# FAST AND ROBUST TEXT DETECTION IN IMAGES AND VIDEO FRAMES

**Mini Project submitted in partial fulfilment of the requirements for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

**Bollempally Laxman Reddy (221710302011)**

**Bompelli Sai Krishna (221710302012)**

**Sripath Cherukuri (221710302061)**

**Yelgonda Aniketh Reddy (221710302065)**

**Under the esteemed guidance of**

**Mr. Jethya**


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**GITAM**

**(DEEMED TO BE UNIVERSITY)**

**HYDERABAD**

**DECEMBER 2020**

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# GITAM

**(Deemed to be University)**

## DECLARATION

We hereby declare that the mini project entitled “FAST AND ROBUST TEXT DETECTION IN IMAGES AND VIDEO FRAMES” is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Bollempally Laxman Reddy
Bompelli Sai Krishna
Sripath Cherukuri
Yelgonda Aniketh Reddy

# DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# GITAM

**(Deemed to be University)**

## CERTIFICATE

This is to certify that Mini Project entitled “FAST AND ROBUST TEXT DETECTION IN IMAGES AND VIDEO FRAMES” is submitted by Bollempally Laxman Reddy (221710302011), Bompelli Sai Krishna (221710302012), Sripath Cherukuri (221710302061), Yelgonda Aniketh Reddy (221710302065) in partial fulfilment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering. The Mini Project has been approved as it satisfies the academic requirements.

**Mr. Jethya**

**Assistant Professor**

**Dept. of Computer**

**Science and Engineering**

# ACKNOWLEDGEMENT

Our Mini Project would not have been successful without the help of several people. we would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice Chancellor, **Prof. N. Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of our seminar.

We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this seminar and encouragement in completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Assistant Professor, Department of Computer Science and Engineering, School of Technology and to **Mr. Jethya**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the summer Internship.

We are also thankful to all the staff members of Computer Science and Engineering department who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our seminar work.

Sincerely,
Bollempally Laxman Reddy
Bompelli Sai Krishna
Sripath Cherukuri
Yelgonda Aniketh Reddy

# CONTENTS

# ABSTRACT

Text is any object that can be "read". It is a set of signs that transmit an informative message. Text in videos or images contains useful information for automatic annotation. Previous approaches for scene text detection have already achieved promising performances across various benchmarks. However, they usually don't perform well when dealing with challenging scenarios, even when equipped with deep neural network models, because the overall performance is determined by multiple stages and components in the framework. In this paper, we propose a simple yet robust framework that yields fast and accurate text detection in natural scenes. The framework directly predicts words or text lines of arbitrary orientations and quadrilateral shapes, eliminating unnecessary intermediate steps (e.g., candidate aggregation and word partitioning), with a single neural network. Experimental results showed that text detection can be performed in real-time. Although, the detection accuracy of the model requires improvement.

Keywords: Text detection, Deep learning, OpenCV, Neural network

| Student Pin no: | Name of the student: |
|---|---|
| 221710302011 | Bollempally Laxman Reddy |
| 221710302012 | Bompelli Sai Krishna |
| 221710302061 | Sripath Cherukuri |
| 221710302065 | Yelgonda Aniketh Reddy |

Name and Signature
of project guide

**Mr. Jethya**

Name and Signature of Project coordinator

**Dr. S. Aparna**

# LIST OF FIGURES

# LIST OF TABLES



# SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| NMS | Non-Maximum Suppression |
| ICDAR | International Conference on Document Analysis and Recognition |
| COCO | Common Objects in Context |
| EAST | Efficient and Accurate Scene Text Detector |

# 1. INTRODUCTION

Over the years, technology has revolutionized our world and daily lives. Technology has created amazing tools and resources, putting useful information at our fingertips.

Modern technology has paved the way for multi-functional devices like the smartwatch and the smartphone. Computers are increasingly faster, more portable, and higher-powered than ever before. With all of these revolutions, technology has also made our lives easier, faster, better, and more fun.

These days many tasks performed by humans are being carried out by machines and robots. Artificial intelligence and machine learning, deep learning have become buzz words recently in the technology sector.

There are many companies and organisations trying to create very good applications based on Artificial intelligence and Machine learning, Deep learning. Organisations namely Google, Microsoft and other IT companies working on these projects.

We also have good number of resources to work on these AI and ML projects, there are huge number of libraries developed by many organisations. Python is most used language these days and its usage has been growing since last 5 years. This programming language is known for its simplicity and the number of libraries it offers to user are simply amazing. Some of the libraries are named below:

- **TensorFlow library**: This library is used to work with Ai and ML models, it offers a bunch of tools and utilities. It is developed by Google.
- **Pytorch library:** This library is also most popular one and offers huge number of methods to work on AI and ML projects. It is developed by Facebook's AI Research Lab.

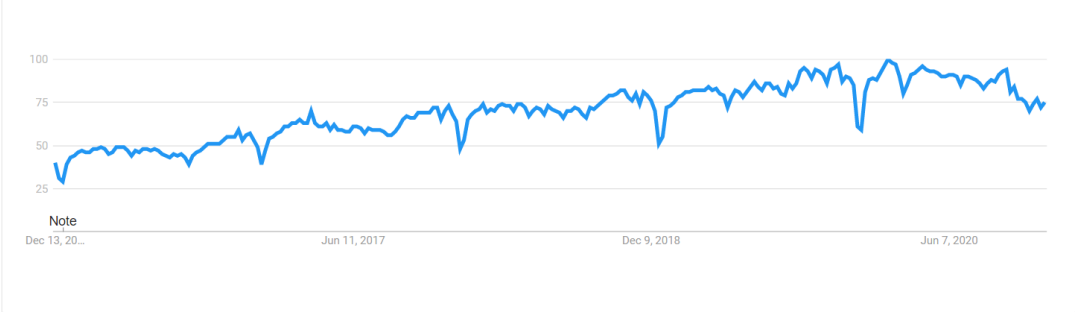


Figure 1.1 Python usage over last 5 years

## 1.1 ARTIFICIAL INTELLIGENCE (AI)

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

The ideal characteristic of artificial intelligence is its ability to rationalize and take actions that have the best chance of achieving a specific goal.

## 1.2 MACHINE LEARNING (ML)

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning.

## 1.3 DEEP LEARNING

Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

Deep-learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, machine vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance. The adjective "deep" in deep learning comes from the use of multiple layers in the network.

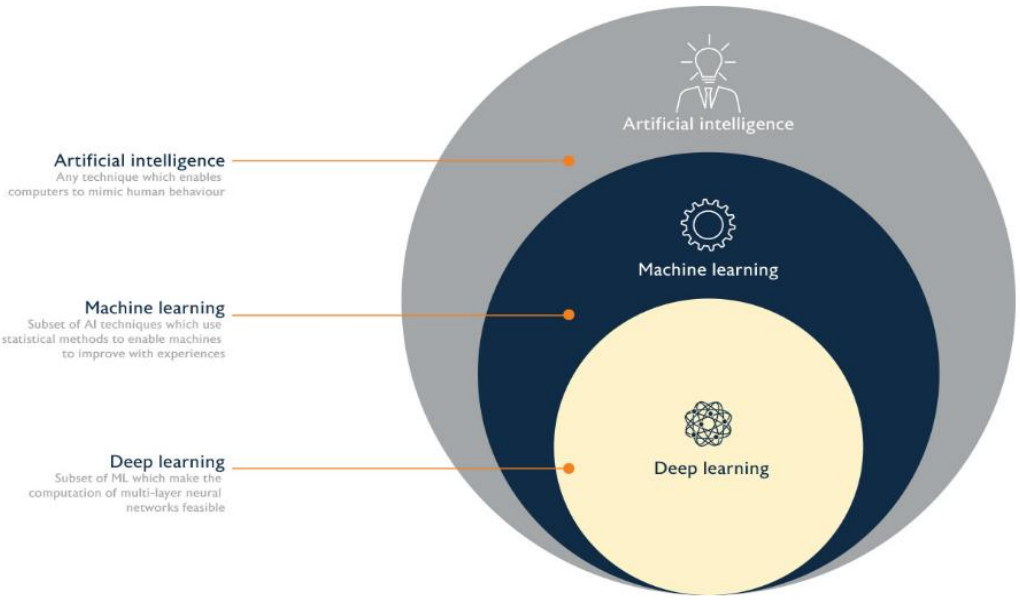


Figure 1.2 Relation Between AI, ML and Deep Learning

## 1.4 TYPES OF LEARNING ALGORITHMS

### 1.4.1 SUPERVISED LEARNING

Supervised learning is the most popular paradigm for machine learning. It is the easiest to understand and the simplest to implement. It is very similar to teaching a child with the use of flash cards.

Given data in the form of examples with labels, we can feed a learning algorithm these example-label pairs one by one, allowing the algorithm to predict the label for each example, and giving it feedback as to whether it predicted the right answer or not. Over time, the algorithm will learn to approximate the exact nature of the relationship between examples and their labels. When fully-trained, the supervised learning algorithm will be able to observe a new, never-before-seen example and predict a good label for it.

Supervised learning is often described as task-oriented because of this. It is highly focused on a singular task, feeding more and more examples to the algorithm until it can accurately perform on that task. This is the learning type that you will most likely encounter, as it is exhibited in many of the common applications such as Advertisement Popularity, Spam Classification etc.

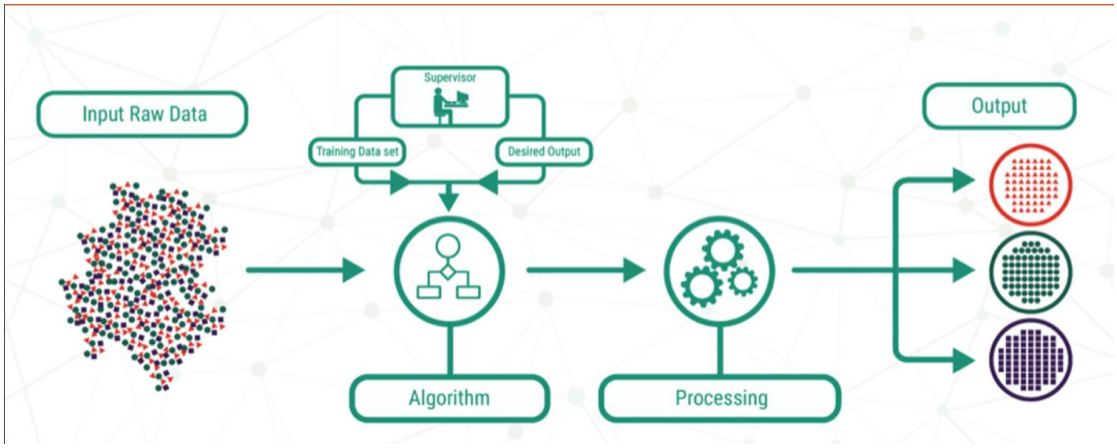


Figure 1.3 Supervised learning

### 1.4.2 UNSUPERVISED LEARNING

Unsupervised learning is very much the opposite of supervised learning. It features no labels. Instead, our algorithm would be fed a lot of data and given the tools to understand the properties of the data. From there, it can learn to group, cluster, and/or organize the data in a way such that a human (or other intelligent algorithm) can come in and make sense of the newly organized data.

For example, what if we had a large database of every research paper ever published and we had an unsupervised learning algorithm that knew how to group these in such a way so that you were always aware of the current progression within a particular domain of research.

Now, you begin to start a research project yourself, hooking your work into this network that the algorithm can see. As you write your work up and take notes, the algorithm makes suggestions to you about related works, works you may wish to cite, and works that may even help you push that domain of research forward. With such a tool, your productivity can be extremely boosted.

Because unsupervised learning is based upon the data and its properties, we can say that unsupervised learning is data-driven. The outcomes from an unsupervised learning task are controlled by the data and the way it’s formatted. Some areas you might see unsupervised learning crop up are Recommender systems, buying habits, grouping user log.

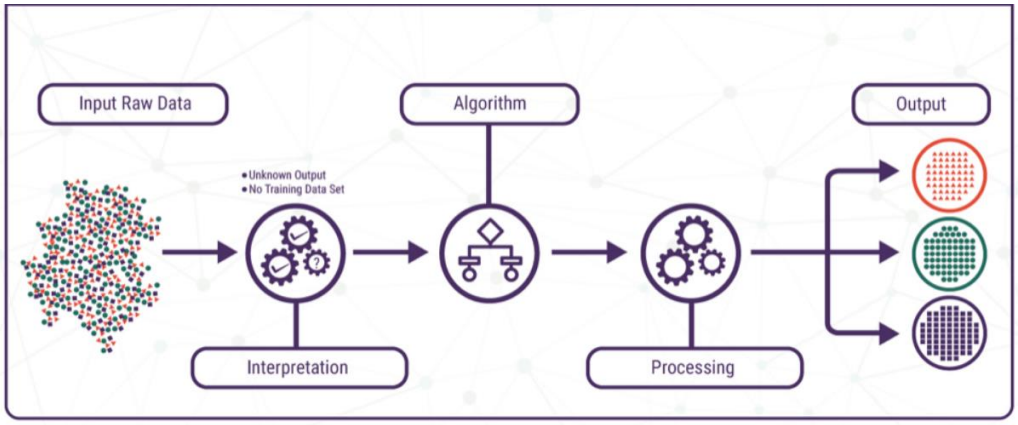


Figure 1.4 Unsupervised learning

### 1.4.3 REINFORCEMENT LEARNING

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.

Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It's up to the model to figure out how to perform the task.

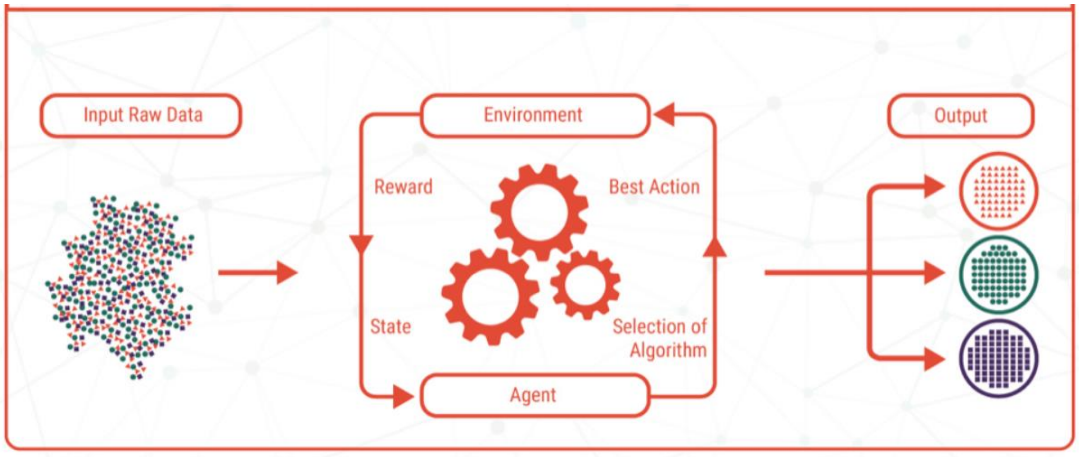


Figure 1.5 Reinforcement learning

## 1.5 PROBLEM DESCRIPTION

To detect text in images and video frames using machine learning models which are developed and trained on different sets of images to improve their accuracy. The role of text detection is to find the image regions containing only text that can be directly highlighted to the user or fed into an optical character reader module for recognition. It is an essential step for text recognition. In some cases, text detection becomes even meaningful by itself. For example, finding the appearance of a caption in news video can help to locate the beginning of a news item.

### 1.5.1 OBJECTIVE

The main aim or objective of this project is to predict where the text could be in the given image and then display a bounding box around that text detected area on the image. This is also applicable to videos as videos are a collection of frames (images).

### 1.5.2 LIMITATIONS

Detecting text in constrained, controlled environments can typically be accomplished by using heuristic-based approaches, such as exploiting gradient information or the fact that text is typically grouped into paragraphs and characters appear on a straight line.

Natural scene text detection is different and much more challenging. Due to the proliferation of cheap digital cameras, and not to mention the fact that nearly every smartphone now has a camera, we need to be highly concerned with the conditions the image was captured under and furthermore, what assumptions we can and cannot make.

- **Image/sensor noise**: Sensor noise from a handheld camera is typically higher than that of a traditional scanner. Additionally, low-priced cameras will typically interpolate the pixels of raw sensors to produce real colours.

- **Viewing angles**: Natural scene text can naturally have viewing angles that are not parallel to the text, making the text harder to recognize.

- **Blurring**: Uncontrolled environments tend to have blur, especially if the end user is utilizing a smartphone that does not have some form of stabilization.

- **Lighting conditions**: We cannot make any assumptions regarding our lighting conditions in natural scene images. It may be near dark, the flash on the camera may be on, or the sun may be shining brightly, saturating the entire image.

- **Resolution**: Not all cameras are created equal — we may be dealing with cameras with sub-par resolution.

- **Non-paper objects**: Most, but not all, paper is not reflective (at least in context of paper you are trying to scan). Text in natural scenes may be reflective, including logos, signs, etc.

- **Non-planar objects**: Consider what happens when you wrap text around a bottle — the text on the surface becomes distorted and deformed. While humans may still be able to easily "detect" and read the text, our algorithms will struggle. We need to be able to handle such use cases.

- **Unknown layout**: We cannot use any a priori information to give our algorithms “clues” as to where the text resides.

These all factors mentioned above affect the performance or the accuracy of the machine learning model and sometimes the model even though trained on large set of samples may not be able to detect text exactly as expected.

### 1.5.3 MOTIVATION FOR THE PROJECT

Images and videos on webs and in databases are increasing. It is a pressing task to develop effective methods to manage and retrieve these multimedia resources by their content. Text, which carries high-level semantic information, is a kind of important object that is useful for this task. For example, text in web images can reflect the content of the web pages. Text on book and journal covers can be helpful to retrieve these digital resources. Caption text in news videos usually annotates information on where, when and who of the happening events. Sub-title in sport videos often annotates information of score, athlete and highlight.

Scene text often suggests the presence of a fact such as advertisement board, traffic warning, etc. Compared with the other image features, text is embedded into images or scenes by human, which can directly reveal the image content in a certain point of view without requiring complex computation. Therefore, it has inspired a lot of research on text detection and recognition in images and videos.

Although a lot of approaches have been developed on text detection in real applications, fast and robust algorithms for detecting text under various conditions need to be further investigated. Here ‘robust’ is referred to the following criteria:

- Good performance (high recall rate and low false alarm rate) for text of various font-size, font-colour, orientations, languages and in complex background.
- Constant performance with minimized need for manually fine-tuning the parameters or rebuilding a model.

## 1.6 APPLICATIONS OF TEXT DETECTION

There are various applications for text detection. They are mentioned as follows:

- **Image understanding**: When images can be automatically understood and indexed by computer, the efficiency of running digital libraries and video database system will be greatly improved.

- **Content based image filtering**: In content-based filtering, image spam can be detected and fraud words can be easily filtered.

- **Vehicle testing**: Vehicle license and scene subtitles have many features in common, so text extraction can be used to supervise the traffic in real time. After text extraction from highway video flow, the traffic situation can be overseen and vehicle licenses can be recognized easily from traffic accidents, which can improve the efficiency of the transportation systems.

- **Super map**: Text extraction technology can be applied to detect scene text from images taken with laptops, phones and other equipment's, so as to be applied to maps, navigation, automatic translation, foreign-related tour guides, walking robots and intelligent monitoring system and also used as visual impaired people's assistance.

- **Optical character reading**: Reads text from paper and translates images into a form that computer can manipulate (for example, into ASCII codes). An OCR system enables to take a book, feed it directly into an electronic computer file, and then edit the file using a word processor.

- **Wearable applications**: Wearable devices such as goggles, phones, cameras are created for detecting text elements and can be converted into voice for blind peoples.

# 2. PROBLEM STUDY/LITERATURE

Text detection and recognition has emerged as an important problem in the past few years. Advancements in the field of computer vision and machine learning as well as increase in the applications based on text detection and recognition has resulted in this trend. Various workshops and conferences like International Conference on Document Analysis and Recognition (ICDAR) are being organized on international level giving further rise to developments in field of text processing from imagery.

Text detection and recognition from video captions as well as web pages is also getting attention. Huge work has been done in the field of text detection and extraction from natural scenes imagery. Various optical character recognition techniques are also available. Still problem of text detection and recognition is not thoroughly solved. Segmentation and extraction of text from natural scenes is still very difficult to achieve.

Scene text recognition has generated significant interest from many branches of research. While it is now possible to achieve extremely high performance on tasks such as digit recognition in controlled settings the task of detecting and labelling characters in complex scenes remains an active research topic. However, many of the methods used for scene text detection and character recognition are predicated on cleverly engineered systems specific to the new task. For text detection, for instance, solutions have ranged from simple off-the-shelf classifiers trained on hand coded features to multi-stage pipelines combining many different algorithms.

## 2.1 PREVIOUSLY DONE RESEARCH TO DETECT TEXT IN IMAGES

In 2013 Huang introduced a new technique for text localization in natural image using Stroke Feature Transform and Text Covariance Descriptors. At first they used a low-level filter “Stroke Feature Transformation (SFT) for removing background pixels. Afterwards, the candidates are obtained using colour homogeneity and consistency between stroke widths. And for component detection, two maps are generated using SFT: stroke width map and a stroke colour map. Then two Text

covariance descriptors (TCD) were used for text region and text line classification. So, by using the SFT and two TCDs, their system's performance improved many folds. The dataset of ICDAR 2005 and ICDAR 2011 are used. The proposed low-level SFT filter, leads to high recall, and the effectiveness of the two-level TCDs, leads to high precision. But this method can only predict text in horizontal manner not vertical.

Minetto in 2013 have proposed a Snooper text in multi scale fashion. First of all, it locates the candidate characters on the images by image segmentation and shape based binary classification. The segmentation algorithm used, was developed by Fabrizio to define local foreground and back ground. Using the SVM classifier, the character filtering is done. Next, the candidate characters are grouped by simple geometric criteria to form either candidate words or candidate text lines. The grouping module in iTown does not work well when come across wide spaces so to overcome this issue, this module was run twice. One more advantage of the multi-scale approach is that it makes the segmentation algorithm insensitive to character texture like high frequency details. The advantage of multi-scale fashion is, it makes segmentation algorithm insensitive to character texture like high frequency details. The method has limitations: The grouping module doesn't work well for wide spaces. And it can't detect tilted or vertical text.

In static natural image, in 2014 Iqbal suggested an adaptive classifier threshold to detect text in images. Adaptive classifier threshold is based on the geometric mean and standard deviation of posterior probabilities of a MSER-based candidate characters using Bayesian network scores. A candidate character is discarded if posterior probability is below the adaptive classifier threshold value. This method is evaluated on an ICDAR 2013 and achieved a significant competitive performance with a comparison of recently published algorithms. However, the proposed method requires improvements in performance by testing another dataset. In addition, this method only detects on a given dataset rather than any natural image.

In 2014 Yin presented a robust method for detecting texts in natural images and designed an effective pruning algorithm using the strategy of minimizing regularized

variations to extract Maximally Stable Extremal Regions (MSERs). Further, they proposed distance matric learning algorithm and then used single link clustering algorithm to group candidate character into text region. Finally, non-text candidates were eliminated using Adaboost classifier based on posterior probabilities. The method is evaluated on ICDAR 2011 Robust Reading Competition dataset and also on multilingual (Chinese and English). But the proposed system doesn't work for all-natural scene images containing any orientation of text.

In 2014 Yao have proposed a multi-scale representation of text in images using stroke lets. Over the conventional approaches stroke lets have following four advantages: Usability, robustness, generality and expressivity called the URGE properties. They designed an efficient algorithm for text recognition. First of all the character identification is done by seeking maxima in Hough maps. They used the discriminative clustering algorithm designed by Singh et al, certain changes were made according to requirement in this algorithm. The random forest classifier is used for classification of components. The data sets of SVT and ICDAR 2013 are used to evaluate the results. The proposed algorithm consistently outperformed the existing state-of-the art approaches. But the result could show variations over other datasets. This system has limitations due to random behaviour.

In 2016 Jaderburg does text detection using deep features. They divided the work of text detection into two tasks: spotting the word region and recognizing the words. A Convolutional Neural Network classifier is used for Case sensitive, Case insensitive and bigram classification and generated the saliency maps for these. The convolutional structure of CNN goes through the whole image once. The mining technique goes through the images produces word level and character level annotations. The output is divided as follow: a character/background classifier, a case insensitive classifier, a case insensitive classifier and a bigram classifier. These classifications add to the efficiency of this work. He used the ICDAR (2003, 2005, 2011, 2013) Robust Reading data set and street view text data set. The system has its limitations due to the saliency maps, because these may sometime give bad resolution and wrong result.

In 2016 Ganesh have worked on text detection in the images of big data. They have taken google API to get many street view images those acted as Big Data in their work. After obtaining the images filters are applied to remove noise. First of all, averaging filter is applied, then median filter is applied and finally adaptive filter is applied to eliminate the low frequency regions and noise. Next, image processing is done in two steps: first, colour-based partition method Second, the classifiers were applied to detect whether the above partitions contain text in them or not. Then by using Hough transformation method, text line grouping method. The strength of the technique is dealing of big data that's very crucial. But it has its limitation that result may vary for other big data rather than dataset used in this technique.

Wang in 2018 focused on videos to detect text by using simultaneous localization and mapping (SLAM) to extract planar scene surfaces "tiles". And all the sensor data is input using LCM (Lightweight Communications and Marshalling) then the system maintain the sensor ring's motion using incremental LIDAR scan-matching and a local map is built consisting of line segment. Tiles are then projected onto the camera which generated the observation and then a fronto-parallel view was observed through a homography transformation. These observations are then considered for text detection and decoding. They used DCT and MSER to produce detection regions for character which then provided to Tesseract. Then a clustering process grouped down the characters word candidate. The output came up to be as sequence of characters with each comprising a small number of candidates. They themselves designed a Scene Image Candidate text line location Text/non-text classification Homography based Rectification OCR based Recognition location Restoration and recognition 6 matrix to evaluate the test result of their work. The strength of the method is text detection in images captured in motion but there is a limitation as well: When the distance is 1.5m while taking observation then the decoder may not work.

## 2.2 OUR RESEARCH

Our project proposes algorithm, instead of classifying an image block into text or non-text block by supervised classification models, a new coarse-to-fine detection

framework is proposed by applying different text properties in different detection stages. In the coarse detection, candidate text regions are firstly obtained using property Dense intensity variety and contrast between text and its background by assuming that all text regions have dense intensity variety and contrast with its background. And then these regions are separated into text lines using property Structural information. In the fine detection, texture property is used to discriminate text with other non-text patterns whose dense intensity variety is similar to that of text. We extract four kinds of texture features to identify text lines from the candidate ones including wavelet moment features, wavelet histogram features, wavelet co-occurrence features and crossing count histogram features. A feature selection algorithm is used to find effective features and a classifier is employed to perform text/non-text classification task.

The algorithm mentioned above is a bit long with extra steps in between. So now we propose a fast and accurate scene text detection pipeline that has only two stages. The pipeline utilizes a fully convolutional network (FCN) model that directly produces word or text-line level predictions, excluding redundant and slow intermediate steps. The produced text predictions, which can be either rotated rectangles or quadrangles, are sent to Non-Maximum Suppression to yield final results. Compared with existing methods, the proposed algorithm achieves significantly enhanced performance, while running much faster, according to the qualitative and quantitative experiments on standard benchmarks.

Libraries used: OpenCV, Non-maxima suppression, NumPy, imutils

## 2.3 RELIABILITY

The improvements of the proposed algorithm over previous methods prove that a simple text detection pipeline, which directly targets the final goal and eliminating redundant processes, can beat elaborated pipelines, even those integrated with large neural network models.

# 3. PROBLEM ANALYSIS

Before we get into how this project was implemented and designed, we need to evaluate or analyse the problem on certain criteria. The criteria include:

- Problem definition
- Requirements analysis
- Feasibility study
- Limitations/Boundaries

## 3.1 PROBLEM DEFINITION

The problem here is to identify the text in images and video frames using machine learning model.

Text here is any alphabetical information that we can get from the image or video. Information is very useful and important these days. Generally, an image gives us more understanding and information than the normal text, so what if we can identify text in image and videos which will give us better understanding about the image. So, if we could detect the text in images and videos, we could get answers for the “wh” questions like when, where, what.

Here the main problem comes with the machine learning model like how well it could detect the text in images and videos. So, coming to the problem definition whatever be the image or video given by the user the model should be able to predict the text in the image or video based on training experience it has, and as a result the area of image or video where text might be present should be highlighted with a bounding box.

## 3.2 REQUIREMENTS ANALYSIS

Now coming on to the requirements for this project it depends on which method you will choose while implementation as there are many ways in which this project can be implemented. But the generalized requirements both hardware and software are mentioned below:

**Hardware requirements**:

- **Processor**: Intel-i5/Intel-i7 or any AMD equivalent
- **RAM**: 8gb minimum or higher
- **GPU**: A decent graphic card with CUDA rating 7.5 - 4gb memory (This is only needed if we choose to develop model from scratch and train it on large datasets)
- **Webcam:** This is required if we want to input live video feed to model.

**Software requirements:**

- Anaconda
- Visual studio code or PyCharm
- Open CV

## 3.3 FEASIBILITY STUDY

This feasibility study provides clear understanding on how resources such as time, money etc are spent on this project and also tells what's the future of this project going to be like and how previous attempts have been performing. There are many factors to be considered such as cost, time, technical feasibility, operational feasibility.

### 3.3.1 TECHNICAL FEASIBILITY

The technical feasibility deals with all possible ways to implement this project. As of now there are three ways in which this text detection project can be implemented. Those three are mentioned below:

1. **Everything from scratch**: In this method everything is developed from scratch on our own. The machine learning model is also developed from scratch and trained on image data sets with sufficient samples until good accuracy is achieved by the model. Now after we are done with the model, we are going to write program on how to use this model and get the predictions based on training experience for the given input images.

Here the user will have the ability to tweak the model if he or she finds anything that has to be changed in the later stages.

2. **Using pre-trained model**: In this method to give the user a head starts the pre-trained model code or file will be available already. Here the user has to work upon how to use this model efficiently and design a program that could use this model data to predict the text in new images or user inputted images and display the result as expected.

3. **Using Google Cloud Vision API**: In this method there will be an API available to the user which when implemented in his or her program it automatically detects the features of the image when an input is given and displays the box around the text in given image. Here the programmer should only worry about how to implement this API in his program. This is the simplest method of all the three mentioned above. But there is a certain cost incurred after exceeding the API usage limit.

### 3.3.2 FINANCIAL FEASIBILITY

The cost indulged in this project is zero as this is done on academic purpose. But if this project was to be released into the real-world market even then it would not incur much cost. We can call it as low budget project. The cost incurred on first two methods mentioned above are almost negligible. If the project is implemented using the google cloud vision API it may cost us a few bucks if the minimum usage of API exceeds the limit. The cost charged for using API after exceeding limit is shown below.

| Feature | Price per 1000 units | | |
|---|---|---|---|
| | First 1000 units/month | Units 1001 - 5,000,000 / month | Units 5,000,001 - 20,000,000 / month |
| Label Detection | Free | $1.50 | $1.00 |
| Text Detection | Free | $1.50 | $0.60 |
| Document Text Detection | Free | $1.50 | $0.60 |

Figure 3.1 Google API cost as per usage

### 3.3.3 OPERATIONAL FEASIBILITY

The operational feasibility deals with how well the proposed system solves the problem. Any one of the above methods satisfies the problem partially but need to be investigated further as the machine learning models accuracy may improve overtime or may be in future due to technological advancements. Method one and method 2 mentioned above in technical feasibility, uses the systems resources as mentioned in the requirements section.

This project also fits into the real-world business as machine learning is one of the recent advancements in technology sector, there are various organisations working on this type of projects and have demand for this type of projects. This could be a very good real-world application if it could be enhanced more. It is also affordable in terms of expenditure and moreover it should be continuously refined where ever there is scope for improvement if deployed as a real-world application. System design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.

### 3.3.4 RESOURCE FEASIBILITY

The number of resources used by the program depends on what method is being to train and develop the machine learning model, from the methods mentioned above in technical feasibility section method 1 (Everything from scratch) uses most of the system resources while running because when the model trains on such huge datasets its going to require lot of memory for processing that image data and store the results. The CPU and GPU also are continuously working in the back to provide speed access to the data for the machine learning model to train.

The other 2 methods use limited resources probably less than the first method as they do not have training process in those methods, but while they are running, they may use webcam if the user want to feed video live to the program to predict output.

### 3.3.5 TIME FEASIBILITY

A time feasibility study will take into account the period in which the project is going to take up to its completion. A project will fail if it takes too long to be completed before it is useful. Typically, this means estimating how long the system will take to develop. The time taken for the project completion is completely based on the technical expertise, that is the programmer's knowledge about the project and his or her capacity to do programs.

But one certain thing that is know is that it will take lot of time to train the model on image samples because sometimes the datasets may be huge, for example a dataset may have 10,000 to 20,000 images. So, while working on such large scale it takes time.

## 3.4 BOUNDARIES OF THE PROJECT

This particular project is only specific to text detection in images and videos, other objects and features of the image and video frames are not being detected here. The detected text is show in bounding box on the image on the output but we are not able to edit or make any changes to the detected text.

Even though it is only limited to text-detection this project acts as a base for text extraction which is the next level of this project. In order to extract text, we have to first detect it, so this project helps to do that and after extraction we can edit that, we can perform search on that text in text extraction. There is a lot of scope for improvement for this project and it can be extended to real world applications such as content-based image filtering and search through text in image, optical character recognition and many more.

# 4. DESIGN

This chapter is going to deal with how project was designed and how a structure was formed. Here we are going to discuss about what is the general flow of the program, what is the structure of the machine learning models, and what all are included. There are various machine learning models for detecting text in images and video frames. Many people have contributed to the design of the model architectures over the years. The model we use here is the EAST model (Efficient and accurate scene text detector). First let us see about the model pipeline and terms included in the pipeline or the project design.

## 4.1 EAST MODEL PIPELINE

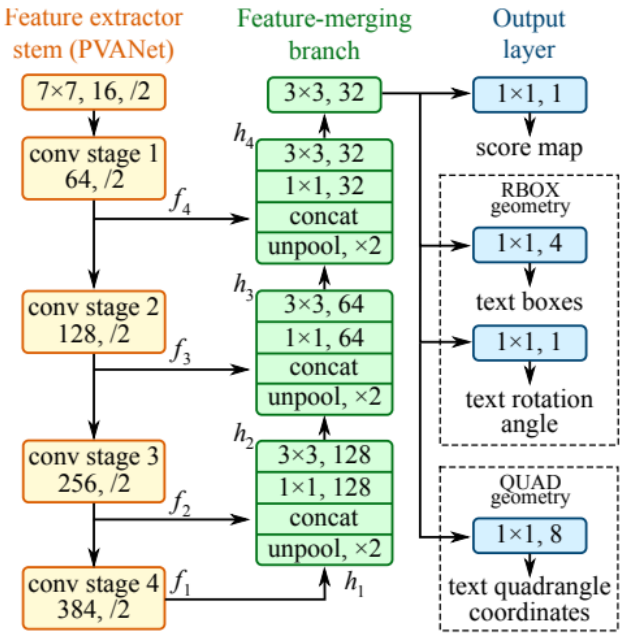


Figure 4.1 EAST model pipeline

This is a very robust deep learning method for text detection. It is worth mentioning as it is only a text detection method. It can find horizontal and rotated bounding boxes. It can be used in combination with any text recognition method. This text detection pipeline has excluded redundant and intermediate steps and only has two stages.

This one utilizes the fully convolutional network to directly produce word or text-line level prediction. The produced predictions which could be rotated rectangles or quadrangles are further processed through the non-maximum-suppression step to yield the final output.

A high-level overview of our pipeline is illustrated in Fig.4.1. The algorithm follows the general design of Dense Box, in which an image is fed into the FCN and multiple channels of pixel-level text score map and geometry are generated. One of the predicted channels is a score map whose pixel values are in the range of [0, 1]. The remaining channels represent geometries that encloses the word from the view of each pixel. The score stands for the confidence of the geometry shape predicted at the same location. We have experimented with two geometry shapes for text regions, rotated box (RBOX) and quadrangle (QUAD), and designed different loss functions for each geometry. Thresholding is then applied to each predicted region, where the geometries whose scores are over the predefined threshold is considered valid and saved for later nonmaximum-suppression. Results after NMS are considered the final output of the pipeline.

Several factors must be taken into account when designing neural networks for text detection. Since the sizes of word regions, vary tremendously, determining the existence of large words would require features from late-stage of a neural network, while predicting accurate geometry enclosing a small word region need low-level information in early stages. Therefore, the network must use features from different levels to fulfil these requirements. We adopt the idea from U-shape to merge feature maps gradually, while keeping the up-sampling branches small. Together we end up with a network that can both utilize different levels of features and keep a small computation cost.

The number of output channels for each convolution is shown in Fig. 3. We keep the number of channels for convolutions in branch small, which adds only a fraction of computation overhead over the stem, making the network computation-efficient. The final output layer contains several conv1×1 operation to project 32 channels of feature maps into 1 channel of score map Fs and a multi-channel geometry

map Fg. The geometry output can be either one of RBOX or QUAD, summarized in Tab. 1 For RBOX, the geometry is represented by 4 channels of axis-aligned bounding box (AABB) R and 1 channel rotation angle θ. The formulation of R is the same, where the 4 channels represents 4 distances from the pixel location to the top, right, bottom, left boundaries of the rectangle respectively.

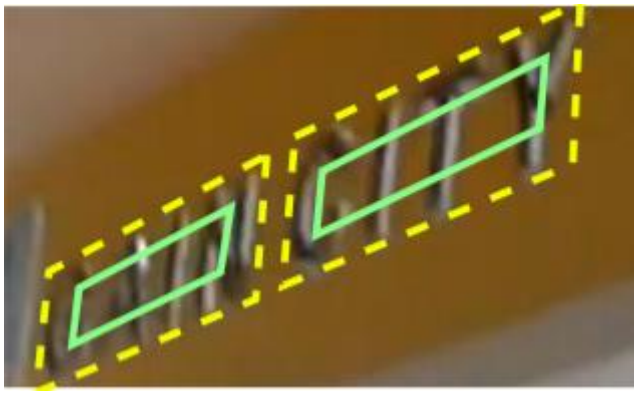

Figure 4.2 Text quadrangle

Figure 4.3 Text Score Map

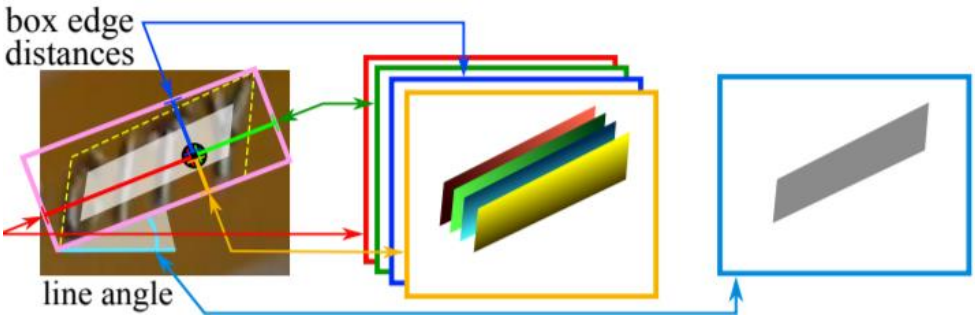


Figure 4.4 RBOX geometry map generation

### 4.1.1 WHAT IS A CONVOLUTION

Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.

The innovation of convolutional neural networks is the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modelling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.

In the context of a convolutional neural network, a convolution is a linear operation that involves the multiplication of a set of weights with the input, much like a traditional neural network. Given that the technique was designed for two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel. The idea of applying the convolutional operation to image data is not new or unique to convolutional neural networks; it is a common technique used in computer vision.

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

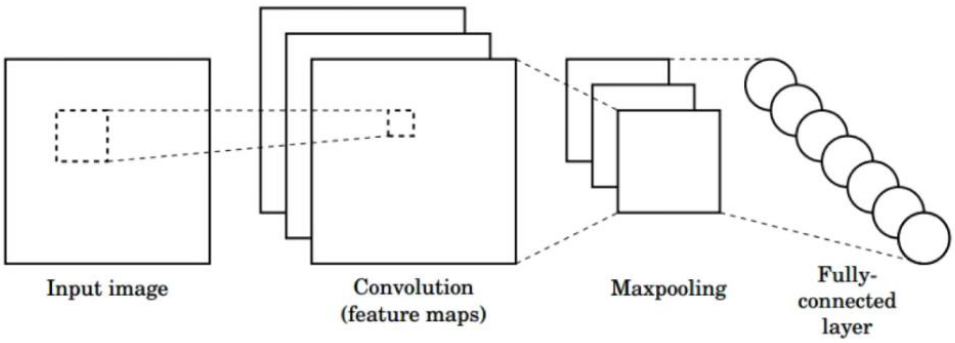


Figure 4.2 Convolution in Machine Learning

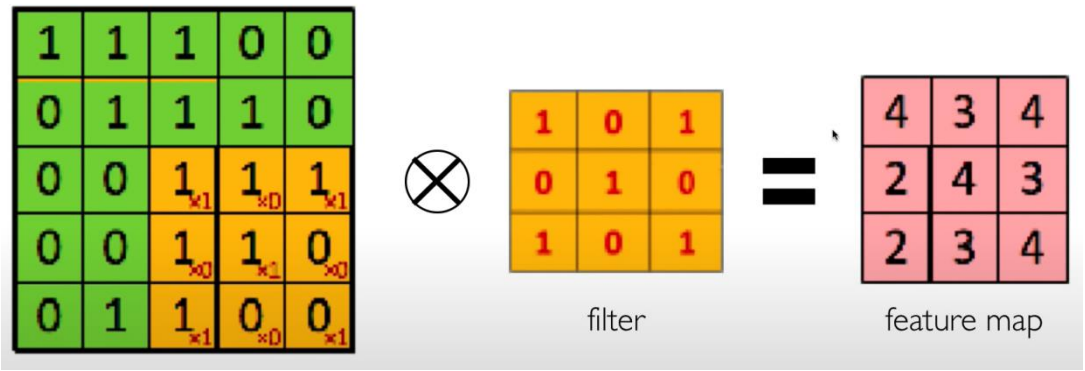


Figure 4.3 Convolution operation

## 4.1.2 WHAT IS POOLING

Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map. Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively. Convolutional layers in a convolutional neural network systematically apply learned filters to input images in order to create feature maps that summarize the presence of those features in the input.

Convolutional layers prove very effective, and stacking convolutional layers in deep models allows layers close to the input to learn low-level features (e.g., lines) and layers deeper in the model to learn high-order or more abstract features, like shapes or specific objects.

A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. This can happen with re-cropping, rotation, shifting, and other minor changes to the input image.

A common approach to addressing this problem from signal processing is called down sampling. This is where a lower resolution version of an input signal is created that still contains the large or important structural elements, without the fine detail that may not be as useful to the task. Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image. A more robust and common approach is to use a pooling layer.

A pooling layer is a new layer added after the convolutional layer. Specifically, after a nonlinearity (e.g., ReLU) has been applied to the feature maps output by a convolutional layer. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.

Pooling involves selecting a pooling operation, much like a filter to be applied to feature maps. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels. This means that the pooling layer will always reduce the size of each feature map by a factor of 2, e.g., each dimension is halved, reducing the number of pixels or values in each feature map to one quarter the size. For example, a pooling layer applied to a feature map of 6×6 (36 pixels) will result in an output pooled feature map of 3×3 (9 pixels).

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

- **Average Pooling**: Calculate the average value for each patch on the feature map.
- **Maximum Pooling (or Max Pooling)**: Calculate the maximum value for each patch of the feature map.
-

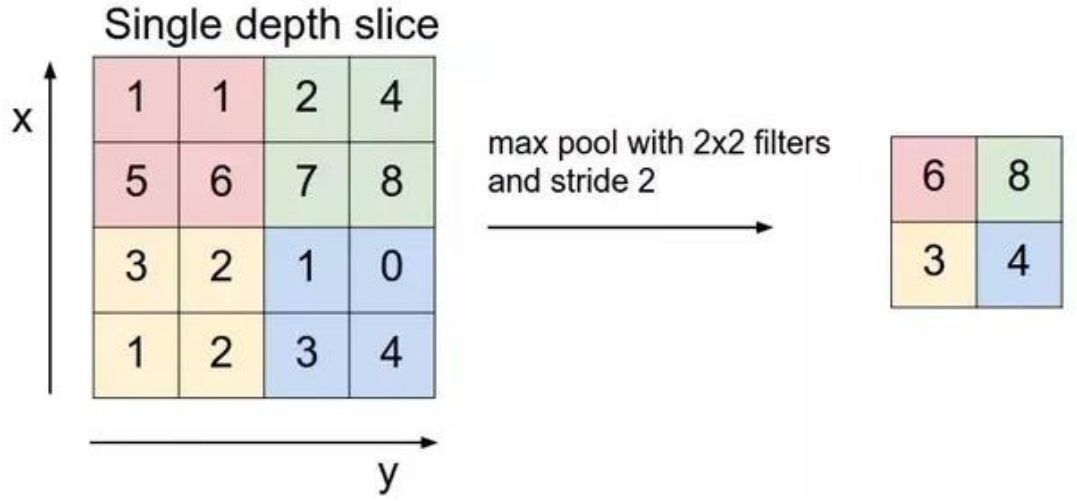


Figure 4.4 Max pooling

### 4.1.3 STRIDING

Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

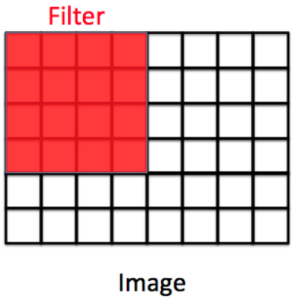


Figure 4.5 No stride length mentioned

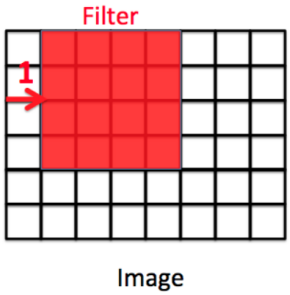


Figure 4.6 Stride length = 1

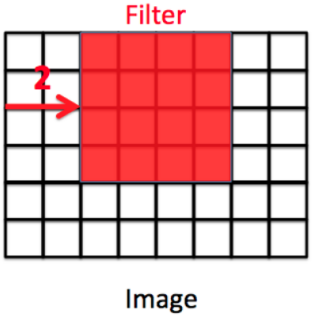


Figure 4.7 Stride length = 2

### 4.1.4 LOSS FUNCTIONS

The loss can be formulated as L = Ls + λgLg where Ls and Lg represents the losses for the score map and the geometry, respectively, and λg weighs the importance between two losses.

$$Ls = \text{balanced-xent}(\hat{Y}, Y*)$$
$$= -\beta Y* \log \hat{Y} - (1 - \beta)(1 - Y*) \log(1 - \hat{Y})$$
$$\beta = 1 - P\ y* \in Y*\ y* / |Y*|$$
$$Lg = LAABB + \lambda\theta L\theta.$$

## 4.2 THE OVERALL PROGRAM DESIGN

The overall design is based on using this trained machine learning model in our program and get the predictions of where the text could be in the given image or video and finally display the output to the user. Here video is nothing but collection of frames so the only difference while dealing with image and video is we loop the program when er are dealing with a video.

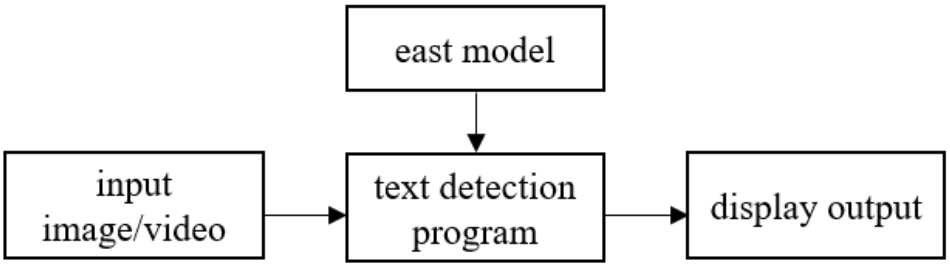


Figure 4.8 Simple Flow of program

# 5. IMPLEMENTATION

## 5.1 GENERAL IMPLEMENTATION OF THE PROJECT

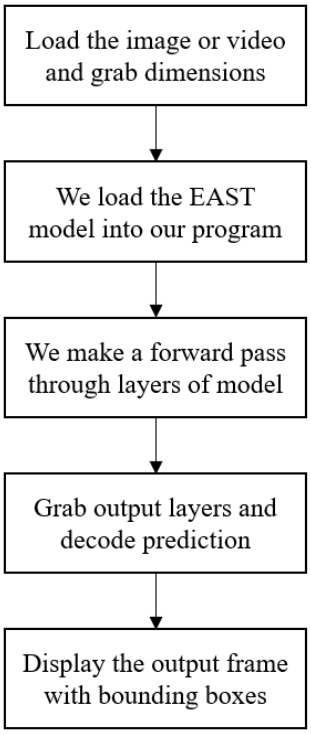


Figure 5.1 General implementation of the project

The first thing is to load all modules required for the image and video handling and some other libraries for other utilities. When the user runs the program and gives any image or video the dimensions of the image or the given video should be captured and resized according to the model input dimensions. Then after we load our EAST model into the program by using function in open cv. Then we take the images or the video given and make a forward pass through two specific layers of model. After doing this we get the output as prediction values, so now we decode these predictions and then display the image with a bounding box on it if text is present.

Now we will look into a detailed code and how we implemented this project. So first let's start with what are all the modules required to run this program. These are call the pre-requisite modules which are required for running the program as expected.

## 5.2 PROGRAM IMPLEMENTATION FOR TEXT DETECTION IN IMAGES

**Imutils:**

The first module we imported here is Imutils. Imutils are a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3. The purpose of non-max suppression is to select the best bounding box for an object and reject or "suppress" all other bounding boxes. The NMS takes two things into account. The objectiveness score is given by the model. The overlap or IOU of the bounding boxes. To select the best bounding box, from the multiple predicted bounding boxes, these object detection algorithms use non-max suppression. This technique is used to "suppress" the less likely bounding boxes and keep only the best one.

**NumPy:**

The second package we imported here is NumPy. NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open-source project and you can use it freely. NumPy stands for Numerical Python. In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

**Argparse:**

The third package imported is argparse. The argparse module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

**Time:**

The next package imported is time. This time module is used to access different time functions and, in our program, we display time taken for model to detect text in images and we also display the time for video and fps counter so we use time module.

**cv2:**

The last and final package we used here was cv2. OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e. whatever operations one can do in NumPy can be combined with OpenCV.

**CODE FOR TEXT DETECTION IN IMAGES (text_detection.py)**

```
# USAGE
# python text_detection.py --image images/lebron_james.jpg --
east frozen_east_text_detection.pb

# we import the necessary packages
from imutils.object_detection import non_max_suppression
import numpy as np
import argparse
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str,
    help="path to input image")
```

```
ap.add_argument("-east", "--east", type=str,
    help="path to input EAST text detector")
ap.add_argument("-c", "--min-
confidence", type=float, default=0.5,
    help="minimum probability required to inspect a region")
ap.add_argument("-w", "--width", type=int, default=320,
    help="resized image width (should be multiple of 32)")
ap.add_argument("-e", "--height", type=int, default=320,
    help="resized image height (should be multiple of 32)")
args = vars(ap.parse_args())

# we load the input image and grab the image dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(H, W) = image.shape[:2]

# we set the new width and height and then determine the ratio
 in change
# for both the width and height
(newW, newH) = (args["width"], args["height"])
rW = W / float(newW)
rH = H / float(newH)

# we resize the image and grab the new image dimensions
image = cv2.resize(image, (newW, newH))
(H, W) = image.shape[:2]

# we define the two output layer names for the EAST detector m
odel that
# we are interested -
- the first is the output probabilities and the
# second can be used to derive the bounding box coordinates of
 text
```

```
layerNames = [
    "feature_fusion/Conv_7/Sigmoid",
    "feature_fusion/concat_3"]

# now we load the pre-trained EAST text detector
print("[INFO] loading EAST text detector...")
net = cv2.dnn.readNet(args["east"])

# we construct a blob from the image and then perform a forwar
d pass of
# the model to obtain the two output layer sets
blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
    (123.68, 116.78, 103.94), swapRB=True, crop=False)
start = time.time()
net.setInput(blob)
(scores, geometry) = net.forward(layerNames)
end = time.time()

# now show timing information on text prediction
print("[INFO] text detection took {:.6f} seconds".format(end -
 start))

# we grab the number of rows and columns from the scores volum
e, then
# initialize our set of bounding box rectangles and correspond
ing
# confidence scores
(numRows, numCols) = scores.shape[2:4]
rects = []
confidences = []

# we loop over the number of rows
for y in range(0, numRows):
```

```
    # extract the scores (probabilities), followed by the geom
etrical
    # data used to derive potential bounding box coordinates t
hat
    # surround text
    scoresData = scores[0, 0, y]
    xData0 = geometry[0, 0, y]
    xData1 = geometry[0, 1, y]
    xData2 = geometry[0, 2, y]
    xData3 = geometry[0, 3, y]
    anglesData = geometry[0, 4, y]

    # we loop over the number of columns
    for x in range(0, numCols):
        # if our score does not have sufficient probability, i
gnore it
        if scoresData[x] < args["min_confidence"]:
            continue

        # we compute the offset factor as our resulting featur
e maps will
        # be 4x smaller than the input image
        (offsetX, offsetY) = (x * 4.0, y * 4.0)

        # we extract the rotation angle for the prediction and
 then
        # compute the sin and cosine
        angle = anglesData[x]
        cos = np.cos(angle)
        sin = np.sin(angle)

        # we use the geometry volume to derive the width and h
eight of
```

```
            # the bounding box
            h = xData0[x] + xData2[x]
            w = xData1[x] + xData3[x]

            # we compute both the starting and ending (x, y)-
coordinates for
            # the text prediction bounding box
            endX = int(offsetX + (cos * xData1[x]) + (sin * xData2
[x]))
            endY = int(offsetY - (sin * xData1[x]) + (cos * xData2
[x]))
            startX = int(endX - w)
            startY = int(endY - h)

            # add the bounding box coordinates and probability sco
re to
            # our respective lists
            rects.append((startX, startY, endX, endY))
            confidences.append(scoresData[x])

    # we apply non-
    maxima suppression to suppress weak, overlapping bounding
    # boxes
    boxes = non_max_suppression(np.array(rects), probs=confidences
    )

    # we loop over the bounding boxes
    for (startX, startY, endX, endY) in boxes:
        # scale the bounding box coordinates based on the respecti
    ve
        # ratios
        startX = int(startX * rW)
        startY = int(startY * rH)
```

```
    endX = int(endX * rW)
    endY = int(endY * rH)

    # draw the bounding box on the image
    cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 25
5, 0), 2)

# now show the output image
cv2.imshow("Text Detection", orig)
cv2.waitKey(0)
```

## 5.3 PROGRAM IMPLEMENTATION FOR TEXT DETECTION IN VIDEOS

The packages used for detecting text in image and text in videos are almost same. The extra packages that we import for detecting text in videos are the VideoStream package and the FPS package again these both are subsets of Imutils module as a whole.

The VideoStream package is used when we deal with videos. It provides different functions and tools which makes us handle videos easily. By having this in our program we can access webcam and live video feeds.

The next package is FPS. This is simply used to monitor the fps (frames per second) in the user inputted video file or on the live video feed i.e., its is helpful while using a webcam instead of local video file.

**CODE FOR TEXT DETECTION IN VIDEOS (text_detection_video.py)**

```
# USAGE
# python text_detection_video.py --
east frozen_east_text_detection.pb
```

```
# import the necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
from imutils.object_detection import non_max_suppression
import numpy as np
import argparse
import imutils
import time
import cv2

def decode_predictions(scores, geometry):
    # we grab the number of rows and columns from the scores v
olume, then
    # we initialize our set of bounding box rectangles and cor
responding
    # confidence scores
    (numRows, numCols) = scores.shape[2:4]
    rects = []
    confidences = []

    # we loop over the number of rows
    for y in range(0, numRows):
        # we extract the scores (probabilities), followed by t
he
        # geometrical data used to derive potential bounding b
ox
        # coordinates that surround text
        scoresData = scores[0, 0, y]
        xData0 = geometry[0, 0, y]
        xData1 = geometry[0, 1, y]
        xData2 = geometry[0, 2, y]
        xData3 = geometry[0, 3, y]
        anglesData = geometry[0, 4, y]
```

```
        # we loop over the number of columns
        for x in range(0, numCols):
            # if our score does not have sufficient probabilit
y,
            # we ignore it
            if scoresData[x] < args["min_confidence"]:
                continue

            # we compute the offset factor as our resulting fe
ature
            # maps will be 4x smaller than the input image
            (offsetX, offsetY) = (x * 4.0, y * 4.0)

            # we extract the rotation angle for the prediction
 and
            # then compute the sin and cosine
            angle = anglesData[x]
            cos = np.cos(angle)
            sin = np.sin(angle)

            # we use the geometry volume to derive the width a
nd height
            # of the bounding box
            h = xData0[x] + xData2[x]
            w = xData1[x] + xData3[x]

            # we compute both the starting and ending (x, y)-
coordinates
            # for the text prediction bounding box
            endX = int(offsetX + (cos * xData1[x]) + (sin * xD
ata2[x]))
```

```
            endY = int(offsetY - (sin * xData1[x]) + (cos * xD
ata2[x]))
            startX = int(endX - w)
            startY = int(endY - h)

            # we add the bounding box coordinates and probabil
ity score
            # to our respective lists
            rects.append((startX, startY, endX, endY))
            confidences.append(scoresData[x])

    # we return a tuple of the bounding boxes and associated c
onfidences
    return (rects, confidences)

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-east", "--east", type=str, required=True,
    help="path to input EAST text detector")
ap.add_argument("-v", "--video", type=str,
    help="path to optinal input video file")
ap.add_argument("-c", "--min-
confidence", type=float, default=0.5,
    help="minimum probability required to inspect a region")
ap.add_argument("-w", "--width", type=int, default=320,
    help="resized image width (should be multiple of 32)")
ap.add_argument("-e", "--height", type=int, default=320,
    help="resized image height (should be multiple of 32)")
args = vars(ap.parse_args())

# we initialize the original frame dimensions, new frame dimen
sions,
# and ratio between the dimensions
```

```
(W, H) = (None, None)
(newW, newH) = (args["width"], args["height"])
(rW, rH) = (None, None)

# we define the two output layer names for the EAST detector m
odel that
# we are interested -
- the first is the output probabilities and the
# second can be used to derive the bounding box coordinates of
 text
layerNames = [
    "feature_fusion/Conv_7/Sigmoid",
    "feature_fusion/concat_3"]

# now load the pre-trained EAST text detector
print("[INFO] loading EAST text detector...")
net = cv2.dnn.readNet(args["east"])

# if a video path was not supplied, grab the reference to the
web cam
if not args.get("video", False):
    print("[INFO] starting video stream...")
    vs = VideoStream(src=0).start()
    time.sleep(1.0)

# otherwise, we grab a reference to the video file
else:
    vs = cv2.VideoCapture(args["video"])

# now start the FPS throughput estimator
fps = FPS().start()

# we loop over frames from the video stream
```

```
while True:
    # we grab the current frame, then handle if we are using a
    # VideoStream or VideoCapture object
    frame = vs.read()
    frame = frame[1] if args.get("video", False) else frame

    # we check to see if we have reached the end of the stream
    if frame is None:
        break

    # we resize the frame, maintaining the aspect ratio
    frame = imutils.resize(frame, width=1000)
    orig = frame.copy()


    # if our frame dimensions are None, we still need to compu
te the
    # ratio of old frame dimensions to new frame dimensions
    if W is None or H is None:
        (H, W) = frame.shape[:2]
        rW = W / float(newW)
        rH = H / float(newH)


    # we resize the frame, this time ignoring aspect ratio
    frame = cv2.resize(frame, (newW, newH))


    # we construct a blob from the frame and then perform a fo
rward pass
    # of the model to obtain the two output layer sets
    blob = cv2.dnn.blobFromImage(frame, 1.0, (newW, newH),
        (123.68, 116.78, 103.94), swapRB=True, crop=False)
```

```
    net.setInput(blob)
    (scores, geometry) = net.forward(layerNames)


    # we decode the predictions, then  apply non-
maxima suppression to
    # suppress weak, overlapping bounding boxes
    (rects, confidences) = decode_predictions(scores, geometry
)
    boxes = non_max_suppression(np.array(rects), probs=confide
nces)


    # we loop over the bounding boxes
    for (startX, startY, endX, endY) in boxes:
        # scale the bounding box coordinates based on the resp
ective
        # ratios
        startX = int(startX * rW)
        startY = int(startY * rH)
        endX = int(endX * rW)
        endY = int(endY * rH)



        # draw the bounding box on the frame
        cv2.rectangle(orig, (startX, startY), (endX, endY), (0
, 255, 0), 2)

    # we update the FPS counter
    fps.update()

    # we show the output frame
```

```
        cv2.imshow("Text Detection", orig)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

# now stop the timer and display FPS information
fps.stop()
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# if we are using a webcam, release the pointer
if not args.get("video", False):
    vs.stop()

# otherwise, release the file pointer
else:
    vs.release()

# close all windows
cv2.destroyAllWindows()
```

# 6. TESTING & VALIDATION

## 6.1 TESTING

To test the proposed algorithm and model we tried running the program on different images in different lighting conditions and scenarios. Our algorithm was able to detect most of the text in images properly. Sometimes when the given input image was dark it could not detect properly. Some of the sample images are shown below.


Figure 6.1 Sample-1

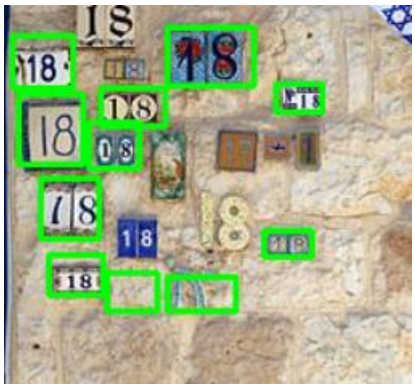

Figure 6.2 Sample-2

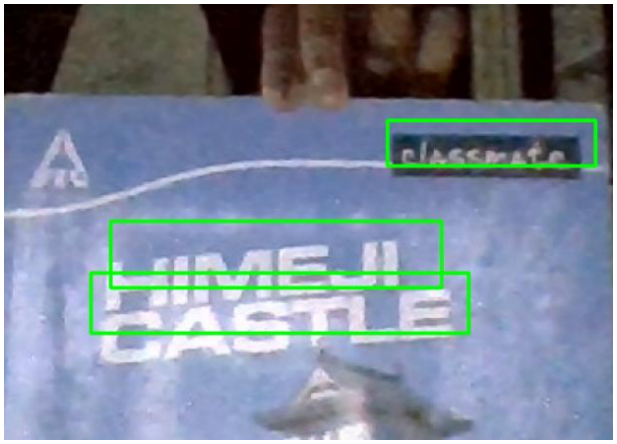


Figure 6.3 Sample-3

## 6.2 VALIDATION

To validate our model, we tested it on different benchmark datasets namely ICDAR2015, COCO-Text and MSRA-TD500.

ICDAR 2015 is used in Challenge 4 of ICDAR 2015 Robust Reading Competition. It includes a total of 1500 pictures, 1000 of which are used for training and the remaining are for testing. The text regions are annotated by 4 vertices of the quadrangle, corresponding to the QUAD geometry in this paper. We also generate RBOX output by fitting a rotated rectangle which has the minimum area. These images are taken by Google Glass in an incidental way. Therefore, text in the scene can be in arbitrary orientations, or suffer from motion blur and low resolution. We also used the 229 training images from ICDAR 2013.

COCO-Text is the largest text detection dataset to date. It reuses the images from MS-COCO dataset. A total of 63,686 images are annotated, in which 43,686 are chosen to be the training set and the rest 20,000 for testing. Word regions are annotated in the form of axis-aligned bounding box, which is a special case of RBOX. For this dataset, we set angle $\theta$ to zero. We use the same data processing and test method as in ICDAR 2015.

MSRA-TD500 is a dataset comprises of 300 training images and 200 test images. Text regions are of arbitrary orientations and annotated at sentence level.

Different from the other datasets, it contains text in both English and Chinese. The text regions are annotated in RBOX format. Since the number of training images is too few to learn a deep model, we also harness 400 images from HUSTTR400 dataset as training data.

As shown in Table.6.1 Our approach outperforms previous state-of-the-art methods by a large margin on ICDAR 2015 and COCO-Text. In ICDAR 2015 Challenge 4, when images are fed at their original scale, the proposed method achieves an F-score of 0.7820. When tested at multiple scales using the same network, our method reaches 0.8072 in F-score, which is nearly 0.16 higher than the best method in terms of absolute value (0.8072 vs. 0.6477).

As observed from Table.6.2 MSRA-TD500 all of the three settings of our method achieve excellent results. The F-score of the best performer (Ours+PVANET2x) is slightly higher. Compared with the method of Zhang et al., the previous published state-of-the-art system, the best performer (Ours+PVANET2x) obtains an improvement of 0.0208 in F-score and 0.0428 in precision.

| **Algorithm** | **Recall** | **Precision** | **F-score** |
|---|---|---|---|
| Ours + PVANET2x RBOX MS* | 0.7833 | 0.8327 | 0.8072 |
| Ours + PVANET2x RBOX | 0.7347 | 0.8357 | 0.7820 |
| Ours + PVANET2x QUAD | 0.7419 | 0.8018 | 0.7707 |
| Ours + VGG16 RBOX | 0.7275 | 0.8046 | 0.7641 |
| Ours + PVANET RBOX | 0.7135 | 0.8063 | 0.7571 |
| Ours + PVANET QUAD | 0.6856 | 0.8119 | 0.7434 |
| Ours + VGG16 QUAD | 0.6895 | 0.7987 | 0.7401 |
| Yao *et al*. [41] | 0.5869 | 0.7226 | 0.6477 |
| Tian *et al*. [34] | 0.5156 | 0.7422 | 0.6085 |
| Zhang *et al*. [48] | 0.4309 | 0.7081 | 0.5358 |

| | | | |
|---|---|---|---|
| StradVision2 [15] | 0.3674 | 0.7746 | 0.4984 |
| StradVision1 [15] | 0.4627 | 0.5339 | 0.4957 |
| NJU [15] | 0.3625 | 0.7044 | 0.4787 |
| AJOU [20] | 0.4694 | 0.4726 | 0.4710 |
| Deep2Text-MO [45, 44] | 0.3211 | 0.4959 | 0.3898 |
| CNN MSER [15] | 0.3442 | 0.3471 | 0.3457 |

Table 6.1 ICDAR Challenge 4

| **Algorithm** | **Recall** | **Precision** | **F-score** |
|---|---|---|---|
| Ours + PVANET2x | 0.6743 | 0.8728 | 0.7608 |
| Ours + PVANET | 0.6713 | 0.8356 | 0.7445 |
| Ours + VGG16 | 0.6160 | 0.8167 | 0.7023 |
| Yao et al. [41] | 0.7531 | 0.7651 | 0.7591 |
| Zhang et al. [48] | 0.67 | 0.83 | 0.74 |
| Yin et al. [44] | 0.63 | 0.81 | 0.71 |
| Kang et al. [14] | 0.62 | 0.71 | 0.66 |
| Yin et al. [45] | 0.61 | 0.71 | 0.66 |
| TD-Mixture [40] | 0.63 | 0.63 | 0.60 |
| TD-ICDAR [40] | 0.52 | 0.53 | 0.50 |
| Epshtein et al. [5] | 0.25 | 0.25 | 0.25 |

Table 6.2 MSRA-TD500

# 7. RESULTS ANALYSIS

## 7.1 QUALITATIVE RESULT

It is able to handle various challenging scenarios, such as non-uniform illumination, low resolution, varying orientation and perspective distortion. Moreover, due to the voting mechanism in the NMS procedure, the proposed method shows a high level of stability on videos with various forms of text instances. The intermediate results of the proposed method. As can be seen, the trained model produces highly accurate geometry maps and score map, in which detections of text instances in varying orientations are easily formed.

## 7.2 QUANTITATIVE RESULTS

While the proposed method significantly outperforms state-of-the-art methods, the computation cost is kept very low, attributing to the simple and efficient pipeline. The fastest setting of our method runs at a speed of 16.8 FPS, while slowest setting runs at 6.52 FPS. Even the best performing model Ours+PVANET2x runs at a speed of 13.2 FPS. This confirm that our method is among the most efficient text detectors that achieve state-of-the-art performance on benchmarks.

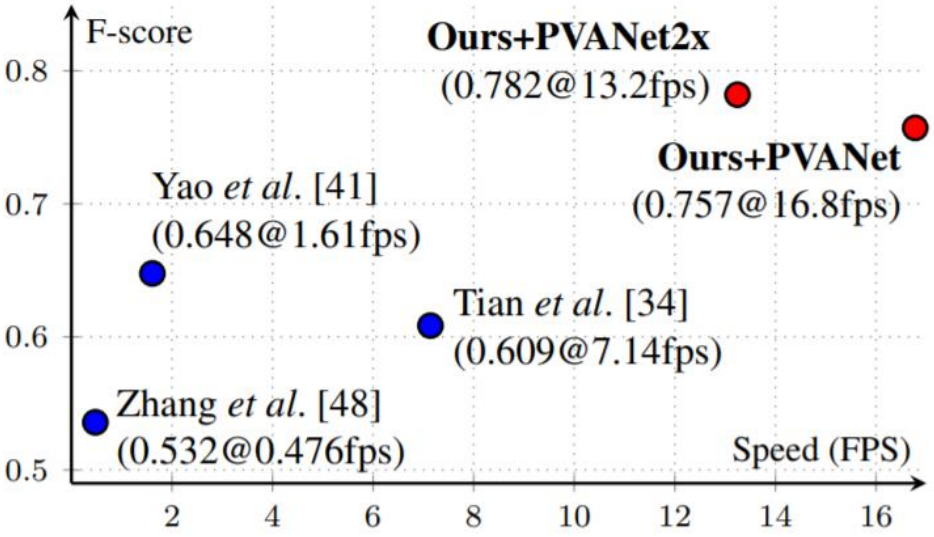


Figure 7.1 F-score vs Speed

Here are some more sample images where text was detected.

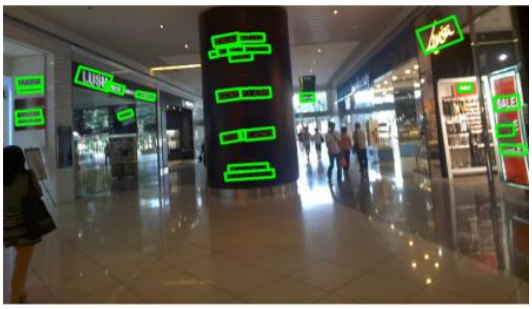

Figure 7.2 Image-1


Figure 7.3 Image-2

Figure 7.4 Image-3

## 7.3 LIMITATIONS

The maximal size of text instances the detector can handle is proportional to the receptive field of the network. This limits the capability of the network to predict even longer text regions like text lines running across the images. Also, the algorithm might miss or give imprecise predictions for vertical text instances as they take only a small portion of text regions in the training set.

# 8. CONCLUSION

We have presented a scene text detector that directly produces word or line level predictions from full images with a single neural network. By incorporating proper loss functions, the detector can predict either rotated rectangles or quadrangles for text regions, depending on specific applications. The experiments on standard benchmarks confirm that the proposed algorithm substantially outperforms previous methods in terms of both accuracy and efficiency.

Even though the model is able to detect most of text in images and videos, there will be always scope for improvement in the technology sector. So, this model can be further refined and modified to achieve better results. Here are some possible directions for future research:

- adapting the geometry formulation to allow direct detection of curved text
- integrating the detector with a text recognizer.
- extending the idea to general object detection.

# REFERENCES

[1] K. Sobottka, H. Bunke, H. Kronenberg, Identification of text on colored book and journal covers, International Conference on Document Analysis and Recognition 1999; 57–63.

[2] H. Li, D. Doermann, O. Kia, Automatic text detection and tracking in digitalvideo,IEEETransactionsonImageProcessing9(2000)147–156.

[3] K.I. Kim, K. Jung, H. Kim, Texture-based approach for text detection in images using support vector machines and continuously adaptive mean shift algorithm, IEEE Transactions on PAMI 25 (2003) 1631– 1639.

[4] X. Tang, X.B. Gao, J. Liu, H. Zhang, Spatial-temporal approach for video caption detection and recognition, IEEE Transactions on Neural Networks 13 (2002) 961–971.

[5] B. Luo, X. Tang, J. Liu, H. Zhang, Video caption detection and extraction using temporal feature vector, International Conference on Image Processing, Spain, September 2003; 297–300.

[6] D. Chen, J.-M. Odobez, H. Bourlard, Text segmentation and recognition in complex background based on Markov random field, Proceedings of the International Conference on Pattern Recognition, April 2002; 227–230.

[7] Q.X. Ye, W. Gao, Q.M. Huang, Automatic text segmentation from complex background, IEEE International Conference on Image Processing, Singapore, October 2004; 2305–2308.

**Websites referred:**

[1]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2] https://deepai.org/machine-learning-glossary-and-terms/stride

[3] https://en.wikipedia.org/wiki/Convolutional_neural_network

[4] https://cloud.google.com/vision/docs/ocr

[5] https://www.tensorflow.org/

[6] https://stackoverflow.com/questions/24385714/detect-text-region-in-image-using-opencv