers are also called dense layers. Output from the convolution or pooling layers is fed to these layers and output from the last fully connected layer is the output of the network. Fixed sized output from RoI pooling layers is flattened and fed to these layers. These layers provides Output from convolution-pooling layers is flattened before feeding fully connected layer.

3.4 Learning of Object Detector

After designing the architecture of the region based convolutional neural network, the network is implemented for learning so that it learns to detect objects. For this purpose, supervised learning model is used.

Training

The training of designed CNN architecture included training of both region proposal network and object regressor/classifier network. RPN operates in a sliding-window fashion, the fully-connected layers were shared across all spatial locations. At each sliding-window location multiple region proposals were predicted. During training all cross-boundary anchors contribution were not considered for the loss. For a typical 1000×600 image, there were be roughly 20k ($\approx 60 \times 40 \times 9$) anchors in total. With the cross-boundary anchors ignored, there were about 6000 anchors per image for training. Some RPN proposals highly overlapped with each other. To reduce redundancy, non-maximum suppression (NMS) on the proposal regions based on their classification scores. Intersection over Union (IoU) threshold for NMS was taken as 0.7 and only top 300 boundary boxes were taken for the training. For Intersection over Union computation two sets of boundary boxes were used: ground truth boxes from training annotated dataset and predicted boundary boxes. IoU was calculated by dividing area of overlap/intersection by area of union between ground truth boxes and predicted boundary boxes.

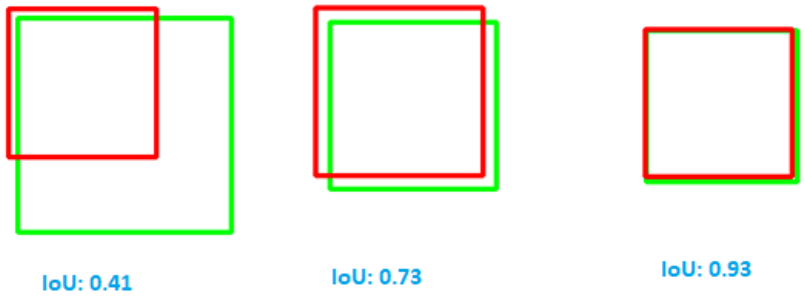


Figure 3.6: Intersection over Union (IoU) Calculation

For training RPNs a binary class label of being an object or not to each anchor had been assigned. A positive label had been assigned to two kinds of anchors as the anchor/anchors with the highest Intersection-over-Union (IoU) overlap with a ground-truth box and an anchor that has an IoU overlap higher than 0.7 with any ground-truth

box. A single ground-truth box may assign positive labels to multiple anchors. A negative label had been assigned to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes. Anchors that were neither positive nor negative do not contribute to the training objective. The objective function had been minimized following the multi-task loss in Fast R-CNN [8] defined as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (3.7)$$

Here, i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. The ground-truth label p_i^* is 1 if the anchor is positive, and is 0 if the anchor is negative. t_i is a vector representing the 4 parameterized coordinates of the predicted bounding box, and t_i^* is that of the ground-truth box associated with a positive anchor. The classification loss L_{cls} is log loss over two classes (object vs. not object). For the regression loss $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$ where R is the robust loss function (smooth L1) defined in [8]. The term $p_i^* L_{reg}$ means the regression loss is activated only for positive anchors ($p_i^* = 1$) and is disabled otherwise ($p_i^* = 0$).

The RPN had been trained end-to-end by backpropagation and stochastic gradient descent (SGD). Unlike gradient descent where gradients were calculated after running on entire dataset, SGD calculates over few examples at a time.

Then, for each object proposal a region of interest (RoI) pooling layer had been extracted as fixed-length feature vector from the feature map. Each feature vector had been fed into a sequence of fully connected layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

Then RPN and CNN networks had been merged into one network during training. In each SGD iteration, the forward pass had generates region proposals which were treated just like fixed, precomputed proposals when training a CNN detector. The backward propagation had taken place as usual, where for the shared layers the backward propagated signals from both the RPN loss and the CNN loss had been combined.

The output from the network has two component: classifier that provides class of the detected objects and regressor that provides four co-ordinates of the detected objects boundary boxes. Apart from change in model architecture, the training process for object detection is same as that for object detection in images.

3.4 Region Based CNN Architecture Design and Implementation

Region based CNN architecture was designed based on ResNET50 network. The layers after average pool layer of ResNET50 were modified to implement Region proposal network and classifier/regressor network. The architecture of region based CNN for detection of objects in Videos was further refined by addition of LSTM recurrent layer after RoI pooling in the network. The details of these designs are given below:

3.4.1 Object Detection in Images

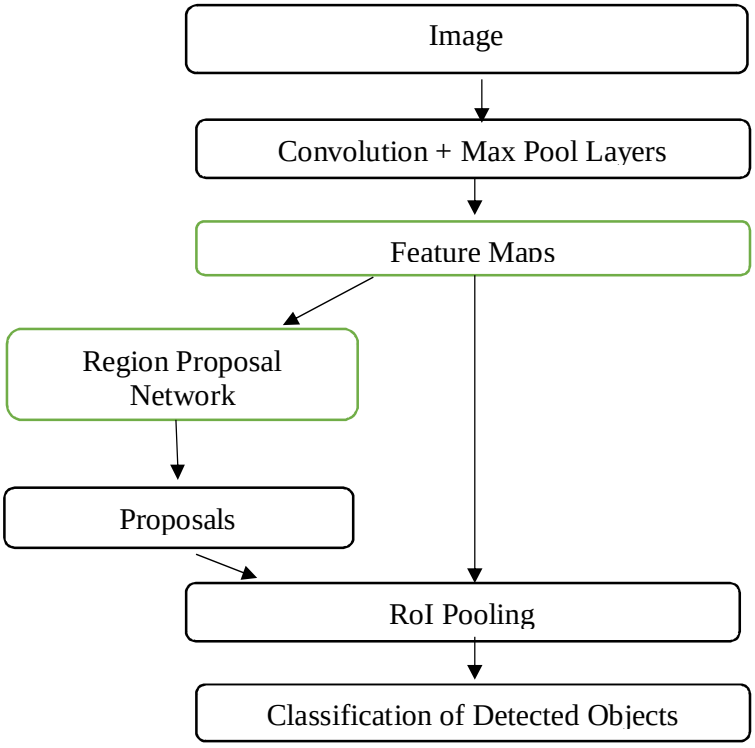


Figure 3.7: Block diagram of Object detection in images using region based CNN

The block diagram of the system design for detection of objects in images is shown in figure 3.7

Architecture Design of the CNN

ResNet architecture was modified to implement Faster R-CNN for object detection in images. The architecture of the modified ResNet Architecture is given in table 3.3.

Table 3.3: Architecture of the System Implementation for object detection in images

Layer (type)	Output Shape	No of Parameters	Connected to
input_1 (InputLayer)	(None, None, None, 3)	0	zero_padding2d_1
zero_padding2d_1	(None, None, None, 3)	0	conv1 (Conv2D)
conv1 (Conv2D)	(None, None, None, 64)	9472	bn_conv1 zatio
bn_conv1 zatio	(None, None, None, 64)	256	activation_1 (Activation)
activation_1 (Activation)	(None, None, None, 64)	0	max_pooling2d_1
max_pooling2d_1	(None, None, None, 64)	0	res2a_branch2a (Conv2D)
res2a_branch2a (Conv2D)	(None, None, None, 64)	4160	bn2a_branch2a
bn2a_branch2a	(None, None, None, 64)	256	activation_2 (Activation)
activation_2 (Activation)	(None, None, None, 64)	0	res2a_branch2b (Conv2D)
res2a_branch2b (Conv2D)	(None, None, None, 64)	36928	bn2a_branch2b
bn2a_branch2b	(None, None, None, 64)	256	activation_3 (Activation)
activation_3 (Activation)	(None, None, None, 64)	0	res2a_branch2c (Conv2D)
res2a_branch2c (Conv2D)	(None, None, None, 25)	16640	res2a_branch1 (Conv2D)
res2a_branch1 (Conv2D)	(None, None, None, 25)	16640	bn2a_branch2c
bn2a_branch2c	(None, None, None, 25)	1024	bn2a_branch1 z
bn2a_branch1 z	(None, None, None, 25)	1024	add_1 (Add)
add_1 (Add)	(None, None, None, 25)	0	activation_4 (Activation)

activation_4 (Activation)	(None, None, None, 25)	0	res2b_branch2a (Conv2D)
res2b_branch2a (Conv2D)	(None, None, None, 64)	16448	bn2b_branch2a
bn2b_branch2a	(None, None, None, 64)	256	activation_5 (Activation)
activation_5 (Activation)	(None, None, None, 64)	0	res2b_branch2b (Conv2D)
res2b_branch2b (Conv2D)	(None, None, None, 64)	36928	bn2b_branch2b
bn2b_branch2b	(None, None, None, 64)	256	activation_6 (Activation)
activation_6 (Activation)	(None, None, None, 64)	0	res2b_branch2c (Conv2D)
res2b_branch2c (Conv2D)	(None, None, None, 25)	16640	bn2b_branch2c
bn2b_branch2c	(None, None, None, 25)	1024	add_2 (Add)
add_2 (Add)	(None, None, None, 25)	0	activation_7 (Activation)
activation_7 (Activation)	(None, None, None, 25)	0	res2c_branch2a (Conv2D)
res2c_branch2a (Conv2D)	(None, None, None, 64)	16448	bn2c_branch2a
bn2c_branch2a	(None, None, None, 64)	256	activation_8 (Activation)
activation_8 (Activation)	(None, None, None, 64)	0	res2c_branch2b (Conv2D)
res2c_branch2b (Conv2D)	(None, None, None, 64)	36928	bn2c_branch2b
bn2c_branch2b	(None, None, None, 64)	256	activation_9 (Activation)
activation_9 (Activation)	(None, None, None, 64)	0	res2c_branch2c (Conv2D)
res2c_branch2c (Conv2D)	(None, None, None, 25)	16640	bn2c_branch2c
bn2c_branch2c	(None, None, None, 25)	1024	add_3 (Add)
add_3 (Add)	(None, None, None, 25)	0	activation_10 (Activation)
activation_10 (Activation)	(None, None, None, 25)	0	res3a_branch2a (Conv2D)
res3a_branch2a (Conv2D)	(None, None, None, 12)	32896	bn3a_branch2a

bn3a_branch2a	(None, None, None, 12)	512	activation_11 (Activation)
activation_11 (Activation)	(None, None, None, 12)	0	res3a_branch2b (Conv2D)
res3a_branch2b (Conv2D)	(None, None, None, 12)	147584	bn3a_branch2b
bn3a_branch2b	(None, None, None, 12)	512	activation_12 (Activation)
activation_12 (Activation)	(None, None, None, 12)	0	res3a_branch2c (Conv2D)
res3a_branch2c (Conv2D)	(None, None, None, 51)	66048	res3a_branch1 (Conv2D)
res3a_branch1 (Conv2D)	(None, None, None, 51)	131584	bn3a_branch2c
bn3a_branch2c	(None, None, None, 51)	2048	bn3a_branch1 z
bn3a_branch1 z	(None, None, None, 51)	2048	add_4 (Add)
add_4 (Add)	(None, None, None, 51)	0	activation_13 (Activation)
activation_13 (Activation)	(None, None, None, 51)	0	res3b_branch2a (Conv2D)
res3b_branch2a (Conv2D)	(None, None, None, 12)	65664	bn3b_branch2a
bn3b_branch2a	(None, None, None, 12)	512	activation_14 (Activation)
activation_14 (Activation)	(None, None, None, 12)	0	res3b_branch2b (Conv2D)
res3b_branch2b (Conv2D)	(None, None, None, 12)	147584	bn3b_branch2b
bn3b_branch2b	(None, None, None, 12)	512	activation_15 (Activation)
activation_15 (Activation)	(None, None, None, 12)	0	res3b_branch2c (Conv2D)
res3b_branch2c (Conv2D)	(None, None, None, 51)	66048	bn3b_branch2c
bn3b_branch2c	(None, None, None, 51)	2048	add_5 (Add)
add_5 (Add)	(None, None, None, 51)	0	activation_16 (Activation)
activation_16 (Activation)	(None, None, None, 51)	0	res3c_branch2a (Conv2D)
res3c_branch2a (Conv2D)	(None, None, None, 12)	65664	bn3c_branch2a

bn3c_branch2a	(None, None, None, 12)	512	activation_17 (Activation)
activation_17 (Activation)	(None, None, None, 12)	0	res3c_branch2b (Conv2D)
res3c_branch2b (Conv2D)	(None, None, None, 12)	147584	bn3c_branch2b
bn3c_branch2b	(None, None, None, 12)	512	activation_18 (Activation)
activation_18 (Activation)	(None, None, None, 12)	0	res3c_branch2c (Conv2D)
res3c_branch2c (Conv2D)	(None, None, None, 51)	66048	bn3c_branch2c
bn3c_branch2c	(None, None, None, 51)	2048	add_6 (Add)
add_6 (Add)	(None, None, None, 51)	0	activation_19 (Activation)
activation_19 (Activation)	(None, None, None, 51)	0	res3d_branch2a (Conv2D)
res3d_branch2a (Conv2D)	(None, None, None, 12)	65664	bn3d_branch2a
bn3d_branch2a	(None, None, None, 12)	512	activation_20 (Activation)
activation_20 (Activation)	(None, None, None, 12)	0	res3d_branch2b (Conv2D)
res3d_branch2b (Conv2D)	(None, None, None, 12)	147584	bn3d_branch2b
bn3d_branch2b	(None, None, None, 12)	512	activation_21 (Activation)
activation_21 (Activation)	(None, None, None, 12)	0	res3d_branch2c (Conv2D)
res3d_branch2c (Conv2D)	(None, None, None, 51)	66048	bn3d_branch2c
bn3d_branch2c	(None, None, None, 51)	2048	add_7 (Add)
add_7 (Add)	(None, None, None, 51)	0	activation_22 (Activation)
activation_22 (Activation)	(None, None, None, 51)	0	res4a_branch2a (Conv2D)
res4a_branch2a (Conv2D)	(None, None, None, 25)	131328	bn4a_branch2a
bn4a_branch2a	(None, None, None, 25)	1024	activation_23 (Activation)
activation_23 (Activation)	(None, None, None, 25)	0	res4a_branch2b (Conv2D)

res4a_branch2b (Conv2D)	(None, None, None, 25)	590080	bn4a_branch2b
bn4a_branch2b	(None, None, None, 25)	1024	activation_24 (Activation)
activation_24 (Activation)	(None, None, None, 25)	0	res4a_branch2c (Conv2D)
res4a_branch2c (Conv2D)	(None, None, None, 10)	263168	res4a_branch1 (Conv2D)
res4a_branch1 (Conv2D)	(None, None, None, 10)	525312	bn4a_branch2c
bn4a_branch2c	(None, None, None, 10)	4096	bn4a_branch1 z
bn4a_branch1 z	(None, None, None, 10)	4096	add_8 (Add)
add_8 (Add)	(None, None, None, 10)	0	activation_25 (Activation)
activation_25 (Activation)	(None, None, None, 10)	0	res4b_branch2a (Conv2D)
res4b_branch2a (Conv2D)	(None, None, None, 25)	262400	bn4b_branch2a
bn4b_branch2a	(None, None, None, 25)	1024	activation_26 (Activation)
activation_26 (Activation)	(None, None, None, 25)	0	res4b_branch2b (Conv2D)
res4b_branch2b (Conv2D)	(None, None, None, 25)	590080	bn4b_branch2b
bn4b_branch2b	(None, None, None, 25)	1024	activation_27 (Activation)
activation_27 (Activation)	(None, None, None, 25)	0	res4b_branch2c (Conv2D)
res4b_branch2c (Conv2D)	(None, None, None, 10)	263168	bn4b_branch2c
bn4b_branch2c	(None, None, None, 10)	4096	add_9 (Add)
add_9 (Add)	(None, None, None, 10)	0	activation_28 (Activation)
activation_28 (Activation)	(None, None, None, 10)	0	res4c_branch2a (Conv2D)
res4c_branch2a (Conv2D)	(None, None, None, 25)	262400	bn4c_branch2a
bn4c_branch2a	(None, None, None, 25)	1024	activation_29 (Activation)
activation_29 (Activation)	(None, None, None, 25)	0	res4c_branch2b (Conv2D)
res4c_branch2b (Conv2D)	(None, None, None, 25)	590080	bn4c_branch2b

bn4c_branch2b	(None, None, None, 25)	1024	activation_30 (Activation)
activation_30 (Activation)	(None, None, None, 25)	0	res4c_branch2c (Conv2D)
res4c_branch2c (Conv2D)	(None, None, None, 10)	263168	bn4c_branch2c
bn4c_branch2c	(None, None, None, 10)	4096	add_10 (Add)
add_10 (Add)	(None, None, None, 10)	0	activation_31 (Activation)
activation_31 (Activation)	(None, None, None, 10)	0	res4d_branch2a (Conv2D)
res4d_branch2a (Conv2D)	(None, None, None, 25)	262400	bn4d_branch2a
bn4d_branch2a	(None, None, None, 25)	1024	activation_32 (Activation)
activation_32 (Activation)	(None, None, None, 25)	0	res4d_branch2b (Conv2D)
res4d_branch2b (Conv2D)	(None, None, None, 25)	590080	bn4d_branch2b
bn4d_branch2b	(None, None, None, 25)	1024	activation_33 (Activation)
activation_33 (Activation)	(None, None, None, 25)	0	res4d_branch2c (Conv2D)
res4d_branch2c (Conv2D)	(None, None, None, 10)	263168	bn4d_branch2c
bn4d_branch2c	(None, None, None, 10)	4096	add_11 (Add)
add_11 (Add)	(None, None, None, 10)	0	activation_34 (Activation)
activation_34 (Activation)	(None, None, None, 10)	0	res4e_branch2a (Conv2D)
res4e_branch2a (Conv2D)	(None, None, None, 25)	262400	bn4e_branch2a
bn4e_branch2a	(None, None, None, 25)	1024	activation_35 (Activation)
activation_35 (Activation)	(None, None, None, 25)	0	res4e_branch2b (Conv2D)
res4e_branch2b (Conv2D)	(None, None, None, 25)	590080	bn4e_branch2b
bn4e_branch2b	(None, None, None, 25)	1024	activation_36 (Activation)
activation_36 (Activation)	(None, None, None, 25)	0	res4e_branch2c (Conv2D)

res4e_branch2c (Conv2D)	(None, None, None, 10)	263168	bn4e_branch2c
bn4e_branch2c	(None, None, None, 10)	4096	add_12 (Add)
add_12 (Add)	(None, None, None, 10)	0	activation_37 (Activation)
activation_37 (Activation)	(None, None, None, 10)	0	res4f_branch2a (Conv2D)
res4f_branch2a (Conv2D)	(None, None, None, 25)	262400	bn4f_branch2a
bn4f_branch2a	(None, None, None, 25)	1024	activation_38 (Activation)
activation_38 (Activation)	(None, None, None, 25)	0	res4f_branch2b (Conv2D)
res4f_branch2b (Conv2D)	(None, None, None, 25)	590080	bn4f_branch2b
bn4f_branch2b	(None, None, None, 25)	1024	activation_39 (Activation)
activation_39 (Activation)	(None, None, None, 25)	0	res4f_branch2c (Conv2D)
res4f_branch2c (Conv2D)	(None, None, None, 10)	263168	bn4f_branch2c
bn4f_branch2c	(None, None, None, 10)	4096	add_13 (Add)
add_13 (Add)	(None, None, None, 10)	0	activation_40 (Activation)
activation_40 (Activation)	(None, None, None, 10)	0	input_2 (InputLayer)
input_2 (InputLayer)	(None, 16, 4)	0	roi_pooling_conv_1
roi_pooling_conv_1	(None, 16, 14, 14, 10)	0	res5a_branch2a
res5a_branch2a	(None, 16, 7, 7, 512)	524800	bn5a_branch2a
bn5a_branch2a	(None, 16, 7, 7, 512)	2048	activation_41 (Activation)
activation_41 (Activation)	(None, 16, 7, 7, 512)	0	res5a_branch2b
res5a_branch2b	(None, 16, 7, 7, 512)	2359808	bn5a_branch2b
bn5a_branch2b	(None, 16, 7, 7, 512)	2048	activation_42 (Activation)
activation_42 (Activation)	(None, 16, 7, 7, 512)	0	res5a_branch2c
res5a_branch2c	(None, 16, 7, 7, 2048)	1050624	res5a_branch1
res5a_branch1	(None, 16, 7, 7, 2048)	2099200	bn5a_branch2c
bn5a_branch2c	(None, 16, 7, 7, 2048)	8192	bn5a_branch1
bn5a_branch1	(None, 16, 7, 7, 2048)	8192	add_14 (Add)

add_14 (Add)	(None, 16, 7, 7, 2048)	0	activation_43 (Activation)
activation_43 (Activation)	(None, 16, 7, 7, 2048)	0	res5b_branch2a
res5b_branch2a	(None, 16, 7, 7, 512)	1049088	bn5b_branch2a
bn5b_branch2a	(None, 16, 7, 7, 512)	2048	activation_44 (Activation)
activation_44 (Activation)	(None, 16, 7, 7, 512)	0	res5b_branch2b
res5b_branch2b	(None, 16, 7, 7, 512)	2359808	bn5b_branch2b
bn5b_branch2b	(None, 16, 7, 7, 512)	2048	activation_45 (Activation)
activation_45 (Activation)	(None, 16, 7, 7, 512)	0	res5b_branch2c
res5b_branch2c	(None, 16, 7, 7, 2048)	1050624	bn5b_branch2c
bn5b_branch2c	(None, 16, 7, 7, 2048)	8192	add_15 (Add)
add_15 (Add)	(None, 16, 7, 7, 2048)	0	activation_46 (Activation)
activation_46 (Activation)	(None, 16, 7, 7, 2048)	0	res5c_branch2a
res5c_branch2a	(None, 16, 7, 7, 512)	1049088	bn5c_branch2a
bn5c_branch2a	(None, 16, 7, 7, 512)	2048	activation_47 (Activation)
activation_47 (Activation)	(None, 16, 7, 7, 512)	0	res5c_branch2b
res5c_branch2b	(None, 16, 7, 7, 512)	2359808	bn5c_branch2b
bn5c_branch2b	(None, 16, 7, 7, 512)	2048	activation_48 (Activation)
activation_48 (Activation)	(None, 16, 7, 7, 512)	0	res5c_branch2c
res5c_branch2c	(None, 16, 7, 7, 2048)	1050624	bn5c_branch2c
bn5c_branch2c	(None, 16, 7, 7, 2048)	8192	add_16 (Add)
add_16 (Add)	(None, 16, 7, 7, 2048)	0	activation_49 (Activation)
activation_49 (Activation)	(None, 16, 7, 7, 2048)	0	avg_pool
avg_pool	(None, 16, 1, 1, 2048)	0	time_distributed_1
time_distributed_1	(None, 16, 2048)	0	dense_class_31
dense_class_31	(None, 16, 31)	63519	dense_regress_31te
dense_regress_31	(None, 16, 120)	245880	

In above table, ‘None’ denotes the size of feature map which depends upon batch size.

The implementation of the above architecture of Faster R-CNN based on ResNet is summarized in figure 3.8.

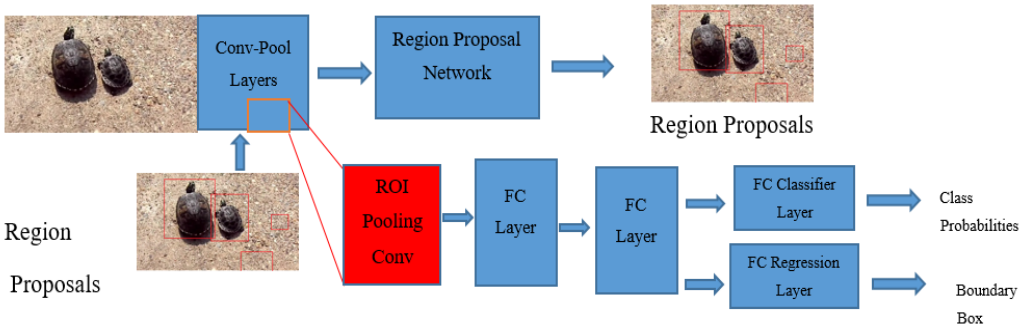


Figure 3.8: Implementation of Architecture of the Faster R-CNN based on ResNet

The input images were resized such that the lowest side is equal to 600 pixel while aspect ratio was kept same as original image. The image sets were divided into batch of 8 images and fed to ResNet network, the output from last convolution and pooling layer is forwarded to Region Proposal Network (RPN). RPN consists of fully connected layers which took base convolutional feature maps and number of anchor and their size and aspect ratios as input. For anchors 3 scales with box areas of 128 , 256 , and 512 pixels, and 3 aspect ratios of 1:1, 1:2, and 2:1 were used.

3.4.2 Object Detection from Videos

For detection of objects in videos, the Faster R-CNN model used for detection of objects in images was modified to incorporate temporal information between consecutive video frames using recurrent network. Recurrent Neural Network pass useful information to later time steps and also propagate the gradient backward to the previous time steps. LSTM (Long Short Term Memory) is a model of Recurrent Neural Network. LSTM layer was added after first fully connected layer in the ResNet architecture. This layer passes useful temporal information of consecutive video frames and useful for detecting objects in videos. The block diagram for the system of object detection from Videos is shown in figure 3.9.

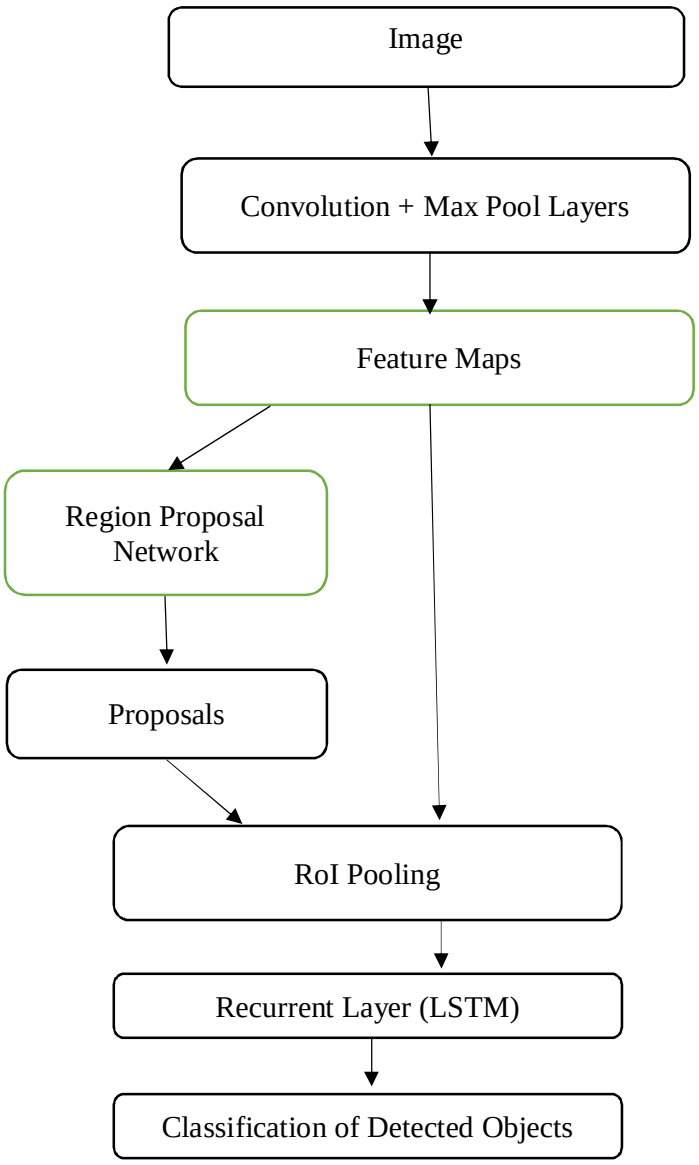


Figure 3.9: Block Diagram of Object Detection in Videos using region based CNN

Architecture Design of the CNN

ResNet architecture was further modified to implement Faster R-CNN for object detection in images with introduction of Recurrent Layer i.e Long Short Term Memory (LSTM). The architecture shown in table 3.3 is modified after last average pooling layer by adding a LSTM layer as shown in table 3.4.

Table 3.4: Architecture of CNN for Detection of Objects in Videos

Layer	Size	No of Parameter	Connected To
….Similar to Table 4.1			
time_distributed_1	(None, 16, 2048)	0	Lstm_256
lstm_1	(None,16, 2046)	83886	dense_class_31
dense_class_31	(None, 16, 31)	63519	dense_regress_31te
dense_regress_31	(None, 16, 120)	245880	

Video frames from video had been taken as input and the network had processes the whole video frame with several convolutional and max pooling layers to produce a convolution feature map. Convolution feature maps had been fed to Region proposal network (RPN).

The ResNet architecture is modified to implement Faster R-CNN by adding RPN, classifier network and regression network after the average pooling layer before fully connected layers.

Faster R-CNN performed object detection in a single independent frame. For object detection in videos, this method lacks temporal information due to processing of frames independently. Temporal information from bounding boxes over time had been taken into consideration such that the objects in the given frame could be more accurately localized. The temporal information had been leveraged by adding a recurrent layer i.e. LSTM layer between consecutive frames. Temporal information of successive video frames had been used to correctly localize the objects in the current frame.

Long short-term memory (LSTM) is a recurrent neural network (RNN) architecture that remembers values over arbitrary intervals. Stored values are not modified as learning proceeds. RNNs allow forward and backward connections between neurons. The architecture of refined model of Faster RCNN with LSTM is shown in figure 3.10.

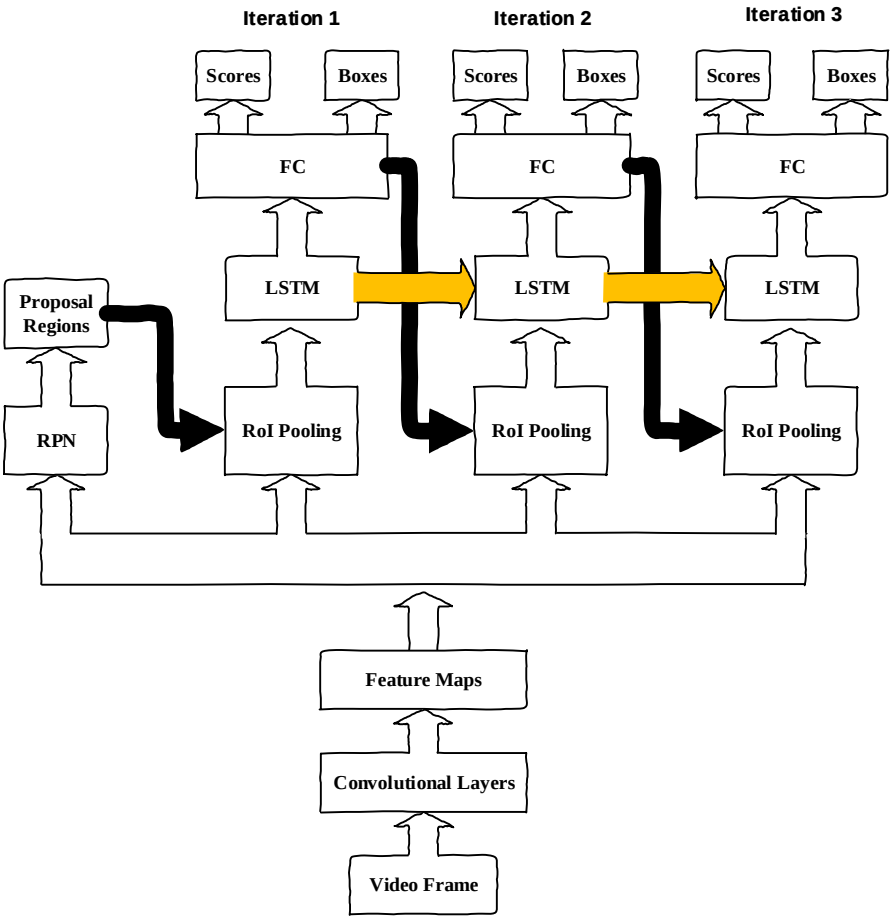


Figure 3.10: Refinement in Faster R-CNN with addition of LSTM

A LSTM layer with 2046 neuron unit was added right before the final fully connected (output) layer, and pass the hidden states of LSTM to the next iteration.

$$f = ResNet(image) \tag{3.8}$$

$$RP_1 = RPN(f) \tag{3.9}$$

If the iteration number = T, then in each iteration i =1, 2, ...T,

$$r_i = RoIPooling(f, RoI_i) \tag{3.10}$$

$$a_i = Flatten(r_i) \tag{3.11}$$

$$h_i = LSTM(h_{i-1}, a_i) \tag{3.12}$$

$$scores_i , boxes_i = FC(h_i) \tag{3.13}$$

where h_i is the hidden state of LSTM at iteration i, and the input of the LSTM layer is h_{i-1}, the hidden state of previous iteration, and a_i , the output of two-layer fully

connected network in current iteration. The benefit of adding a LSTM layer is that the hidden state of previous iteration contains information that is useful to improve classification and bounding box regression results in current iteration. Another benefit is that the gradient of loss from the later iterations is back propagated to the earlier ones, since LSTM is differentiable with respect to the previous hidden state.

3.5 Testing and Performance Evaluations Metrics

The model is tested using the PASCAL VOC Dataset for images and Imagenet VID Dataset for Videos. The performance of system is evaluated in term of mean average precision or detection rate, recall and F1-score as in the following formulas:

1. Precision or Detection Rate:

It is rate of correctly detected objects out of total objects in the system.

$$Precision = \frac{TP}{TP+FP} \tag{3.14}$$

The mean average precision metric is used for evaluating multiclass object detection. Mean average precision for a set of classes is the mean of the average precision scores for each class.

$$MeanAveragePrecision = \frac{\sum_{q=1}^{NumberofClass} AveragePrecision(q)}{NumberofClass} \tag{3.14}$$

2. Recall:

The recall is the fraction of correctly detected objects to the number of correctly detected objects and missed objects.

$$Recall = \frac{TP}{TP+FN} \tag{3.15}$$

3. F1-Measure:

It is the harmonic mean of precision and recall.

$$F - Measure = 2.\frac{Precision.Recall}{Precision+Recall} \quad (3.16)$$

Where,

True Positive (TP) = Objects that are correctly detected as given objects class and boundary

True Negative (TN) = Background detected as background in the image/video

False Positive (FP) = Background data that are incorrectly detected as objects

False Negative (FN) = Objects that are incorrectly detected as background

The developed object detection is evaluated against the speed of processing images/frames per second.

3.6 Experimental Setup and Tools

The datasets used for this this research are PASCAl VOC and ImageNet. All the experiments were done in Amazon EC2 cloud-computing service. EC2 instance g2.8xlarge instance was used for the experiment. The EC2 instance has 32 vCPUs., 60 GiB of memory, 240 GB (2 x 120) of SSD storage and Four NVIDIA GRID GPUs, each with 1,536 CUDA cores and 4 GB of video memory. Ubuntu 16.04 Linux distribution platform with Python programming language and Tensorflow deep learning framework was used for the experiments.The tools and software's that are used in this thesis work are listed below:

- Ubuntu 16.04 Linux
- Python 2.7
- Tensorflow 1.1.0
- SSH Secure Shell
- Pycharm IDE
- Sublime Text

CHAPTER FOUR: RESULT and ANALYSIS

4.1 Objects Detection in Images

The system for detecting objects in images is trained and evaluated in PASCAL VOC dataset. This dataset contains fully annotated images with 5000 images tagged as train/validation set and 5000 images tagged as testing set. Faster R-CNN based on ResNet with 50 layers is trained with this dataset for 50 epochs. The images were resized as the lowest side is 600 pixels keeping aspect ratios. The training and validation loss for RPN classifier 50 epochs is shown in Figure 4.1.

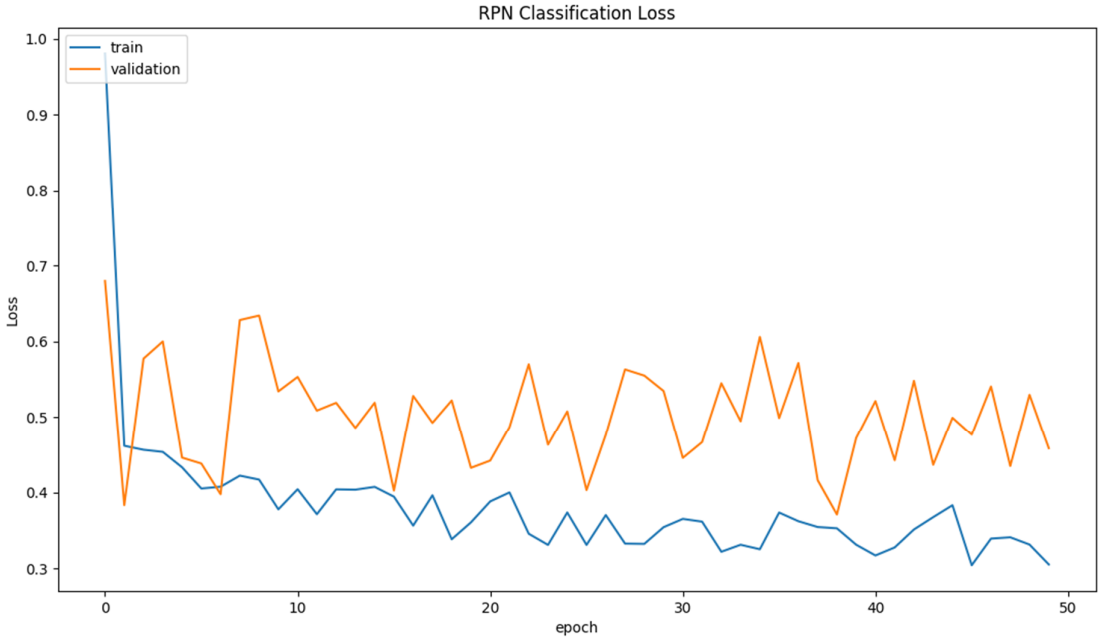


Figure 4.1: RPN Classifier Loss in PASCAL VOC Dataset

Figure 4.5 shows the RPN Classifier loss vs epoch number. Training loss is reduced to 0.30 while validation loss is reduced to 0.5 in 50 epochs.

After 50 epochs of training, the results are shown in table 4.1

Table 4.1 Training results on PASCAL VOC image Dataset

Metrics	Training	Validation
Classifier accuracy for bounding boxes from RPN	0.69	0.5
Loss RPN classifier	0.30	0.45
Loss RPN regression	0.15	0.40
Loss Detector classifier	1.20	2.68
Loss Detector regression	0.38	0.33

The system was then evaluated using test samples of PASCAL VOC dataset. The evaluation was done using mean average precision (mAP), which is widely used in object detection/classification. The mean average precision for all twenty categories and total mAP is shown in table 4.2

Table 4.2: Performance of Faster R-CNN with ResNet50 in PASCAL VOC image dataset

Class ID	Class	mAP VGG-16 Net	mAP ResNET50
1	Aeroplane	0.7	0.81
2	Bicycle	0.806	0.9
3	Bird	0.701	0.85
4	Boat	0.573	0.84
5	Bottle	0.499	0.86
6	Bus	0.782	0.77
7	Car	0.804	0.86
8	Cat	0.82	0.82
9	Chair	0.522	0.86
10	Cow	0.753	0.84
11	DiningTable	0.672	0.85
12	Dog	0.803	0.68
13	Horse	0.798	0.51
14	Motorbike	0.75	0.91
15	Person	0.763	0.51
16	PottedPlant	0.391	0.72
17	Sheep	0.683	0.58
18	Sofa	0.673	0.79
19	Train	0.811	0.71
20	Tv/monitor	0.676	0.87
	Total mAP	0.699	0.78

The mean average precision of 0.78 on PASCAL VOC dataset was achieved using Faster R-CNN with ResNet50 which is better than mean average precision of Faster R-CNN with VGG-16 Net i.e. 69.9 [1]. The comparison between the mean average precision of Faster R-CNN with VGG-16 Net and mean precision obtained in this research work is shown in figure 4.2

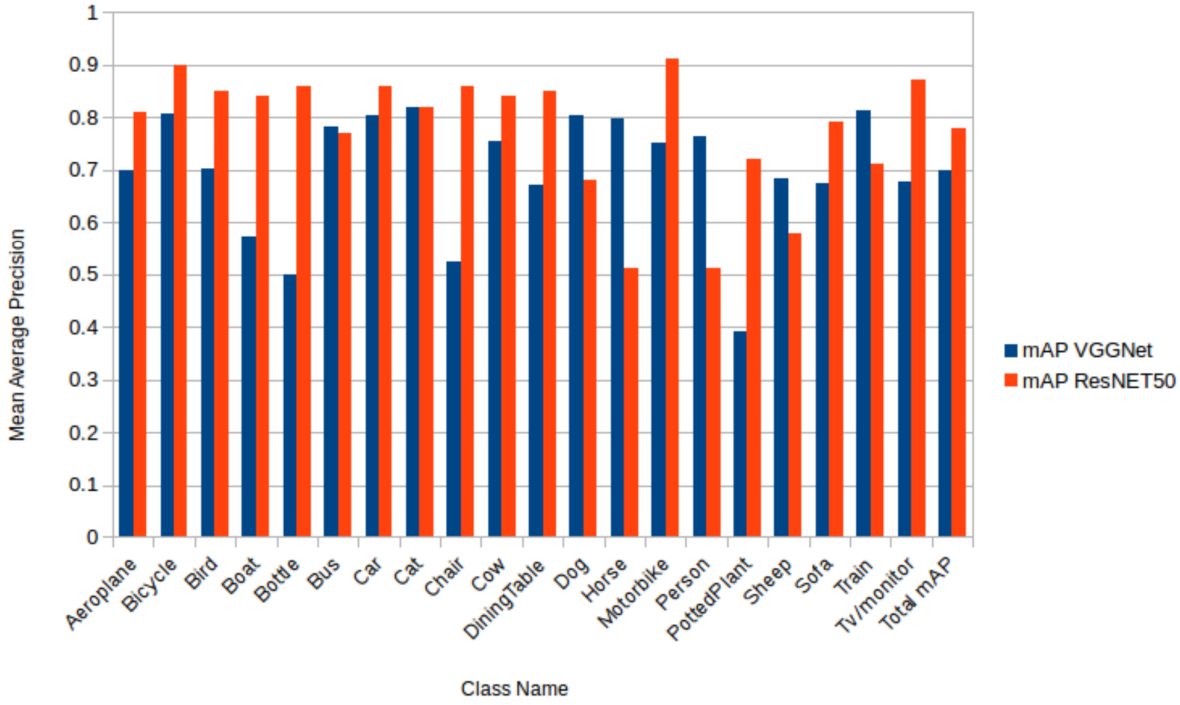


Figure 4.2: Mean Average Precision Comparison between VGG-16 Net and ResNET50

The performance evaluation of the system based on accuracy, precision and recall is shown in table 4.3

Table 4.3 Performance evaluation of object detection using VGG-16 Net and ResNET50

Metrics	VGG Net	ResNET50
Precision	0.69	0.78
Recall	0.72	0.82
F-Mesure	0.70	0.79

The timing comparison of Faster R-CNN method based on ResNET50 is compared with that of VGG Net in table 4.4

Table 4.4: Timing analysis of Faster R-CNN method using ResNET50 and VGG Net

Model	System	Rate
VGG-16	RPN + Fast R-CNN	5 fps
ResNET50	RPN + Fast R-CNN	12 fps

Output from the object detection model in some sample images are shown in figure 4.3.

Figure 4.3: Output from Object Detector

4.2 Objects detection in Videos

For training, validating and testing object detection in videos, ImageNet VID Dataset was used. From this dataset 60000 video frames were used for training, 20000 video frames were used for validation and another 20000 video frames were used for testing the model. The aspect ratio of image was kept unchanged while, image is resized as smaller side of image as 600 pixels.

Faster R-CNN network was implemented with ResNet50 with 50 layers, the training and validation loss for RPN are shown in figure 4.4

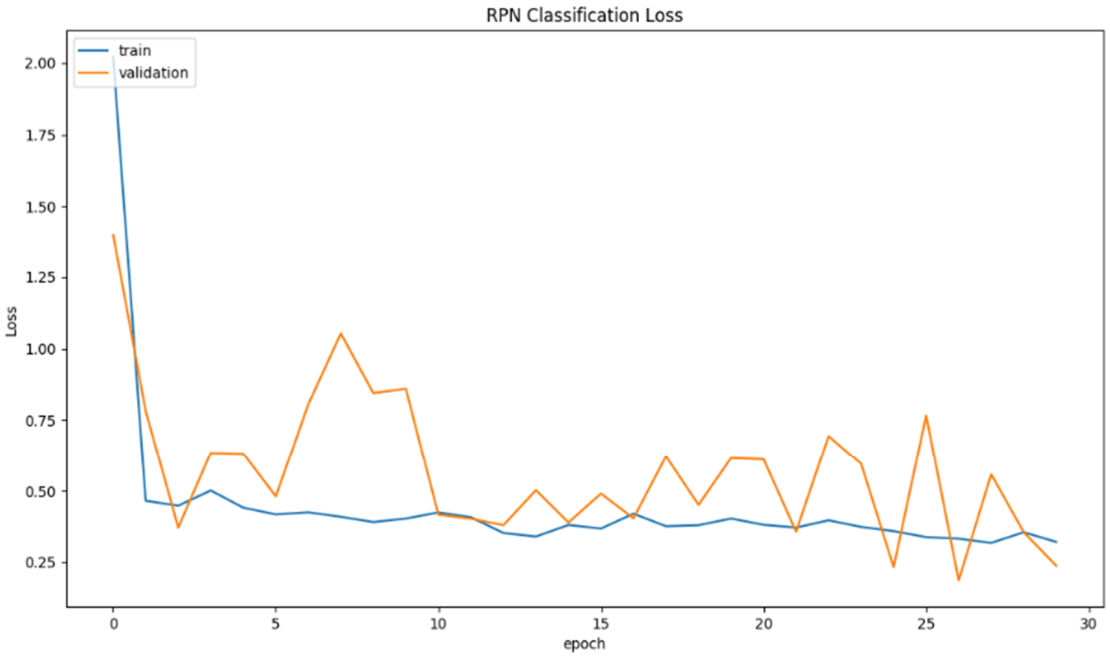


Figure 4.4: RPN Classification Loss of Faster RCNN based on ResNet

Figure 4.4 shows the graph of RPN loss for 30 epochs both training and validation loss are gradually reduced over epochs. At epoch 30, training loss is reduced to 0.34 while validation loss is 0.32. After 30 epochs of training, the results are shown in table 4.5.

Table 4.5: Training results on ImageNet VID video dataset

Metrics	Training	Validation
Classifier accuracy for bounding boxes from RPN	0.68	0.72
Loss RPN classifier	0.32	0.23
Loss RPN regression	0.16	0.15
Loss Detector classifier	1.29	0.39
Loss Detector regression	0.39	0.31

The system was evaluated using annotated ImageNet VID Dataset video frames test samples based on mean average precision metric. The performance of the system is shown in table 4.6 below:

Table 4.6: Test Results of Faster R-CNN based on ResNet on ImageNet VID video dataset

Class ID	Class Name	mAP Faster R-CNN
1	airplane	0.78
2	antelope	0.80
3	Bear	0.50
4	Bicycle	0.56
5	Bird	0.38
6	Bus	0.68
7	Car	0.39
8	cattle	0.60
9	dog	0.30
10	Domestic cat	0.65
11	elephant	0.44
12	Fox	0.94
13	Giant panda	0.66
14	hamster	0.93
15	horse	0.75
16	lion	0.15
17	lizard	0.71
18	monkey	0.04
19	motorcycle	0.69
20	rabbit	0.15
21	Red panda	0.32
22	sheep	0.79
23	snake	0.55
24	squirrel	0.10
25	tiger	0.30
26	train	0.94
27	turtle	0.78
28	watercraft	0.73
29	whale	0.40
30	zebra	0.71
mAP		0.56

The mean average precision for the test samples was found 0.56.

4.3 Refinement in Faster R-CNN with addition of Recurrent Layer (LSTM)

The system was refined with introduction of LSTM layer with 2046 units after average pooling layer. The system with LSTM layer leveraged the temporal information of consecutive video frames. The refined system was again trained, validated and tested on the ImageNet VID Dataset. The training and validation loss for RPN is shown in figure 4.5.

Figure 4.5 shows the RPN classification loss of Faster R-CNN with LSTM layer on ResNet50. The training loss is reduced to 0.30 while validation loss is reduced to 0.26 in 30 epochs.

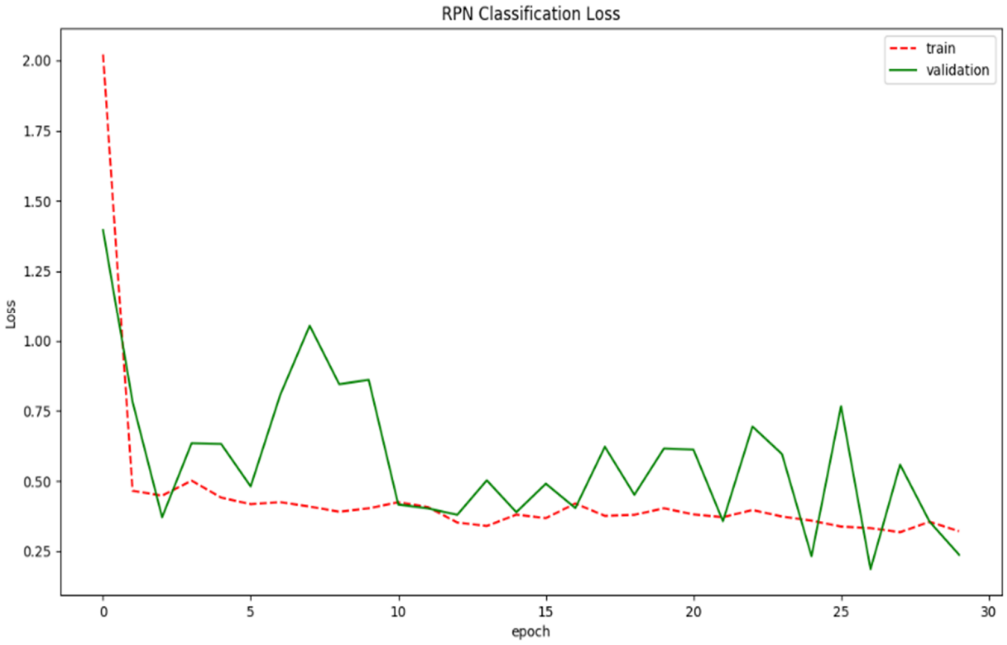


Figure 4.5: RPN Classification Loss of Faster R-CNN with LSTM based on ResNet50

After 30 epochs of training, the results are shown in table 4.7

Table 4.7: Training results on ImageNet VID video dataset with Refined system

Metrics	Training	Validation
Classifier accuracy for bounding boxes from RPN	0.69	0.76
Loss RPN classifier	0.30	0.20
Loss RPN regression	0.15	0.16
Loss Detector classifier	1.30	0.35
Loss Detector regression	0.35	0.29

The refined system was evaluated using annotated ImageNet VID Dataset video frames test samples based on mean average precision metric. The performance of the system is shown in table 4.8 below:

Table 4.8: Test Results of Faster R-CNN with LSTM based on ResNet50 on ImageNet VID

Class ID	Class Name	mAP Faster R-CNN with LSTM
1	airplane	0.82
2	antelope	0.45
3	Bear	0.64
4	Bicycle	0.63
5	Bird	0.49
6	bus	0.72
7	car	0.49
8	cattle	0.70
9	dog	0.48
10	Domestic cat	0.74
11	elephant	0.50
12	fox	0.95
13	Giant panda	0.75
14	hamster	0.94
15	horse	0.86
16	lion	0.38
17	lizard	0.82
18	monkey	0.05
19	motorcycle	0.80
20	rabbit	0.37
21	Red panda	0.36

22	sheep	0.91
23	snake	0.63
24	squirrel	0.43
25	tiger	0.35
26	train	0.96
27	turtle	0.89
28	watercraft	0.84
29	whale	0.45
30	zebra	0.82
mAP		0.64

The refined Faster R-CNN with LSTM layer provided mAP of 0.64 over ImageNet VID Dataset video frames, which is better than the performance of the Faster R-CNN i.e. 0.56

The comparison of mean average precision for original Faster R-CNN and refined Faster R-CNN with LSTM layer is shown in figure 4.6.

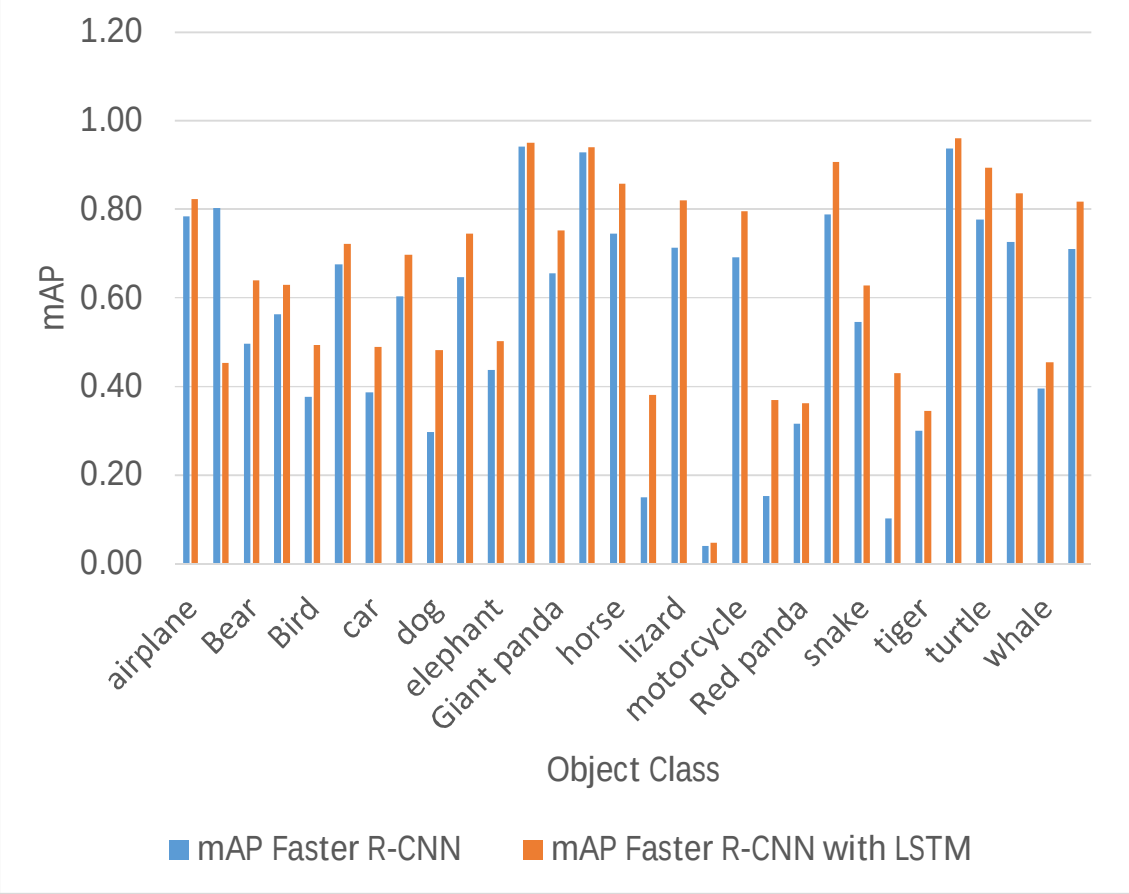


Figure 4.6: Mean average precision of Faster R-CNN and Refined Faster R-CNN

The performance evaluation of the system based on accuracy, precision and recall is shown in table 4.9.

Table 4.9: Performance evaluation of object detection for Faster R-CNN and Faster R-CNN with LSTM

Metrics	Faster R-CNN	Faster R-CNN LSTM
Precision	0.56	0.64
Recall	0.63	0.75
F-Measure	0.59	0.69

Above results showed that Faster R-CNN with LSTM provided better detection results than Faster R-CNN without LSTM.

Detection results of Faster R-CNN on ImageNet VID test set using different settings of anchors was analyzed in table 4.10.

Table 4.10: Detection results using different settings of anchors

Settings	Anchor scales	Aspect ratios	mAP
1 scale, 1 ratio	128x128	1:1	0.591
	256x256	1:1	0.604
1 scale, 3 ratios	128x128	{2:1,1:1,1:2}	0.613
	256x256	{2:1,1:1,1:2}	0.624
3 scales, 3 ratios	{128x128,256x256,512x512}	{2:1,1:1,1:2}	0.640

The default setting of using 3 scales and 3 aspect ratios (0.640 mAP) is the same as that in Table 4.9. Setting of 3 scales and 3 aspect ratios of anchors provided the best results.

The precision recall curve analysis of the system with and without use of LSTM layer in Faster R-CNN method is shown in figure 4.7

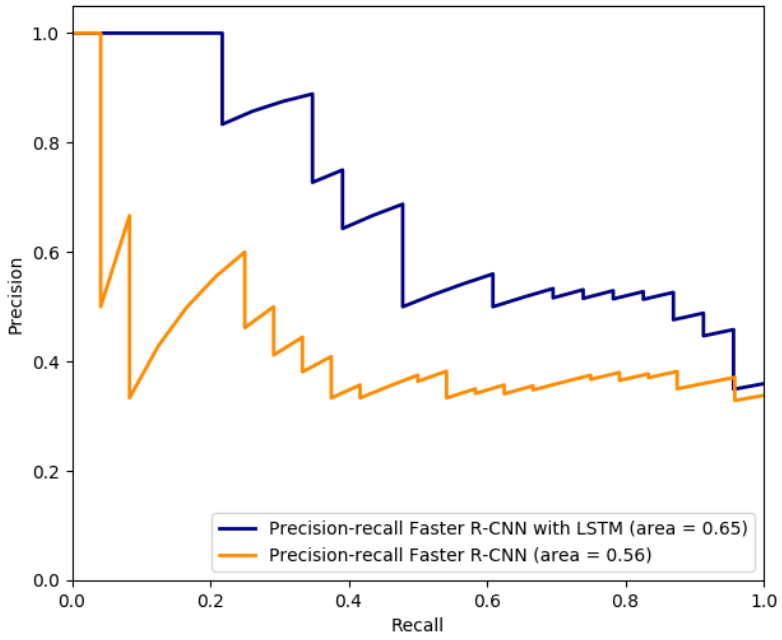


Figure 4.7: Precision Recall curve analysis of Faster R-CNN with and without LSTM

Above curve showed the area under the curve (AUC) for system with LSTM is larger than that of system without LSTM layer. Thus use of LSTM in Faster R-CNN enhanced the performance of the object detection in videos. The analysis of threshold of Intersection-over-union vs recall between ground truth boxes and predicted boxes for considering positive samples while training is shown in table 4.11.

Table 4.11: Detection results for Different IoU Threshold

IoU Threshold	Recall Faster R-CNN	Recall Faster R-CNN with LSTM
0.5	0.63	0.75
0.6	0.53	0.62
0.7	0.48	0.56

The above results showed that the system provided best recall when intersection over union threshold was set to 0.5.

Some of the sample output of the system are visualized in following figures:

Figure 4.8: Sample output from the system on ImageNet VID dataset

The system was also used to evaluate sample video snippets out of the ImageNet Dataset. The results of object detection are satisfactory for that real scenario cluttered video.

Figure 4.9: System evaluation sample output on another video frame

CHAPTER FIVE: CONCLUSION

5.1 Conclusion

This thesis work focuses on finding an efficient method of region based convolution neural network for detecting objects in videos. The method developed here implements deep neural network architecture. Objects in video frames were localized and classified into various categories. The system was trained and evaluated using Imagenet VID Dataset. The system yield mean average precision of 0.64 after refinement with addition of LSTM layer and 0.56 for without refinement in Faster R-CNN method on Imagenet VID Dataset. It is seen that addition of LSTM layer in R-CNN architecture improved the performance of the system for detection of objects.

Also, an efficient method is determined to detect objects in images by designing R-CNN architecture based on Residual Network Architecture. ResNET architecture is modified to implement faster R-CNN with 50 layers. ResNET50 based architecture provided higher precision and timing (frames per second rate) for object detection in PASCAL VOC dataset that original VGG-16 Net of Faster R-CNN method. It is seen that in spite of being a deeper architecture, residual network model is found smaller in model weight size and provided faster frames processing rate than VGG-16 Network model.

It is also seen that while designing architecture of region proposal network to proposal region of interests in an image, anchor boxes of three aspect ratio and size i.e aspect ratios of 1:1, 1:2 and 2:1 & size of 128, 256 and 512 provided highest precision for detection of objects.

It is also seen that for training of region proposal network, the threshold 0.5 for intersection-over-union to consider a sample feature map as positive provided the best results for detection of objects in images/video frames.

5.2 Limitations and Future Enhancements

In this research work, region based CNN method is refined with addition of Recurrent (LSTM) layer. Despite of this, implementation of this method has some limitations. Since, this method implements a deep architecture of CNN, it was very time consuming and resource consuming task. Imagenet VID dataset is very huge dataset with annotated video frames and snippets. Due to constraints on computing resource (GPU, RAM), system was trained and evaluated using a part of Imagenet VID dataset and thus limits the performance and generalization of the detector model.

Further work can be done to improve the performance of the object detector with use of full dataset of Imagenet VID. Performance of the system for detection of occluded objects in videos can also be analyzed. Also, the performance of the system can be analyzed with use of deeper CNN architecture than ResNET50. The architecture used in this method can be further analyzed with varying parameters and using different optimizers than SGD. Also, memory requirements and CPU utilization can be analyzed for system resource optimization.

REFERENCES

[1] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, "Faster R-CNN: Towards RealTime Object Detection with Region Proposal Networks", IEEE Transactions on Pattern Analysis & Machine Intelligence, vol. 39, no. , pp. 1137-1149, June 2017, doi:10.1109/TPAMI.2016.2577031

[2] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv:*
1409.0575, 2014.

[3] Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation.
DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.

[4] LeCun, Yann. LeNet-5, convolutional neural networks. Retrieved 16 November 2013

[5] K. Alex, S. Ilya and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, 2012.

[6] Zeiler, M. D and F. Rob, Visualizing and Understanding Convolutional Networks, arXiv, 2013.

[7] Simonyan, Karen, and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[8] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778. doi: 10.1109/CVPR.2016.90

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014.

[10] R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 1440-1448. doi: 10.1109/ICCV.2015.169

[11] Zhu, Gao, Fatih Porikli, and Hongdong Li. "Tracking randomly moving objects on edge box proposals." arXiv preprint arXiv:1507.08085 (2015).

[12] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A.W. Smeulders. Selective search for object recognition. IJCV, 2013.

[13] Kosuke Mizuno, Yosuke Terachi, Kenta Takagi, Shintaro Izumi, Hiroshi Kawaguchi, and Masahiko Yoshimoto. Architectural study of hog feature extraction processor for real-time object detection. In Signal Processing Systems (SiPS), 2012 IEEE Workshop on, pages 197–202. IEEE, 2012.

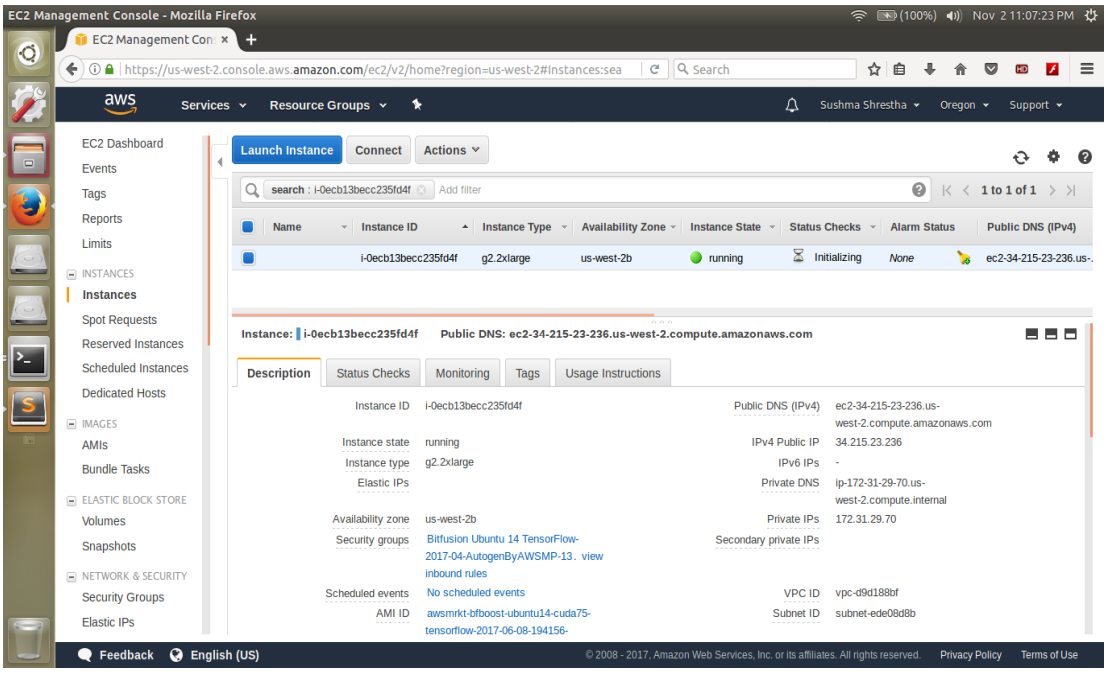
[14] Ramachandran, Prajit. "Object Detection in Video using Faster R-CNN." University of Illinois. 2015

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, 2007.

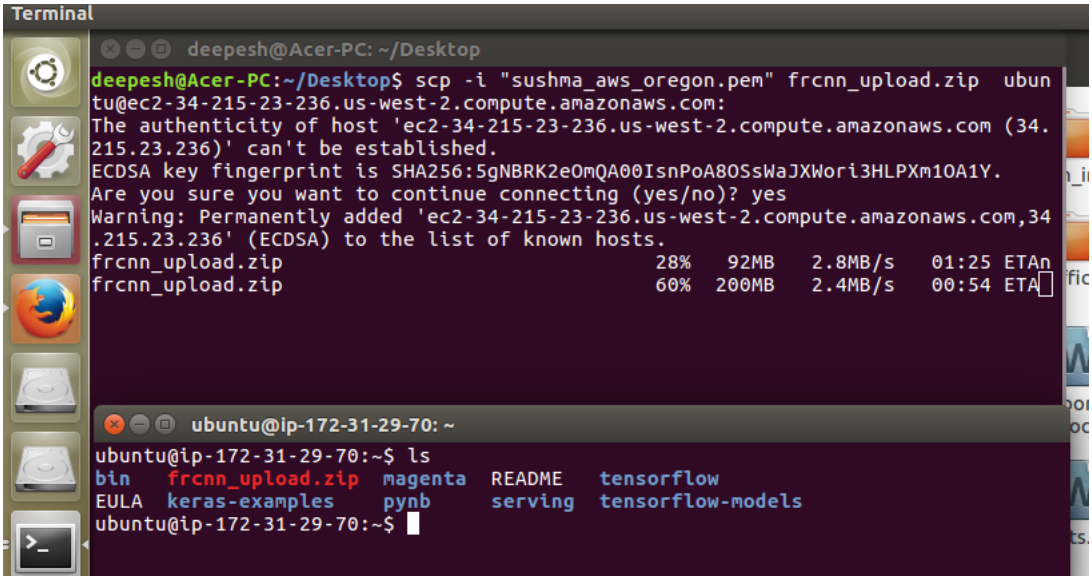
APPENDIX

A. Experimental Setup in Amazon Web Service

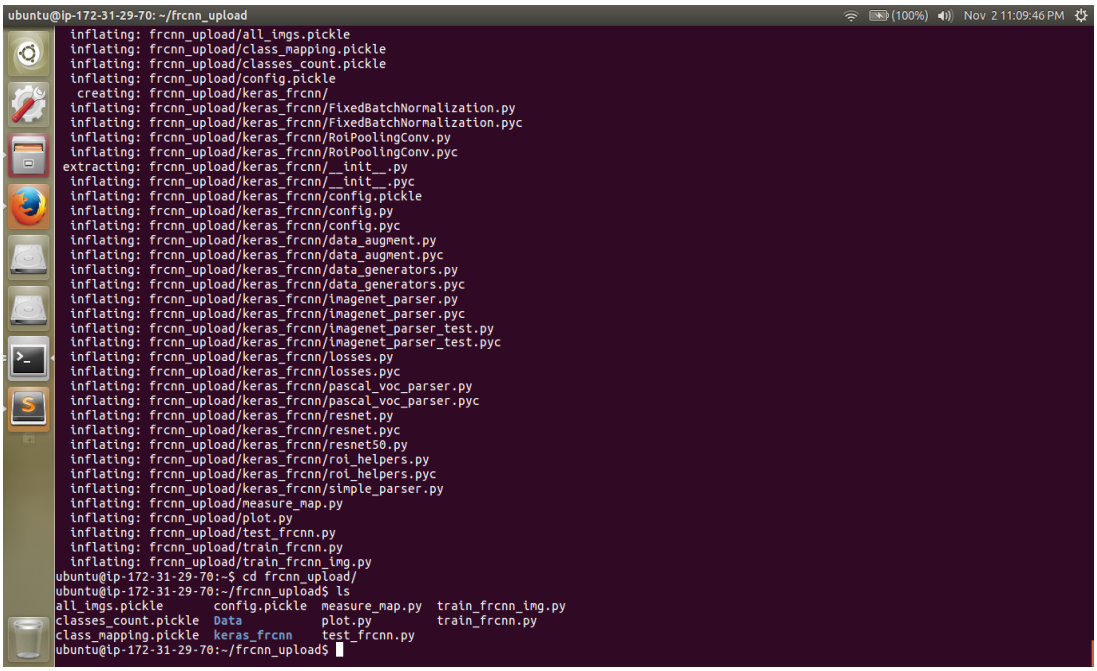
A1. AWS setup



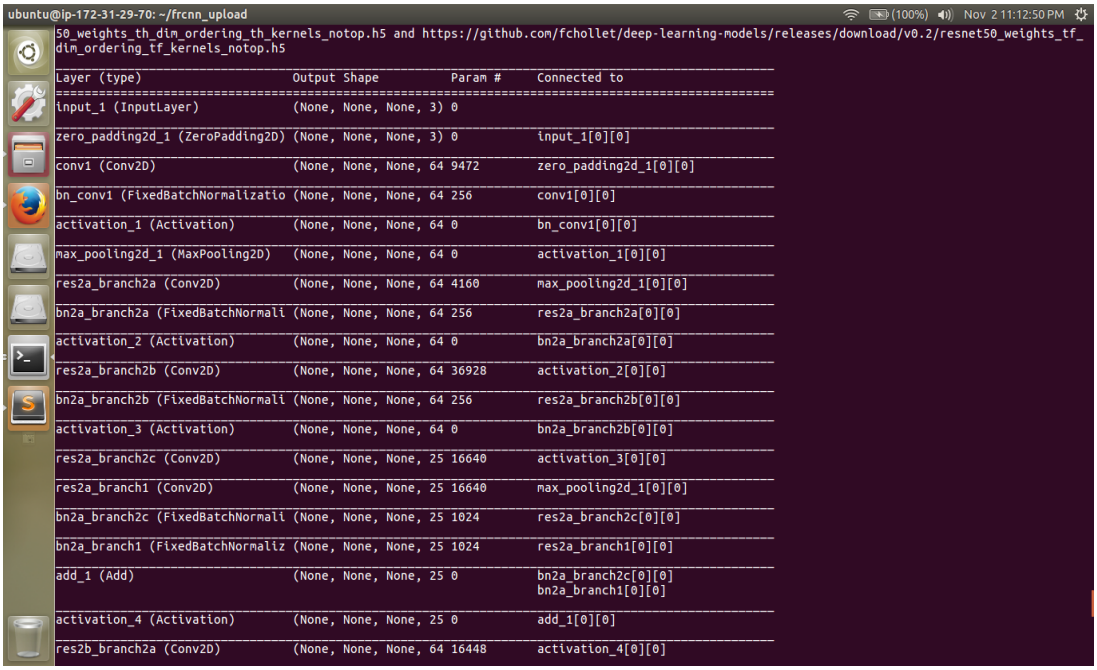
A2. Project Transfer to AWS using SSH



A3. Unzip Project files

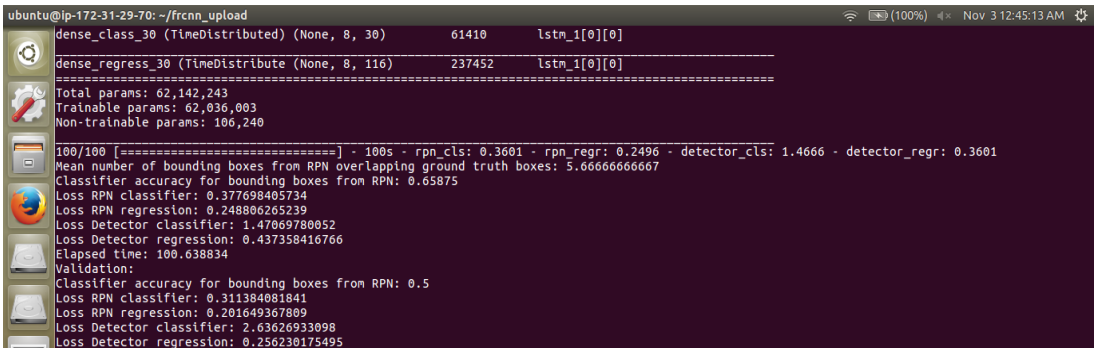


A4. Model Summary



B. Training and Testing

B1. Training output on Terminal



B2. A Portion of Training Output Log

/usr/bin/python2.7 /home/deepesh/Documents/frcnn_aws/train_frcnn_img.py -p /home/deepesh/ -o imagenet
Using TensorFlow backend.
Training images per class:
{'airplane': 17737,
'antelope': 2571,
'bear': 1907,
'bg': 0,
'bicycle': 9544,
'bird': 3927,
'bus': 447,
'car': 11280,
'cattle': 8128,
'dog': 4792,
'domestic_cat': 2211,
'elephant': 260,
'fox': 1069,
'giant_panda': 4451,
'hamster': 1749,
'horse': 2931,
'lion': 1865,
'lizard': 727,
'monkey': 2143,
'motorcycle': 4290,
'rabbit': 450,
'red_panda': 5543,
'sheep': 5524,
'snake': 2656,
'squirrel': 1531,
'tiger': 1478,
'train': 3367,
'turtle': 1868,
'watercraft': 3804,
'whale': 2550,
'zebra': 1695}
Num classes (including bg) = 31

Starting training
Epoch 1/30
2017-09-04 12:40:02.032193: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computations.
2017-09-04 12:40:02.032217: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computations.
2017-09-04 12:40:02.032222: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computations.

2017-09-04 12:40:02.032227: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computations.
2017-09-04 12:40:02.032231: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computations.
2017-09-04 12:40:02.108950: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:901] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2017-09-04 12:40:02.109264: I tensorflow/core/common_runtime/gpu/gpu_device.cc:887] Found device 0 with properties:
name: Nvidia K520 Grid
pciBusID 0000:01:00.0
Total memory: 15.96GiB
Free memory: 15.74GiB
2017-09-04 12:40:02.109277: I tensorflow/core/common_runtime/gpu/gpu_device.cc:908] DMA: 0
2017-09-04 12:40:02.109282: I tensorflow/core/common_runtime/gpu/gpu_device.cc:918] 0:   Y
2017-09-04 12:40:02.109289: I tensorflow/core/common_runtime/gpu/gpu_device.cc:977] Creating TensorFlow device (/gpu:0) -> (device: 0, name: GeForce GT 740M, pci bus id: 0000:01:00.0)
2017-09-04 12:40:03.672683: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.05GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:04.046712: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.05GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:04.446181: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.07GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:05.054214: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.15GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:05.502571: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.15GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:06.280367: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.07GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:06.468686: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.06GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:07.029589: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.05GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:07.180045: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.03GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
2017-09-04 12:40:07.750381: W tensorflow/core/common_runtime/bfc_allocator.cc:217] Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.05GiB. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory is available.
  1/100 [..............................] - ETA: 1645s - rpn_cls: 0.2429 - rpn_regr: 0.1046 - detector_cls: 3.4340 - detector_regr: 0.24292017-09-04 12:40:32.445041: I tensorflow/core/common_runtime/gpu/pool_allocator.cc:247] PoolAllocator: After 2424 get requests, put_count=2209 evicted_count=1000 eviction_rate=0.452694 and unsatisfied allocation rate=0.542492
2017-09-04 12:40:32.445068: I tensorflow/core/common_runtime/gpu/pool_allocator.cc:259] Raising pool_size_limit_ from 100 to 110

  2/100 [..............................] - ETA: 1677s - rpn_cls: 1.0549 - rpn_regr: 0.1513 - detector_cls: 2.9898 - detector_regr: 1.0549
  3/100 [..............................] - ETA: 1334s - rpn_cls: 1.4686 - rpn_regr: 0.1619 - detector_cls: 2.6709 - detector_regr: 1.4686
  4/100 [>.............................] - ETA: 1164s - rpn_cls: 1.8812 - rpn_regr: 0.1989 - detector_cls: 2.3843 - detector_regr: 1.8812
  5/100 [>.............................] - ETA: 1053s - rpn_cls: 2.1525 - rpn_regr: 0.2163 - detector_cls: 2.2320 - detector_regr: 2.1525
  6/100 [>.............................] - ETA: 982s - rpn_cls: 2.3478 - rpn_regr: 0.2219 - detector_cls: 2.0854 - detector_regr: 2.3478
  7/100 [=>............................] - ETA: 974s - rpn_cls: 2.5043 - rpn_regr: 0.2298 - detector_cls: 1.9530 - detector_regr: 2.5043
  8/100 [=>............................] - ETA: 1164s - rpn_cls: 2.6280 - rpn_regr: 0.2529 - detector_cls: 1.8357 - detector_regr: 2.6280
  9/100 [=>............................] - ETA: 1122s - rpn_cls: 2.7532 - rpn_regr: 0.2788 - detector_cls: 1.7319 - detector_regr: 2.7532
 10/100 [==>...........................] - ETA: 1061s - rpn_cls: 2.8639 - rpn_regr: 0.2995 - detector_cls: 1.6398 - detector_regr: 2.8639
 11/100 [==>...........................] - ETA: 1033s - rpn_cls: 2.9226 - rpn_regr: 0.3151 - detector_cls: 1.5578 - detector_regr: 2.9226
 12/100 [==>...........................] - ETA: 976s - rpn_cls: 2.9773 - rpn_regr: 0.3253 - detector_cls: 1.4983 - detector_regr: 2.9773
 13/100 [==>...........................] - ETA: 937s - rpn_cls: 3.0369 - rpn_regr: 0.3380 - detector_cls: 1.4907 - detector_regr: 3.0369
 14/100 [===>..........................] - ETA: 903s - rpn_cls: 3.1024 - rpn_regr: 0.3483 - detector_cls: 1.5027 - detector_regr: 3.1024
 15/100 [===>..........................] - ETA: 871s - rpn_cls: 3.1700 - rpn_regr: 0.3557 - detector_cls: 1.5057 - detector_regr: 3.1700
 16/100 [===>..........................] - ETA: 844s - rpn_cls: 3.2342 - rpn_regr: 0.3608 - detector_cls: 1.5338 - detector_regr: 3.2342
 17/100 [====>.........................] - ETA: 819s - rpn_cls: 3.3064 - rpn_regr: 0.3665 - detector_cls: 1.5518 - detector_regr: 3.3064
 18/100 [====>.........................] - ETA: 796s - rpn_cls: 3.3914 - rpn_regr: 0.3717 - detector_cls: 1.5622 - detector_regr: 3.3914
 19/100 [====>.........................] - ETA: 775s - rpn_cls: 3.4635 - rpn_regr: 0.3761 - detector_cls: 1.5666 - detector_regr: 3.4635
 20/100 [=====>........................] - ETA: 755s - rpn_cls: 3.5266 - rpn_regr: 0.3791 - detector_cls: 1.5665 - detector_regr: 3.5266
 21/100 [=====>........................] - ETA: 786s - rpn_cls: 3.5805 - rpn_regr: 0.3811 - detector_cls: 1.5628 - detector_regr: 3.5805
 22/100 [=====>........................] - ETA: 759s - rpn_cls: 3.6247 - rpn_regr: 0.3827 - detector_cls: 1.5564 - detector_regr: 3.6247
 23/100 [=====>........................] - ETA: 740s - rpn_cls: 3.6680 - rpn_regr: 0.3838 - detector_cls: 1.5479 - detector_regr: 3.6680
 24/100 [======>.......................] - ETA: 722s - rpn_cls: 3.7094 - rpn_regr: 0.3842 - detector_cls: 1.5377 - detector_regr: 3.7094
 25/100 [======>.......................] - ETA: 705s - rpn_cls: 3.7440 - rpn_regr: 0.3843 - detector_cls: 1.5279 - detector_regr: 3.7440
 26/100 [======>.......................] - ETA: 685s - rpn_cls: 3.7705 - rpn_regr: 0.3838 - detector_cls: 1.5169 - detector_regr: 3.7705
 27/100 [=======>......................] - ETA: 684s - rpn_cls: 3.7935 - rpn_regr: 0.3830 - detector_cls: 1.5050 - detector_regr: 3.7935
 28/100 [=======>......................] - ETA: 669s - rpn_cls: 3.8102 - rpn_regr: 0.3822 - detector_cls: 1.4963 - detector_regr: 3.8102
 29/100 [=======>......................] - ETA: 654s - rpn_cls: 3.8250 - rpn_regr: 0.3811 - detector_cls: 1.4867 - detector_regr: 3.8250
 30/100 [========>.....................] - ETA: 640s - rpn_cls: 3.8368 - rpn_regr: 0.3798 - detector_cls: 1.4764 - detector_regr: 3.8368
 31/100 [========>.....................] - ETA: 626s - rpn_cls: 3.8494 - rpn_regr: 0.3786 - detector_cls: 1.4655 - detector_regr: 3.8494
 32/100 [========>.....................] - ETA: 621s - rpn_cls: 3.8630 - rpn_regr: 0.3773 - detector_cls: 1.4591 - detector_regr: 3.8630
 33/100 [========>.....................] - ETA: 607s - rpn_cls: 3.8724 - rpn_regr: 0.3760 - detector_cls: 1.4519 - detector_regr: 3.8724
 34/100 [=========>....................] - ETA: 594s - rpn_cls: 3.8828 - rpn_regr: 0.3744 - detector_cls: 1.4448 - detector_regr: 3.8828
 35/100 [=========>....................] - ETA: 582s - rpn_cls: 3.8927 - rpn_regr: 0.3728 - detector_cls: 1.4370 - detector_regr: 3.8927
 36/100 [=========>....................] - ETA: 569s - rpn_cls: 3.9017 - rpn_regr: 0.3711 - detector_cls: 1.4288 - detector_regr: 3.9017

37/100 [==========>.....] - ETA: 557s - rpn_cls: 3.9088 - rpn_regr: 0.3693 - detector_cls: 1.4203 - detector_regr: 3.9088
38/100 [==========>.....] - ETA: 545s - rpn_cls: 3.9140 - rpn_regr: 0.3675 - detector_cls: 1.4114 - detector_regr: 3.91402017-09-04 12:45:42.083346: I tensorflow/core/common_runtime/gpu/pool_allocator.cc:247] PoolAllocator: After 7464 get requests, put_count=7425 evicted_count=1000 eviction_rate=0.13468 and unsatisfied allocation rate=0.142283
2017-09-04 12:45:42.083371: I tensorflow/core/common_runtime/gpu/pool_allocator.cc:259] Raising pool_size_limit_ from 256 to 281

39/100 [==========>.....] - ETA: 536s - rpn_cls: 3.9185 - rpn_regr: 0.3656 - detector_cls: 1.4046 - detector_regr: 3.9185
40/100 [==========>.....] - ETA: 524s - rpn_cls: 3.9235 - rpn_regr: 0.3637 - detector_cls: 1.4024 - detector_regr: 3.9235
41/100 [==========>.....] - ETA: 511s - rpn_cls: 3.9264 - rpn_regr: 0.3617 - detector_cls: 1.4011 - detector_regr: 3.9264
42/100 [==========>.....] - ETA: 500s - rpn_cls: 3.9272 - rpn_regr: 0.3597 - detector_cls: 1.3999 - detector_regr: 3.9272
43/100 [==========>.....] - ETA: 493s - rpn_cls: 3.9266 - rpn_regr: 0.3577 - detector_cls: 1.3992 - detector_regr: 3.9266
44/100 [==========>.....] - ETA: 487s - rpn_cls: 3.9246 - rpn_regr: 0.3557 - detector_cls: 1.3990 - detector_regr: 3.9246
45/100 [==========>.....] - ETA: 478s - rpn_cls: 3.9207 - rpn_regr: 0.3537 - detector_cls: 1.3982 - detector_regr: 3.9207
46/100 [==========>.....] - ETA: 467s - rpn_cls: 3.9155 - rpn_regr: 0.3517 - detector_cls: 1.3980 - detector_regr: 3.9155
47/100 [==========>.....] - ETA: 457s - rpn_cls: 3.9088 - rpn_regr: 0.3497 - detector_cls: 1.3981 - detector_regr: 3.9088
48/100 [==========>.....] - ETA: 446s - rpn_cls: 3.9012 - rpn_regr: 0.3476 - detector_cls: 1.3983 - detector_regr: 3.9012
49/100 [==========>.....] - ETA: 436s - rpn_cls: 3.8928 - rpn_regr: 0.3456 - detector_cls: 1.3989 - detector_regr: 3.8928
50/100 [==========>.....] - ETA: 425s - rpn_cls: 3.8835 - rpn_regr: 0.3437 - detector_cls: 1.3997 - detector_regr: 3.8835
51/100 [==========>.....] - ETA: 415s - rpn_cls: 3.8735 - rpn_regr: 0.3417 - detector_cls: 1.4009 - detector_regr: 3.8735
52/100 [==========>.....] - ETA: 405s - rpn_cls: 3.8628 - rpn_regr: 0.3398 - detector_cls: 1.4019 - detector_regr: 3.8628
53/100 [==========>.....] - ETA: 395s - rpn_cls: 3.8517 - rpn_regr: 0.3379 - detector_cls: 1.4028 - detector_regr: 3.8517
54/100 [==========>.....] - ETA: 389s - rpn_cls: 3.8399 - rpn_regr: 0.3360 - detector_cls: 1.4039 - detector_regr: 3.8399
55/100 [==========>.....] - ETA: 380s - rpn_cls: 3.8278 - rpn_regr: 0.3342 - detector_cls: 1.4046 - detector_regr: 3.8278
56/100 [==========>.....] - ETA: 371s - rpn_cls: 3.8153 - rpn_regr: 0.3323 - detector_cls: 1.4055 - detector_regr: 3.8153
57/100 [==========>.....] - ETA: 361s - rpn_cls: 3.8029 - rpn_regr: 0.3306 - detector_cls: 1.4064 - detector_regr: 3.8029
58/100 [==========>.....] - ETA: 352s - rpn_cls: 3.7900 - rpn_regr: 0.3288 - detector_cls: 1.4075 - detector_regr: 3.7900
59/100 [==========>.....] - ETA: 341s - rpn_cls: 3.7770 - rpn_regr: 0.3270 - detector_cls: 1.4084 - detector_regr: 3.7770
60/100 [==========>.....] - ETA: 332s - rpn_cls: 3.7637 - rpn_regr: 0.3253 - detector_cls: 1.4090 - detector_regr: 3.7637
61/100 [==========>.....] - ETA: 323s - rpn_cls: 3.7502 - rpn_regr: 0.3236 - detector_cls: 1.4098 - detector_regr: 3.7502
62/100 [==========>.....] - ETA: 314s - rpn_cls: 3.7366 - rpn_regr: 0.3219 - detector_cls: 1.4105 - detector_regr: 3.7366
63/100 [==========>.....] - ETA: 305s - rpn_cls: 3.7229 - rpn_regr: 0.3205 - detector_cls: 1.4108 - detector_regr: 3.7229
64/100 [==========>.....] - ETA: 296s - rpn_cls: 3.7091 - rpn_regr: 0.3192 - detector_cls: 1.4113 - detector_regr: 3.7091
65/100 [==========>.....] - ETA: 287s - rpn_cls: 3.6950 - rpn_regr: 0.3178 - detector_cls: 1.4122 - detector_regr: 3.6950
66/100 [==========>.....] - ETA: 278s - rpn_cls: 3.6808 - rpn_regr: 0.3165 - detector_cls: 1.4127 - detector_regr: 3.6808
67/100 [==========>.....] - ETA: 269s - rpn_cls: 3.6666 - rpn_regr: 0.3154 - detector_cls: 1.4132 - detector_regr: 3.6666
68/100 [==========>.....] - ETA: 260s - rpn_cls: 3.6523 - rpn_regr: 0.3143 - detector_cls: 1.4137 - detector_regr: 3.6523
69/100 [==========>.....] - ETA: 251s - rpn_cls: 3.6380 - rpn_regr: 0.3132 - detector_cls: 1.4142 - detector_regr: 3.6380
70/100 [==========>.....] - ETA: 244s - rpn_cls: 3.6236 - rpn_regr: 0.3121 - detector_cls: 1.4148 - detector_regr: 3.6236
71/100 [==========>.....] - ETA: 236s - rpn_cls: 3.6092 - rpn_regr: 0.3110 - detector_cls: 1.4156 - detector_regr: 3.6092
72/100 [==========>.....] - ETA: 227s - rpn_cls: 3.5947 - rpn_regr: 0.3100 - detector_cls: 1.4165 - detector_regr: 3.5947
73/100 [==========>.....] - ETA: 219s - rpn_cls: 3.5803 - rpn_regr: 0.3089 - detector_cls: 1.4172 - detector_regr: 3.5803
74/100 [==========>.....] - ETA: 211s - rpn_cls: 3.5659 - rpn_regr: 0.3079 - detector_cls: 1.4177 - detector_regr: 3.5659
75/100 [==========>.....] - ETA: 202s - rpn_cls: 3.5517 - rpn_regr: 0.3069 - detector_cls: 1.4179 - detector_regr: 3.5517
76/100 [==========>.....] - ETA: 194s - rpn_cls: 3.5376 - rpn_regr: 0.3058 - detector_cls: 1.4183 - detector_regr: 3.5376
77/100 [==========>.....] - ETA: 185s - rpn_cls: 3.5236 - rpn_regr: 0.3048 - detector_cls: 1.4185 - detector_regr: 3.5236
78/100 [==========>.....] - ETA: 177s - rpn_cls: 3.5096 - rpn_regr: 0.3038 - detector_cls: 1.4184 - detector_regr: 3.5096
79/100 [==========>.....] - ETA: 168s - rpn_cls: 3.4958 - rpn_regr: 0.3028 - detector_cls: 1.4188 - detector_regr: 3.4958
80/100 [==========>.....] - ETA: 160s - rpn_cls: 3.4821 - rpn_regr: 0.3018 - detector_cls: 1.4193 - detector_regr: 3.4821
81/100 [==========>.....] - ETA: 152s - rpn_cls: 3.4683 - rpn_regr: 0.3008 - detector_cls: 1.4198 - detector_regr: 3.4683
82/100 [==========>.....] - ETA: 144s - rpn_cls: 3.4547 - rpn_regr: 0.2998 - detector_cls: 1.4204 - detector_regr: 3.4547Average number of overlapping bounding boxes from RPN = 2.64 for 100 previous iterations

83/100 [==========>.....] - ETA: 135s - rpn_cls: 3.4411 - rpn_regr: 0.2988 - detector_cls: 1.4210 - detector_regr: 3.4411
84/100 [==========>.....] - ETA: 127s - rpn_cls: 3.4276 - rpn_regr: 0.2978 - detector_cls: 1.4218 - detector_regr: 3.4276
85/100 [==========>.....] - ETA: 119s - rpn_cls: 3.4142 - rpn_regr: 0.2968 - detector_cls: 1.4226 - detector_regr: 3.4142
86/100 [==========>.....] - ETA: 111s - rpn_cls: 3.4008 - rpn_regr: 0.2958 - detector_cls: 1.4236 - detector_regr: 3.4008
87/100 [==========>.....] - ETA: 103s - rpn_cls: 3.3875 - rpn_regr: 0.2948 - detector_cls: 1.4246 - detector_regr: 3.3875
88/100 [==========>.....] - ETA: 94s - rpn_cls: 3.3743 - rpn_regr: 0.2939 - detector_cls: 1.4255 - detector_regr: 3.3743
89/100 [==========>.....] - ETA: 86s - rpn_cls: 3.3611 - rpn_regr: 0.2929 - detector_cls: 1.4264 - detector_regr: 3.3611
90/100 [==========>.....] - ETA: 78s - rpn_cls: 3.3479 - rpn_regr: 0.2920 - detector_cls: 1.4274 - detector_regr: 3.3479
91/100 [==========>.....] - ETA: 70s - rpn_cls: 3.3349 - rpn_regr: 0.2910 - detector_cls: 1.4285 - detector_regr: 3.3349
92/100 [==========>.....] - ETA: 62s - rpn_cls: 3.3220 - rpn_regr: 0.2901 - detector_cls: 1.4297 - detector_regr: 3.3220
93/100 [==========>.....] - ETA: 54s - rpn_cls: 3.3092 - rpn_regr: 0.2892 - detector_cls: 1.4307 - detector_regr: 3.3092
94/100 [==========>.....] - ETA: 47s - rpn_cls: 3.2966 - rpn_regr: 0.2882 - detector_cls: 1.4316 - detector_regr: 3.2966
95/100 [==========>.....] - ETA: 39s - rpn_cls: 3.2840 - rpn_regr: 0.2873 - detector_cls: 1.4325 - detector_regr: 3.2840
96/100 [==========>.....] - ETA: 31s - rpn_cls: 3.2716 - rpn_regr: 0.2864 - detector_cls: 1.4333 - detector_regr: 3.2716
97/100 [==========>.....] - ETA: 23s - rpn_cls: 3.2592 - rpn_regr: 0.2855 - detector_cls: 1.4343 - detector_regr: 3.2592
98/100 [==========>.....] - ETA: 15s - rpn_cls: 3.2469 - rpn_regr: 0.2846 - detector_cls: 1.4350 - detector_regr: 3.2469
99/100 [==========>.....] - ETA: 7s - rpn_cls: 3.2347 - rpn_regr: 0.2837 - detector_cls: 1.4358 - detector_regr: 3.2347
100/100 [==============================] - 780s - rpn_cls: 3.2226 - rpn_regr: 0.2828 - detector_cls: 1.4366 - detector_regr: 3.2226
Mean number of bounding boxes from RPN overlapping ground truth boxes: 3.58823529412
Classifier accuracy for bounding boxes from RPN: 0.79625
Loss RPN classifier: 2.02161489191
Loss RPN regression: 0.194934143592
Loss Detector classifier: 1.51350333355

Loss Detector regression: 1.9001730258
Elapsed time: 785.089758873
Validation:
Classifier accuracy for bounding boxes from RPN: 0.875
Loss RPN classifier: 1.39686632156
Loss RPN regression: 1.29178774357
Loss Detector classifier: 0.865628838539
Loss Detector regression: 0.413135677576
Total loss decreased from inf to 5.63022539486, saving weights
Epoch 2/30
……

C. Output Samples

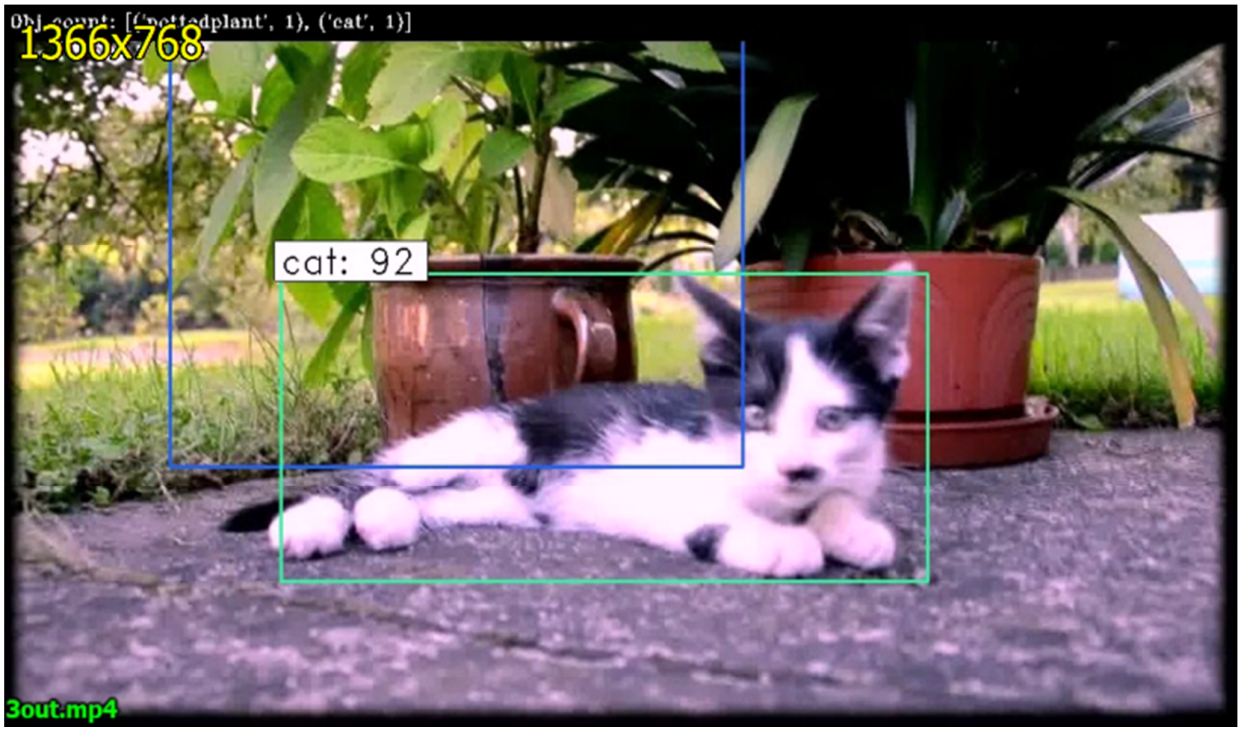
C1. Output from object detector in images

C2. Output from object detector in Videos

i. Object Aeroplane

ii. Object Car

iii. Object Cat



iv. Class Person and Motorbike