ROTATION - INVARIANT SCENE TEXT EXTRACTION USING TRANSFORMER NETWORKS

A PROJECT REPORT

Submitted by

MANOJ R	**(Reg. No. 9517202104082)**
SREENIVASAN S R	**(Reg. No. 9517202104158)**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

(An Autonomous Institution affiliated to Anna University Chennai)

APRIL 2025

BONAFIDE CERTIFICATE

Certified that this project report titled **"ROTATION - INVARIANT SCENE TEXT EXTRACTION USING TRANSFORMER NETWORKS"** is the bonafide work of **Mr. R. MANOJ (Reg. No.:9517202104082)** and **Mr. S. R. SREENIVASAN (Reg. No.: 9517202104158)** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Internal Guide	**Head of the Department**
Dr.S.VANITHA SIVAGAMI, M.E., Ph.D. Professor Dept. of Computer Science and Engg. Mepco Schlenk Engg. College (Autonomous) Sivakasi.	**Dr. J. RAJA SEKAR,** M.E., Ph.D. Professor and Head Dept. of Computer Science and Engg. Mepco Schlenk Engg. College (Autonomous) Sivakasi.

Submitted for Viva-Voce Examination held at **Mepco Schlenk Engineering College (Autonomous), Sivakasi** on ____/____/ 2025**.**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

A novel approach to scene text recognition is presented in this work, utilizing a transformer-based framework that enhances accuracy by addressing geometric distortions and background noise. Various forms of text, such as those found on logos, signs, banners, advertisements, and more, are encountered in our daily lives. These texts are not always displayed horizontally, which becomes increasingly important when images of such texts are captured. The overall view can be significantly altered by the perspective from which an image is taken. Furthermore, text may appear against complex backgrounds filled with clutter, irregularities, and bumps, making accurate extraction a considerable challenge for computers. The proposed model operates in two phases: text detection and text recognition. Text detection involves the identification of text regions within the provided image, while text recognition focuses on the actual extraction of words from these detected regions. To address these challenges, deep learning techniques are employed in this project to improve the efficiency and accuracy of scene text recognition. Unlike traditional optical character recognition (OCR) systems, the proposed approach is tailored for unstructured and real-world scenarios. The transformer-based architecture allows for parallel processing, significantly reducing inference time compared to conventional sequential methods. Additionally, the model is designed to adapt to various text formats, enhancing its robustness across different real-world applications.

ACKNOWLEDGEMENT

First and foremost, we thank the **LORD ALMIGHTY** for his abundant blessings that are showered upon our past, present and future successful endeavours.

We express our heartiest gratitude and sincere thanks to our college management and Principal **Dr. S. ARIVAZHAGAN, M.E., Ph.D.,** for allowing and providing us with all resources for doing this project successfully.

We would like to extend our gratitude to **Dr. J. RAJA SEKAR, M.E., Ph.D.,** Professor and Head, Department of Computer Science and Engineering, Mepco Schlenk Engineering College for giving us the golden opportunity to undertake a project of this nature.

We thank our project coordinator **Dr. P. VINOTH, M.E., Ph.D.,** Assistant Professor (Sl. Grade), Department of Computer Science and Engineering, Mepco Schlenk Engineering College for being our Project Coordinator and for directing us throughout our project.

We would also like to extend our heartfelt gratitude and sincere thanks to our project guide **Dr. S. VANITHA SIVAGAMI, M.E., Ph.D.,** Professor, Department of Computer Science and Engineering, Mepco Schlenk Engineering College for guiding us in the right way with full commitment and energy to complete our project successfully and for giving her valuable time to guide us.

We would also like to thank all the **staff and technicians** in the department for helping us to complete our work.

Last but not least, we are very grateful to our beloved **family and friends** for their consistent support and love which made the project a successful one.

TABLE OF CONTENTS

CHAPTER NO.	CONTENTS	PAGE NO.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS		EXPANSION
OCR	-	**O**ptical **C**haracter Recognition
STR	-	**S**cene **T**ext **R**ecognition
CNN	-	**C**onvolutional **N**eural **N**etwork
I2C2W	-	**I**mage-to-**C**haracter-to-**W**ord
I2C	-	**I**mage-to-**C**haracter
C2W	-	**C**haracter-to-**W**ord
SIUN	-	**S**cale **I**terative **U**pscaling **N**etwork
IOT	-	**I**nternet **O**f **T**hings
SR	-	**S**uper **R**esolution
GAN	-	**G**enerative **A**dversial **N**etwork
TGDT	-	**T**oken **G**uided **D**ual **T**ransformer
CMC	-	**C**onsistent **M**ultimodal **C**ontrastive
TSE	-	**T**arget **S**hape **E**xtraction
SFE	-	**S**tyle **F**eature **E**xtraction
ICDAR	-	**I**nternational **C**onference on **D**ocument **A**nalysis and **R**ecognition
TPGSR	-	**T**ext **P**rior **G**uided **S**cene text **R**ecognition
HR	-	**H**igh **R**esolution
LR	-	**L**ow **R**esolution
LAU	-	**L**atitude **A**daptive **U**pscaling
ODI	-	**O**mni **D**irectional **I**mages
DDM	-	**D**rop-**B**and **D**ecision **M**odule
ViT	-	**V**ision **T**ransformer
VCP	-	**V**ision **C**ontent **P**osition
LSM	-	**L**ine **S**egmentation **M**ask
RSMS	-	**R**einforcement **S**hrink **M**ask
LSTM	-	**L**ong **S**hort-**Term** **M**emory
SPW	-	**S**uper **P**ixel **W**indow
CTC	-	**C**onnectionist **T**emporal **C**lassification

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

This chapter provides an overview of the project, discussing its purpose, objectives, outcomes, and applications. This work features a unique approach to scene text recognition using a transformer-based framework that improves accuracy by overcoming geometric distortions and background noise. In our practical life, numerous words in various forms, such as on logos, signs, banners, advertisements, and more, are encountered by us. These texts are not always presented horizontally, and this becomes more vital when images of such texts are captured. The overview of the image changes depending on the position from which it is taken. Additionally, texts may appear against complex backgrounds filled with clutter, irregularities, and bumps. It becomes a great challenge for computers to accurately extract. To explain these issues, this project provides deep learning techniques to enhance the efficiency and accuracy of scene text recognition. Unlike the conventional optical character recognition (OCR) systems, the proposed approach is made for unstructured and real-world scenarios. The transformer-based architecture enables parallel processing and reduces inference time compared to traditional sequential methods. Additionally, the model is designed to be adaptable to multiple text variants and making it more robust across different real-world applications.

1.2 PROBLEM DESCRIPTION

Scene Text Recognition (STR) is a difficult task in computer vision and it enables various applications such as autonomous navigation, document digitization, and assistive technologies. Traditional STR models struggle with challenges like complex backgrounds, geometric distortions, and low-resolution text images. The most existing methods works on sequential decoding or segmentation-based approaches, both of which have significant limitations. Sequential decoding aims at one character at a time and it is prone to misalignment issues due to image noise. On the other hand, segmentation-based methods require extensive character-level annotations, which are expensive to obtain. They are also often inaccurate.

Additionally, CNN-RNN-based models undergo numerous problems with text occlusion, overlapping characters, and blurred images, leading to recognition errors. Most existing models lack generalization when tested on new datasets, making them less effective for real-world scenarios. To address these challenges, a new Vision Transformer-based [2] model has been developed. This approach introduces a two-stage recognition process:

1. **Text Detection:** The process of identification of text regions in the provided image.
2. **Text Recognition:** The process of actual extraction of word from the detected text regions.

This method abolishes the need for character-level annotations and heuristic grouping rules, making it more robust in recognizing scene text in distorted, curved, or low-resolution conditions. The transformer-based architecture effectively creates spatial dependencies and character relationships. It improves overall recognition accuracy. Additionally, the proposed method reduces dependency on predefined lexicons, extends its flexibility for real-world applications such as smart surveillance, retail automation, and augmented reality.

1.3 OBJECTIVES OF THE PROJECT

- To build up robustness in recognizing text affected by geometric distortions, text occlusions, and background noise.
- To find a system that does not require character-level annotations while maintaining high accuracy in text recognition.
- To develop a framework to efficiently extract text from curved, multi-oriented, and distorted text images.

1.4 OUTCOMES OF THE PROJECT

- The locations of the text regions are identified.
- Identified text regions are cropped from the overall image.
- The cropped images are processed for text recognition.
- Curved and distorted texts are extracted from the image.

1.5 ORGANIZATION OF THE PROJECT REPORT

- **Chapter 2:** Reviews existing scene text recognition methodologies and their limitations.
- **Chapter 3:** Explains the study of system architecture and its components.
- **Chapter 4:** Describes the proposed **Vision transformer-based framework**.
- **Chapter 5:** Details the implementation methodology of the model.
- **Chapter 6:** Presents experimental results and performance evaluation.
- **Chapter 7:** Summarizes the findings, conclusions, and potential future enhancements.

1.5 SUMMARY

This chapter provides an introduction to the project, defining its purpose, objectives, and expected outcomes. The next chapter will discuss existing methodologies and their challenges, leading to the need for the proposed Vision transformer-based approach for accurate scene text recognition.

CHAPTER 2

LITERATURE SURVEY

2.1 OVERVIEW

This section discusses the methodologies from the existing state of art work. The techniques discussed here consists of CNN, transformers and deep learning approaches. The advantages and disadvantages of various existing techniques were reviewed.

2.2 Hierarchical Transformer Framework for Precise Scene Text Interpretation

Chuhui Xue et al., [28] proposed an Image to character to word transformers for accurate scene text recognition. To tackle these challenges, a novel Image-to-Character-to-Word (I2C2W) Transformer-based model has been designed. This approach introduces a two-stage recognition process such that Image-to-Character Mapping (I2C) which identifies potential character candidates and rates their relative locations within a word. Character-to-Word Mapping (C2W) that uses the detected character candidates to refine and anticipate the eventual word, boosting accuracy by changing semantic linkages.

This method abolishes the demand for character-level annotations and heuristic grouping criteria, making it more robust in detecting scene text in distorted, curved, or low-resolution scenarios. The transformer-based architecture successfully establishes spatial relationships and character links. It boosts overall recognition accuracy. Additionally, the recommended technique lowers dependency on predetermined lexicons, enhances its versatility for real-world applications such as smart surveillance, retail automation, and augmented reality.

Advantages:

- Better performance is achieved compared to current methods, especially for text images that are distorted, curved, or of low quality.
- Training is made easier than segmentation-based methods because expensive character-annotations is not needed.
- A unified teaching technique is used, making the method more efficient for real-world applications. It works well with a wide range of datasets, including bending, multi-oriented, and normal text datasets.

Disadvantages:

- Performance depends on successful pre training on large-scale datasets, requiring considerable computer resources.
- While powerful against moderate noise, it may struggle with ultra-low-resolution or severely occluded letters.
- The additional processing steps (I2C + C2W) may increase latency, making it less suitable for real-time apps.

2.3 Progressive Multi-Scale Network for Enhanced Image Deblurring

Minyuan Ye et al., [29] proposed a progressive multi-scale network for enhanced image deblurring which is a new deep learning-based technology works continuously to restore clear images from noise images inputs using an iterative up scaling approach. Not like standard multi-scale deblurring designs, which work with a known number of levels, SIUN dynamically changes its processing depth according on the strength of the blur, making it better to real-world scenarios. Other than working with typical up sampling layers, a super-resolution (SR) framework is added into the model to help replace small characteristics lost in the blurring process, considerably increasing visual clarity. One of the main objects of SIUN is its curricular learning method, by which the hardness of the deblurring job is constantly increased throughout training, allowing the network to learn little by little and gather faster convergence with better generalization.

A scale-iterative approach is applied by the model, where images are first evaluated at a reduced resolution before being gradually up scaled through multiple refinement steps, considerably decreases processing costs while maintaining high-frequency data. This methodology renders SIUN highly flexible, with remarkable working exhibited across numerous domains, including text images, facial images, and natural scene photographs, where the conservation of structural integrity is crucial. Extensive testing on benchmark datasets and real-world blurry images have stated that state-of-the-art performance is produced by SIUN, outperforming preceding methods in both qualitative and quantitative dimensions.

And some parameters are needed by it than by scale-cascaded structures, as weights are shared across repetition, making it computationally better. This work emphasizes the potential of SIUN for multiple practical applications, such as document reinstatement, face picture enhancing, and normal photography deblurring, opening the way for further developments in adaptive image restoration systems**.**

Advantages:

- Uses a super-resolution network alternative of simple up sampling, helping to replace fine image details more effectively.
- The model decreases the number of parameters distinguished to traditional multi-scale deblurring networks, improving better computational benefits.
- Tasks well on text images, face images, and natural scene images, considering it useful for diverse real-world applications.

Disadvantages:

- The continuous upscaling process needs numerous passes, making it more computationally needed than single-pass deblurring models.
- Difficult to implement and makes it high complex.
- Meanwhile effective for most blurry images, it may suffer with extremely low-resolution or severely distorted images.

2.4 Diffusion-Driven Dual-Phase OCR for IIoT Applications

Chae-Won Park et al., [30] investigated on diffusion-driven dual-phase OCR which is a determining technology in Industrial Internet of Things applications, enables the extraction of textual data for active manufacturing, agriculture and healthcare. Moreover, analyzing text in low-determination images is highly crucial due to blur, image noise, poor lighting, distortions, and varying orientations. To overcome this issue, OCR-Diff is carried out in this work as a two-stage deep learning framework that influence generative diffusion models to overcome OCR accuracy in worst environments.

In the initial stage, a customized conditional U-Net architecture is trained before using a forward diffusion process, where noise is continuously added to an image and then drive back using a reverse diffusion process. The quality of low determination text images is significantly improved in this working method, with fine details being restored that traditional super-resolution methods fail to overcome. The second method stage involves the fine-tuning of the previously trained network along with a feature extractor, and its optimization for an off-the-shelf text recognizer to come up with precise text identification. Not like before deep learning approaches, which provide only marginal quality development, a text-aware diffusion model is implemented by OCR-Diff, which better conserves character structure and readability.

The proposed model is evaluated on the TextZoom dataset, where it is found to outperform existing state-of-the-art OCR enhancement models in both image quality restoration and text recognition accuracy. Unlike general diffusion-based super-resolution models which tend to distort text structures, textual integrity is maintained by OCR-Diff, ensuring high OCR performance even in extremely degraded conditions. Extensive experiments are conducted to validate its superiority in handling industrial OCR challenges, making it a robust and effective solution for real-world IIoT applications.

Advantages:

- The image quality is enhanced during the pretraining stage, while recognition performance is refined during the fine-tuning stage, ensuring higher accuracy and robustness.
- Blur, noise, lighting variations, and distortions are handled, making it ideal for smart manufacturing, agriculture, and surveillance applications.
- Text-awareness is maintained by OCR-Diff, ensuring that character shapes and spatial relationships are preserved.

Disadvantages:

- Required to undergo multiple iterations, making it slower than traditional deep learning OCR methods.

- High computational resources are required for integrating diffusion models with OCR pipelines
- Handwritten text or artistic fonts are a challenge for the model to accurately extract.

2.5 Token-Aware Multimodal Retrieval via Aligned Contrastive Learning

Chong Liu et al., [31] proposed a work on token-aware multimodal retrieval via aligned contrastive learning. Unlike conventional super-resolution models, textual integrity is mainly observed on being preserved by OCR-Diff, with recognition working ability being improved in industrial environments. The image quality is improved during the previous training module, while recognition performance is redefined during the fine-tuning stage, able to produce higher accuracy and robustness. The fine text details are re modularization by the reverse diffusion process, outperforming conventional GAN-based and CNN-based image enhancing methods. Blur, noise images, lighting variations, and distortions are handled, making it ideal for smart manufacturing, agriculture, and surveillance applications.

State-of-the-art OCR models on TextZoom are surpassed, demonstrating its working condition across various OCR architecture. Not like other diffusion models that able to distort text structure, text-awareness is maintained by OCR-Diffusion, capable that character shapes an Image-text retrieval is a fundamental task in multimodal machine learning, requiring models to precisely align textual contents with corresponding images. Conventional approaches are often worked upon, either using coarse-grained retrieval, which connects entire images and sentences, or fine-grained retrieval, which aligns local image regions with individual words. Meanwhile, both methods have disadvantages coarse retrieval lacks detailed mentioning, while fine retrieval is computationally expensive. A novel framework called Token-Guided Dual Transformer (TGDT) is introduced in this work, which merges coarse- and fine-grained retrieval into a unified approach, leveraging the advantages of both.

TGDT holds for two homogeneous branches: one for image encoding and other one for text encoding method, both works based on transformer networks. The image encoder mentions both global and local features, ensuring that large-scale scene information and object-level details are conserved. The text encoder works along sentence-level and word-level representations to enhance text-image alignment. A main innovation of this work is the

Consistent Multimodal Contrastive (CMC) loss, which is capable to semantic consistency across intra- and inter-modal representations in the embedding space.

Advantages:

- Global and local semantic correlation between images and text are simultaneously encapsulated, improving retrieval accuracy for the work.
- Already available retrieval models on Flickr30K and MS-COCO are outperformed, considering superior accuracy and efficiency.
- The decreased computational cost makes it ideal for real-world search engines.
- Complex scenes such as noisy backgrounds are handled well.

Disadvantages:

- Significant computational power is still required for local re-ranking.
- Large-scale pre training is relied upon by the transformer-based model, making it resource-intensive for smaller datasets.
- Careful fine-tuning is required for the CMC loss function.

2.6 Universal CNN Architecture for Detecting and Stylizing Scene Text in Social Media Content

Palaiahnakote Shivakumaran et al., [3] proposed a universal CNN architecture for detecting and stylizing scene text in social media content. Social media images are often overblown by compression artifacts, noisy images, and resolution reduction, making text detection and style transfer more complexing, mainly for multilingual and non-Latin scripts. A novel deep learning-based framework is performed in this work, which integrates text detection and style transfer into a single end-to-end model, productively handling low resolution social media images. The proposed architecture holds for two key components they are EffiUNet++ for text detection and TESP-Net for text style transfer. EffiUNet++ merges EfficientNet and UNet++ architectures to conservatively detect text regions, even in low-quality images with complex backgrounds and noisy images, meanwhile a differential binarization head is also build in this to improve text boundary precision.

TESP-Net, a generative adversarial network (GAN)-based style transfer model, collects both the structural and stylistic features of text using Target Shape Extraction (TSE) and Style Feature Extraction (SFE) modules, confirms that the original text characteristics are preserved during transformation stages. Not like conventional text detection methods that works hard with varied scripts and artistic fonts, this model is made language-independent, holds for multiple scripts, including English, Arabic, Devanagari, Bengali, Cyrillic, and Kana. And thus a new social media text dataset is also introduced to train and evaluate the model , with results showing that it is outperformed by the existing state-of-the-art methods on benchmark datasets like ICDAR, STEFANN, and TE141K. Experimental evaluation results ensures that superior accuracy in both text detection and style transfer is obtained by the proposed framework, making it a strong solution for applications in social media content processing, digital forensics, and artistic text generation.

Advantages:

- Numerous scripts are supported, which enables that text style transfer works across various languages without the need for language-specific models.
- Accurately finds out even when collapsed with cluttered social media backgrounds.
- Effective in real-world applications.
- Text structure and style are accurately conserved.

Disadvantages:

- Large amount of GPU resources is needed due to the integration of frequency-domain processing, EfficientNet and UNet++.
- The model is weaker than simpler CNN-based text detection models due to its two-stage text processing approach.

2.7 Scene Text Enhancement with Prior-Guided Super-Resolution Techniques

Jianqi Ma et al., [4] developed a novel framework called scene text enhancement with prior-guided super-resolution techniques to enrich low-resolution scene text images while parallelly improving Optical Character Recognition (OCR) accuracy. Not like conventional super-resolution (SR) methods that treat text images as natural images, text recognition priors are submerged directly into the super-resolution process by TPGSR, making it more consistent in restoring text clarity.

The main concept behind TPGSR is the need of text recognition priors gathered from a previously trained text recognition model, which gives categorical information data about the text characters. These priors serve as manual for reconstructing high-resolution (HR) images, enables that text readability is increased beyond simple upscaling.

To further re define the methodology process, a multi-stage refinement strategy is carried out, where the output HR image is used iteratively to refine the text prior and improve the next super-resolution cycle. This projects in the progressive enhancement of image quality and OCR performance over multiple stages. The architecture of TPGSR holds up for two main components such as Text Prior Generation Branch which is a previously trained text recognition model that is used to generate text priors from the input LR image. The other one is Super-Resolution Branch where the LR image and the result in text prior are taken as inputs, and an enhanced HR image is re developed.

Main module in this framework is the Text Prior Transformer, which combines the text prior features with the super-resolution network to produce better text re processing. Experimental validation is carried on the TextZoom dataset, where it is delivered that TPGSR significantly increases both image quality and OCR accuracy compared to existing STISR models.

Advantages:

- Enables character clarity and significantly increases recognition accuracy in hard text images.
- Improves text visibility and recognition working.
- TPGSR leverages text structure and semantics, enabling correct character shapes and positioning in the result.

Disadvantages:

- This method is hard to implement and optimize.
- It may toil with severely distorted, artistic, or decorative fonts.

2.8 Adaptive Latitude Network for Enhancing Omnidirectional Image Resolution

Xin Deng et al., [32] worked on adaptive latitude network for enhancing omnidirectional image resolution where the omnidirectional images are mainly used in applications such as virtual reality, augmented reality, and 360-degree panoramic imaging, but they often tend to struggle from low resolution due to holding capacity, transmission, and processing constraints. Conventional super-resolution (SR) methods, designed for 2D planar images, stop to account for the geometric distortions and uneven pixel density present in ODIs, making them not effective for high-quality reconstruction. To address this, LAU-Net+ (Latitude Adaptive Upscaling Network) is established in the project as a novel deep learning framework tailored for Omnidirectional Image Super-Resolution (ODI-SR). Not like conventional SR models that uses a uniform upscaling factor across a whole image, different scaling factors are dynamically assigned to various latitude bands by LAU-Net+, enables that low-latitude regions, which hold more details, are enhanced with greater precision, meanwhile computational resources are decreased for high-latitude regions, where distortion is more used.

This methodology is built upon a multi-level Laplacian pyramid structure, which continuously refines image resolution in numerous stages, enabling for better feature extraction and reconstruction of high-frequency data. An important innovation in LAU-Net+ is the Drop-Band Decision Module (DDM), which ensures reinforcement learning to find and destroy redundant high-latitude bands dynamically, significantly reducing computational overhead without reducing image quality. And moreover, the High-Latitude Enhancement Module (HEM) is established to refine and enable the dropped bands through a lightweight processing unit, enabling that crucial visual information is not lost.

Advantages:

- The Drop-Band Decision Module eliminates unwanted high-latitude bands, reducing computation while holding high image quality.
- Utilizes reinforcement learning to search the optimal upscaling factors, maintaining image quality and processing efficiency.
- Low-resolution images are gradually enhanced, reducing artifacts and distortion.

Disadvantages:

- Even though optimized, the multi-level framework and reinforcement learning scheme still asks for significant processing power.
- This methodology requires hard training procedures.
- This methodology relies heavily on the diversity and quality of the training dataset.
- The model fails to accurately extract text for images that are heavily occluded.

2.9 The Reordering Based Decoding Strategy for Scene Text Recognition Efficiency

Dajian Zhong et al., [5] introduced the reordering based decoding strategy for scene text recognition efficiency where the given occlusion, low resolution, and noisy images in real-world text photos, scene text recognition is referred as a difficult task. Main current recognition method works on a set left-to-right decoding sequence, which omits contextual information when handling less-quality or obstructed characters. This work presents NDOrder, a unique scene text recognition algorithm that decodes clear characters first so that acquired contextual information may predict obstructed or distorted letters.

Under a Vision Transformer (ViT) encoder-decoder architecture, the suggested approach extracts visual features utilizing a ViT-based encoder. Two original modules are included into the decoder unlike conventional techniques: This module generates a random decoding order instead of a rigid left-to- right sequencing, therefore ensuring that the model learns to decode characters dynamically. Secondly, module for Vision-Content-Position (VCP) Strong relationships between character location, content, and visual information are developed, therefore enhancing the recognition accuracy for low-quality or obstructed text.

Furthermore, included is the OLQT (Occluded and Low-Quality Text) dataset, which comprises difficult text pictures with distortions and occlusions. Experimental results on OLQT and other benchmark datasets reveal that NDOrder outperforms state-of-the-art scene text recognition methods. Unlike conventional systems that depend exclusively on sequential feature processing, clear text sections are prioritized by NDOrder, leading to increased recognition accuracy and better handling of occluded text.

Advantages:

- Increased confidence, explicit characters are prioritized before working occluded or degraded characters, reducing mistakes.
- Allows better finding for distorted text.
- Increased accuracy than conventional method left-to-right decoding models is gained.
- Provide good visual feature representation, increasing text recognition across diverse backgrounds.

Disadvantages:

- Higher computational load.
- When high accuracy is achieved, speed is relinquished for better recognition.
- Performance on other scripts (e.g., Chinese, Arabic) remains less than ideal.

- The integration of ROG and VCP modules needs specialized training methods, making real-world deployment challenging.

2.10 Accurate Container Text Reading at Terminals through the Line Mask Augmented OCR.

Zhichao Zhang et al., [33] introduced accurate container text reading at terminals through the line mask augmented OCR is addressed as having a main role in automating container terminal process, improving efficiency in container tracking, logistics, and port administration. Moreover, specific problems such as skewed text, text stickiness, and hard container fonts are worked by OCR in container terminals, which diminish recognition accuracy. In this work, an Enhanced OCR (EOCR) system is constructed, built specifically to overcome these issues by combining Line Segmentation Mask (LSM)-based detection with Scanline-based recognition. The proposed LSM-based detection methodology is developed upon compared to standard OCR by segregating overlapped and warped text lines, thus reducing mistakes caused by previous segmentation methods. Furthermore, a Scanline-based recognition algorithm is included to increase text processing performance, enabling that real-time OCR operations may be done efficiently in busy port scenarios. Additionally, the Arbitrary Angle Quadrilateral Fitting (AAQF) technique is given, enabling more correct localization of container text that appears at sloping angles due to camera location and uneven container geometries.

To correct the productiveness of the proposed approach, extensive experiments were performed using a dataset of 12,000 container photos needed from the Shanghai Port. State-of-the-art performance was done by the EOCR system, obtaining a recognition accuracy of up to 98.7%, greatly surpassing standard OCR methods. Ablation research moreover confirmed the individual offering of the LSM module, the AAQF algorithm, and the Scanline-based identification method, revealing their collective effect on increasing OCR efficiency in container terminals.

By overcoming crucial issues in real-time container text recognition, a better OCR solution is provided by this research that can increase throughput, overcome processing delays, and provide automation in port operations. The discovery shows that the EOCR system can potentially be expanded to other industrial applications where OCR must come up with distorted, overlapping, or irregularly positioned text.

Advantages:

- A recognition accuracy of 98.7% is obtained.
- Reducing faults in detection and segmentation.
- Skewed and rotated text is processed correctly, increasing detection for text at different angles.
- Accurate recognition is enabled under hard environmental conditions common in ports and logistics methods.

Disadvantages:

- Needs high-performance hardware for real-time processing.
- Extensive parameter tuning is needed to widen the performance of segmentation.
- Toil when working images with noise or highly compressed text regions.
- Reduce adaptability to OCR applications with less structured or unannotated datasets.

2.11 Generative Scene and Tattoo Text Spotting via Frequency Domain Learning

Ayan Banerjee et al., [6] investigated on generative scene and tattoo text spotting via frequency domain learning is used for Tattoo Text Spotting (TTS) is carried out in this work, convey an upcoming and complex problem in text recognition. No like standard scene text recognition works, tattoo text spotting is described more struggle due to calligraphic handwriting, artistic designs, skin distortions, and complex noisy images.

To overcome these problems, an end-to-end text spotting method is established that provides Hilbert Transform (HT) for fine detail identification, which is used to extract important curved edges and data when reducing background complexity. Optimum Phase Congruency (OPC) is used to improve key text-related features while stamp out background noise images. A Generative Adversarial Network (GAN) is developed to generate synthetic text samples for effective model working, pay back for the lack of huge tattoo text datasets.

The GAN-based text spotting framework is processed using a Tattoo Text Spotting Dataset (TTSD), specifically introduced for this work. Furthermore, tests are conducted on benchmark scene text datasets, including Total-Text, CTW1500, and ICDAR2015. The proposed methodology is shown to better perform state-of-the-art methods in both tattoo and general scene text spotting. This is the initial method specifically constructed for tattoo text spotting, which has forensic and identification applications. The unification of Hilbert

Transform is introduced to enhance text structures and overcome the background complexity. GAN-based data generation is used to make the work stronger to variations in tattoo text styles. A new Tattoo Text Spotting Dataset (TTSD) is established for further work in this domain.

Advantages:

- This method is tailored for tattoo text, better handling calligraphy, distortions, and complex backgrounds.
- Text is identified and recognized even in low-resolution and irregularly shaped tattoos.
- Generalize well to different types of text spotting tasks.
- Improve model robustness.
- Higher accuracy than previous approaches on tattoo text datasets is achieved.

Disadvantages:

- Increases processing time compared to traditional text spotting models.
- Challenges are faced with highly distorted or occluded characters.
- Recognition errors may be introduced by skin distortions and varying tattoo placements, necessitating additional image enhancement techniques.

2.12 Reinforcement Masking Strategy for Scene Text Detection

Chuang Yang et al, [34] proposed a novel method named reinforcement masking strategy for scene text detection. Conventional text detection models often toil with hard backgrounds images and text of arbitrary shapes. RSMS focuses to address this by moderately diminishing the text area by reinforcement learning, allows more precise boundary estimation for uneven and curved text instances. This is mainly important in conditions where accurate text outline affects the downstream text recognition work.

The RSMS module is unified into a detection pipeline, where it comes up to bring out masks that shrink text regions adjustable. A reward function is planned to encourage the model to work mainly on critical regions of text while reducing background noise of the images. This technique supports both horizontal and arbitrarily-shaped text, which betters flexibility and versatile over standard segmentation-based methods.

By encompassing reinforcement learning, the system imitates human-like iterative refinement of found out regions. The moderate mask reducing helps to separate core text regions, improving performance on hard benchmarks such as ICDAR 2015 and Total-Text. The method surpasses many existing techniques in terms of accuracy and robustness across many datasets.

Advantages:

- Increased detection precision
- Increased Robustness.
- Background interference has been remarkably reduced.

Disadvantages:

- Increased computational need.
- Hyper parameter tuning is needed.
- Real-time deployment is hard

2.13 Unified Transformer Architecture for End-to-End Multi Scale Scene Text Parsing.

Tianyu Geng [35] presents a unified transformer architecture for end-to-end multi scale scene text parsing. Not like traditional pipelines that serve detection and recognition as individual stages, this way engages a multi-scale encoder-decoder process, allows shared feature learning for the two tasks. This combination helps decrease information loss and helps better performance in real-world applications.

A transformer backbone is used for its higher modeling of long-range dependencies and determined relationships in images. The network works along multi-scale features through distortable attention, allowing it to localize and make clear text of varying sizes and fonts productively.

Detection is mentored by an anchor-free way while recognition is carried through a sequence modeling decoder. This tight incorporating leads to mutual enhancement between tasks, and performance evaluations reflect increased accuracy when distinguished to many two-stage or separate systems.

Advantages:

- Text detection and recognition have been worked as a single end-to-end model.
- Texts of various sizes and fonts are extracted accurately.

Disadvantages:

- High memory consumption has been noted.
- Higher computational resources are in need.
- Over fitting is carried on smaller datasets.

2.14 SUMMARY

This chapter has given the description about the literature survey for various existing techniques involved in scene text recognition. Next chapter deals with the system study for the proposed system.

CHAPTER 3

SYSTEM STUDY

3.1 OVERVIEW

This chapter deals with the detailed description of the proposed system and how it overcome the drawback in the existing system.

3.2 EXISTING SYSTEM

In earlier, text recognition systems have been developed using multiple deep learning techniques, such as CNN (Convolutional Neural Networks), LSTM (Long Short Term Memory), and Transformer. High accuracy in text detection and recognition has been achieved by these methods according to their visual features like shapes and orientations. However, one limitation of these models is that curved and multi-oriented text may not be correctly recognized. For instance, slightly curved text on the image with different orientation is not overcome by other models. This is because the model is relied upon solely for linear text present in the image. Therefore, better and accurate text recognition systems still need to be constructed that can account for this text recognition and produce more precise outputs.

3.3 PROPOSED SYSTEM

High accuracy is aimed to be achieved by our proposed text recognition system through the integration of both low-level features and high-level concepts. The shapes and orientations of the text, which are the basic visual characteristics used by most text recognition systems, are included as low-level features. In addition to these features, high-level concepts are held for the text recognition phase so that better performance is ensured in rotary, low-resolution, and multi-oriented texts. Multi-oriented texts are referred to as the text that is not only present as horizontal texts but also appears at multiple angles and orientations or even vertically within an image. By both low-level features and high-level concepts being integrated, greater accuracy in recognizing texts can be achieved, as variations in low-resolution image appearance and other factors that may affect text identification can be accounted for.

The system is introduced mainly to work with extracting text from images in real-time scene text images and is developed around the transformer networks. It is aimed to recognize and extract text productively while imperfections in existing methods are discussed. A main goal of this system is set to reduce reliance on large character-level annotations and to overcome the complexities that are brought up by multiple backgrounds and distorted text. By the search being narrowed down to specific collections of texts, text can quickly and accurately be identified and best matched with the ground truth for the straightforwardly mentioned text in the images. This approach is assumed particularly needful in conditions where real-time applications such as serial number detection on harbour containers are involved.

3.4 TEXT DETECTION

At the main phase of the work, the Text Detection Module is taken as the main role in identifying and then detecting the text regions within a provided image. This phase is drafted mainly to be robust and is held capable of picking up text that is emerged in multiple orientations and under various distortions. The detection process is divided into two main components.

The first component is worked on for the generation of a shrink mask. This phase is used with a ResNet50 architecture [36] as its backbone for feature extraction. The shrink mask is presented in the form of a binary representation, representing the probability of each pixel being part of a text region. In addition to the shrink mask, a reinforcement offset map is introduced, which computes the nearest distance from each pixel to the nearest text contour. This additional layer of data is used to enhance the accuracy of text localization by giving regional context. By feature maps from various stages of the ResNet being combined, high-level semantic insights are effectively merged with low-level features, improving the ability to find out text even in hard conditions.

The second phase is introduced as the Super Pixel Window (SPW), which is used to increase the model's prospective to distinguish text pixels from close background areas. This is considered mainly useful in situations where the variance between text and background is found to be fine. The SPW is brought into work during the training phase to filter the model's learning process, but it is not used during speculation, which helps to smooth the detection process and increase speed. By the regions between the shrink mask and the text contour being handled as background, issues like “text adhesion,” where nearby text instances might incorrectly be combined, are helped to be stopped. This method not only is used to increase the accuracy of text detection but also brings down false positives, assuming that the model is focused on genuine text regions.

3.5 TEXT RECOGNITION

The Text Recognition Module is assigned the responsibility of converting the localized text regions into coherent text sequences. The power of vision transformers is tackled in this phase to find out characters in parallel, improving both the speed and accuracy of text recognition remarkably. Rather than conventional word embedding being depended upon, the loaded image is split up into 2D patches, which are then flattened for refining by the transformer. This method allows the model to progressively have spatial relationships detained among various parts of the image, smoothing the recognition of characters in multiple orientations.

The multi-head self-attention mechanism is made use of by the vision transformer, which permits various areas of the image to be provided attention at the same time. This potential is considered critical for admitting characters that may appear in multiple orientations or combinations. A series of characters in the correct order and length is trained to be come up with by the model, using special tokens to represent the start and end of the text. An order of vectors is held in the final outcome from the vision transformer, each constituting the probability of a character. To have the predicted sequences come across, a CTC decoder [19] is used, which gradually addresses hard challenges such as non-matching characters and padding tokens. Variable-length outputs are allowed to be produced by the CTC decoder, making it particularly appropriate for text recognition tasks where the length of the text can differ notably.

The training process is constructed to reduce an overall loss function that merges the losses from both the text detection and recognition works. The loss related to text detection is reduced using a dice loss function for the shrink mask, while the recognition loss is calculated using cross-entropy loss for character predictions. This unified approach is made sure to have both modules trained to accessory each other, leading to enhanced overall performance. The use of a merged loss function not only is used to smooth the training operation but also is motivated to have the model study features that are useful for both detection and recognition, resulting in a more balanced system.

Furthermore, the model’s strength is increased by the incorporation of artificially generating new data from existing data methods during training. By multiple scenarios such as low resolution, rotary, and distortions being come across, the model is taught to identify and recognize text under a broad range of conditions. This not only enhances the model's production on the training dataset but also is used to improve its capability to observe unseen data, making it more productive in real-time applications. The augmentation techniques employed can be taken in as random cropping, scaling, and color adaptation, which help the model to become unchangeable to common variations that are brought up in real-world image applications.

The need for real-time working potential is also mentioned by the proposed system. In scenarios such as independent vehicles or augmented reality, the power to have text fast and accurately drawn out from images is needed for timely details to be provided to users.

Infinite and varied applications are considered possible for this model. In the domain of autonomous driving, for example, the way to have road signs and navigation aids read and spelled out can progressively improve vehicle well-being. In augmented reality, appropriate details can be covered on real-world text applications, enhancing user relations and experiences. Moreover, in content-based image capturing, the ability to have text collected and indexed from images can enhance search ability and information retrieval processes.

At last, the hard situations of scene text recognition are labeled completely by the proposed system for rotation-invariant scene text extraction. Enhanced deep learning techniques are used and attention is given to the working of detection and recognition modules, so that the system is developed well to manage the complexities of real-world text extraction tasks.

3.6 SUMMARY

This chapter discussed about the existing models and its limitations, followed by a description of the proposed model and its advantages. The next chapter will deal with the detailed description about the proposed system and explanation of the various models utilized within the system.

CHAPTER 4

SYSTEM DESIGN

4.1 OVERVIEW

In this chapter, system architecture, and various modules involved in the work are discussed.

4.2 SYSTEM ARCHITECTURE DESIGN

A robust text extraction model has been designed to detect texts and extract the texts contained in an image. The text extraction problem has been divided majorly into two tasks – text localization (detection) and text recognition. Two individual modules have been designed to solve these two tasks.

The proposed text extraction model has been specifically designed to accurately detect and crop text region on an image with complex background and text distortion. The proposed model is based on a robust vision transformer network, as illustrated in figure 4.1. This diagram is used as a blueprint for understanding the overall functioning and structure of the proposed solution, with key components being highlighted that contribute to its effectiveness in the identification and extraction of texts from images.

The architecture diagram of the proposed system is depicted in figure 4.1 and discussed below.

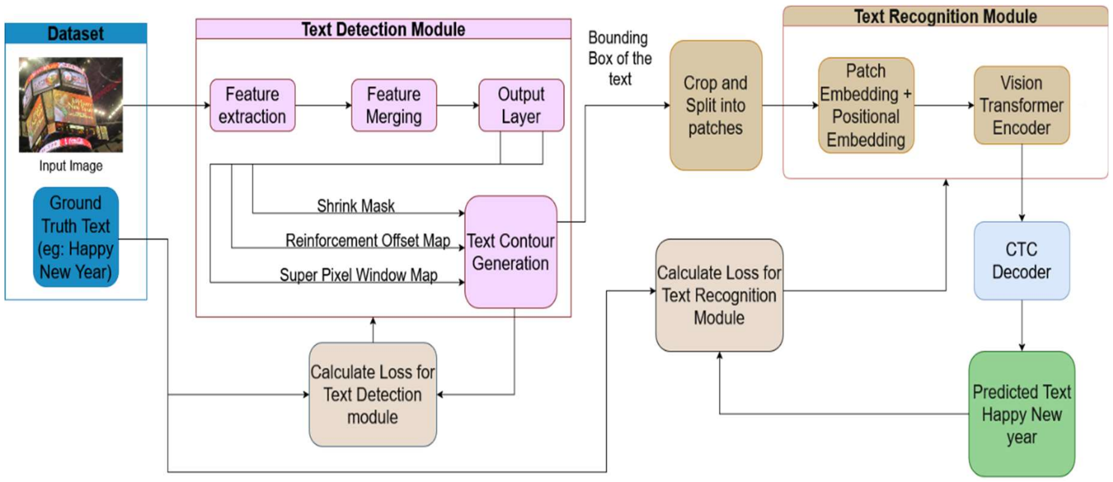

Dataset
Input Image
Ground Truth Text (eg: Happy New Year)
Text Detection Module
Feature extraction
Feature Merging
Output Layer
Shrink Mask
Reinforcement Offset Map
Super Pixel Window Map
Text Contour Generation
Calculate Loss for Text Detection module
Bounding Box of the text
Crop and Split into patches
Text Recognition Module
Patch Embedding + Positional Embedding
Vision Transformer Encoder
CTC Decoder
Calculate Loss for Text Recognition Module
Predicted Text Happy New year

Figure 4.1 Rotation-Invariant Scene Text Extraction Architecture Diagram

The text extraction system is trained on various scene text images from many publicly available datasets. The features from the image are extracted using a CNN backbone like the ResNet50 architecture [19]. The extracted features are merged using a Feature Pyramid Network [37]. Three tensors—shrink mask, reinforcement offset map, and super-pixel window map—are mapped from the merged features. The predicted shrink mask and reinforcement offset map are used to generate contours of text regions as a binary mask. The predicted text regions are cropped out and fed into the text recognition module for text extraction. A Vision Transformer model [2] is employed to accurately predict the text in the cropped image patch.

4.3 FEATURE EXTRACTION

The publicly available scene text image datasets are included, with images containing texts that are synthetically added to them to simulate complex environments and images that are captured in various public places, containing texts in various orientations and perspectives. The visual features from these images are extracted using a CNN backbone like the ResNet50. The ResNet50, which was originally proposed for image classification, is known to contain classification layers. The architecture diagram of ResNet50 is provided in Figure 4.2.

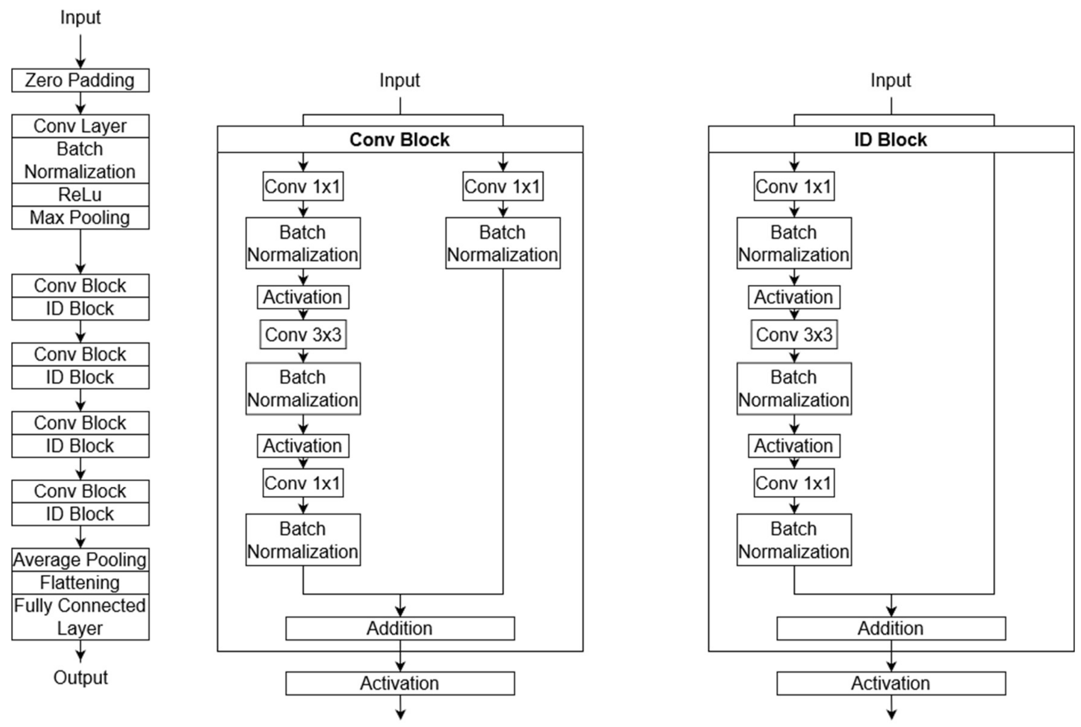


Figure 4.2 ResNet50 Architecture Diagram

The proposed system is designed to use the ResNet architecture as a feature extractor, and the classification layers are not needed. Hence, they are removed from the proposed architecture. The ResNet, or residual network, is structured to contain multiple residual blocks. These residual blocks are incorporated with skip connections. These connections are allowed to preserve information from earlier layers, which helps the network learn better representations of the input data. The skip connection is simply designed to perform an addition of the identity function to the output, as shown in the ID block in Figure 4.2.

$$y = x + F(x) \tag{4.1}$$

Each Convolution layer is followed by an activation function like ReLU and batch normalization. The ReLU activation function is given as following

$$f(x) = \begin{cases} x \ \ if \ x > 0 \\ 0 \ \ if \ x \leq 0 \end{cases} \tag{4.2}$$

The batch normalization is given in equation 4.1

$$\hat{x} = \frac{(x - \mu)}{\sqrt{(\sigma^2 + \varepsilon)}} \tag{4.3}$$

where x denotes input, μ denotes the mean value, σ^2 represents the variance and ε denotes epsilon constant added for numerical stability.

The ResNet is used to generate multi-level features whose sizes are ¼, 1/8, 1/16, and 1/32. These multi-level features capture high-level semantic information and low-level apparent features at various resolutions. The backbone is pretrained on the ImageNet dataset for better semantics.

4.4 FEATURE MERGING

The feature merging branch is mainly composed of convolutional layers and upsampling operations. The exact architecture of the merging branch is shown in Figure 4.3. In a Convolutional Neural Network architecture such as ResNet, low-level information such as edges, corners, and small objects is captured in the early layers, while high-level abstract features like complex shapes and objects are extracted in the deeper layers.

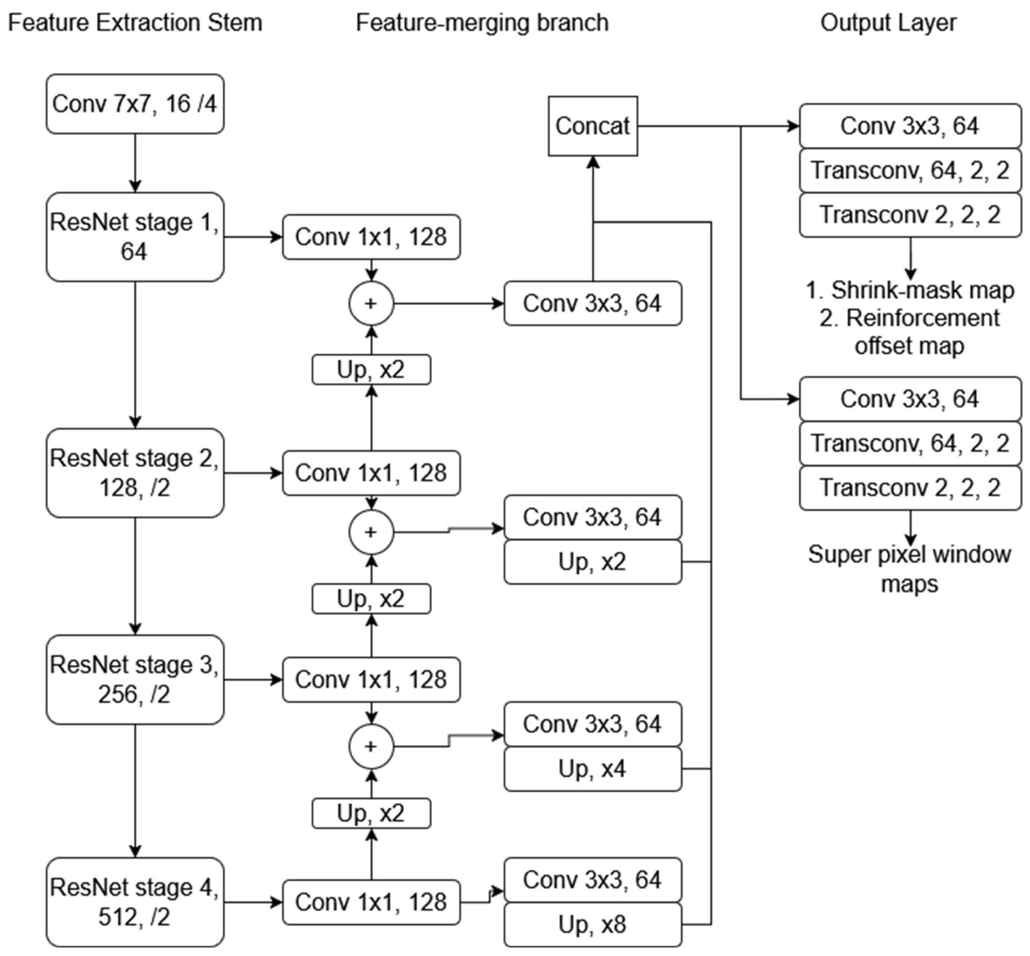


Figure 4.3 Text Detection Module Architecture Diagram

The multi-scale features are merged using a Feature Pyramid Network. As a result, the fused feature map contains information about both low-level and high-level features, leading to a better representation of the image. The Feature Pyramid Network is also designed as a lightweight merging branch, which significantly reduces computational cost and thereby improves inference speed. The multi-level features are merged as follows

$$H_i^1 = conv_{1x1,128}(F_i), \ i = 1,2,3,4 \tag{4.4}$$

$$H_i^2 = \begin{cases} H_i^1, \ i = 1 \\ up_{x2}(H_i^1) + H_{i-1}^1, \ i = 2, \ 3, \ 4 \end{cases} \tag{4.5}$$

$$H_i^3 = \begin{cases} conv_{3x3,\,64}(H_i^1), \; i = 1 \\ up_{x2^{(i-1)}}\left(conv_{3x3,\,64}(H_i^2)\right), \; i = 2,\, 3,\, 4 \end{cases} \tag{4.6}$$

$$H_f = concatenate(H_i^3), \; i = 1,\, 2,\, 3,\, 4 \tag{4.7}$$

Where F_i is the feature map from the feature extracting stem. H_i^1, H_i^2, H_i^3 are the hidden feature maps. H_f is the fused multi-level feature map. This fused feature map provides a better representation of the image. $Up_2^{(i-1)}$ denotes the upsampling operator which upsamples the feature map to $2^{(i-1)}$ times the size of the original feature map. The hidden feature maps at each stage of the ResNet are upsampled to the same size and then concatenated to form the fused feature map. A Conv 1x1, 128 is a convolution layer with kernel size 1x1 and output size 128.

4.5 TEXT CONTOUR DETECTION

The fused feature map is used to predict three tensors known as the shrink mask map, the reinforcement offset map, and the super pixel window map (SPW). A combination of convolution layers and transpose convolution layers is employed to predict the three maps. The exact architecture for the output layer is illustrated in Figure 4.3. As shown in the diagram, a tensor of size $\frac{H}{4} * \frac{W}{4} * 2$ is predicted by the first network. This tensor consists of the predicted shrink mask map and the reinforcement offset map each of size $\frac{H}{4} * \frac{W}{4} * 1$. The second network predicts a tensor of size $\frac{H}{4} * \frac{W}{4} * 9$ which is composed of 9 SPW maps.

The text contour can be accurately generated using the combination of the predicted shrink mask and the reinforcement offsets by extending the shrink mask contours outward by the reinforcement offsets. This method allows for accurate prediction of the text contour even if the shrink mask deviates from the ground truth. Since the prediction of the reinforcement offset forces the model to recognize the text shapes and orientations, thereby preventing interference with other text regions.

Existing shrink mask methods extend the mask outward by a fixed offset calculated as below

$$o_f = \frac{S_s}{L_s}\delta_t \tag{4.8}$$

Where S_s and L_s are the area and perimeter of the predicted shrink mask. The offset o_f will deviate from the ground-truth when the predicted shrink mask is larger or smaller than the ground-truth, leading to interference or lack of complete information. The proposed model fixes this issue by predicting the offset o_r itself.

$$O_r = TConv_{2x2,1}(TConv_{2x2,64}\left(conv_{3x3,64}(H_f)\right)) \tag{4.9}$$

$$o_r = \frac{O_r^{i,j} + O_r^{k,v}}{2} \tag{4.10}$$

The third tensor that is predicted by the text detection module is the Super Pixel Window (SPW) map which is predicted by a network of convolution and transpose convolution operations by taking the fused feature map H_f as the input. The fused feature map is passed through a 3x3 convolution layer with 64 outputs, a transpose convolution layer of output size 64 and kernel 2x2 and stride 2 and another transpose convolution layer of output size 9 and kernel 2x2 and stride 2 to finally give a tensor of size $\frac{H}{4} * \frac{W}{4} * 9$.

In many images, the background is characterized by similar low-level features as the text itself, such as colour or texture. In such images, the background may be misclassified as text regions. The proposed Super Pixel Window (SPW) map is encouraged to be utilized by the network to incorporate information from surrounding pixels to enhance pixelwise predictions, thereby allowing confusion between background and text regions to be eliminated. SPW is activated only during the training phase of the model. It is used to calculate how well certain predefined areas (called anchors) overlap with simplified text regions (shrink-masks), thereby enabling the model to be guided in focusing on relevant parts. This process is designed not to affect the inference speed, as SPW is removed during inference. Shrink-masks are treated as foreground, and interval regions between shrink-masks and actual text contours are suppressed, thus allowing interference to be reduced and preventing text adhesion. This is particularly effective in scenarios where complex text layouts, such as adhesive or curved texts, are present. The SPW is defined as follows:

$$SPW = \frac{S_{(A_v \cap A_{1S})} + S_{(A_v \cap A_{2S})}}{S_{A_v}} \tag{4.11}$$

Where A_{1_s} and A_{2_s} are considered as shrink mask regions, and A_v is indicated as the region of the predefined anchor. In traditional text detection models, the network is encouraged to utilize surrounding information of each pixel by being optimized under the supervision of the Intersection over Union (IoU). IoU is given as follows:

$$IoU = \frac{max(S_{(A_v \cap A_1)}, S_{(A_v \cap A_2)})}{S_{A_v} \cup argmax_A(S_{(A_v \cap A_1)}, S_{(A_v \cap A_2)})} \tag{4.12}$$

Here the A_1 and A_2 are considered as the regions of text instances. The symbols $\cup$ and $\cap$ represent the union and intersection operations, respectively. $S_{(.)}$ defined as the area of a region. The entire text instance region is considered for the calculation of IoU. However, many interferences are introduced when long text instances are dealt with, and smaller texts are ignored as they are treated as background.

These problems are resolved by SPW due to following reasons.

1. Only the shrink masks within the range of the anchor are recognized, by which interference from long texts is avoided.
2. The interval region between the text contour and the shrink mask is treated as background by SPW. This allows shrink masks to be distinguished from the corresponding reinforcement offset.
3. Shrink masks within the range of the anchor are recognized regardless of their scales (small or large). By considering all scales, small texts are effectively detected.

4.6 LABEL GENERATION FOR TEXT DETECTION

The proposed model is trained and evaluated on publicly available datasets for scene text recognition. Most of these datasets are provided with ground truth word-level bounding box coordinates, character-level bounding box coordinates, and the ground truth text. For the training of the model to predict the shrink mask, reinforcement offset, and super pixel window, the bounding box coordinates cannot be directly utilized. Therefore, ground truth labels for the shrink mask, reinforcement offset, and super pixel window map are generated separately.

For the shrink mask, the corresponding contour is generated by moving the text instance contour inward by a shrink offset o_s which is computed using the Vatti clipping algorithm. The Vatti clipping is used to produce a new polygon (or polygons) that represent the intersection, union, difference, or exclusive-or of the input polygons. The shrink offset o_s is calculated as follows:

$$o_s = \frac{S_t}{L_t}(1 - \delta_s^2) \qquad (4.13)$$

Where S_t and L_t are the area and perimeter of text instance respectively. δ_s^2 defined as the shrinking coefficient and is set to 0.4.

For the reinforcement offset ground truth label o_r, the minimum distance between the text contour and the pixels in the text instance. Mathematically it is calculated as following:

$$o_r = \min\{\left|\left|p - p_m\right|\right|_2^2\}, \quad m = 1,2,3,..,M \qquad (4.14)$$

Where p and p_m are the coordinates of pixels of text instance and M is denoted as the number of contour points.

For SPW, shrink masks are treated as valid regions, and the corresponding pixel values of the map are computed using the formula provided above. The algorithm used to generate the ground truth labels from the boundary box coordinates given in the dataset is presented in Algorithm 5.1.

4.7 TEXT RECOGNITION MODULE

Traditional segmentation techniques are used to classify each pixel as either a text or non-text region in order to localize the position of characters in the image. This process requires large character-level annotations (coordinates of characters in the image), which are time-consuming to collect. Difficulties are also encountered in grouping the detected characters due to complex geometry and alignment. The proposed text recognition module, which is based on a vision transformer, does not require extensive character-level annotations. The network is trained to predict a sequence of characters with the correct order and length.

The text regions in a scene text image are detected using the text contour detection module, as discussed previously. The detected text regions are cropped individually to form N separate images, where N is defined as the number of texts in an image. The cropped texts are resized to 100×32. Each of these resized images is passed through the text recognition module based on the vision transformer. The vision transformer is structured similarly to the transformer architecture [1] introduced by Vaswani et al. The difference between the transformer and the vision transformer lies in the fact that only the encoder network is employed in the vision transformer. Originally, the vision transformer was introduced for object class recognition. In this case, the vision transformer is required to predict multiple characters with the correct sequence and length. The architecture diagram of the proposed vision transformer is depicted in Figure 4.4.

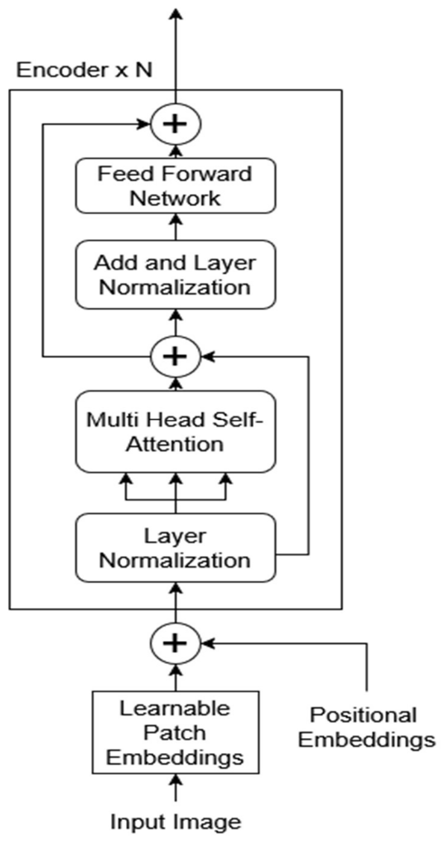


Figure 4.4 Vision Transformer Architecture

Unlike the original transformer in which word embeddings were used, the vision transformer is designed to utilize flattened 2D patch embeddings. The input image $x \epsilon \mathbb{R}^{HxWxC}$ is reshaped into flattened 2D patches $x^p \epsilon \mathbb{R}^{NxP^2C}$. The image is defined to be of size H×W with C channels. The patch dimension is set as P×P. The resulting patch sequence length is denoted as N. The embedding size is represented by D, which is treated as a tunable hyperparameter. The flattened 2D patches are linearly projected to the embedding size D. A learnable class embedding of the same dimension D is prepended to the sequence.

The prediction in a vision transformer is performed in parallel. Due to this, the positional information or the sequence order of text tends to be lost. To overcome this, a learnable positional embedding is added to the previously created embedding. The resulting vector sum is used as the input to the encoder.

The input of the encoder is normalized using Equation 4.1. The normalized input is then passed onto the attention network.

The encoder network is primarily based on the principle of attention. A multi-head self-attention network is employed by the vision transformer. The attention mechanism is used to enable the model to learn to focus on the relevant features of the image. For scene text extraction, greater attention is given to the character features. The multi-head self-attention layers are defined as follows:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (4.15)$$

Where Q, K and V refer to the Query, Key and Value respectively. d_k is dimension of Q and K. Multi Head attention layer is a concatenation of several single self-attention heads. It linearly projects queries, keys, values M times. This is defined by

$$MultiHead(Q,K,V) = Concat(head_1, \ldots, head_M)W^O \quad (4.16)$$

where, $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$.

Here W_i^Q, W_i^K, W_i^V are learnt projections of queries, keys and values.

The output of the attention network is added with the input of the network and is normalized. This tensor is passed onto a feed forward network. The feed forward network consists of two hidden layers with ReLU activation function between them.

$$FFN(x) = \max(0, xW_1 + b_1)\, W_2 + b_2 \tag{4.17}$$

A [START] token is used, instead of the learnable class embedding in the original vision transformer, to mark the start of the text prediction. Another special token [PAD] is used to indicate the end of the text and whitespace. The [PAD] token is padded to the end of each text prediction up to the maximum prediction length. Multiple vectors are extracted from the encoder, and the exact number for the maximum length is determined as the maximum length of the text in the dataset plus 2 for the [START] and [PAD] token.

The output of the vision transformer encoder is composed of a number of vectors equal to the max_length, each of size D, which corresponds to the embedding size. Each vector is mapped to a character by selecting the character with the highest probability. The detected characters may include [PAD] tokens in between or may contain duplicate characters. These issues are resolved by using a Connectionist Temporal Classification (CTC) Decoder.

4.8 LOSS FUNCTIONS

The aim of the network is to minimize the following loss function

$$L = L_{det} + L_{recog} \tag{4.18}$$

where L_{det} and L_{recog} are the loss of text detection and text recognition tasks respectively.

The shrink mask segmentation is optimized using the dice loss L_{sm}. The dice loss evaluates the difference between the binary segmentation masks and the corresponding ground truth labels.

$$L_{sm} = 1 - \frac{2\, X\, |Y \cap \hat{Y}| + 1}{|Y| + |\hat{Y}| + 1} \tag{4.19}$$

Ratio loss is used to optimize both reinforcement-offset map module and the super pixel window module.

$$L_{ratio}(P, \hat{P}) = \log \frac{\max(P, \hat{P})}{min(P, \hat{P})} \tag{4.20}$$

Where P and $\hat{P}$ are the ground truth and the prediction respectively. Hence the losses for reinforcement offset map and the super pixel window are calculated as follows

$$L_{o_r} = L_{ratio}(o_r, \hat{o_r}) \tag{4.21}$$

$$L_{SPW} = L_{ratio}(SPW, \widehat{SPW}) \tag{4.22}$$

Hence the overall text detection loss is given by

$$L_{det} = \lambda_1 L_{sm} + \lambda_2 L_{o_r} + \lambda_3 L_{SPW} \tag{4.23}$$

The $\lambda_1, \lambda_2, \lambda_3$ are tunable hyperparameters. These values are set as 1, 0.25, and 0.25 respectively.

The character prediction is a multiclass classification problem and hence cross-entropy loss is used.

$$L_{recog} = L_{CE}(P, \hat{P}) \tag{4.24}$$

where L_{CE} is the standard cross-entropy loss, P is ground-truth and $\hat{P}$ is the predicted output.

4.9 SUMMARY

This chapter explained about architecture design, modules and overall process of the proposed system. The next chapter discusses about the procedures and algorithms used in the proposed system.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 OVERVIEW

A detailed description of each of the different modules in the system design was provided in the previous chapter. This chapter discusses about various algorithms and procedures involved in implementation of the proposed system.

5.2 IMPLEMENTATION OF LABEL GENERATION FOR TEXT DETECTION

Most publicly available datasets for scene text recognition have been provided with ground truths for word-level bounding box coordinates, character-level bounding box coordinates and the ground truth text. For the text detection module, only word-level bounding box information is required. However, the coordinates should be converted to labels for shrink-mask maps, reinforcement offset and Super Pixel Window (SPW), which are explained in section 4.5.

The preprocessing procedure is designed to take an image and a set of bounding box coordinates as input and returns ground truth labels for the shrink-mask map, reinforcement offset map and Super Pixel Window as output. The pseudocode for preprocessing is depicted below.

Algorithm: Label Generation

Input: Dataset D of scene text images I, word-level bounding box coordinates B

Output: Ground truth labels for shrink-mask binary mask map, heat map for reinforcement offset and heatmap for SPW.

```
*Create empty matrices for:  shrink_mask SM, reinforcement_offset RO, SPW_heatmap SPW*

$\delta_s = 0.4$

for each image I in dataset D:

        I = resize_img(I, 640, 640)

        B = load_bounding_box(D, I)


        for each bounding_box $b_i$ in B:

            $c_i$ = extract_contour($b_i$)

            $o_s = (Area(b_i) / Perimeter(b_i)) * (1 - \delta_s^2)$

            shrink text contour inward by $o_s$

            $s_i$ = shrink_contour($b_i$, $o_s$ )

            $s_i$ = fill ($s_i$, 1)


            for each pixel (i,j) inside $b_i$:

                d_min = min($\sqrt{[(i - x)^2 + (j - y)^2]}$ for all (x,y) on contour)

                reinforcement_offset[i,j] = d_min


            scales = [2, 4, 8]

            aspect_ratios = [0.5, 1, 2]

            anchor region A = generate_achor_regions(scales, aspect_ratios)

            for each anchor region $A_v$ in A:
```

```
            a = intersection ($A_v, s_i$)
            $SPW_v$ = a / area ($A_v$)
            update SPW_heatmap with SPW_value at anchor location
            SPW [$A_v$] = $SPW_v$

    Return SM, RO, SPW
```

5.3 IMPLEMENTATION OF TEXT CONTOUR DETECTION

The text contour detection module is designed to process an input image to predict shrink-mask, reinforcement offset map and super pixel window map. The shrink_mask is extended by reinforcement offset to form the text contour.

The algorithm takes the dataset images D as input and the corresponding binary masks containing the text regions identified in the images are outputted.

```
Algorithm: Text Contour Detection
Input: D: Dataset images
Output: $b_{mask}$: binary mask of the text regions in the images
for each img in D do
    I = resize_img(I, 640, 640)
    feature_maps = ResNetBackbone(image)
    fused_features = FeatureMerging(feature_maps)
    shrink_mask, reinforcement_offset, SPW = PredictionHead(fused_features)
    extended_contour = extend_contour(shrink_mask_contour,  reinforcement_offset)
    $b_{mask}$ = zeros_like(image)
```

```
        for contour in filtered_contours:
            fill_polygon(b_mask, contour, value=1)
```

5.4 IMPLEMENTATION OF TEXT RECOGNITION

The text contour detection algorithm detected the text regions in an image. The detected regions are cropped from the image and are passed on to the text recognition module. The pseudocode for the text detection module is presented below:

Algorithm: Text Recognition

Input: *img:* Dataset image,

Output: *str*: list of detected texts in the image

```
resize image to standardized resolution (100x32)
patches = split_patches(img)
for l = 1 to num_encoders:
    $z'_l$ = MultiHeadAttention(LayerNorm($z_{l1}$)) + $z_{l1}$
    $z_l$ = MLP(LayerNorm($z'_l$)) + $z'_l$
char_logits = parallel_linear_projection(sequence_features, num_classes=96)
char_probs = softmax(char_logits)
predicted_tokens = []
for t in range(sequence_length):
    if char_probs[t] not in {[START], [PAD]}:
        predicted_tokens.append(argmax(char_probs[t]))

str = ctc_decoder(predicted_tokens)
```

```
return str
```

5.5 IMPLEMENTATION OF CTC DECODER

The CTC decoder or Connectionist Temporal Classification Decoder [19] is used to remove redundant characters, unwanted [PAD] tokens and fixes alignment issues.

```
Algorithm: CTC Decoder
Input: Probabilities P of shape $(T \times C)$, Beam Width B
Output: Best decoded sequence
Initialize Beam Set: $B_t = \{(\text{""}, 1.0)\}$
for t = 1 to T do
    Initialize New Beam Set: $B_{t+1} = \{\}$
    for each (seq, prob) in $B_t$ do
        for each character c in possible characters do
            new seq = seq + c
            new prob = prob × P(t, c)
            if new_seq exists in Bt+1 then
                $B_{t+1}$[new_seq] += new_prob
            else
                $B_{t+1}$[new_seq] = new_prob
    Prune $B_{t+1}$ to keep only top B sequences
    Normalize probabilities
    $B_T = B_{t+1}$
return the highest probability sequence in $B_T$
```

5.6 IMPLEMENTATION OF LEVENSHTEIN RATIO

The performance of text extraction is evaluated using Levenshtein ratio. The Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other. The ratio of Levenshtein distance over alignment length is known as the Levenshtein ratio. It is essentially provided as a similarity score between any two strings.

The algorithm takes two strings and a similarity score between them is returned.

```
Algorithm: Levenshtein Ratio
Input: String a of length N, String b of length M
Output: Similarity score between two strings
Initialize Matrix D of size (N+1) × (M+1)
D[0][0] = 0
for i = 1 to N do
    D[i][0] = i
for j = 1 to M do
    D[0][j] = j
for i = 1 to N do
    for j = 1 to M do
        substitution_cost = 0 if a[i-1] == b[j-1] else 1
        D[i][j] = min(
            D[i-1][j] + 1,     // Deletion
            D[i][j-1] + 1,     // Insertion
```

```
                    D[i-1][j-1] + substitution_cost  // Substitution
          )
  edit_distance = D[N][M]
  max_length = max(N, M)
  levenshtein_ratio = 1 - (edit_distance / max_length)
  return levenshtein_ratio
```

5.7 SUMMARY

This chapter discussed about the procedures and algorithm employed in the proposed system. An extensive experiment was carried out on the proposed system alongside with state-of-the algorithms, that is presented in the next chapter.

CHAPTER 6

RESULTS AND DISCUSSION

6.1 OVERVIEW

This chapter discusses about the experimental results of the proposed methodology.

6.2 DATASET DESCRIPTION

Scene Text Recognition is considered a difficult field of Image Processing due to the lack of large real-world datasets. Most scene text recognition models are therefore trained on synthetic datasets. These synthetic datasets are often found to lack texts with complex backgrounds, rotations, and distortions. Hence, fine-tuning on real-world datasets is required for the model to be made robust. All the datasets used are made publicly available. Various data augmentation steps such as random cropping, random flipping, and random rotation are applied to increase the dataset variety, thereby ensuring that the model is made robust. As the proposed model consists of both text detection and recognition parts, the dataset is required to have ground truth labels for word-level bounding box annotations and ground truth text labels. Each dataset is observed to have a different image size, so they are converted to a common size range to enable further processing. The ratio of training images to testing images of scene text images in real-world datasets is depicted in Figure 6.1. The list of datasets and the number of images for training and testing is provided in Table 6.1.

TABLE 6.1 Datasets used

Dataset	Number of training images	Number of testing images

SynthText	8 million	-
ICDAR15	1,000	500
SCUT-CTW1500	1,000	500
CUTE80	-	80

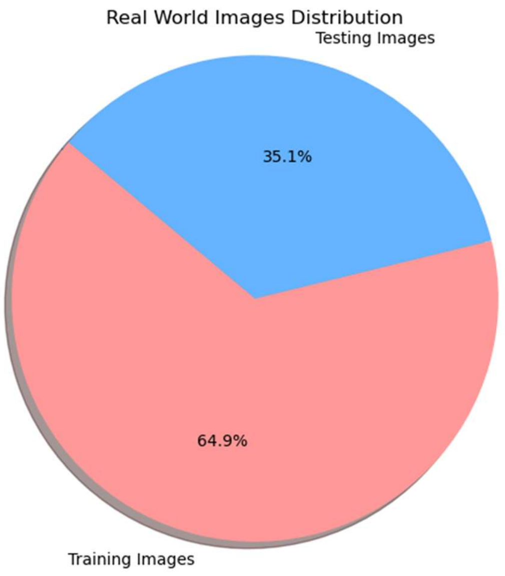
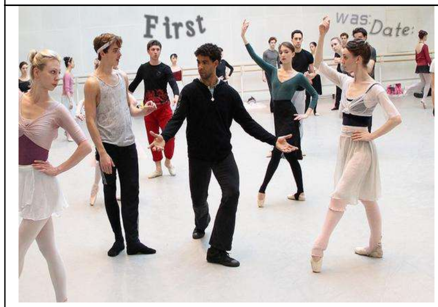
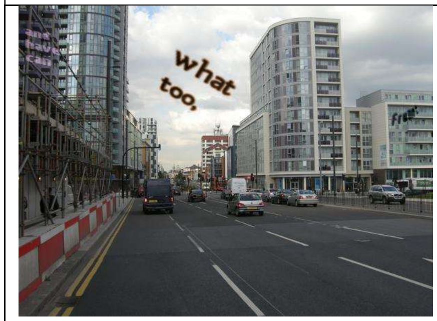


Fig 6.1. Dataset Collection

6.2.1 SYNTHTEXT DATASET

The SynthText dataset is a synthetic dataset created for Scene Text Recognition problem to compensate for the lack of large real-world datasets. The dataset contains 858,750 synthetic scene-image files (.jpg) split into 200 directories, with 7,266,866 word-instances, and 28,971,487 characters. The dataset contains word-level bounding box annotations and character-level bounding box annotations. The word-level bounding-boxes for each image, represented by tensors of size 2x4xNWORDS_i, where the first dimension is 2 for x and y respectively, the second dimension corresponds to the 4 points (clockwise, starting from top-left), and the third dimension of size NWORDS_i, corresponds to the number of words in the i^{th} image.

TABLE 6.2 SynthText Dataset Sample

Image	**Word-level Bounding Box**	**Ground-truth text**
	[[420.589,512.332, 511.9220, 420.1795], [21.0638,23.0695,41.8271,39.8214]] [[448.0848,519.4515,518.7375,447.3708], [47.5704,50.1302,70.0363,67.4764]]	Date: First was:
	[[392.0631,458.5410,457.0992,390.6212], [273.7418,276.7329,308.7788,305.7877]] [[489.5330,546.2872,546.1216,489.3673], [20.1453,21.0693,31.2418,30.3178]]	What too

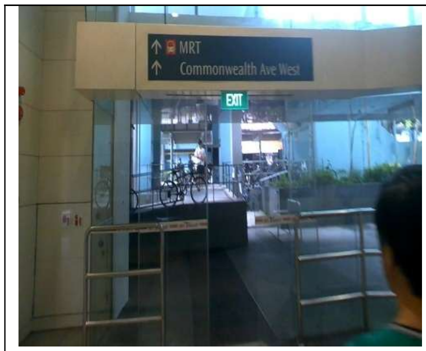
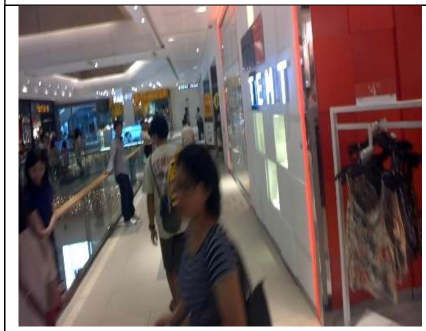
	[[107.7010,156.4480,163.7692,115.0222] [317.7537,274.2737,282.4817,325.9617]]	Shout of

6.2.2 ICDAR15 DATASET

The ICDAR15 dataset is a real-world dataset created for Scene Text Recognition problem as a part of the ICDAR challenge 2015 edition. The dataset contains real scene images captured without the user having taken any specific prior action to cause its appearance or improve its positioning / quality in the frame. While focused scene text is the expected input for applications such as translation on demand, incidental scene text covers another wide range of applications linked to wearable cameras or massive urban captures where the capture is difficult or undesirable to control. Such datasets enhance the robustness of the model in recognizing arbitrary texts in that is randomly oriented, distorted or blurred. Texts are often not clear in the image, thereby making the dataset effective in testing a scene text recognition model's capability. Samples from the dataset are provided in Table 6.3.

TABLE 6.3 ICDAR15 Dataset Sample

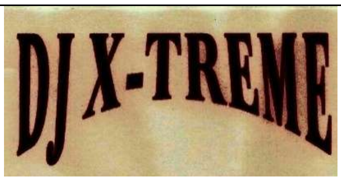
Image	**Word-level Bounding Box**	**Ground-truth text**
	[405,233,478,228,478,241,405,246]	SMOKIN G

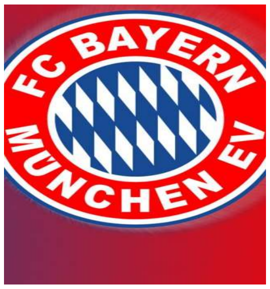
	[506,71,581,73,581,104,506,102] [511,111,763,113,760,148,507,146] [762,117,820,117,822,147,764,147] [821,116,898,117,901,148,823,147] [651,181,713,181,715,213,653,213]	MRT Common wealth Ave West EXIT
	[723,160,863,116,867,225,728,234]	TENT

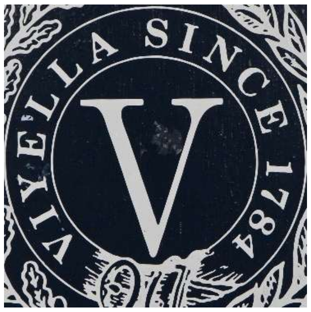
6.2.3 SCUT-CTW1500 DATASET

The SCUT-CTW1500 dataset is a real-world Scene Text Recognition dataset that mainly focuses on curved and rotated texts. Curved and rotated texts often prove a challenge for scene text models to exactly recognize the text. Images include texts in posters, banners, advertisements, placards and logos. Many texts in real-world are curved due to printing on curved surfaces or intentionally curved for design purpose. Hence it is important for a text recognition model to accurately extract text from such images. Sample images from the dataset are provided in Table 6.4.

TABLE 6.4 SCUT-CTW1500 Dataset Sample

Image	**Word-level Bounding Box**	**Ground-truth text**
	height="321" left="25" top="59" width="626"	DJX-TREME

	height="90" left="2" top="73" width="79"
	height="124" left="74" top="29" width="241"
	height="160" left="0" top="201" width="275"
	height="102" left="254" top="190" width="67"
		FC
		BAYERN
		MUNCHEN
		EV

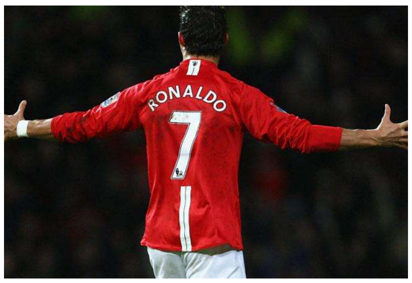
	top="57" left="10" width="696" height="1490"
	top="23" left="748" width="950" height="761"
	top="870" left="1233" width="488" height="690"
	top="529" left="419" width="852" height="907"
		VIYELLA
		SINCE
		1784
		V

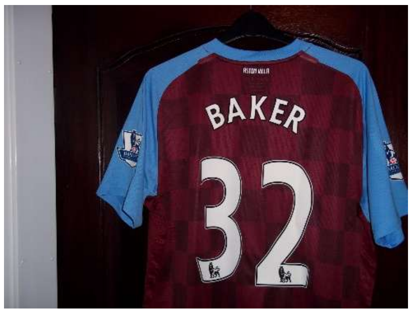
6.2.4 CUTE80 DATASET

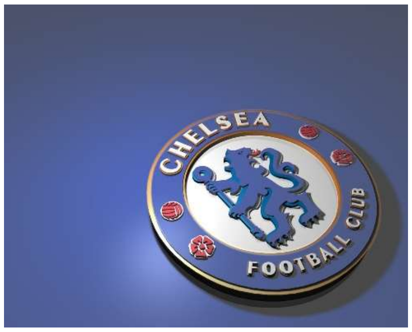
The CUTE80 that consists of 80 curved text line images with complex background, perspective distortion effect and poor resolution effect. CUTE80 is necessary in order to show the capability of the current text detection method in handling curved texts. These images are either indoor or outdoor images captured with a digital camera or retrieved from the Internet. Due the dataset's small sample size it is used only for evaluation of the model. Sample images are provided in the Table 6.5.

TABLE 6.5 CUTE80 Dataset Sample

Image	Word-level Bounding Box	Ground-truth text

	x="232 237 248 265 280 298 320 336 355 359 358 352 344 339 321 309 302 290 281 270 259 248 241"
	
	y="159 152 141 133 129 129 131 139 156 160 167 172 171 168 153 149 148 148 149 151 156 163 172"
	
	x="273 317 317 289 270 299 267"
	
	y="171 172 188 279 279 192 189"
		RONALDO
		
		7

	x="753 850 928 972 1034 1119 1092 1032 958 916 886 833 780"
	
	y="384 345 338 343 354 394 477 444 426 426 426 440 465"
	
	x="727 1029 1124 1151 1126 737 716"
	
	y="576 576 613 709 1048 1025 933"
		BAKER
		
		32

	x="499 526 548 585 625 690 747 819 839 757 707 662 613 574 554 519"
	y="508 460 433 403 378 354 344 344 382 386 390 413 439 476 523 537"
		CHELSEA
		FOOTBALL
		CLUB

6.3 PRE-PROCESSING

The datasets used in the proposed model are publicly available datasets and are widely used for scene text related models. The datasets include images of various locations with texts with various orientation and distortion to maintain balance while training. The images in the datasets are in different resolutions. First, the images have been resized to a standard scale so that it improves performance while training. Our proposed model has two tasks – text detection and text recognition. The text detection phase requires word-level bounding box coordinates. Different datasets have different way of representing the bounding boxes. Some datasets provide the x and y coordinates for the four vertices of the bounding box. Some datasets provide the coordinates of a polygon that exactly trace the curved text in an image. These coordinates are standardized and are converted into usable ground truth labels for the prediction of shrink mask, reinforcement offset and the super pixel window using the algorithm in section 5.2. Standard data augmentation steps have been applied like random cropping without cropping the text out, random rotation of -10 to +10 degrees to create variations in text orientations, random perspective distortions to enrich the robustness of the model.

6.4 IMPLEMENTATION RESULTS

6.4.1 Text Detection

As mentioned in Table 6.1 three datasets have been used for testing. ICDAR15, SCUT-CTW1500 and CUTE80 are the three datasets. Images of size 640x640 are passed through the text detection module. The outputs from the text detection modules are presented below for the sample input in figure 6.2. The sample image is from the CUTE80 dataset.

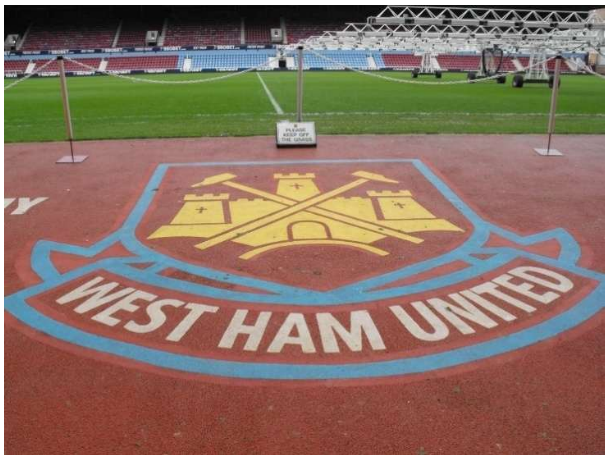


Figure 6.2 Sample Input image

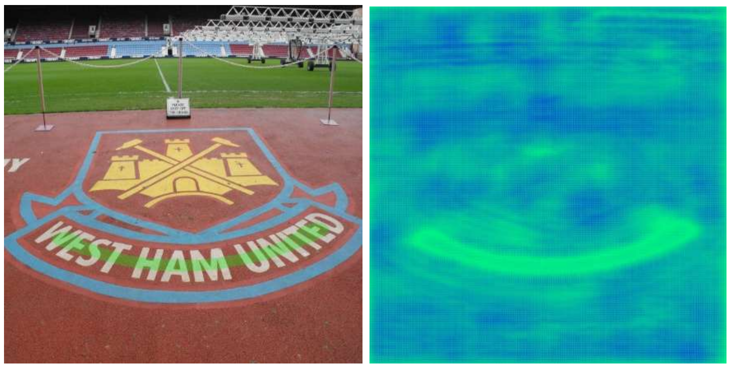


Figure 6.3 Predicted Shrink mask and Predicted reinforcement offset map

The depicted in the Figure 6.2 the text in the image is curved and our model predicted the shrink mask accurately. All the characters are covered within the shrink-mask map. The shrink mask contour is extended outward by the reinforcement offset and the detected text region is visualized in Figure 6.4.

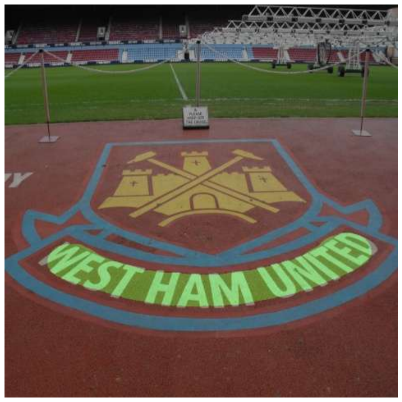
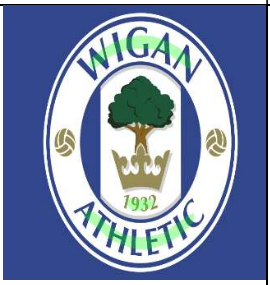
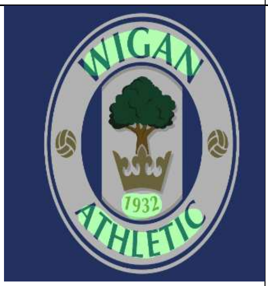
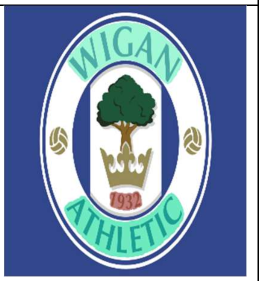


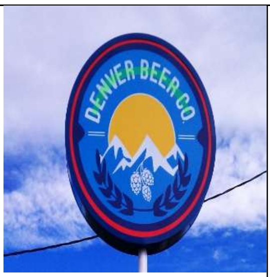
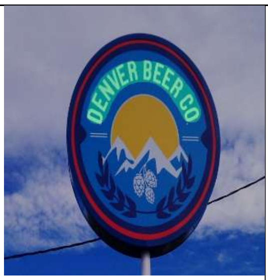
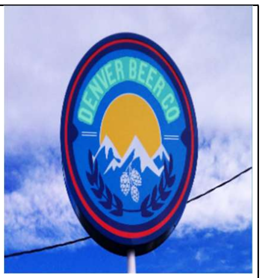
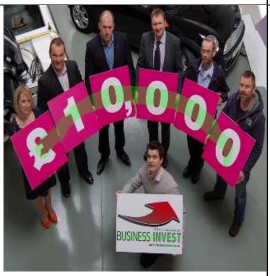
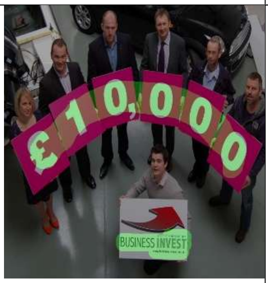
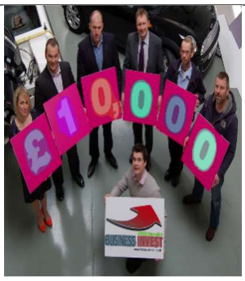
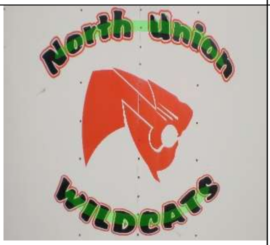
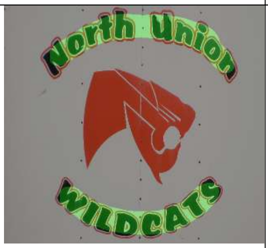
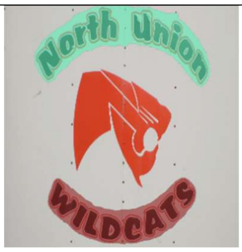
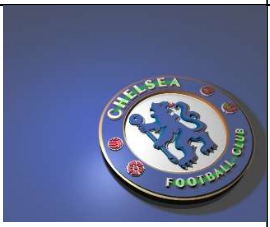
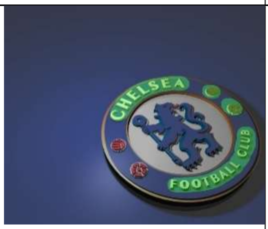
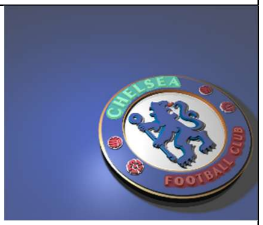
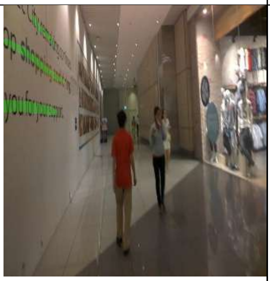
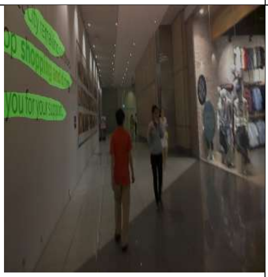
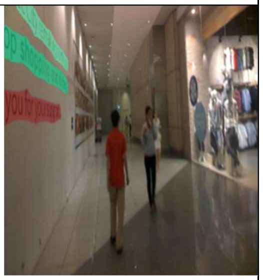
Figure 6.4 Detected Text Region

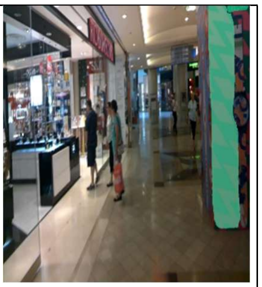
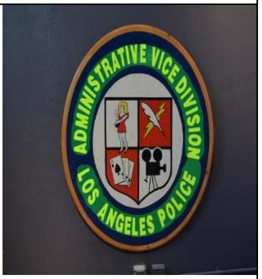
The shrink mask was a tensor of size 160 x 160 x 1 ($\frac{H}{4} * \frac{W}{4} * 1$) and the reinforcement offset map was also a tensor of size 160 x 160 x 1 ($\frac{H}{4} * \frac{W}{4} * 1$). The Super pixel window was omitted in inference to improve inference speed.

Text detection results for various images are provided in the Table 6.6.

TABLE 6.6 Text Detection Results

Input Image	**Predicted Shrink Mask**	**Predicted Text region**	**Ground-Truth Text Region**
			

6.4.2 Text Extraction

The text extraction phase was done on the detected text regions cropped from the original image. The cropped image is depicted in Figure 6.5.

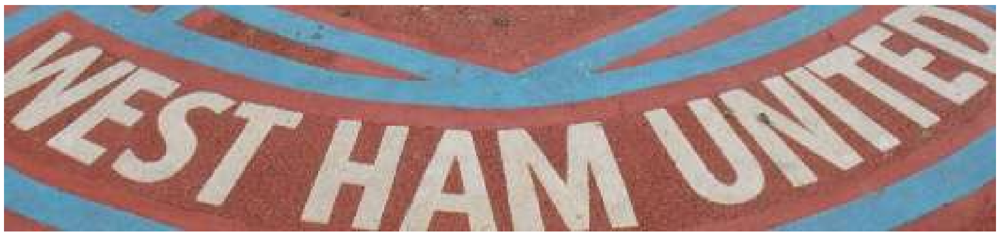


Figure 6.5 Cropped image

The text extraction result for the image in Figure 6.5 was “WEST HAM UNITED”. The attention for the image is depicted in Figure 6.6.

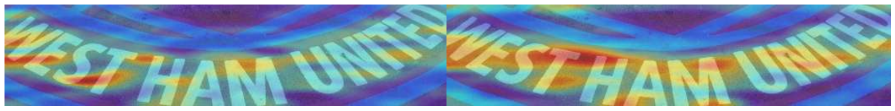
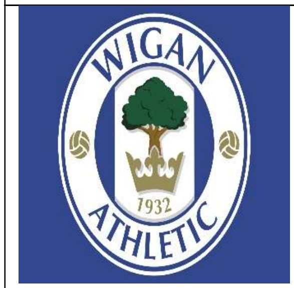
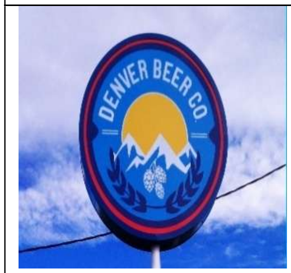
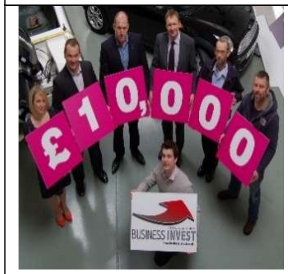
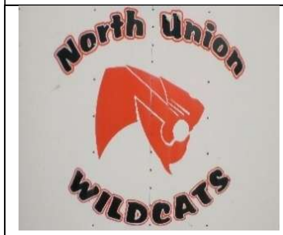
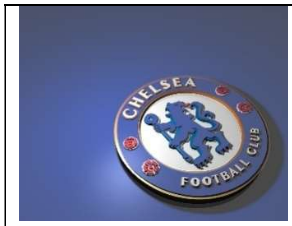
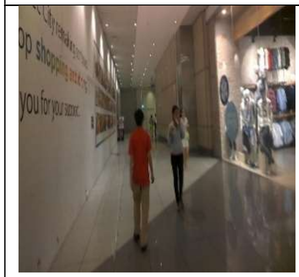
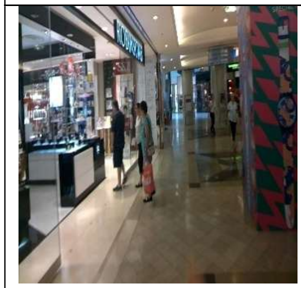
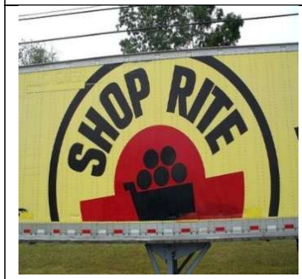
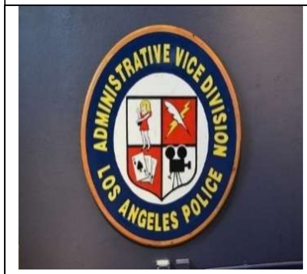


Figure 6.6 Attention map

TABLE 6.7 Text Recognition Results

Input Image	Predicted Text	Ground-truth Text	Similarity Score
	Wigan athletic	WIGAN ATHLETIC	0.9285
	Denver beer co	DENVER BEER CO	0.9321
	10000	10,000	0.9091
	Northunion wildcats	North Union WILDCATS	0.923

	Chelsea Footballclu8	CHELSEA FOOTBALL CLUB	0.829
	ity remaking continues shopping and dinin you tor your support	City remaking continues Shopping and dining You for your support	0.968
	Robinsons special	ROBINSONS SPECIAL	0.941
	Shop rite	SHOP RITE	1.0
	Admnstrative vice divson Loo angeles police	ADMINISTRATIVE VICE DIVISION LOS ANGELES POLIC	0.876

6.4.3 Evaluation Metrics

The evaluation metrics used for text detection are precision, recall and f-measure. The text extraction model is evaluated using the levenshtein ratio metric.

Precision:

Precision refers to the proportion of correctly identified instances of a text region out of all the regions that the model predicted. Precision is calculated using the Eq (6.1).

$$Precision = \frac{TP}{TP+FP} \tag{6.1}$$

Where,

TP – True Positive

FP – False Positive

Recall:

Recall represents the proportion of correctly identified instances of a text region out of all the regions in the image. Recall is calculated using the Eq (6.2).

$$Recall = \frac{TP}{TP+FN} \tag{6.2}$$

Where,

TP – True Positive

FN – False Negative

F1 Score:

The f1 score is the harmonic mean of precision and. F1 is calculated using the Eq (6.3).

$$F1\ score = 2 * \frac{Precision*Recall}{Precision+Recal} \tag{6.3}$$

Intersection of Union:

The Intersection of Union is the proportion of overlap of two regions in an image over the entire regions combined. IoU is calculated using the Eq (6.4).

$$IoU = \frac{max(\,S_{(A_v \cap A_1)}, S_{(A_v \cap A_2)})}{S_{A_v} \cup argmax_A(S_{(A_v \cap A_1)}, S_{(A_v \cap A_2)})} \tag{6.4}$$

Levenshtein Ratio:

The levenshtein ratio is the similarity score between any two strings. It is calculated using the Eq (6.5).

$$lev(a,b) = \begin{cases} |a|, \ if\ |b| = 0 \\ |b|, \ if\ |a| = 0 \\ lev\big(tail(a), tail(b)\big), \ if\ head(a) = head(b) \\ 1 + \min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev\big(tail(a), tail(b)\big) \end{cases}, \ otherwise \end{cases} \tag{6.5}$$

6.4.4 Text Detection Metrics

Using the above equations the evaluation metrics for the text detection module has been calculated for all datasets and the results are presented in Table 6.7. The confusion matrix for each dataset is presented in Figures 6.7, 6.8 and 6.9.

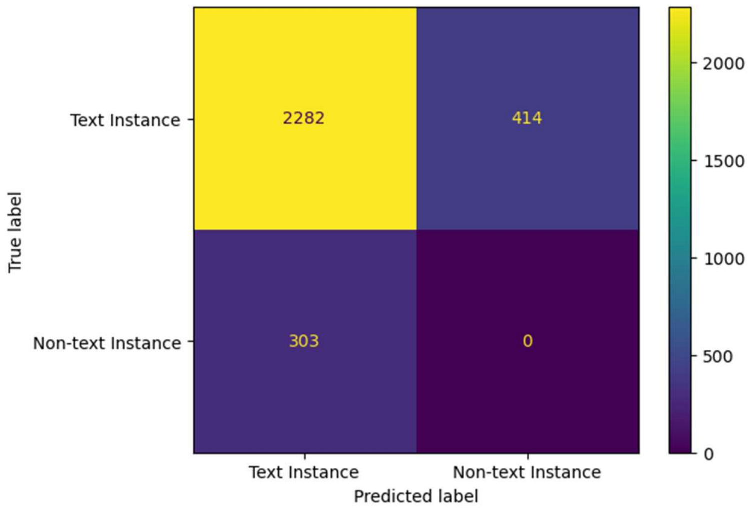


Figure 6.7 Confusion Matrix for SCUT-CTW1500 dataset

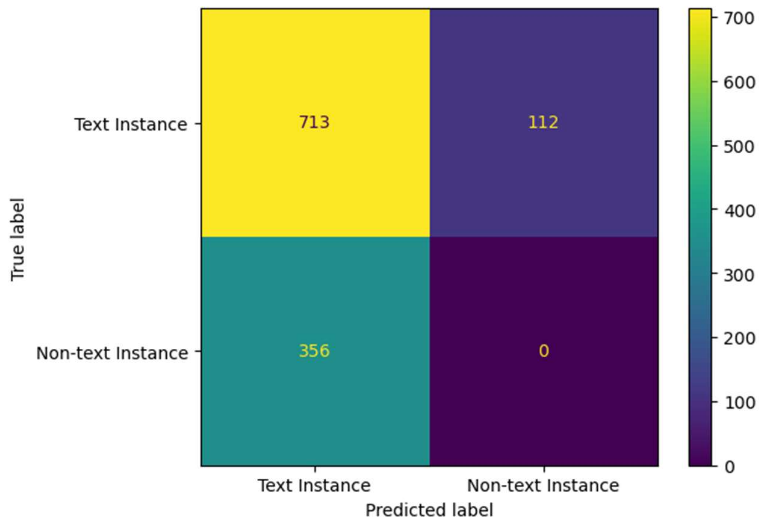


Figure 6.8 Confusion Matrix for ICDAR15 dataset

TABLE 6.8 Text Detection Metrics over multiple datasets

Dataset	Precision %	Recall %	F1 Score %
ICDAR15	86.42	66.70	75.29
SCUT-CTW1500	84.64	88.28	86.42

The results of the text detection module have been compared with other State of The Art (SOTA) models for various datasets and the results are presented in Table 6.8.

TABLE 6.9 Text Detection Metrics over multiple SOTA methods on SCUT-CTW1500

SOTA Methods	**Precision %**	**Recall %**	**F1 Score %**
MixNet [20]	91.4	88.3	89.8
SRFormer [21]	91.6	87.7	89.6
DPText-DETR [22]	91.7	86.2	88.8
TextFuseNet [23]	89.7	85.1	87.4

Ours	84.64	88.28	86.42

6.4.5 Text Recognition Metrics

Using the equation (6.5) the evaluation metrics for the text detection module has been calculated for SCUT-CTW1500 dataset and ICDAR15 dataset and the results are presented in Table 6.9.

TABLE 6.10 Text Recognition Metrics over multiple datasets

Dataset	**Average Levenshtein Ratio**
ICDAR15	85.1
SCUT-CTW1500	84.6

The results of the text recognition module have been compared with other State-Of-The Art (SOTA) models for various datasets and the results are presented in Table 6.10.

TABLE 6.11 Text Recognition Metrics over multiple SOTA methods on ICDAR2015

SOTA Methods	**Accuracy**
DTrOCR 105M [27]	93.5
CLIP4STR-L* [24]	92.6
CPPD [25]	91.7
MGP-STR [26]	90.9
Ours	80.4

6.4.6 Results Discussion

The experimental results have been presented in the above tables and figures. Texts that are in various rotation and orientation are accurately detected by our model, as is evident from table 6.6. However, some regions are also falsely detected as texts when, in fact, there is no text in that particular region in the image. Some examples are given below:

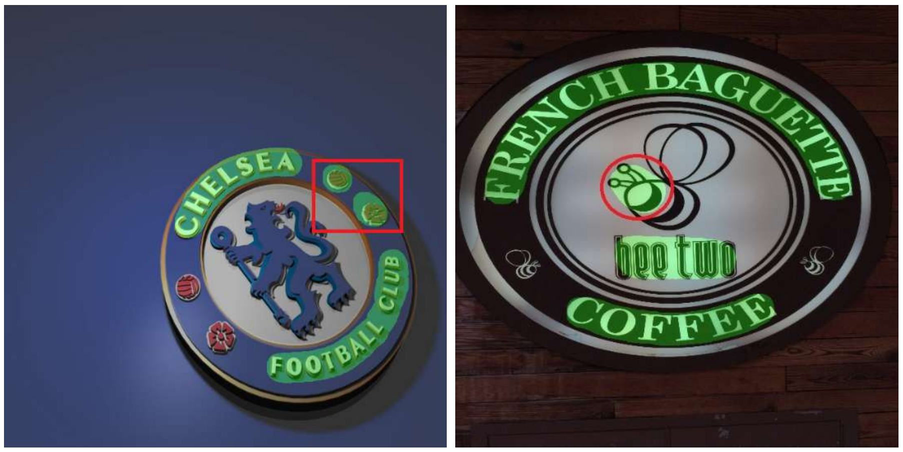


Figure 6.9 Cases of false predictions

The model confuses designs or patterns that resemble characters in shape or size as text regions.

6.5 SUMMARY

This chapter discussed about experimental results of the proposed system compared with various state-of-the-art algorithms. The next chapter discusses about Conclusion & Future enhancement.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

Enhancements in text detection methodologies are addressed by two works, though the problem is approached from different perspectives. In the first work a reinforcement learning-based method is brought forward to improve text localization by finding out regions which progressively shrink, thereby permitting false positives to be decreased and detection accuracy to be increased. In the second work, another aspect of text detection and recognition is explored, mostly focusing on the hard conditions involved in multi-oriented text or scene text recognition.

When the models from both works are merged, it is made notable that better enhancement in text detection under hard environments can be obtained using new deep learning techniques, mainly those based on reinforcement learning and CNN architectures. Detection outcomes are additionally processed by the use of better methodologies such as the shrink-mask mechanism, makes them more strengthen towards noise and low-resolution images. By such techniques being combined, state-of-the-art performance can be gained in real time applications where text is aligned in arbitrary orientations and diverse backgrounds.

7.2 FUTURE ENHANCEMENTS

While the approach of utilizing shrink mask and vision transformer to recognize the text enhances the accuracy of the model, it has some limitations that require addressing in future work. One significant limitation is that the model is trained using only English characters and digits. Therefore, to expand the scope of the model and make it more versatile, it is necessary to increase the training dataset by including additional languages. By expanding the dataset, the model can be trained to identify more languages like Tamil, Chinese, etc. and special symbols like #, @, $, which would increase its usefulness and applicability in various scenarios. Additionally, the model has been found to underperform for images captured in low light conditions, such as during night time. Such scenarios could be explored further.

7.3 SOCIAL IMPACTS

1. Number Plate Recognition:

The numbers and letters imprinted on vehicles are detected, and this information is used for operations at toll plazas for fee collection. Serial numbers of containers from harbors are collected through the recognition process.

2. Improved Education

Students are assisted by allowing images of notes, textbooks, and other educational stuffs to be converted into editable text and used for further study thereby enhancing learning opportunities. The growth of educational apps that make use of scene text recognition for language learning and literacy programs is encouraged.

3. Aesthetic and Ancient Text Conservation

Historical texts and records are computerized, making them useful to investigators and the observer. This allows the preservation of cultural heritage.

7.4 APPLICABILITY OF THE PROJECT

1. Document Digitization and STR

Accuracy in self-operated document scanning and computerization is enhanced. STR systems are improved for greater character recognition in hard layouts.

2. Self-Driving Vehicles and Navigation networks

Road indication signs, hoardings, and license number plates can be analyzed in real-time applications. Blind navigation support systems are possible.

3. Surveillance and Security

Self-programmed number plate recognition systems are improved. Forensic text survey in security footage is improved.

4. Robotic Assistance and Smart Devices

Intelligent assistants qualified of reading text from multiple surfaces are possible. Real time application text recognition is allowed in wearable devices.

APPENDIX 1

HARDWARE REQUIREMENTS

Hard Disk	: 1 TB
RAM	: 16 GB
GPU	: Nvidia A40

SOFTWARE REQUIREMENTS

Programming Language	: Python
Operating System	: Windows 11
Platform	: Visual Studio Code

APPENDIX 2

text_detector.py

```
import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import Dataset, DataLoader

from torchvision.models import resnet50, ResNet50_Weights

from torchvision import transforms

import numpy as np

import cv2

import os

from PIL import Image

from scipy.io import loadmat

import pyclipper

import random

import matplotlib.pyplot as plt

import xml.etree.ElementTree as ET

import glob


# ------------------------------

# Model Components

# ------------------------------

# Backbone: Feature Extractor Stem

class FeatureExtractorStem(nn.Module):
    def __init__(self):
        super().__init__()
        resnet = resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)
        self.layer0 = nn.Sequential(resnet.conv1, resnet.bn1, resnet.relu, resnet.maxpool)
        self.layer1 = resnet.layer1  # [B, 256, H/4, W/4]
        self.layer2 = resnet.layer2  # [B, 512, H/8, W/8]
        self.layer3 = resnet.layer3  # [B, 1024, H/16, W/16]
        self.layer4 = resnet.layer4  # [B, 2048, H/32, W/32]
    def forward(self, x):
        x = self.layer0(x)
        f1 = self.layer1(x)  # 1/4 resolution, 256 channels
        f2 = self.layer2(f1) # 1/8 resolution, 512 channels
        f3 = self.layer3(f2) # 1/16 resolution, 1024 channels
        f4 = self.layer4(f3) # 1/32 resolution, 2048 channels
        return f1, f2, f3, f4
# Feature Merging Branch (Equations 1-4 in the paper)
class FeatureMergingBranch(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1_1 = nn.Conv2d(256, 128, kernel_size=1)
        self.conv1_2 = nn.Conv2d(512, 128, kernel_size=1)
        self.conv1_3 = nn.Conv2d(1024, 128, kernel_size=1)
        self.conv1_4 = nn.Conv2d(2048, 128, kernel_size=1)
        self.conv3x3 = nn.Conv2d(128, 64, kernel_size=3, padding=1)
        self.upsample = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
        # Initialize weights to prevent NaNs
        nn.init.kaiming_normal_(self.conv1_1.weight, mode='fan_out', nonlinearity='relu')
        nn.init.kaiming_normal_(self.conv1_2.weight, mode='fan_out', nonlinearity='relu')
        nn.init.kaiming_normal_(self.conv1_3.weight, mode='fan_out', nonlinearity='relu')
        nn.init.kaiming_normal_(self.conv1_4.weight, mode='fan_out', nonlinearity='relu')
        nn.init.kaiming_normal_(self.conv3x3.weight, mode='fan_out', nonlinearity='relu')
    def forward(self, f1, f2, f3, f4):
        H1_1 = self.conv1_1(f1)  # [B, 128, H/4, W/4]
        H1_2 = self.conv1_2(f2)  # [B, 128, H/8, W/8]
        H1_3 = self.conv1_3(f3)  # [B, 128, H/16, W/16]
        H1_4 = self.conv1_4(f4)  # [B, 128, H/32, W/32]

        H2_1 = H1_1
        H2_2 = self.upsample(H1_2) + H1_1  # [B, 128, H/4, W/4]
        H2_3 = self.upsample(H1_3) + H1_2  # [B, 128, H/8, W/8]
        H2_4 = self.upsample(H1_4) + H1_3  # [B, 128, H/16, W/16]

        H3_1 = self.conv3x3(H2_1)  # [B, 64, H/4, W/4]
        H3_2 = self.conv3x3(H2_2)  # [B, 64, H/4, W/4]
        H3_3 = self.upsample(self.conv3x3(H2_3))  # [B, 64, H/4, W/4]
        H3_4 = self.upsample(self.upsample(self.conv3x3(H2_4)))  # [B, 64, H/4, W/4]

        Hf = torch.cat([H3_1, H3_2, H3_3, H3_4], dim=1)  # [B, 256, H/4, W/4]
```

```
        return Hf
# Output Layer
class OutputLayer(nn.Module):
    def __init__(self):
        super().__init__()
        self.head1 = nn.Sequential(
            nn.Conv2d(256, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 64, kernel_size=2, stride=2),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 2, kernel_size=2, stride=2)
        )
        self.head2 = nn.Sequential(
            nn.Conv2d(256, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 64, kernel_size=2, stride=2),
            nn.ReLU(),
            nn.ConvTranspose2d(64, 9, kernel_size=2, stride=2)
        )
        # Initialize weights
        for module in self.modules():
            if isinstance(module, nn.Conv2d) or isinstance(module, nn.ConvTranspose2d):
                nn.init.kaiming_normal_(module.weight, mode='fan_out', nonlinearity='relu')
    def forward(self, x):
        O1 = self.head1(x)  # [B, 2, H, W]
        O2 = self.head2(x)  # [B, 9, H, W]
        shrink_mask = O1[:, 0:1, :, :]
        offsets = O1[:, 1:2, :, :]
        return shrink_mask, offsets, O2
# Combined RSMTD Model
class RSMTD(nn.Module):
    def __init__(self):
        super().__init__()
        self.backbone = FeatureExtractorStem()
        self.merging = FeatureMergingBranch()
        self.headers = OutputLayer()
    def forward(self, x):
        f1, f2, f3, f4 = self.backbone(x)
        merged_features = self.merging(f1, f2, f3, f4)  # [B, 256, H/4, W/4]
        shrink_mask, offsets, spw = self.headers(merged_features)  # [B, 1, H, W], [B, 1, H,
W], [B, 9, H, W]
        return shrink_mask, offsets, spw
# -------------------------------
# Loss Functions
# -------------------------------
class DiceLoss(nn.Module):
    def __init__(self, eps=1e-5):
        super().__init__()
        self.eps = eps
    def forward(self, pred, target):
        pred = torch.sigmoid(pred)
        pred_flat = pred.reshape(-1)
        target_flat = target.reshape(-1)
        intersection = (pred_flat * target_flat).sum()
        dice = (2. * intersection + self.eps) / (pred_flat.sum() + target_flat.sum() + self.eps)
        return 1 - dice
class RatioLoss(nn.Module):
    def __init__(self, eps=1e-8):
        super().__init__()
        self.eps = eps
    def forward(self, pred, target, mask):
        pred = torch.relu(pred)  # Ensure non-negative predictions
        mask = mask.expand_as(pred)  # Match mask shape to pred/target
        pred_masked = pred * mask
        target_masked = target * mask
        max_val = torch.max(pred_masked, target_masked)
        min_val = torch.max(target_masked, torch.zeros_like(target_masked))  # Ensure
min_val >= 0
        ratio = torch.log((max_val + self.eps) / (min_val + self.eps))
        return torch.sum(ratio * mask) / (torch.sum(mask) + self.eps)
def total_loss(pred_masks, pred_offsets, pred_spw, target_masks, target_offsets,
target_spw):
    dice_loss = DiceLoss()(pred_masks, target_masks)
    offset_ratio_loss = RatioLoss()(pred_offsets, target_offsets, target_masks)
    spw_ratio_loss = RatioLoss()(pred_spw, target_spw, target_masks)
    return dice_loss + 0.25 * offset_ratio_loss + 0.25 * spw_ratio_loss
# ------------------------------
# Dataset with Label Generation and Data Augmentation
# ------------------------------
class SynthTextDataset(Dataset):
    def __init__(self, root_dir, transform=None, augment=True, sample_size=10000):
        self.root_dir = root_dir
        self.transform = transform
        self.augment = augment
        self.gt = loadmat(os.path.join(root_dir, 'gt.mat'))
        self.imnames = self.gt['imnames'][0]
        self.wordBB = self.gt['wordBB'][0]
        self.shrink_coef = 0.4
        total_samples = len(self.imnames)
        if total_samples < sample_size:
            raise ValueError(f"Requested sample size {sample_size} exceeds total samples
{total_samples}.")
        self.sample_indices = random.sample(range(total_samples), sample_size)

    def __len__(self):
        return len(self.sample_indices)
```

```
def __getitem__(self, idx):
    actual_idx = self.sample_indices[idx]
    img_path = os.path.join(self.root_dir, self.imnames[actual_idx][0])
    image = Image.open(img_path).convert('RGB')
    word_boxes = self.wordBB[actual_idx]
    if word_boxes.ndim == 2:
        word_boxes = word_boxes[:, :, np.newaxis]
    num_words = word_boxes.shape[2]
    H, W = image.size[::-1]
    shrink_mask = np.zeros((H, W), dtype=np.float32)
    offsets = np.zeros((H, W), dtype=np.float32)
    for i in range(num_words):
        box = word_boxes[:, :, i]
        polygon = self.bb_to_polygon(box)
        area = cv2.contourArea(polygon)
        perimeter = cv2.arcLength(polygon, True)
        if perimeter == 0:
            continue
        delta = (area / perimeter) * (1 - self.shrink_coef**2)  # Per paper’s Equation 8
        shrinked_polygon = self.shrink_polygon(polygon, delta)
        cv2.fillPoly(shrink_mask, [shrinked_polygon.astype(np.int32)], 1)
        original_mask = np.zeros((H, W), dtype=np.uint8)
        cv2.fillPoly(original_mask, [polygon.astype(np.int32)], 1)
        dist = cv2.distanceTransform(original_mask, cv2.DIST_L2, 3)
```

```
            offsets[original_mask == 1] = dist[original_mask == 1]
        spw = self.generate_spw(shrink_mask)
        if self.augment:
            image, shrink_mask, offsets, spw = self.augment_data(image, shrink_mask, offsets, spw, target_size=(640, 640))
        else:
            image = image.resize((640, 640))
            shrink_mask = cv2.resize(shrink_mask, (640, 640), interpolation=cv2.INTER_NEAREST)
            spw = np.stack([cv2.resize(spw[i], (640, 640), interpolation=cv2.INTER_LINEAR) for i in range(9)], axis=0)
            offsets = cv2.resize(offsets, (640, 640), interpolation=cv2.INTER_LINEAR)
        if self.transform:
            image = self.transform(image)
        return {
            'image': image,
            'shrink_mask': torch.from_numpy(shrink_mask).unsqueeze(0),  # [1, 640, 640]
            'offsets': torch.from_numpy(offsets).unsqueeze(0),        # [1, 640, 640]
            'spw': torch.from_numpy(spw)                              # [9, 640, 640]
        }
    def bb_to_polygon(self, box):
        return np.array(box.T, dtype=np.int32).reshape((-1, 2))
    def shrink_polygon(self, polygon, delta):
        pco = pyclipper.PyclipperOffset()
        pco.AddPath(polygon.tolist(),                                          pyclipper.JT_ROUND,
pyclipper.ET_CLOSEDPOLYGON)
        shrunk = pco.Execute(-delta)
        return np.array(shrunk[0]) if shrunk else polygon
    def generate_spw(self, shrink_mask):
        window_configs = [
            (3, 5), (3, 3), (5, 3),  # Scale 2: r=1/2, 1, 2
            (5, 7), (5, 5), (7, 5),  # Scale 4: r=1/2, 1, 2
            (9, 11), (9, 9), (11, 9) # Scale 8: r=1/2, 1, 2
        ]
        spw_maps = []
        for kh, kw in window_configs:
            spw = cv2.boxFilter(shrink_mask, ddepth=-1, ksize=(kh, kw))
            spw = spw / (kh * kw)
            spw_maps.append(spw)
        return np.stack(spw_maps, axis=0).astype(np.float32)  # [9, H, W]
    def augment_data(self, image, shrink_mask, offsets, spw, target_size=(640, 640)):
        img_np = np.array(image)
        h, w = img_np.shape[:2]
        # Random Horizontal Flip
        if random.random() < 0.5:
            img_np = cv2.flip(img_np, 1)
            shrink_mask = np.fliplr(shrink_mask)
            spw = np.fliplr(spw)
            offsets = np.fliplr(offsets)
        # Random Rotation
        angle = random.uniform(-10, 10)
        center = (w / 2, h / 2)
        M = cv2.getRotationMatrix2D(center, angle, 1.0)
        img_np = cv2.warpAffine(img_np, M, (w, h), flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT)
        shrink_mask = cv2.warpAffine(shrink_mask, M, (w, h), flags=cv2.INTER_NEAREST,
borderMode=cv2.BORDER_REFLECT)
        spw = np.stack([cv2.warpAffine(spw[i], M, (w, h), flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT) for i in range(9)], axis=0)
        offsets = cv2.warpAffine(offsets, M, (w, h), flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT)
        # Random Scaling
        scale = random.uniform(0.8, 1.2)
        new_h, new_w = int(h * scale), int(w * scale)
        img_np = cv2.resize(img_np, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
        shrink_mask = cv2.resize(shrink_mask, (new_w, new_h),
interpolation=cv2.INTER_NEAREST)
        spw = np.stack([cv2.resize(spw[i], (new_w, new_h),
interpolation=cv2.INTER_LINEAR) for i in range(9)], axis=0)
        offsets = cv2.resize(offsets, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
        # Random Cropping or Padding
        target_h, target_w = target_size
        if new_h > target_h and new_w > target_w:
            top = random.randint(0, new_h - target_h)
```

```
            left = random.randint(0, new_w - target_w)
            img_np = img_np[top:top + target_h, left:left + target_w]
            shrink_mask = shrink_mask[top:top + target_h, left:left + target_w]
            spw = spw[:, top:top + target_h, left:left + target_w]
            offsets = offsets[top:top + target_h, left:left + target_w]
        else:
            pad_h = max(0, target_h - new_h)
            pad_w = max(0, target_w - new_w)
            img_np = cv2.copyMakeBorder(img_np, 0, pad_h, 0, pad_w,
cv2.BORDER_CONSTANT, value=0)
            shrink_mask = cv2.copyMakeBorder(shrink_mask, 0, pad_h, 0, pad_w,
cv2.BORDER_CONSTANT, value=0)
            spw = np.pad(spw, ((0, 0), (0, pad_h), (0, pad_w)), mode='constant',
constant_values=0)
            offsets = cv2.copyMakeBorder(offsets, 0, pad_h, 0, pad_w,
cv2.BORDER_CONSTANT, value=0)
            img_np = img_np[:target_h, :target_w]
            shrink_mask = shrink_mask[:target_h, :target_w]
            spw = spw[:, :target_h, :target_w]
            offsets = offsets[:target_h, :target_w]

        aug_image = Image.fromarray(img_np)
        return aug_image, shrink_mask, offsets, spw
class ICDAR15DetectionDataset(Dataset):
    def __init__(self, img_dir, gt_dir, transform=None, augment=True, sample_size=None):
```

```
"""
Args:
    img_dir (str): Path to the directory containing images.
    gt_dir (str): Path to the directory containing per-image ground truth files.
    transform (callable, optional): Optional transform to be applied on an image.
    augment (bool): Whether to apply data augmentation.
    sample_size (int, optional): Number of samples to use; if None, use all.
"""
self.img_dir = img_dir
self.gt_dir = gt_dir
self.transform = transform
self.augment = augment
self.shrink_coef = 0.4  # Shrinkage coefficient, same as SynthTextDataset
self.data = []
# List all image files and pair with ground truth files
image_files = [f for f in os.listdir(img_dir) if f.endswith(('.jpg', '.png'))]
for image_file in image_files:
    image_path = os.path.join(img_dir, image_file)
    base_name = os.path.splitext(image_file)[0]
    gt_name = f"gt_{base_name}.txt"
    gt_path = os.path.join(gt_dir, gt_name)
    if os.path.exists(gt_path):
        word_boxes = self.parse_gt_file(gt_path)
        self.data.append((image_path, word_boxes))
```

```
            else:
                print(f"Warning: GT file {gt_path} not found for {image_file}")
        total_samples = len(self.data)
        if sample_size is not None and sample_size < total_samples:
            self.sample_indices = random.sample(range(total_samples), sample_size)
        else:
            self.sample_indices = list(range(total_samples))
        print(f"Loaded {len(self.sample_indices)} samples from {img_dir}")
    def parse_gt_file(self, gt_file):
        """Parse a ground truth file to extract bounding boxes."""
        boxes = []
        with open(gt_file, 'r', encoding='utf-8-sig') as f:
            for line in f:
                parts = line.strip().split(',')
                if len(parts) >= 9:
                    try:
                        coords = list(map(float, parts[:8]))
                        label = ','.join(parts[8:]).strip()
                        if label != '###':  # Skip "don't care" regions
                            x = coords[0::2]  # x1, x2, x3, x4
                            y = coords[1::2]  # y1, y2, y3, y4
                            box = np.array([x, y])
                            boxes.append(box)
                    except ValueError:
```

```
                print(f"Invalid coordinates in {gt_file}")
        return np.stack(boxes, axis=2) if boxes else np.zeros((2, 4, 0))
    def __len__(self):
        return len(self.sample_indices)
    def __getitem__(self, idx):
        """
        Args:
            idx (int): Index of the sample to fetch.
        Returns:
            dict: Contains image, shrink_mask, offsets, and spw as tensors.
        """
        actual_idx = self.sample_indices[idx]
        image_path, word_boxes = self.data[actual_idx]
        image = Image.open(image_path).convert('RGB')
        if word_boxes.ndim == 2:  # Ensure 3D array even for one box
            word_boxes = word_boxes[:, :, np.newaxis]
        num_words = word_boxes.shape[2]
        H, W = image.size[::-1]
        shrink_mask = np.zeros((H, W), dtype=np.float32)
        offsets = np.zeros((H, W), dtype=np.float32)
        # Generate shrink_mask and offsets
        for i in range(num_words):
            box = word_boxes[:, :, i]
            polygon = self.bb_to_polygon(box)
            area = cv2.contourArea(polygon)
            perimeter = cv2.arcLength(polygon, True)
            if perimeter == 0:
                continue
            delta = (area / perimeter) * (1 - self.shrink_coef**2)
            shrinked_polygon = self.shrink_polygon(polygon, delta)
            cv2.fillPoly(shrink_mask, [shrinked_polygon.astype(np.int32)], 1)
            original_mask = np.zeros((H, W), dtype=np.uint8)
            cv2.fillPoly(original_mask, [polygon.astype(np.int32)], 1)
            dist = cv2.distanceTransform(original_mask, cv2.DIST_L2, 3)
            offsets[original_mask == 1] = dist[original_mask == 1]
        spw = self.generate_spw(shrink_mask)
        # Apply augmentation or resize
        if self.augment:
            image, shrink_mask, offsets, spw = self.augment_data(image, shrink_mask, offsets,
spw, target_size=(640, 640))
        else:
            image = image.resize((640, 640))
            shrink_mask = cv2.resize(shrink_mask, (640, 640),
interpolation=cv2.INTER_NEAREST)
            spw = np.stack([cv2.resize(spw[i], (640, 640), interpolation=cv2.INTER_LINEAR)
for i in range(9)], axis=0)
            offsets = cv2.resize(offsets, (640, 640), interpolation=cv2.INTER_LINEAR)
        if self.transform:
            image = self.transform(image)
        return {
            'image': image,
            'shrink_mask': torch.from_numpy(shrink_mask).unsqueeze(0),  # [1, 640, 640]
            'offsets': torch.from_numpy(offsets).unsqueeze(0),         # [1, 640, 640]
            'spw': torch.from_numpy(spw)                               # [9, 640, 640]
        }
    def bb_to_polygon(self, box):
        """Convert bounding box [2, 4] to polygon [4, 2]."""
        return np.array(box.T, dtype=np.int32).reshape((-1, 2))
    def shrink_polygon(self, polygon, delta):
        """Shrink a polygon by a given delta."""
        pco = pyclipper.PyclipperOffset()
        pco.AddPath(polygon.tolist(),                                  pyclipper.JT_ROUND,
pyclipper.ET_CLOSEDPOLYGON)
        shrunk = pco.Execute(-delta)
        return np.array(shrunk[0]) if shrunk else polygon
    def generate_spw(self, shrink_mask):
        """Generate shrinkage probability weights with multi-scale filters."""
        window_configs = [
            (3, 5), (3, 3), (5, 3),  # Scale 2
            (5, 7), (5, 5), (7, 5),  # Scale 4
            (9, 11), (9, 9), (11, 9) # Scale 8
        ]
        spw_maps = []
        for kh, kw in window_configs:
            spw = cv2.boxFilter(shrink_mask, ddepth=-1, ksize=(kh, kw))
            spw = spw / (kh * kw)
            spw_maps.append(spw)
        return np.stack(spw_maps, axis=0).astype(np.float32)
    def augment_data(self, image, shrink_mask, offsets, spw, target_size=(640, 640)):
        """Apply random augmentations to image and masks."""
        img_np = np.array(image)
        h, w = img_np.shape[:2]
        # Random Horizontal Flip
        if random.random() < 0.5:
            img_np = cv2.flip(img_np, 1)
            shrink_mask = np.fliplr(shrink_mask)
            spw = np.fliplr(spw)
            offsets = np.fliplr(offsets)
        # Random Rotation
        angle = random.uniform(-10, 10)
        center = (w / 2, h / 2)
        M = cv2.getRotationMatrix2D(center, angle, 1.0)
        img_np = cv2.warpAffine(img_np, M, (w, h), flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT)
        shrink_mask = cv2.warpAffine(shrink_mask, M, (w, h), flags=cv2.INTER_NEAREST,
borderMode=cv2.BORDER_REFLECT)
        spw = np.stack([cv2.warpAffine(spw[i], M, (w, h), flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REFLECT) for i in range(9)], axis=0)
```

```
offsets = cv2.warpAffine(offsets, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT)

# Random Scaling

scale = random.uniform(0.8, 1.2)

new_h, new_w = int(h * scale), int(w * scale)

img_np = cv2.resize(img_np, (new_w, new_h), interpolation=cv2.INTER_LINEAR)

shrink_mask = cv2.resize(shrink_mask, (new_w, new_h), interpolation=cv2.INTER_NEAREST)

spw = np.stack([cv2.resize(spw[i], (new_w, new_h), interpolation=cv2.INTER_LINEAR) for i in range(9)], axis=0)

offsets = cv2.resize(offsets, (new_w, new_h), interpolation=cv2.INTER_LINEAR)


# Random Cropping or Padding

target_h, target_w = target_size

if new_h > target_h and new_w > target_w:

    top = random.randint(0, new_h - target_h)

    left = random.randint(0, new_w - target_w)

    img_np = img_np[top:top + target_h, left:left + target_w]

    shrink_mask = shrink_mask[top:top + target_h, left:left + target_w]

    spw = spw[:, top:top + target_h, left:left + target_w]

    offsets = offsets[top:top + target_h, left:left + target_w]

else:

    pad_h = max(0, target_h - new_h)

    pad_w = max(0, target_w - new_w)
```

```
            img_np = cv2.copyMakeBorder(img_np, 0, pad_h, 0, pad_w, cv2.BORDER_CONSTANT, value=0)
            shrink_mask = cv2.copyMakeBorder(shrink_mask, 0, pad_h, 0, pad_w, cv2.BORDER_CONSTANT, value=0)
            spw = np.pad(spw, ((0, 0), (0, pad_h), (0, pad_w)), mode='constant', constant_values=0)
            offsets = cv2.copyMakeBorder(offsets, 0, pad_h, 0, pad_w, cv2.BORDER_CONSTANT, value=0)
            img_np = img_np[:target_h, :target_w]
            shrink_mask = shrink_mask[:target_h, :target_w]
            spw = spw[:, :target_h, :target_w]
            offsets = offsets[:target_h, :target_w]

        return Image.fromarray(img_np), shrink_mask, offsets, spw
class SCUTCTW1500Dataset(Dataset):
    def __init__(self, root_dir, transform=None, augment=True, sample_size=None):
        """
        Initialize the SCUT-CTW1500 dataset.
        Args:
            root_dir (str): Root directory containing 'train_images' and 'train_labels' folders.
            transform (callable, optional): Optional transform to be applied to the image.
            augment (bool): Whether to apply data augmentation. Default is True.
            sample_size (int, optional): Number of samples to use. If None, use all samples.
        """
        self.root_dir = root_dir
        self.transform = transform
        self.augment = augment
        self.xml_files = sorted(glob.glob(os.path.join(root_dir, 'train_labels', '*.xml')))
        total_samples = len(self.xml_files)
        if sample_size is None:
            self.sample_indices = list(range(total_samples))
        else:
            if sample_size > total_samples:
                raise ValueError(f"Requested sample size {sample_size} exceeds total samples {total_samples}.")
            self.sample_indices = random.sample(range(total_samples), sample_size)
        self.shrink_coef = 0.4  # Shrink coefficient for text region polygons
    def __len__(self):
        """Return the number of samples in the dataset."""
        return len(self.sample_indices)
    def __getitem__(self, idx):
        """
        Get a sample from the dataset.
        Args:
            idx (int): Index of the sample.
        Returns:
            dict: Dictionary containing the image, shrink mask, offsets, and scale pyramid weights.
        """
        actual_idx = self.sample_indices[idx]
        xml_path = self.xml_files[actual_idx]
        image_file, polygons = self.parse_xml(xml_path)
        img_path = os.path.join(self.root_dir, 'train_images', image_file)
        image = Image.open(img_path).convert('RGB')
        H, W = image.size[1], image.size[0]  # PIL Image.size is (width, height)
        # Initialize masks
        shrink_mask = np.zeros((H, W), dtype=np.float32)
        offsets = np.zeros((H, W), dtype=np.float32)
        # Process each text region polygon
        for polygon in polygons:
            shrinked_polygon = self.shrink_polygon(polygon)
            cv2.fillPoly(shrink_mask, [shrinked_polygon.astype(np.int32)], 1)
            original_mask = np.zeros((H, W), dtype=np.uint8)
            cv2.fillPoly(original_mask, [polygon.astype(np.int32)], 1)
            dist = cv2.distanceTransform(original_mask, cv2.DIST_L2, 3)
            offsets[original_mask == 1] = dist[original_mask == 1]
        # Generate scale pyramid weights
        spw = self.generate_spw(shrink_mask)
        # Apply augmentation if enabled
        if self.augment:
            image, shrink_mask, offsets, spw = self.augment_data(image, shrink_mask, offsets,
spw, target_size=(640, 640))
        else:
            image = image.resize((640, 640))
            shrink_mask = cv2.resize(shrink_mask, (640, 640), interpolation=cv2.INTER_NEAREST)
            spw = np.stack([cv2.resize(spw[i], (640, 640), interpolation=cv2.INTER_LINEAR) for i in range(9)], axis=0)
            offsets = cv2.resize(offsets, (640, 640), interpolation=cv2.INTER_LINEAR)
        # Apply transform if provided
        if self.transform:
            image = self.transform(image)
        return {
            'image': image,
            'shrink_mask': torch.from_numpy(shrink_mask).unsqueeze(0),  # Shape: [1, 640, 640]
            'offsets': torch.from_numpy(offsets).unsqueeze(0),        # Shape: [1, 640, 640]
            'spw': torch.from_numpy(spw)                              # Shape: [9, 640, 640]
        }
    def parse_xml(self, xml_path):
        """
        Parse the XML annotation file to extract image filename and polygons.
        Args:
            xml_path (str): Path to the XML file.
        Returns:
            tuple: (image_file, polygons), where image_file is the filename and polygons is a list of numpy arrays.
        """
        tree = ET.parse(xml_path)
        root = tree.getroot()
        image_file = root.find('image').attrib['file']
        polygons = []
        for box in root.findall('image/box'):
            segs = box.find('segs').text
            coords = list(map(int, segs.split(',')))
            polygon = np.array(coords).reshape(-1, 2)  # Shape: [N, 2], where N is the number
of points
            polygons.append(polygon)
        return image_file, polygons
    def shrink_polygon(self, polygon):
        """
        Shrink the polygon using Pyclipper.
        Args:
            polygon (np.ndarray): Polygon coordinates of shape [N, 2].
        Returns:
            np.ndarray: Shrunk polygon coordinates.
        """
        pco = pyclipper.PyclipperOffset()
        pco.AddPath(polygon.tolist(),                pyclipper.JT_ROUND,
pyclipper.ET_CLOSEDPOLYGON)
        area = cv2.contourArea(polygon)
        perimeter = cv2.arcLength(polygon, True)
        if perimeter == 0:
            return polygon
        delta = (area / perimeter) * (1 - self.shrink_coef**2)
        shrunk = pco.Execute(-delta)
        return np.array(shrunk[0]) if shrunk else polygon
    def generate_spw(self, shrink_mask):
        """
        Generate scale pyramid weights from the shrink mask.
        Args:
            shrink_mask (np.ndarray): Binary mask of shape [H, W].
        Returns:
            np.ndarray: Scale pyramid weights of shape [9, H, W].
        """
        window_configs = [
            (3, 5), (3, 3), (5, 3),  # Scale 2: r=1/2, 1, 2
            (5, 7), (5, 5), (7, 5),  # Scale 4: r=1/2, 1, 2
            (9, 11), (9, 9), (11, 9)  # Scale 8: r=1/2, 1, 2
        ]
        spw_maps = []
        for kh, kw in window_configs:
            spw = cv2.boxFilter(shrink_mask, ddepth=-1, ksize=(kh, kw))
            spw = spw / (kh * kw)
            spw_maps.append(spw)
        return np.stack(spw_maps, axis=0).astype(np.float32)
```

```
def augment_data(self, image, shrink_mask, offsets, spw, target_size=(640, 640)):
    """
    Apply data augmentation to the image and masks.
    Args:
        image (PIL.Image): Input image.
        shrink_mask (np.ndarray): Shrink mask of shape [H, W].
        offsets (np.ndarray): Offset map of shape [H, W].
        spw (np.ndarray): Scale pyramid weights of shape [9, H, W].
        target_size (tuple): Target size (height, width).
    Returns:
        tuple: Augmented (image, shrink_mask, offsets, spw).
    """
    img_np = np.array(image)
    h, w = img_np.shape[:2]
    # Random Horizontal Flip
    if random.random() < 0.5:
        img_np = cv2.flip(img_np, 1)
        shrink_mask = np.fliplr(shrink_mask)
        spw = np.fliplr(spw)
        offsets = np.fliplr(offsets)
    # Random Rotation
    angle = random.uniform(-10, 10)
    center = (w / 2, h / 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
```

```
img_np = cv2.warpAffine(img_np, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT)
shrink_mask = cv2.warpAffine(shrink_mask, M, (w, h), flags=cv2.INTER_NEAREST, borderMode=cv2.BORDER_REFLECT)
spw = np.stack([cv2.warpAffine(spw[i], M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT) for i in range(9)], axis=0)
offsets = cv2.warpAffine(offsets, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REFLECT)
# Random Scaling
scale = random.uniform(0.8, 1.2)
new_h, new_w = int(h * scale), int(w * scale)
img_np = cv2.resize(img_np, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
shrink_mask = cv2.resize(shrink_mask, (new_w, new_h), interpolation=cv2.INTER_NEAREST)
spw = np.stack([cv2.resize(spw[i], (new_w, new_h), interpolation=cv2.INTER_LINEAR) for i in range(9)], axis=0)
offsets = cv2.resize(offsets, (new_w, new_h), interpolation=cv2.INTER_LINEAR)
# Random Cropping or Padding
target_h, target_w = target_size
if new_h > target_h and new_w > target_w:
    top = random.randint(0, new_h - target_h)
    left = random.randint(0, new_w - target_w)
    img_np = img_np[top:top + target_h, left:left + target_w]
    shrink_mask = shrink_mask[top:top + target_h, left:left + target_w]
    spw = spw[:, top:top + target_h, left:left + target_w]
    offsets = offsets[top:top + target_h, left:left + target_w]
```

```
        else:
            pad_h = max(0, target_h - new_h)
            pad_w = max(0, target_w - new_w)
            img_np = cv2.copyMakeBorder(img_np, 0, pad_h, 0, pad_w,
cv2.BORDER_CONSTANT, value=0)
            shrink_mask = cv2.copyMakeBorder(shrink_mask, 0, pad_h, 0, pad_w,
cv2.BORDER_CONSTANT, value=0)
            spw = np.pad(spw, ((0, 0), (0, pad_h), (0, pad_w)), mode='constant',
constant_values=0)
            offsets = cv2.copyMakeBorder(offsets, 0, pad_h, 0, pad_w,
cv2.BORDER_CONSTANT, value=0)
            img_np = img_np[:target_h, :target_w]
            shrink_mask = shrink_mask[:target_h, :target_w]
            spw = spw[:, :target_h, :target_w]
            offsets = offsets[:target_h, :target_w]
        aug_image = Image.fromarray(img_np)
        return aug_image, shrink_mask, offsets, spw
#---------------------------------
# Visualization code
#---------------------------------
def unnormalize(img_tensor):
    # Assuming normalization with mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = img_tensor.permute(1, 2, 0).cpu().numpy()
    img = std * img + mean
    img = np.clip(img, 0, 1)
    return img
def visualize_sample(sample):
    # Get the first image from the batch
    image = unnormalize(sample['image'][0])
    shrink_mask = sample['shrink_mask'][0, 0].cpu().numpy()
    spw = sample['spw'][0, 0].cpu().numpy()
    offsets = sample['offsets'][0].cpu().numpy()  # shape: [2, H, W]
    offset_mag = np.sqrt(offsets[0]**2 + offsets[1]**2)
    fig, axs = plt.subplots(1, 4, figsize=(20, 5))
    axs[0].imshow(image)
    axs[0].set_title("Input Image")
    axs[0].axis('off')
    axs[1].imshow(shrink_mask, cmap='gray')
    axs[1].set_title("Shrink Mask GT")
    axs[1].axis('off')
    axs[2].imshow(offset_mag, cmap='viridis')
    axs[2].set_title("Offset Magnitude GT")
    axs[2].axis('off')
    axs[3].imshow(spw, cmap='jet')
    axs[3].set_title("SPW GT")
    axs[3].axis('off')
    plt.show()
# ------------------------------
# Training Script
# ------------------------------

config = {
    'data_path': r'F:\SynthText\SynthText',
    'batch_size': 8,
    'num_workers': 1,
    'lr': 1e-3,
    'epochs': 15,
    'img_size': (640, 640),
    'device': 'cuda' if torch.cuda.is_available() else 'cpu',
    'n_samples': 1500
}
def train():
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
    print(f"Device = {config['device']}")
    # dataset = SynthTextDataset(config['data_path'], transform=transform, augment=True,
sample_size=config['n_samples'])
    dataset = ICDAR15DetectionDataset(
        img_dir=r'D:\Projects\RPP\Sem 7\RISTE\data\raw\icdar_og\ch4_training_images',
        gt_dir=r'D:\Projects\RPP\Sem
7\RISTE\data\raw\icdar_og\ch4_training_localization_transcription_gt',
        transform=transform,  # Add your PyTorch transforms here
        augment=True,
        sample_size=1000
    )
    dataloader = DataLoader(
        dataset,
        batch_size=config['batch_size'],
        shuffle=True,
        num_workers=config['num_workers'],
        pin_memory=True
    )
    model = RSMTD().to(config['device'])
    if 'checkpoint_path' in config and config['checkpoint_path']:
        checkpoint_path = config['checkpoint_path']
        model.load_state_dict(torch.load(checkpoint_path, map_location=config['device']))
        print(f"Loaded model from {checkpoint_path} for fine-tuning")
    optimizer = optim.AdamW(model.parameters(), lr=config['lr'], weight_decay=1e-4)
    scheduler = optim.lr_scheduler.OneCycleLR(
        optimizer,
        max_lr=config['lr'],
        total_steps=config['epochs'] * len(dataloader),
        pct_start=0.3
    )
    scaler = torch.amp.GradScaler('cuda')
    for epoch in range(config['epochs']):
        model.train()
        running_loss = 0.0
        for batch_idx, batch in enumerate(dataloader):
            # print(f"Step: [{batch_idx}/{len(dataloader)}]")
            torch.cuda.empty_cache()
            images = batch['image'].to(config['device'])
            masks = batch['shrink_mask'].to(config['device'])
            offsets = batch['offsets'].to(config['device'])
            spw = batch['spw'].to(config['device'])
            optimizer.zero_grad()
            pred_masks, pred_offsets, pred_spw = model(images)
            loss = total_loss(pred_masks, pred_offsets, pred_spw, masks, offsets, spw)
            scaler.scale(loss).backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=5.0)
            scaler.step(optimizer)
            scaler.update()
            scheduler.step()
            running_loss += loss.item()
            print(f"Batch {batch_idx}, Loss: {loss.item():.4f}")
        avg_loss = running_loss / len(dataloader)
```

```
        print(f'Epoch {epoch+1}/{config["epochs"]} Loss: {avg_loss:.4f}')
        torch.save(model.state_dict(), f'./model/shrink_detector/rsmtd_{epoch+1}.pth')
    torch.save(model.state_dict(), './model/shrink_detector/rsmtd_final.pth')
if __name__ == '__main__':
    train()
```

dataloader.py

```
import os
import cv2
import numpy as np
import scipy.io
from scipy.optimize import linear_sum_assignment
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, random_split
from torchvision import transforms, models
from torchvision.models import resnet50, ResNet50_Weights
from PIL import Image
import pandas as pd
from utils import *
class CroppedSynthTextDataset(Dataset):
    def __init__(self, csv_file, image_dir, transform=None):
        self.data = pd.read_csv(csv_file).iloc[6000:7500,:]
        self.image_dir = image_dir
        self.transform = transform


    def __len__(self):
        return len(self.data)
    def __getitem__(self, idx):
        row = self.data.iloc[idx]
        image_name = row['image_name']
        label = row['text']
        img_path = os.path.join(self.image_dir, image_name)
        image = Image.open(img_path).convert('RGB')
        if self.transform:
            image = self.transform(image)
        return image, label
class SynthTextRecognitionDataset(Dataset):
    def __init__(self, gt_mat_path, image_root, transform=None, max_samples=None):
        super(SynthTextRecognitionDataset, self).__init__()
        self.image_root = image_root
        self.transform = transform
        print("Loading ground truth .mat file...")
        mat = scipy.io.loadmat(gt_mat_path)
        self.imnames = mat['imnames'][0]
        self.wordBB = mat['wordBB'][0]
        self.txt = mat['txt'][0]
        self.samples = []
        num_images = self.imnames.shape[0]
        print(f"Found {num_images} images in gt.mat")
        for i in range(num_images):
            im_item = self.imnames[i]
            imname = im_item[0] if isinstance(im_item, (np.ndarray, list)) else str(im_item)
            img_path = os.path.join(self.image_root, imname)


            txt_item = self.txt[i]
            if isinstance(txt_item, np.ndarray):
                if txt_item.dtype.type is np.str_:
                    text_raw = str(txt_item.squeeze())
                else:
                    text_raw = ''.join(chr(c) for c in txt_item.squeeze())
            else:
                text_raw = str(txt_item)
            words = [w.strip() for w in text_raw.split('\n') if w.strip()]
            bbox = self.wordBB[i]
            if bbox.size == 0:
                continue
            if bbox.ndim == 2:
                bboxes = [bbox]
            else:
                bboxes = [bbox[:, :, j] for j in range(bbox.shape[2])]
            if len(words) != len(bboxes):
                min_len = min(len(words), len(bboxes))
                words = words[:min_len]
                bboxes = bboxes[:min_len]
            for j in range(len(words)):
                self.samples.append((img_path, bboxes[j], words[j]))
        if max_samples is not None and len(self.samples) > max_samples:
            np.random.seed(42)
            indices = np.random.choice(len(self.samples), max_samples, replace=False)
            self.samples = [self.samples[i] for i in indices]
        print(f"Total word instances: {len(self.samples)}")
    def __len__(self):
        return len(self.samples)


    def __getitem__(self, idx):
        img_path, bbox, word_text = self.samples[idx]
        img = cv2.imread(img_path)
        if img is None:
            raise FileNotFoundError(f"Image not found: {img_path}")
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        try:
            word_img = crop_word(img, bbox)
        except Exception as e:
            print(f"Error cropping image {img_path} with bbox {bbox}: {e}")
            word_img = np.zeros((32, 128, 3), dtype=np.uint8)
            word_text = ""
        word_img = Image.fromarray(word_img)
        if self.transform:
            word_img = self.transform(word_img)
        return word_img, word_text
class IIIT5KDataset(Dataset):
    def __init__(self, img_dir, labels_file, mode, transform=None):
        """
        Args:
            img_dir (str): Path to the directory containing images.
            labels_file (str): Path to the .mat file containing image names and labels.
            transform (callable, optional): Optional transform to be applied on an image.
        """
        self.img_dir = img_dir
        self.transform = transform
        # Load .mat file and extract necessary fields
        data = scipy.io.loadmat(labels_file)
        if mode == "train":
            annotations = data['traindata'][0]
        else:
            annotations = data['testdata'][0]  # testdata is a 1x2000 struct array
        # Extract image names and ground truth labels
        self.image_names = [str(entry['ImgName'][0]) for entry in annotations][:100]
        self.labels = [str(entry['GroundTruth'][0]) for entry in annotations][:100]
    def __len__(self):
        return len(self.image_names)
    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.image_names[idx])
        label = self.labels[idx]
        # Load image
        image = Image.open(img_path).convert("RGB")  # Ensure 3-channel format
        # Apply transformations if provided
        if self.transform:
            image = self.transform(image)
        return image, label
class ICDAR15Dataset(Dataset):
    def __init__(self, img_dir, gt_file, transform=None):
        """
        Args:
            img_dir (str): Path to the directory containing images.
            gt_file (str): Path to the text file containing image names and labels.
            transform (callable, optional): Optional transform to be applied on an image.
        """
        self.img_dir = img_dir
        self.gt_file = gt_file
        self.transform = transform
        self.data = []
        # Read and parse the ground truth file
        with open(self.gt_file, 'r', encoding='utf-8-sig') as f:
            for line in f:
                # Split on the first occurrence of ', ' to handle labels with commas
                parts = line.strip().split(', ', 1)
                if len(parts) == 2:
                    image_name = parts[0].strip()
                    # Remove quotes from the label
                    label = parts[1].strip().strip('"')
                    self.data.append((image_name, label))
        print(f"Loaded {len(self.data)} samples from {self.gt_file}")
    def __len__(self):
        """Return the total number of samples in the dataset."""
        return len(self.data)
    def __getitem__(self, idx):
        """
        Args:
            idx (int): Index of the sample to fetch.
        Returns:
            tuple: (image, label) where image is a PIL Image and label is a string.
        """
        # Get image name and label for the given index
        image_name, label = self.data[idx]
        # Construct the full image path
        img_path = os.path.join(self.img_dir, image_name)
        # Load the image and convert to RGB
        image = Image.open(img_path).convert('RGB')
        # Apply transformations if provided
        if self.transform:
            image = self.transform(image)
        return image, label
```

vision_transformer.py

```
import torch
import torch.nn as nn
import logging
from copy import deepcopy
from functools import partial
from timm.models.vision_transformer import VisionTransformer, _cfg
from timm.models import create_model
class TextRecognizer(VisionTransformer):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
    def reset_classifier(self, num_classes):
        self.num_classes = num_classes
        self.head = nn.Linear(self.embed_dim, num_classes) if num_classes > 0 else
nn.Identity()
```

```
    def forward_features(self, x):
        B = x.shape[0]
        x = self.patch_embed(x)
        cls_tokens = self.cls_token.expand(B, -1, -1)  # stole cls_tokens impl from Phil Wang,
thanks
        x = torch.cat((cls_tokens, x), dim=1)
        x = x + self.pos_embed
        x = self.pos_drop(x)
        for blk in self.blocks:
            x = blk(x)
        x = self.norm(x)
        return x
    def forward(self, x, seqlen: int =25):
        x = self.forward_features(x)
        x = x[:, :seqlen]
        # batch, seqlen, embsize
        b, s, e = x.size()
        x = x.reshape(b*s, e)
        x = self.head(x).view(b, s, self.num_classes)
        return x
```

utils.py

```
import torch
import argparse
from PIL import Image
from torchvision import transforms

class TextRecognitionFeatureExtractor:

    def __init__(self, input_channel=1, imgH=224, imgW=224):

        self.imgH = imgH

        self.imgW = imgW

        self.transform = NormalizePAD((input_channel, imgH, imgW))

    def __call__(self, img_path):

        img = Image.open(img_path).convert('L')

        img = img.resize((self.imgW, self.imgH), Image.BICUBIC)

        img = self.transform(img)

        img = torch.unsqueeze(img, dim=0)

        return img

class NormalizePAD:

    def __init__(self, max_size, PAD_type='right'):

        self.toTensor = transforms.ToTensor()

        self.max_size = max_size

        self.max_width_half = max_size[2] // 2

        self.PAD_type = PAD_type

    def __call__(self, img):

        img = self.toTensor(img)

        img.sub_(0.5).div_(0.5)

        c, h, w = img.size()

        pad_img = torch.FloatTensor(*self.max_size).fill_(0)

        pad_img[:, :, :w] = img  # right pad
        if self.max_size[2] != w:  # add border Pad
            pad_img[:, :, w:] = img[:, :, w - 1].unsqueeze(2).expand(c, h, self.max_size[2] - w)
        return pad_img
class TokenLabelConverter:
    """ Convert between text-label and text-index """
    def __init__(self, args):
        # character (str): set of the possible characters.
        # [GO] for the start token of the attention decoder. [s] for end-of-sentence token.
        self.SPACE = '[PAD]'
        self.GO = '[START]'
        self.list_token = [self.GO, self.SPACE]
        self.character = self.list_token + list(args.character)

        self.dict = {word: i for i, word in enumerate(self.character)}
        self.batch_max_length = args.batch_max_length + len(self.list_token)
    def encode(self, text):
        """ convert text-label into text-index.
        """
         length = [len(s) + len(self.list_token) for s in text]  # +2 for [GO] and [s] at end of
sentence.
                                                batch_text          =          torch.LongTensor(len(text),
self.batch_max_length).fill_(self.dict[self.GO])
        for i, t in enumerate(text):
            txt = [self.GO] + list(t) + [self.SPACE]
            txt = [self.dict[char] for char in txt]

            batch_text[i][:len(txt)] = torch.LongTensor(txt)  # batch_text[:, 0] = [GO] token

        return batch_text.to(device)


    def decode(self, text_index, length):

        """ convert text-index into text-label. """

        texts = []

        for index, l in enumerate(length):

            text = ''.join([self.character[i] for i in text_index[index, :]])

            texts.append(text)

        return texts


def get_args():

    parser = argparse.ArgumentParser(description='ViTSTR evaluation')

    parser.add_argument('--image', default=None, help='path to input image')

    parser.add_argument('--batch_max_length', type=int, default=25, help='maximum-label-
length')

    parser.add_argument('--imgH', type=int, default=224, help='the height of the input image')

    parser.add_argument('--imgW', type=int, default=224, help='the width of the input image')

    parser.add_argument('--rgb', action='store_true', help='use rgb input')

    parser.add_argument('--character', type=str,

                        default='0123456789abcdefghijklmnopqrstuvwxyz', help='character label')

    parser.add_argument('--input-channel', type=int, default=1,

                        help='the number of input channel of Feature extractor')
    parser.add_argument('--model',  help='text recognition model')
    parser.add_argument('--gpu', action='store_true', help='use gpu for model inference')
    args = parser.parse_args()
    return args
```

ctc_decoder.py

```
class CTCLabelConverter(object):
    """ Convert between text-label and text-index """
    def __init__(self, character):
        # character (str): set of the possible characters.
        dict_character = list(character)
        self.dict = {}
        for i, char in enumerate(dict_character):
            # NOTE: 0 is reserved for 'CTCblank' token required by CTCLoss
            self.dict[char] = i + 1
          self.character = ['[CTCblank]'] + dict_character  # dummy '[CTCblank]' token for
CTCLoss (index 0)
    def encode(self, text, batch_max_length=25):
        length = [len(s) for s in text]
        # The index used for padding (=0) would not affect the CTC loss calculation.
        batch_text = torch.LongTensor(len(text), batch_max_length).fill_(0)
        for i, t in enumerate(text):
            text = list(t)
            text = [self.dict[char] for char in text]
            batch_text[i][:len(text)] = torch.LongTensor(text)
        return (batch_text.to(device), torch.IntTensor(length).to(device))

    def decode(self, text_index, length):

        """ convert text-index into text-label. """

        texts = []

        for index, l in enumerate(length):

            t = text_index[index, :]

            char_list = []

            for i in range(l):

                if t[i] != 0 and (not (i > 0 and t[i - 1] == t[i])):  # removing repeated characters and
blank.

                    char_list.append(self.character[t[i]])

            text = ''.join(char_list)

            texts.append(text)

        return texts
```

REFERENCES

1. Ashish Vaswani , Noam Shazeer, Niki Parmar, Jakob Uszkoreit , Llion Jones, Aidan N. Gomez, and Łukasz Kaiser, "Attention Is All You Need", 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. https://arxiv.org/abs/1706.03762

2. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weis- senborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, "AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE", ICLR 2021. https://arxiv.org/abs/2010.11929

3. P. Shivakumara, A.Banerjee, U.Pal, L.Nandanwar and Cheng-Lin Liu "A New Language-Independent Deep CNN for Scene Text Detection and Style Transfer in Social Media Images", IEEE Transactions on Image Processing, Vol. 32, 2023. https://doi.org/10.1109/TIP.2023.3287038

4. Jianqi Ma, Shi Guo, and Lei Zhang, "Text Prior Guided Scene Text Image", IEEE Transactions On Image Processing, Vol. 32, 2023

5. D.Zhong, H.Zhan, S.Lyu, C.Liu, B.Yin, P. Shivakumara, U.Pal and Y.Lu "NDOrder: Exploring a novel decoding order for scene text recognition", Expert Systems With Applications 249 (2024). https://www.sciencedirect.com/science/article/pii/S0957417424006377

6. Ayan Banerjee, Shivakumara Palaiahnakote, Umapada Pal, Apostolos Antonacopoulos, Tong Lu and Josep Llados Canet "TTS: Hilbert Transform-Based Generative Adversarial Network for Tattoo and Scene Text Spotting", Engineering Applications of Artificial Intelligence 133 (2024). https://ieeexplore.ieee.org/document/10474185/

7. Xuejian Rong, Chucai Yi, and Yingli Tian "Unambiguous Text Localization, Retrieval, and recognition for Cluttered Scenes", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 44, NO. 3, MARCH 2022. https://ieeexplore.ieee.org/document/9173716

8. Shi-Xue Zhang , Chun Yang, Xiaobin Zhu , Hongyang Zhou , Hongfa Wang, and Xu-Cheng Yin, "Inverse-Like Antagonistic Scene Text Spotting via Reading-Order Estimation and Dynamic Sampling", IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 33, 2024. https://arxiv.org/abs/2401.03637

9. Lintai Wu, Yong Xu, Junhui Hou, C. L. Philip Chen and Cheng-Lin Liu "A Two-Level Rectification Attention Network for Scene Text Recognition", IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 25, 2023. http://ieeexplore.ieee.org/document/9695247/

10. Yuliang Liu , Chunhua Shen , Lianwen Jin , Tong He , Peng Chen, Chongyu Liu , and Hao Chen "ABCNet v2: Adaptive Bezier-Curve Network for Real-Time End-to-End Text Spotting", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 44, NO. 11, NOVEMBER 2022. https://arxiv.org/abs/2105.03620

11. Meng Cao , Can Zhang , Dongming Yang, and Yuexian Zou "All You Need Is a Second Look: Towards Arbitrary-Shaped Text Detection", IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 32, NO. 2, FEBRUARY 2022. https://arxiv.org/abs/2106.12720

12. Hengyue Bi , Canhui Xu , Cao Shi , Guozhu Liu , Honghong Zhang , Yuteng Li , and Junyu Dong “HGR-Net: Hierarchical Graph Reasoning Network for Arbitrary Shape Scene Text Detection”, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 32, 2023. https://ieeexplore.ieee.org/document/10185179/

13. Chuang Yang , Mulin Chen , Zhitong Xiong, Yuan Yuan, and Qi Wang “CM-Net: Concentric Mask Based Arbitrary-Shaped Text Detection”, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 31, 2022. https://ieeexplore.ieee.org/document/9744124/

14. Wenhai Wang, Enze Xie, Xiang Li, Xuebo Liu, Ding Liang, Zhibo Yang, Tong Lu and Chunhua Shen “PAN++: Towards Efficient and Accurate End-to-End Spotting of Arbitrarily-Shaped Text”, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 44, NO. 9, SEPTEMBER 2022. https://arxiv.org/abs/2105.00405

15. Qitong Wang, Bin Fu , Ming Li, Junjun He , Xi Peng , and Yu Qiao “Region-Aware Arbitrary-Shaped Text Detection With Progressive Fusion”, IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 25, 2023. https://ieeexplore.ieee.org/document/9850390

16. Yu Zhou , Hongtao Xie , Shancheng Fang, and Yongdong Zhang “Semi-Supervised Text Detection With Accurate Pseudo-Labels”, IEEE SIGNAL PROCESSING LETTERS, VOL. 29, 2022. https://ieeexplore.ieee.org/document/9779451/

17. Peng Wang , Hui Li , and Chunhua Shen “Towards End-to-End Text Spotting in Natural Scenes”,IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 44, NO. 10, OCTOBER 2022. https://arxiv.org/abs/1906.06013

18. Wenwen Yu , Yuliang Liu, Xingkui Zhu , Haoyu Cao , Xing Sun , and Xiang Bai “Turning a CLIP Model Into a Scene Text Spotter”, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 46, NO. 9, SEPTEMBER 2024. https://arxiv.org/abs/2308.10408

19. Alex Graves, Santiago Fern´andez, Faustino Gomez , J¨urgen Schmidhuber “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks”, Proceedings of the 23 rd International Conference on Machine Learning, Pittsburgh, PA, 2006. https://www.cs.toronto.edu/~graves/icml_2006.pdf

20. Yu-Xiang Zeng, Jun-Wei Hsieh, Xin Li, Ming-Ching Chang “MixNet: Toward Accurate Detection of Challenging Scene Text in the Wild”, arXiv:2308.12817. https://arxiv.org/abs/2308.12817

21. Qingwen Bu, Sungrae Park, Minsoo Khang, Yichuan Cheng “SRFormer: Text Detection Transformer with Incorporated Segmentation and Regression”, arXiv:2308.10531. https://arxiv.org/abs/2308.10531

22. Maoyuan Ye, Jing Zhang, Shanshan Zhao, Juhua Liu, Bo Du, Dacheng Tao, “DPText-DETR: Towards Better Scene Text Detection with Dynamic Points in Transformer”, arXiv:2207.04491v2. https://arxiv.org/abs/2207.04491

23. Jian Ye, Zhe Chen, Juhua Liu and Bo Du, “TextFuseNet: Scene Text Detection with Richer Fused Features”, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20). https://www.ijcai.org/proceedings/2020/72

24. Miao Rang, Zhenni Bi, Chuanjian Liu, Yunhe Wang, Kai Han, Huawei Noah's Ark Lab, "An Empirical Study of Scaling Law for OCR", arXiv:2401.00028v3. https://arxiv.org/abs/2401.00028

25. Yongkun Du, Zhineng Chen, Caiyan Jia, Xiaoting Yin, Chenxia Li, Yuning Du, and Yu-Gang Jiang, "Context Perception Parallel Decoder for Scene Text Recognition", arXiv:2307.12270v2. https://arxiv.org/abs/2307.12270

26. Peng Wang, Cheng Da, and Cong Yao, "Multi-Granularity Prediction for Scene Text Recognition", arXiv:2209.03592v2. https://arxiv.org/abs/2209.03592

27. Masato Fujitake, "DTrOCR: Decoder-only Transformer for Optical Character Recognition", arXiv:2308.15996v1. https://arxiv.org/abs/2308.15996

28. Chuhui Xue, Jiaxing Huang, Wenqing Zhang, Shijian Lu, Changhu Wang, and Song Bai, "Image-to-Character-to-Word Transformers for Accurate Scene Text Recognition", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 45, NO. 11, NOVEMBER 2023, https://arxiv.org/abs/2105.08383

29. Minyuan Ye, Dong Lyu, And Gengsheng Chen, "Scale-Iterative Upscaling Network for Image Deblurring", IEEE Access

30. Chae-Won Park et.al "OCR-Diff: A Two-Stage Deep Learning Framework for Optical Character Recognition Using Diffusion Model in Industrial Internet of Things", IEEE INTERNET OF THINGS JOURNAL, VOL. 11, NO. 15, 1 AUGUST 2024

31. Chong Liu et.al "Efficient Token-Guided Image-Text Retrieval with Consistent Multimodal Contrastive Training", IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 32, 2023

32. Xin Deng et.al "Omnidirectional Image Super-Resolution via Latitude Adaptive Network", IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 25, 2023

33. Zhichao Zhang et.al "Enhancing OCR with line segmentation mask for container text recognition in container terminal", Engineering Applications of Artificial Intelligence 133 (2024)

34. Chuang Yang, Mulin Chen, Yuan Yuan, and Qi Wang, "Reinforcement Shrink-Mask for Text Detection", IEEE TRANSACTIONS ON MULTIMEDIA, VOL. 25, 2023

35. Tianyu Geng, "Transforming Scene Text Detection and Recognition: A Multi-Scale End-to-End Approach With Transformer Framework", IEEE Access, https://ieeexplore.ieee.org/document/10464316

36. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition", arXiv:1512.03385, https://arxiv.org/abs/1512.03385

37. Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie, "Feature Pyramid Networks for Object Detection", arXiv:1612.03144, https://arxiv.org/abs/1612.03144

Sreenivasan S R

UG Project Report

UG Report

2025-2026

CSE

Document Details

Submission ID	trn:oid:::1:3219659569
Submission Date	Apr 17, 2025, 12:20 PM GMT+5:30
Download Date	Apr 17, 2025, 2:27 PM GMT+5:30
File Name	Project_Report_Chapters.docx
File Size	4.5 MB

54 Pages

12,335 Words

72,161 Characters

13% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Submitted works

Match Groups

- **152** Not Cited or Quoted 12%
  Matches with neither in-text citation nor quotation marks
- **3** Missing Quotations 0%
  Matches that are still very similar to source material
- **2** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

6%	Internet sources
10%	Publications
0%	Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.