Minor Project Report

on

TEXT DETECTION FROM SIGNBOARDS IN VERNACULAR LANGUAGES

Submitted in partial fulfilment of the requirement for the degree of

Master of Technology

In

Computer Science and Engineering

Under the Supervision of

Dr. Nupur Prakash

(Professor)

USICT, GGSIPU

Submitted by

Megha Gupta

(Enrollment No.: 00716404819)

University School of Information, Communication & Technology

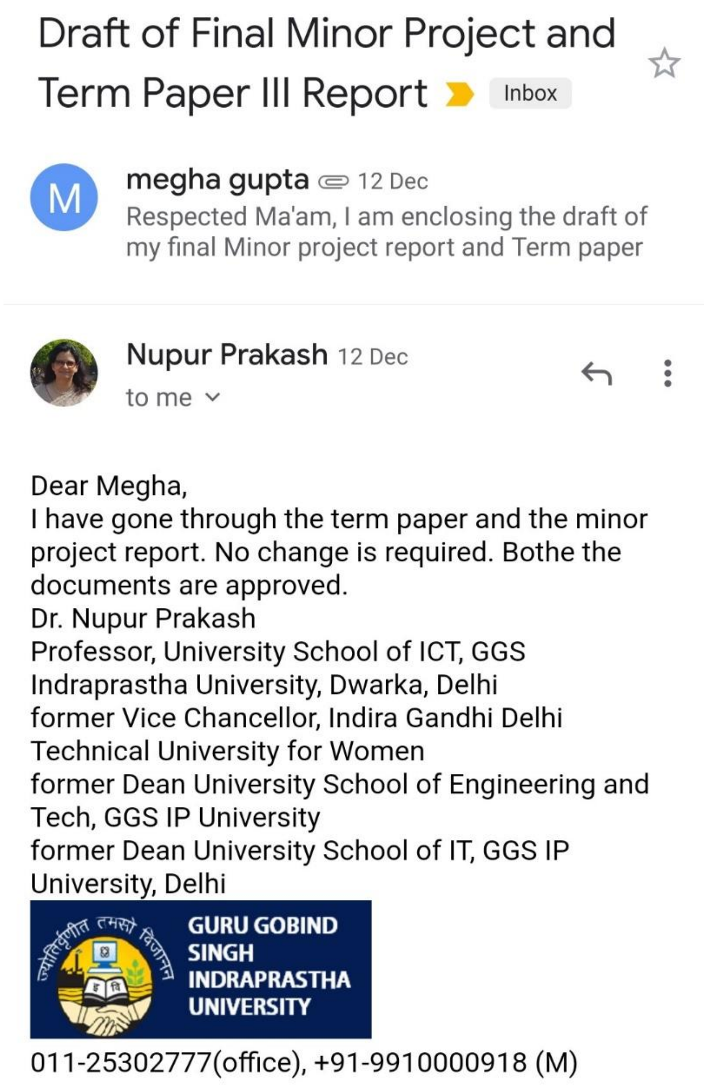
Guru Gobind Singh Indraprastha University

New Delhi – 110078

December, 2020

CERTIFICATE

This is to certify that this Minor Project entitled "**Text Detection from Signboards in Vernacular Languages**" which is submitted by **Megha Gupta (00716404819)** in partial fulfilment of the requirement for award of degree **M. Tech. in Computer Science & Engineering** to USICT, GGSIPU, New Delhi is a record of candidate's own work carried out by her under my supervision.



Draft of Final Minor Project and Term Paper III Report

megha gupta 12 Dec
Respected Ma'am, I am enclosing the draft of my final Minor project report and Term paper

Nupur Prakash 12 Dec
to me

Dear Megha,
I have gone through the term paper and the minor project report. No change is required. Bothe the documents are approved.
Dr. Nupur Prakash
Professor, University School of ICT, GGS Indraprastha University, Dwarka, Delhi
former Vice Chancellor, Indira Gandhi Delhi Technical University for Women
former Dean University School of Engineering and Tech, GGS IP University
former Dean University School of IT, GGS IP University, Delhi

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

011-25302777(office), +91-9910000918 (M)

ACKNOWLEDGEMENT

I would like to express my sincere thanks to those who have contributed significantly to this report. It is a pleasure to extend the deep gratitude to my guide **Prof. Nupur Prakash** for her valuable guidance and support to continuously promote me for the progress of the report. I hereby present my sincere thanks to her for valuable suggestions towards my report, which helped me in making this report more efficient and user friendly.

I would like to thank all faculty members and staff of the Department of Computer Science and Engineering, USICT, GGSIPU, Dwarka for their extreme help throughout course. An Assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in the reference section. I acknowledge my indebtedness to all of them.

Megha Gupta
(00716404819)

ABSTRACT

India has 122 major languages with 22 scheduled languages. A traveller, traveling in an area speaking a language not known to him/her gets frustrated by not being able to understand the text on various signboards. It is impractical to have signboards in different languages at every location. Hence, a system can be built to help the visitors to understand and comprehend the signboards in their own languages. The first and foremost task for developing such a system is to detect the text written on the signboard using object detection. In this project, an object detection engine to detect text objects in the given image and isolating the region containing the text enclosed in a bounding box has been implemented. Dataset for this project has been taken from ai4bharat.org. The dataset contains more than 100k images along with their annotations. YOLOv3, one of the state-of-the-art deep learning based object detector is used to detect the text objects. All the configurations required for YOLOv3 are made in Google Colaboratory. Darknet-53 is cloned from GitHub Repository which is used by YOLOv3 as backbone network. The model predicts bounding boxes and class probabilities for those boxes with mean average precision (mAP) of 95.01%. The detection time is 89.65 milliseconds. Code for the text detection is available at:
https://colab.research.google.com/drive/1BtgTdURTKPiZnulqaUMqqJb5U223nXxe?usp=sharing.

TABLE OF CONTENTS

<u>**LIST OF FIGURES**</u>

1: PROBLEM STATEMENT

In order to perform signboard translation from vernacular languages, the first goal is to detect the text on the signboard. The objective of this project is to develop a system that can detect the message (written in vernacular language) on a signboard.

2: INTRODUCTION

India has 122 major languages out of which 22 are scheduled languages in the constitution. Hindi is the most commonly spoken language in India, followed by Bengali and Marathi languages. An average traveller, traveling in an area speaking a language not known to him/her get frustrated by not being able to understand the text on various signboards. This can spoil the experience of visiting a new place and can lead to confusion in navigation and exploration of a new place. People of one region get a wrong signal that the people of other regions are not thoughtful enough to display signboards in their language and vice versa. In reality, it is just impractical to have every signboard in every location in every other language. It may be possible to put signboards in 2-3 languages only. Hence, a system can be made to help the visitors/tourists to understand and comprehend the signboards in their own languages. The first and foremost task for developing this system is to detect the text written on the signboard using object detection. The second task is to recognize individual characters of the detected text using a CNN-RNN encoder decoder model. Therefore, deep learning methods in two popular fields, computer vision and NLP can be applied to find the solution. For this, the dataset needed should contain many images with some text written in different languages. Additionally, annotations in each image will be done as follows:

(a) A box enclosing the region incorporating text in the image

(b) The text that is inscribed in this box.

For object detection, i.e., identifying region containing text in the image, training will be done with the help of the first set of annotations.

3: LITERATURE REVIEW

To perform text detection task, many methods have been developed and used such as:

- Fully Convolutional Networks [1]

  The proposed methodology is divided into the following components:

  - Text Block Detection: The network architecture of the Text-Block FCN inherits 5 convolutional stages from VGG 16-layer model. Each convolutional stage is followed by a deconvolutional layer to generate feature maps of the same size. At the end, to make pixel-level prediction, the fully-connected layers are replaced with a 1×1 convolutional layer and a sigmoid layer. In the training phase, pixels within the bounding box of each text line or word are considered as the positive region. The cross-entropy loss function and stochastic gradient descent are used to train the model.

  - Multi-Oriented Text Line Candidate Generation: Based on text blocks, multi-oriented text line candidates are formed. The character components are extracted present within the text blocks. Then, by component projection the orientation of the text lines is estimated. Finally, text line candidates are extracted (minimum bounding box).
  - Text Line Candidates Classification: Character-Centroid FCN (a small version of Text-Block FCN) is used to remove false candidates.

- EAST: An Efficient and Accurate Scene Text Detector

  It was proposed by X. Zhou et al. [2], in which an image is fed into the FCN and multiple channels of pixel-level text score map and geometry are generated. The pixel values of the score map are in the range of [0, 1] for one of the predicted channels. The remaining channels represent geometries that encloses the word from the view of each pixel.

- Faster R-CNN (Faster Region based CNN)

  It was proposed by S. Ren et al. [3], as an improved version of Fast R-CNN. They introduced a Region Proposal Network (RPN) which is a FCN that simultaneously predicts object bounds and objectness scores. The RPN generates high-quality region proposals, which are used by fast R-CNN for detection. Therefore, Faster R-CNN consists of two modules. The first module comprises of a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector [4] that uses the proposed regions.

- YOLO (You Only Look Once)

It was developed by Redmon et al. [5] Yolo is a single convolutional network that simultaneously predicts multiple bounding boxes and class probabilities for those boxes. It processes images in real-time at 45 frames per second. The detection is modeled as a regression problem. 416 × 416 images are used for training. The input image is divided into an $S \times S$ grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU between the predicted box and any ground truth box (P(Object) * IOU_{pred}^{truth}). Each grid cell also predicts C conditional class probabilities (P(Class_i|Object)). These probabilities are conditioned on the grid cell containing an object. Only one set of class probabilities per grid cell is predicted, regardless of the number of boxes B. At test time, the conditional class probabilities and the individual box confidence predictions are multiplied.

$$P(\text{Class}_i|\text{Object}) * P(\text{Object}) * IOU_{pred}^{truth} = P(\text{Class}_i) * IOU_{pred}^{truth}$$

This gives the class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

The network architecture has 24 convolutional layers followed by 2 fully connected layers. The network is pre-trained on the ImageNet classification task. A linear activation function is used for the final layer and the leaky rectified linear activation is used for all other layers.

YOLO imposes strong spatial constraints on bounding box predictions. Since the model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations.

- YOLOv2 [6]

  Improved version of Yolov1. Some of the improvements made are:

  - All fully connected layers are removed and anchor boxes are used to predict bounding boxes.
  - Darknet-19 is used as backbone network. It contains 19 convolutional layers.
  - Batch normalization is used on all convolutional layers.
  - 416 × 416 images are used for training.

    - One pooling layer is removed to increase the resolution of output.
    - Uses logistic activation σ to bound the location.

- YOLOv3 [7]

  Improved version of Yolov2. Instead of using Darknet-19 as backbone network, Darknet-53 is used. Darknet-53 consists of 53 convolutional layers, thus increasing the accuracy.

- Curve Text Detector

  Y. Liu et al. [8] constructed a curve text dataset named CTW1500, which includes over 10k text annotations in 1,500 images (1000 for training and 500 for testing). Based on this dataset, they proposed a simple but effective polygon based curve text detector (CTD) which can directly detect curve text without empirical combination.

- Single Shot Text Detector with Regional Attention

  P. He et al. [9] proposed a single-shot text detector that directly outputs word-level bounding boxes in a natural image. The detector is composed of three main parts: a convolutional component, a text-specific component, and a box prediction component. The convolutional and box prediction components mainly inherit from SSD detector. They proposed the text-specific component which contains two modules: a text attention module and a hierarchical inception module.

- Connectionist Text Proposal Network to detect text in natural image

  Z. Tian et al. [10] proposed a novel connectionist text proposal network (CTPN) that accurately localizes text lines in natural image. It includes three key modules: detecting text in fine-scale proposals, recurrent connectionist text proposals, and side-refinement. The CTPN is a fully convolutional network. It allows an input image of arbitrary size. It detects a text line by sliding a small window in the convolutional feature maps and outputs a sequence of fine-scale text proposals.

4: METHODOLOGY

The problem can be implemented as follows:

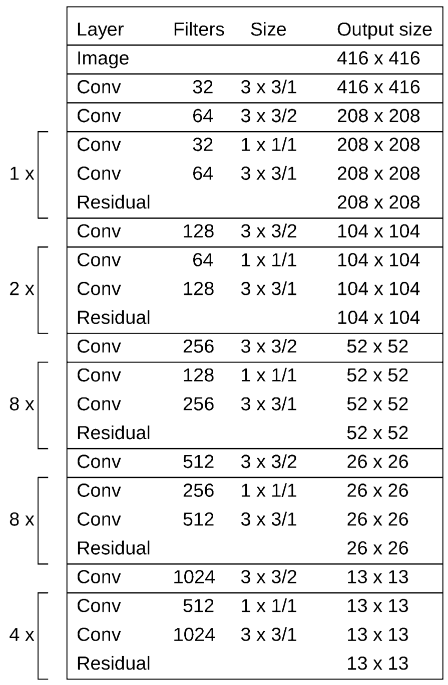
- Text region detection: An object detection engine to detect signboard objects in the given image and isolating the region containing the text enclosed in a bounding box.

Text region detection is implemented using **YOLOv3 model** [7].

It is an improved version of Yolov2. The backbone network used is Darknet-53, which has 53 convolutional layers. The whole system can be divided into two major components: Feature Extractor and Detector. Darknet-53 is used as feature extractor. These features are fed into detector to get bounding boxes and class information. The input image is divided into *S* x *S* grid. Each grid cell predicts three bounding boxes.

Darknet-53

It is mainly composed of a series of convolution layers at dimensions of 1 x 1 and 3 x 3, with a total of 53 layers (including the last fully connected layer but excluding the residual layer). Each convolution layer is followed by a batch normalization (BN) layer and LeakyReLU layer. A number of residual network modules are introduced i.e., the residual layer, which is derived from ResNet. The purpose of adding the residual layer is to solve the gradient disappearance or gradient explosion problem in the network, such that the propagation of gradient can be done in more controlled manner.



	Layer	Filters	Size	Output size
	Image			416 x 416
	Conv	32	3 x 3/1	416 x 416
	Conv	64	3 x 3/2	208 x 208
1 x	Conv	32	1 x 1/1	208 x 208
	Conv	64	3 x 3/1	208 x 208
	Residual			208 x 208
	Conv	128	3 x 3/2	104 x 104
2 x	Conv	64	1 x 1/1	104 x 104
	Conv	128	3 x 3/1	104 x 104
	Residual			104 x 104
	Conv	256	3 x 3/2	52 x 52
8 x	Conv	128	1 x 1/1	52 x 52
	Conv	256	3 x 3/1	52 x 52
	Residual			52 x 52
	Conv	512	3 x 3/2	26 x 26
8 x	Conv	256	1 x 1/1	26 x 26
	Conv	512	3 x 3/1	26 x 26
	Residual			26 x 26
	Conv	1024	3 x 3/2	13 x 13
4 x	Conv	512	1 x 1/1	13 x 13
	Conv	1024	3 x 3/1	13 x 13
	Residual			13 x 13

Figure 1: Structure of Darknet-53 network

A detection head is appended to this feature extractor. Features from last three residual blocks are all passed to the detector (consists of 21 convolutional layers) to get bounding boxes at three different scales. Nine anchor boxes are used per grid cell (Three for each scale). For each anchor box, three things are predicted:

1. The location offset against the anchor box: tx, ty, tw, th.
2. The objectness score to indicate if box contains an object.
3. The class probabilities.

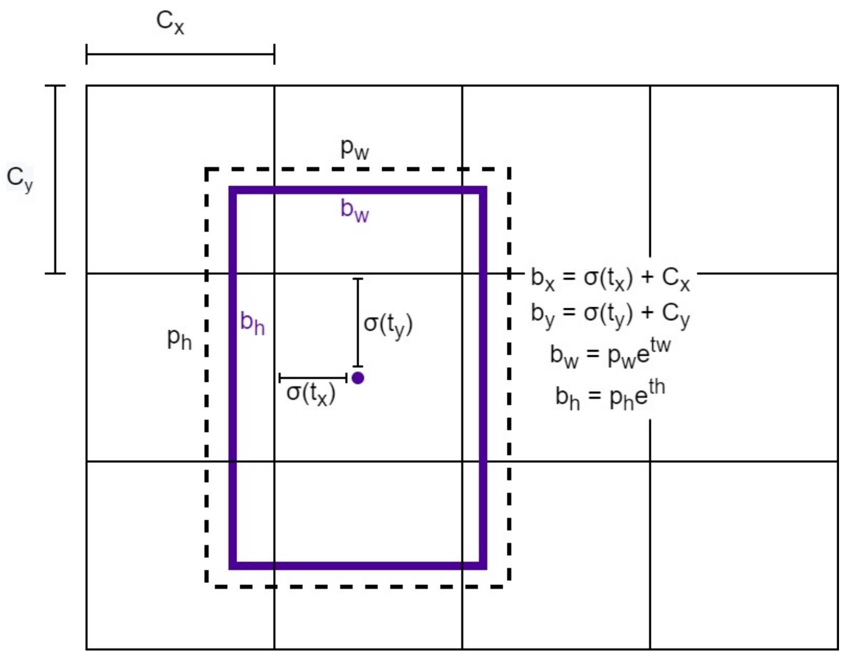


Figure 2: Bounding box location prediction

(c_x, c_y) represents the absolute location of the top-left corner of the current grid cell. The network predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h. b_w and b_h are the absolute width and height to the whole image. p_w and p_h are the width and height of the anchor box. Since it is a regression problem, sum of squared error loss function is used.

5: IMPLEMENTATION

Dataset

Dataset is taken from ai4bharat.org (Link: https://drive.google.com/open?id=1E5kI8CLoC-XffqQMTWwSpBIPp1Wb2tne) [12] which consists of 1,16,132 images along with their annotations. Each image contains some Hindi language words. The dataset is divided into 25 parts. Each part consists of two folders 1. Image 2. Annotation. The Images and Annotations are distributed among 25 folders, which have serial no. 1 to 25. Annotation for a particular image can be found in corresponding serial no.

Understanding Annotation file.

Each line in the annotation file represents a word bounding box. Suppose there are four lines of text in a file, then there will be 4-word bounding boxes in the image. The last word of each line represents ground truth. The first four values of line represent x1, x2, x3, x4, which are x-coordinates in clockwise order. x1 represent top-left, x2 represent top-right, x3 represent bottom-right, x4 represent bottom left. Similarly, after the first four, next four represents y1, y2, y3, y4, which are y-coordinates in clockwise order. y1 represent top-left, y2 represent top-right, y3 represent bottom-right, y4 represent bottom left. A line can be seen as x1_x2_x3_x4_y1_y2_y3_y4_groundTruth, where '_' represents single space.

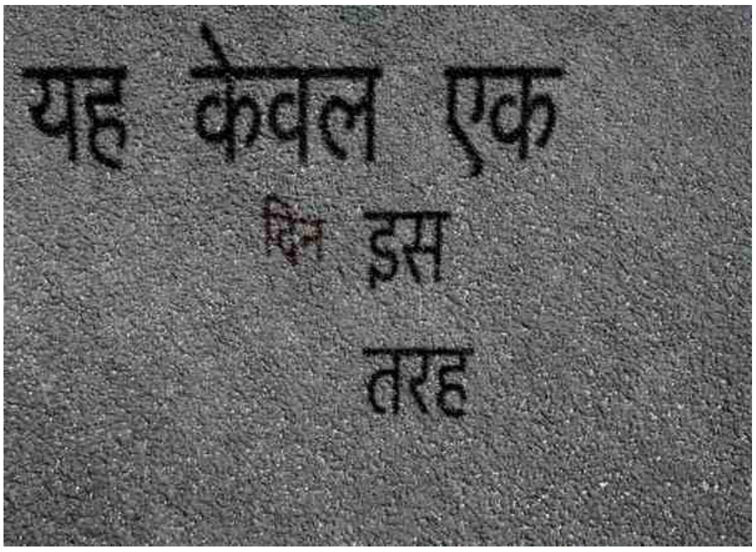


Figure 3: Example image in the dataset

Example of an annotation file is.

```
206.34122 258.02145 258.1178 206.43756 151.50044 151.41408 209.05013 209.13649 दिन
15.015945 103.20639 103.33369 15.143242 49.214314 49.079666 132.45021 132.58485 यह
147.70013 303.9722 304.1776 147.90555 12.001665 11.717926 124.84769 125.131424 केवल
350.8119 449.03406 449.10852 350.88635 48.632027 48.551636 139.54865 139.62903 एक
289.1196 359.50034 359.5085 289.12778 165.57788 165.5686 227.73221 227.74149 इस
288.36942 378.91788 378.88058 288.33212 275.58444 275.64536 331.0419 330.981 तरह
```

For yolo, each line of the annotation file should have following format:

<class> <x_center> <y_center> <width> <height>

Where, <class> = integer number from 0 to (classes - 1)

<x_center> <y_center> <width> <height> - float values relative to width and height of the image. Example: <x_center> = <absolute_x>/<image_width> and <y_center> = <absolute_y>/<image_height>, <width> = <adsolute_width>/<image_width>, <height> = <adsolute_height>/<image_height> [15]

Using python script, the annotations were changed to the above format as needed by yolo.

```
0 0.38704918333333333 0.4153808410138249 0.08629429999999999 0.13300094470046075

0 0.09862469583333333 0.2092909170506912 0.14719624166666667 0.19240825806451609

0 0.37656477499999996 0.15766054147465436 0.26079578333333333 0.2613214239631336

0 0.66660035 0.21679800230414747 0.16382770000000002 0.2098557465437788

0 0.5405234166666667 0.45312222350230413 0.11731483333333334 0.14325550691244238

0 0.5560416666666667 0.698878271889401 0.15097626666666655 0.12778216589861757
```

The annotation files and the images files should be in same directory. The annotation file name should be of same name as the image file name with .txt extension.

Configuring YOLOv3 in Google Colab

1. Installing cuDNN

(Download file - cudnn-10.0-linux-x64-v7.5.0.56.tgz from Nvidia website)

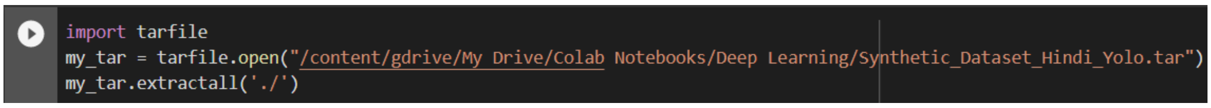
The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN accelerates widely used deep learning frameworks, including Caffe2, Chainer, Keras, MATLAB, MxNet, PyTorch, and TensorFlow.

2. Installing darknet

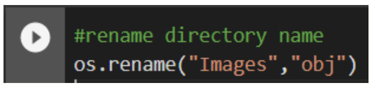
Darknet is cloned from GitHub repository (https://github.com/kriyeng/darknet/ [16]) and compiled.

3. Preparing data and configuration files

Dataset is loaded into the session. Dataset is in the form of tar file. The tar file is extracted.



```
import tarfile
my_tar = tarfile.open("/content/gdrive/My Drive/Colab Notebooks/Deep Learning/Synthetic_Dataset_Hindi_Yolo.tar")
my_tar.extractall('./')
```



```
#rename directory name
os.rename("Images","obj")
```

Figure 4: Dataset Loaded in Google Colab

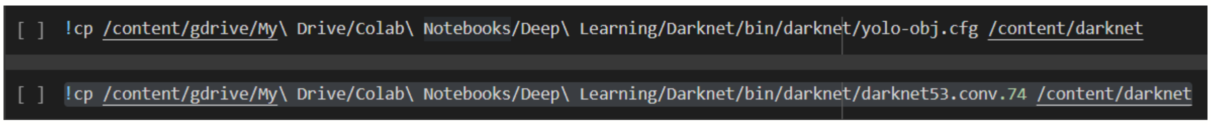
The weights file (darknet53.conv.74) [13] is downloaded and loaded in the darknet directory. The default configuration file (yolov3.cfg) present in the darknet folder is downloaded and changes are made according to the custom dataset. The changes made are:

```
batch=64
subdivisions=16
learning_rate=0.01
```

In the yolo layer, classes variable is set to 1 and in the layer above it filters variable is set to 18 ((classes + 5) * 3). This is done at 3 places.

```
[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear

[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,
156,198,  373,326
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

The modified configuration file (yolo-obj.cfg) is then loaded into the darknet directory.



```
!cp /content/gdrive/My\ Drive/Colab\ Notebooks/Deep\ Learning/Darknet/bin/darknet/yolo-obj.cfg /content/darknet

!cp /content/gdrive/My\ Drive/Colab\ Notebooks/Deep\ Learning/Darknet/bin/darknet/darknet53.conv.74 /content/darknet
```

Figure 5: Weights file and Configuration file loaded in google colab

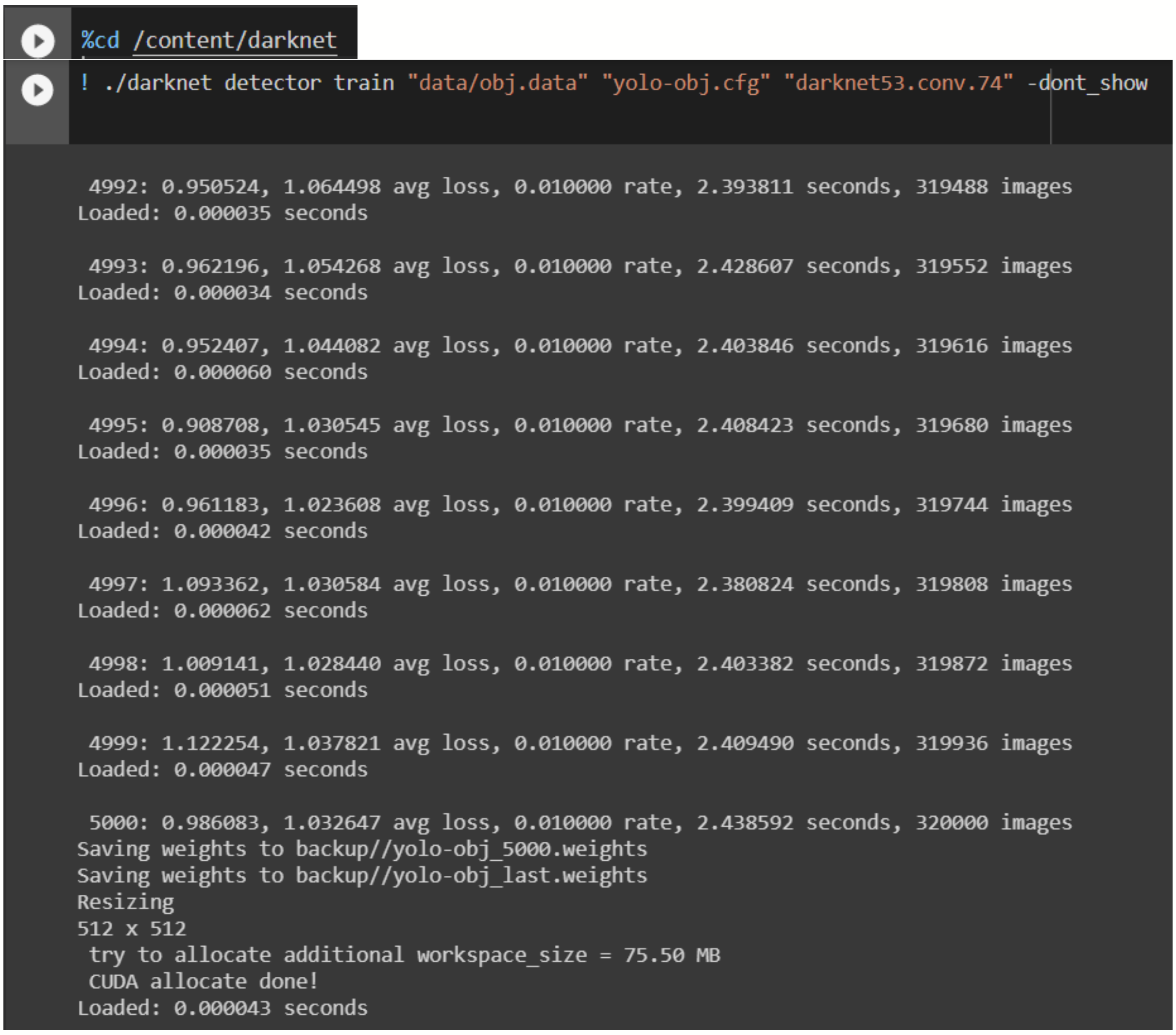
Yolo needs four files – train.txt, test.txt, obj.names, obj.data. The file train.txt consists of the paths of all the images used for training the model and the file test.txt consists of the paths of all the images for testing the model. The dataset is divided into two parts. 95 percent of the dataset is used for training the model and 5 percent is used for testing the model.

In obj.names file the names of all the classes are written in separate lines. For this dataset the file consists of only one line i.e., HINDI. The file obj.data consists of the following:

```
classes = 1
train = data/train.txt
valid = data/test.txt
names = data/obj.names
backup = backup/
```

4. Training the yolov3 model [14]

The training is done using the line:



```
%cd /content/darknet
! ./darknet detector train "data/obj.data" "yolo-obj.cfg" "darknet53.conv.74" -dont_show

 4992: 0.950524, 1.064498 avg loss, 0.010000 rate, 2.393811 seconds, 319488 images
Loaded: 0.000035 seconds

 4993: 0.962196, 1.054268 avg loss, 0.010000 rate, 2.428607 seconds, 319552 images
Loaded: 0.000034 seconds

 4994: 0.952407, 1.044082 avg loss, 0.010000 rate, 2.403846 seconds, 319616 images
Loaded: 0.000060 seconds

 4995: 0.908708, 1.030545 avg loss, 0.010000 rate, 2.408423 seconds, 319680 images
Loaded: 0.000035 seconds

 4996: 0.961183, 1.023608 avg loss, 0.010000 rate, 2.399409 seconds, 319744 images
Loaded: 0.000042 seconds

 4997: 1.093362, 1.030584 avg loss, 0.010000 rate, 2.380824 seconds, 319808 images
Loaded: 0.000062 seconds

 4998: 1.009141, 1.028440 avg loss, 0.010000 rate, 2.403382 seconds, 319872 images
Loaded: 0.000051 seconds

 4999: 1.122254, 1.037821 avg loss, 0.010000 rate, 2.409490 seconds, 319936 images
Loaded: 0.000047 seconds

 5000: 0.986083, 1.032647 avg loss, 0.010000 rate, 2.438592 seconds, 320000 images
Saving weights to backup//yolo-obj_5000.weights
Saving weights to backup//yolo-obj_last.weights
Resizing
512 x 512
 try to allocate additional workspace_size = 75.50 MB
 CUDA allocate done!
Loaded: 0.000043 seconds
```

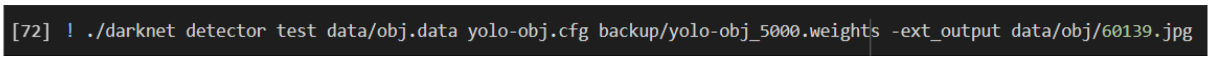
Figure 6: Training YOLOv3

The model is trained till 5000 iterations. The weights after every 100 epochs are stored in backup directory (yolo-obj_last.weights). For every 1000 epochs the weights are stored in a separate file (for example: yolo-obj_5000.weights).

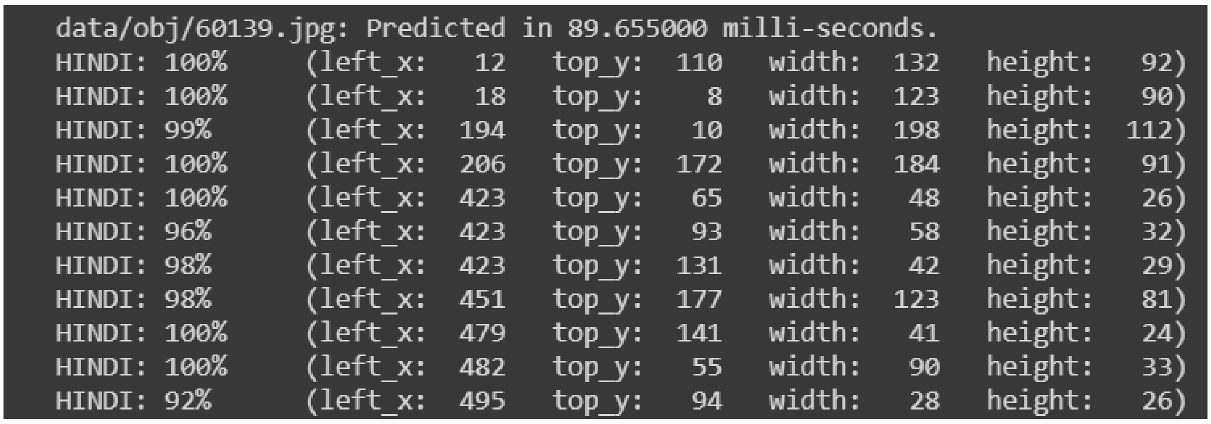
6: RESULTS

Testing the model

The testing is done using the line:



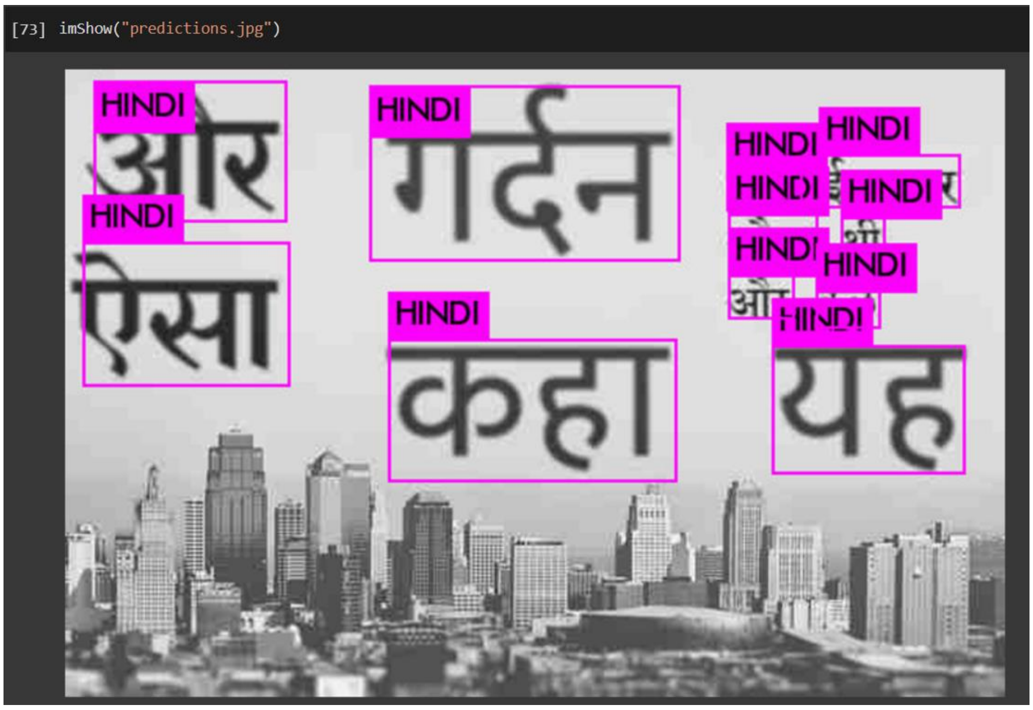
```
[72] ! ./darknet detector test data/obj.data yolo-obj.cfg backup/yolo-obj_5000.weights -ext_output data/obj/60139.jpg
```



```
data/obj/60139.jpg: Predicted in 89.655000 milli-seconds.
HINDI: 100%     (left_x:   12   top_y:  110   width:  132   height:   92)
HINDI: 100%     (left_x:   18   top_y:    8   width:  123   height:   90)
HINDI: 99%      (left_x:  194   top_y:   10   width:  198   height:  112)
HINDI: 100%     (left_x:  206   top_y:  172   width:  184   height:   91)
HINDI: 100%     (left_x:  423   top_y:   65   width:   48   height:   26)
HINDI: 96%      (left_x:  423   top_y:   93   width:   58   height:   32)
HINDI: 98%      (left_x:  423   top_y:  131   width:   42   height:   29)
HINDI: 98%      (left_x:  451   top_y:  177   width:  123   height:   81)
HINDI: 100%     (left_x:  479   top_y:  141   width:   41   height:   24)
HINDI: 100%     (left_x:  482   top_y:   55   width:   90   height:   33)
HINDI: 92%      (left_x:  495   top_y:   94   width:   28   height:   26)
```

Figure 7: Testing

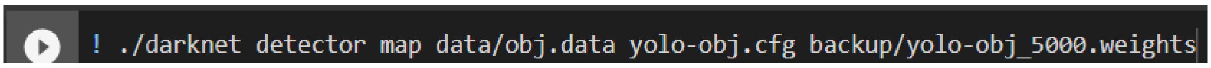
The predicted image is stored as predictions.jpg file.



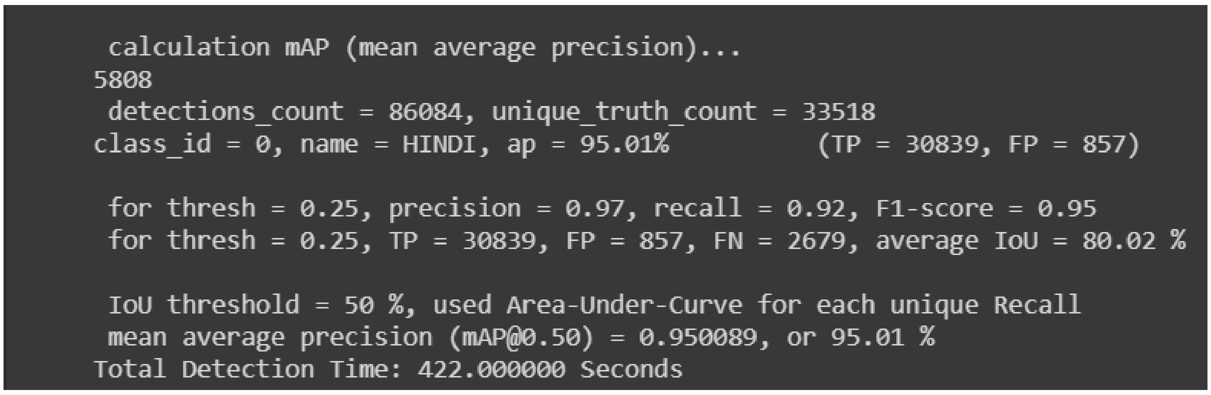
```
[73] imShow("predictions.jpg")
```

Figure 8: Bounding box predicted

For calculation of mean average precision:



```
! ./darknet detector map data/obj.data yolo-obj.cfg backup/yolo-obj_5000.weights
```

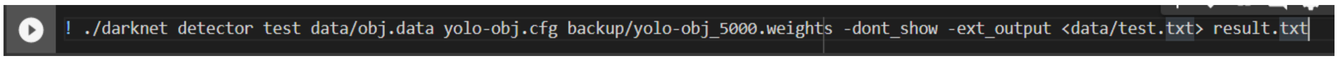


```
 calculation mAP (mean average precision)...
5808
 detections_count = 86084, unique_truth_count = 33518
class_id = 0, name = HINDI, ap = 95.01%          (TP = 30839, FP = 857)

 for thresh = 0.25, precision = 0.97, recall = 0.92, F1-score = 0.95
 for thresh = 0.25, TP = 30839, FP = 857, FN = 2679, average IoU = 80.02 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.950089, or 95.01 %
Total Detection Time: 422.000000 Seconds
```

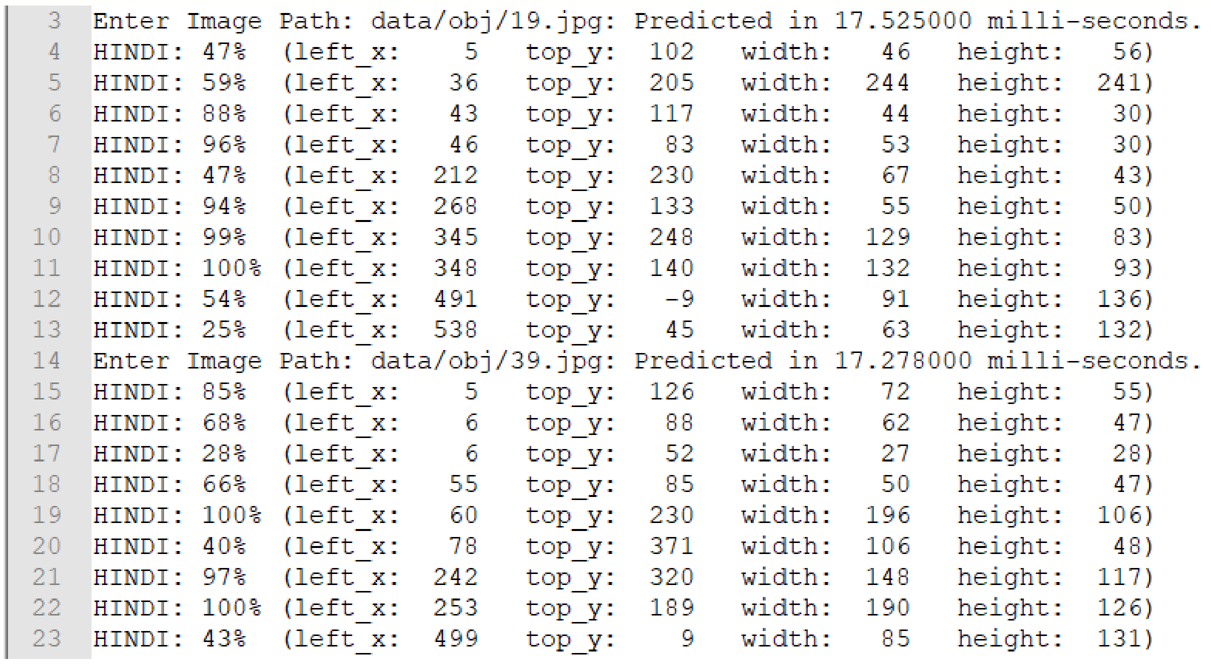
Figure 9: Mean average precision calculated

The mean average precision at IOU 0.5 = **95.01%**

The predictions on the test data can be stored in a separate file (result.txt) as shown below:



```
! ./darknet detector test data/obj.data yolo-obj.cfg backup/yolo-obj_5000.weights -dont_show -ext_output <data/test.txt> result.txt
```



```
Enter Image Path: data/obj/19.jpg: Predicted in 17.525000 milli-seconds.
HINDI: 47%  (left_x:    5   top_y:  102   width:   46   height:   56)
HINDI: 59%  (left_x:   36   top_y:  205   width:  244   height:  241)
HINDI: 88%  (left_x:   43   top_y:  117   width:   44   height:   30)
HINDI: 96%  (left_x:   46   top_y:   83   width:   53   height:   30)
HINDI: 47%  (left_x:  212   top_y:  230   width:   67   height:   43)
HINDI: 94%  (left_x:  268   top_y:  133   width:   55   height:   50)
HINDI: 99%  (left_x:  345   top_y:  248   width:  129   height:   83)
HINDI: 100% (left_x:  348   top_y:  140   width:  132   height:   93)
HINDI: 54%  (left_x:  491   top_y:   -9   width:   91   height:  136)
HINDI: 25%  (left_x:  538   top_y:   45   width:   63   height:  132)
Enter Image Path: data/obj/39.jpg: Predicted in 17.278000 milli-seconds.
HINDI: 85%  (left_x:    5   top_y:  126   width:   72   height:   55)
HINDI: 68%  (left_x:    6   top_y:   88   width:   62   height:   47)
HINDI: 28%  (left_x:    6   top_y:   52   width:   27   height:   28)
HINDI: 66%  (left_x:   55   top_y:   85   width:   50   height:   47)
HINDI: 100% (left_x:   60   top_y:  230   width:  196   height:  106)
HINDI: 40%  (left_x:   78   top_y:  371   width:  106   height:   48)
HINDI: 97%  (left_x:  242   top_y:  320   width:  148   height:  117)
HINDI: 100% (left_x:  253   top_y:  189   width:  190   height:  126)
HINDI: 43%  (left_x:  499   top_y:    9   width:   85   height:  131)
```

Figure 10: Predictions stored in a file

The predicted bounding boxes are then cropped from the image by running a python script. These cropped images can then be passed on to recognition system for recognition of the words.

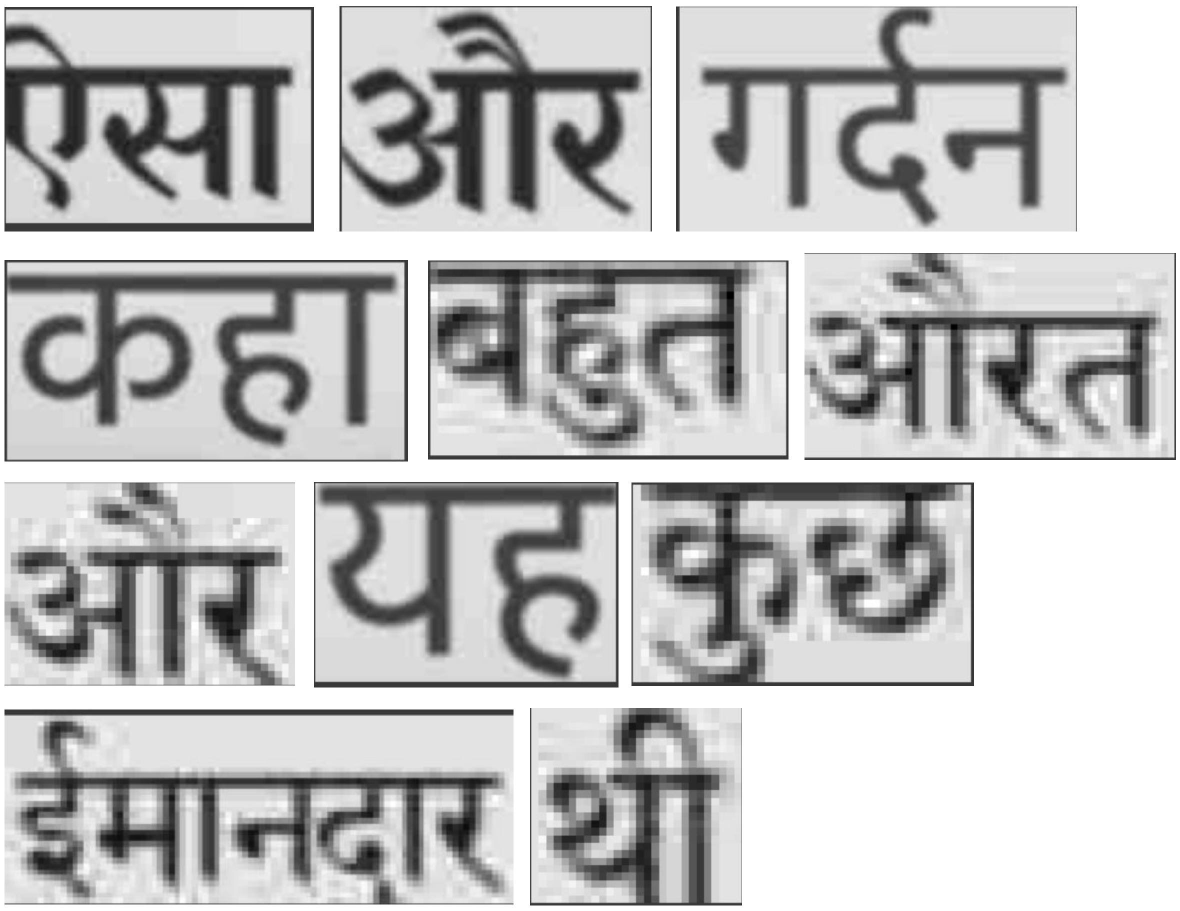


Figure 11: Cropped Bounding box

7: CONCLUSION

The first and foremost task for signboard translation problem is to detect the text objects present in the image. In this project, an object detection engine is implemented which is used to detect text objects in the given image and isolating the region containing the text enclosed in a bounding box. The model used for this task is deep learning based object detector YOLOv3 (You Only Look Once). The model predicts bounding boxes and class probabilities for those boxes with mean average precision (mAP) of 95.01%. The detection time is 89.65 milliseconds. Hence, YOLOv3 can be used for detection of text on the signboards with high precision in a very short time span.

8: FUTURE WORK

The next step for signboard translation problem is text recognition and translation. The output of text detection engine can be passed to text recognition system (encoder-decoder model). The output of text recognition system can then be translated. This task is extended to Major project.

REFERENCES

[1] Z. Zhang, C. Zhang, W. Shen et al., “Multi-Oriented Text Detection with Fully Convolutional Networks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 16 Apr 2016.

[2] X. Zhou, C. Yao, H. Wen et al., “EAST: An Efficient and Accurate Scene Text Detector”, *arXiv:1704.03155v2 [cs.CV]*, 10 Jul 2017.

[3] S. Ren, K. He, R. Girshick, J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, *arXiv:1506.01497v3 [cs.CV]*, 6 Jan 2016.

[4] R. Girshick, “Fast R-CNN”, *arXiv:1504.08083v2 [cs.CV]*, 27 Sep 2015.

[5] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, *arXiv:1506.02640v5 [cs.CV]*, 9 May 2016.

[6] J. Redmon, A. Farhadi, “YOLO9000: Better, Faster, Stronger”, *arXiv:1612.08242v1 [cs.CV]*, 25 Dec 2016.

[7] J. Redmon, A. Farhadi, “YOLOv3: An Incremental Improvement”, *arXiv:1804.02767v1 [cs.CV]*, 2018.

[8] Y. Liu, L. Jin, S. Zhang et al., “Detecting Curve Text in the wild: New Dataset and New Solution”, *arXiv:1712.02170v1 [cs.CV]*, 6 Dec 2017.

[9] P. He, W. Huang, T. He et al., “Single Shot Text Detector with Regional Attention”, in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3047-3055, 2017.

[10] Z. Tian, W. Huang, T. He et al., “Detecting Text in Natural Image with Connectionist Text Proposal Network”, in *European Conference on Computer Vision*. Springer, Cham. pp. 56-72, Oct 2016

[11] B. Shi, X. Bai, C. Yao, “An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and its Application to Scene Text Recognition”, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298-2304, Nov 2017.

[12] “Signboard Translation from Vernacular Languages – Ai4bharat”, AI4Bharat, 2020. [online]. Available at: https://ai4bharat.org/articles/sign-board. [Accessed: 16- Aug-2020].

[13] J. Redmon, “YOLO: Real-Time Object Detection”, *Pjreddie.com*, 2020. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 08- Sep- 2020].

[14] “How to train YOLOv3 using Darknet on Colab 12GB-RAM GPU notebook and speed up load times”, *DEV-ibanyez.info*, 2019. [online]. Available: https://blog.ibanyez.info/blogs/coding/20190410-run-a-google-colab-notebook-to-train-yolov3-using-darknet-in/. [Accessed: 16- Sep- 2020].

[15] “mathieuorhan/darknet”, *Github*, 2018. [Online]. Available: https://github.com/mathieuorhan/darknet. [Accessed: 20-sep- 2020].

[16] “kriyeng/darknet”, GitHub, 2019. [Online]. Available: https://github.com/kriyeng/darknet. [Accessed: 25-Sep- 2020].