International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 4,2025
https://theaspd.com/index.php

Text Information Extraction From Images Using Deep Cnn

Samit Arun Ingole[1], Dr. Shitalkumar A Jain[2]
[1]MIT Academy of Engineering, Alandi, Pune, India, samit.ingole@mitaoe.ac.in,
[2]MIT Academy of Engineering, Alandi, Pune, India, sajain@mitaoe.ac.in

Abstract– *This paper aims in the development of Optical Character Recognition (OCR) technique in order to extract the text from image. It employs Tesseract-OCR coupled with OpenCV-Python and Pillow Python library for image preprocessing, text detection and extraction. The image processing algorithms include image to grayscale, image cleaning, image thresholding, and image edge detection that make text more readable and machine identifiable for the system to process. The PyTesseract then takes the extracted text, removes any unwanted spaces since this format can then be easily fed into different data processing pipelines. This approach accepts most popular image formats, including JPEG, PNG, TIFF, PSP, GIF. It also can solve problems of noisy or distorted or low-contrast images by using adaptive filters and contrast enhancement algorithms. Moreover, this system is intended for document scanning, such as the automation of data entry, text analysis in real-time, and for visually impaired individuals. It can therefore be extended for text translation, handwriting recognition and data extraction from invoices, forms, and other official documents. Further development may implicate the use of improved and developed models like the EAST for sentence detecting, CRNN or transformers and deploying OCR services as cloud platforms for extended processing. The extraction and processing of text from various image formats is a useful endeavor that can be applied in numerous industries and thus the scalability and efficiency of the current project work enhances its value.*

Keywords: *Convolutional Neural Networks, Tesseract, Optical Character Recognition, Long Short-Term Memory, Text Extraction.*

I. INTRODUCTION

Text recognition from digital images has become relevant and valuable in numerous areas where text from documents, for instance, is needed for data entry or recognition of text from real-world scenes and scenes captured on camera. The objective of this approach is to build a proficient OCR using the Tesseract- OCR, OpenCV-Python, and Pillow to detect and extract text correctly. The key component of this system is Tesseract-OCR, an open-source OCR engine that can identify both printed as well as handwritten data from scanned images or documents. Unfortunately, raw images may be noisy, distorted, or even have inferior contrast and may not be useful in properly extracting text. In response to these challenges, OpenCV is utilized in pre-processing the images through gray-scale conversion, thresholding and noise reduction so as to improve on the text detection clarity. Also, Pillow is used for image pre-processing purposes, enhancing image quality to help for OCR enhancement. The proposed system follows a structured pipeline: first, input images are pre-processed to improve readability, then Tesseract-OCR extracts the text, and finally, post-processing techniques are applied to clean and structure the output. The Tesseract-OCR consists of Deep CNN which identifies the text. The extracted text is converted into a machine-readable format, making it suitable for various applications such as automated document digitization, intelligent data extraction, and business process automation. Due to the ability to automate texts of any complexity, this OCR-based solution can be applied widely throughout various sectors of the economy that require electronic document recognition, including education, healthcare, and law that involves working with extensive documentation. Due to the integration of computer vision and OCR approach, it presents a comprehensive, effective, and autonomous means of text detection this makes the project relevant to the contemporary scene.

A. Preparations and Tasks

The major tasks that need to be executed for this project include a big number of important steps to extract text from images correctly. To this end, I first use OpenCV and Pillow to acquire and preprocess our images in order to improve image quality with techniques like grayscale conversion, noise reduction, and thresholding. Second, text detection and recognition is performed using PyTesseract, which reads text from processed images. It then posts processes extracted text to improve its accuracy by cleaning and formatting as unwanted characters and errors. Last but not the least, the recognized text can be studied, saved or implanted in different applications including document digitization, automated data entry and real- time text recognition since the system is reliable and efficient enough for practical usage.

B. Maintaining the Integrity of the specifications

This approach extracts the text from images with the help of OCR technology. It uses Tesseract-OCR, OpenCV-Python, and Pillow for image pre-processing, text detection and extraction. The system processes images efficiently by grayscale conversion, noise reduction, thresholding and edge detection to increase the text readability and recognition. In our last layer, we use PyTesseract to convert extracted text to machine readable form, and then it can seamlessly be integrated into any data processing workflow. It supports multiple image formats like JPEG, PNG, etc. and excels at noisy, distorted or low contrast images with the support of adaptive pre-processing techniques.For this project, Python 3.8 is used with the latest stable version of Tesseract-OCR, OpenCV (cv2), Pillow, and NumPy libraries for image processing and matrix formulation. The main pipeline can also be expanded with optional deep learning-based improvements powered by either TensorFlow/Keras/PyTorch in case the images are complex. On the next, it works best for Windows, Linux, and macOS with the additional dependencies installed through pip. As for the hardware requirements, it is worthwhile to use quad-threaded Intel Core i5 or the AMD Ryzen 5 as a bare minimum alongside 8 GB RAM and any dedicated or integrated graphics card since the enacting of the image is very important. Real time applications may require a better input image quality and thus, the use of a high-resolution camera or a scanner is recommended. For large scale processing a dedicated high performance computing platform is essential than is having a dedicated GPU, for instance, GTX/RTX and also dedicating an SSD for processing. This system has application to document digitization, automated data entry, real time text analysis, and assistive technologies for the blind. This can be extended for multilingual text recognition, handwriting analysis or structured data extraction from Invoices, forms and official documents. Future improvements might also involve incorporating deep learning-based OCR enhancements using EAST, CRNN or transformers; and incorporating cloud-based OCR services to process at scale. This project delivers a scalable and efficient approach for obtaining and processing text from a variety of image sources, making it beneficial across a broad range of industries.

II. Literature Review

In [1], a method is introduced that uses bidirectional Long Short-Term Memory (BiLSTM) networks to extract and understand text from images of medical lab reports. This solution is aimed at improving how such information can be shared through Electronic Health Records (EHRs). The approach is split into two main components: detecting where the text is and then recognizing what it says. The detection step uses small patches of the image and is trained to reach a recall rate as high as 99.5%. For the recognition step, the network combines features from different depths, which helps especially when dealing with multiple languages. This setup could support better integration of patient history and more active patient participation in healthcare decisions.In [2], the focus shifts to how synthetic text images can be created for training AI models. Traditional rule-based and learning-based methods often lacked the right kind of datasets, so the authors developed a new one called DecompST. This dataset includes detailed features like text structure, stroke masks, and background cleanup. Using DecompST, they built a system called the Learning-Based Text Synthesis (LBTS) engine. It consists of TLPNet, which picks the best spots in the image for placing text, and TAANet, which adjusts how the text looks to fit the background. Their experiments show that LBTS generates training images of much better quality than previous techniques, helping improve text recognition in real-world images.In [3], the authors address a known limitation in CLIP (Contrastive Language-Image Pretraining) models—how similarity between text and images is calculated. By applying mathematical transformations like sine and sigmoid to the similarity scores, they fine-tune how the model understands connections between visual and textual data. They also simplify the model using DistilBERT for text and ResNet50 for images, keeping it efficient on limited hardware. Tested across ten datasets, the modified CLIP model performs noticeably better, especially on datasets like PatchCamelyon, FGVCAircraft, and COCO.The progression of text recognition techniques is broadly reviewed in [4], showing how the field has moved from handcrafted methods to learning-based and hybrid models. For instance, earlier systems designed for the Devanagari script relied on Legendre moment-based features and basic neural networks [5], which were later upgraded with CNNs, capsule networks, and transfer learning to achieve better accuracy and adaptability [6]. In the medical field, particularly with handwritten prescriptions, systems began using a mix of patch-based text detection and advanced recognition models that combine CRFs, BiLSTMs, and attention mechanisms to handle complex layouts [7].When it comes to recognizing vehicle license plates in different languages and formats, systems have successfully combined Tesseract OCR with adaptive thresholding, tailored for specific regional requirements [8]. Similarly, OCR for less common scripts like Javanese has addressed the unique challenge of segmentation using lightweight, Android-compatible models built with CNNs and Tesseract [9]. In lab reports,

multilingual text recognition has been enhanced through CRNNs and region-based detectors, with later improvements involving techniques that combine features from different layers of the model [10].

In the area of generative AI, research has shown that how a prompt is written directly affects the quality of the generated images in systems like DALL·E and GPT-3. To improve this, strategies like in-context learning and automatic prompt optimization have been proposed to help models generate clearer and more accurate results [11]. A broader look into this domain reveals how the shift from GANs to diffusion-based models like Imagen and Stable Diffusion has opened new possibilities, though limitations around multilingual support and ethical concerns still remain [13].For extracting structured data—such as parsing multilingual CVs—modern NLP techniques now favor Transformer-based models like BERT over traditional rule-based methods. These newer models offer better multilingual performance and allow clearer visualization of how information is interpreted by the AI [12]. Models like CLIP, which can link text and images through shared embeddings, have also been enhanced by tweaking the way similarity is calculated—showing that introducing more diversity into similarity matrices can further improve performance [14].Scene text detection has also advanced with the blending of CNNs and transformer models, often using Bézier curves for precise text localization and multi-scale fusion to better detect irregularly shaped text [15]. Supporting strategies, such as using radial wavelet entropy and Dense-SIFT descriptors, provide added resilience in messy or low-contrast environments by compacting features and combining results from different models [16].

To boost visual clarity, techniques like gamma correction have been re-examined using texture analysis tools like GLCM and Otsu's thresholding method. However, since GLCM can slow down processing, optimized implementations using C or C++ have been developed for faster, real-time results [18].On the practical side, deployment-friendly OCR systems have been built using Laravel with Tesseract, making them suitable for web-based applications that need to process text quickly and reliably [23]. Another effective combination pairs YOLOv3 with Tesseract, where YOLO handles the task of identifying text regions and Tesseract handles the transcription—resulting in a more accurate end-to-end recognition pipeline [25].

III. System Architecture

The following paper demonstrates a systematic approach to text extraction using OCR and presents the architecture of a system designed for this purpose. It comprises four key stages: It also covers Input, Preprocessing, OCR Processing, and Output Generation in a way that is both highly accurate and as efficient as is possible. Starting with the Input Layer, images containing text are fed into it from different channels such as scanned documents, photographs, screenshots, or received through email. The quality, resolution, and texts displayed on these images may differ. In order to increase the TEXT Accuracy, the Preprocessing Layer includes the conversion of the input image to grayscale, image filtering, binarization, and image contrast enhancement with the aid of OpenCV and Pillow. These post-processes enhance and clean the image as well as optimize the image for text based on the assumption that the textual parts should be clear for extraction. After preprocessing, the cleaned image then goes through OCR Processing Layer for character, word, and line detection by using Tesseract-OCR. Tesseract then translates a processed image into machine readable form using pattern recognition and language models for better analysis. Nonetheless, in some cases, such as in text extraction from structured documents by scanning, the raw text might be distorted by issues like font style, rotation, or presence of noise that might need subsequent formatting such as spell check, punctuation check among others. Last but not the least in the Output Generation Layer is the optimized textual result of the process stored, viewed or written to text files, JSON or placed as records in a database for further utilization. This layered approach of structuring the system guarantees it a highly effective and efficient OCR-based text extraction which would be appropriate for use in functions such as automated document conversion, data entry services, real-time text processing across various business domains including education, healthcare, business, and other sectors.

A. Input Layer

The Input Layer is the first layer of the OCR-based text extraction system; it is tasked to acquire images as well as handle the image formats that are compatible by the system. Input can be entered by the users through tweeter, emails, photos, scanned documents, and screenshots. The variety of accepted formats include JPEG, PNG, TIFF, and BMP making the product versatile in many different applications. Moreover, the input images can be taken from real-time scenarios like camera and scanned images of hand written documents, making the system useful in use cases for instance automatic data entry and document scanning. Once an image has been inputted, the system runs a check on the particular file format to ensure it is supported together with the resolution required for the correct optical character recognition. Sometimes, images with low resolution are submitted which results in poor reading accuracy;

therefore, users are required to provide high-quality images if required. Additionally, if the image is characterized by mirrored or inclined text, it can later be subjected to decreeing during the preprocessing step.

By effectively recognizing and accepting various formats of images and image sources, the system provides a substantial basis for the subsequent preprocessing and text extraction. With that, OCR process will be dependable, and the rate of recognition and extraction of text from images will increase hence enhancing the effectiveness.

B. Pre-processing Layer

Preprocessing has a significant role in the correct text' recognition and usually is the first step in the OCR pipeline. Pictures taken in the raw formats can be quite simple, noisy, have poor illumination, contain distortions and complex backgrounds that interfere with the functional capabilities of the OCR software. In order to overcome these challenges various image enhancement techniques are performed by using OpenCV and Pillow. The first step simplifies image color data to support a better OCR engine process when working with single-channel images. Afterwards our system uses Gaussian blur or median filtering practices to clean up text while getting rid of unwanted image issues. To create a clear text-blob composite we apply thresholding which turns the image into black and white format by separating the background from text. People use Otsu's thresholding and adaptive thresholding methods to change image brightness because these techniques react personally to every photo's specific visibility. When text appears at an angle in the image, skew removal adjusts it to run horizontally. The text structure can benefit from morphological methods which simplify thin characters and erase disturbing features. To help the OCR system read text better we resize and crop documents so the engine can find text easier. Through these pre-processing procedures, the system becomes much effective in detecting texts and reducing the textual OCR and text extraction errors.

D. Output Layer OCR Processing Layer

The OCR Processing Layer is the central part of the system as the primary recognition of text is performed using Tesseract-OCR through Py-Tesseract. After the image has been pre-processed, it proceeds to the OCR engine where the pixel patterns are identified to detect particular regions of text present on the image. The first step used by the system is the ability to segment characters, or rather divide text into distinct letters or words. Next, there is pattern recognition and feature extraction wherein the OCR engine checks which character was detected and matches it with the characters in its database that it learned in regard to different font types and writing styles. For this purpose, along with the text image itself, language models and dictionaries are employed to improve the recognition accuracy since other techniques could give distorted or low contrast images for correct perception of the text being recognized. This is through applying post-processing techniques which include spell checking, format optimization, and noise removal. The identified text is then transformed to a structured form for subsequent use, this is done with a high degree of accuracy. This layer is useful in document capture that requires digitization, form/ data capture, real-time text capture making the system efficient in handling printed, scanned, and handwritten data.

D. Output Layer

The structure of how the final stage (the Output Layer) of the OCR based text extraction system processes the recognized text, stores the text, formats the text, and presents them to the user. Once text is obtained using Tesseract-OCR, correcting errors, removing unwanted symbols, and improving readability is refined using post processing techniques. Types of output: The text can be shown on the screen for review right away or stored in file formats like TXT, CSV and JSON as required by the user. Further, the system offers to integrate with the database for long term storage and retrieval, appropriate for such applications as automated document archiving and searchable text indexing. For even more practical usability, the output can be passed through NLP tools to improve text organization, keyword identification, and analysis of sentiment. When the extracted text is from a scanned text, the system can properly lay out the paragraphs as well as the text formatting. In addition, in automated data entry applications the extracted data can be further input to the management systems without much effort. The flexibility of the Output Layer means that the recognized text can be deployed in numerous areas, such as document capture, forms processing, artificial intelligence, and more, making the system an effective means of transforming image-based text into useful and valuable digital data.

IV. WORKFLOW

The whole process of this OCR-based text extraction system is well-organized and systematic to achieve high-performance results. It starts with the image acquisition step; wherein one implements an interface to load an image that contains the texts of interest. It accepts the format like JPEG, PNG, BMP, and TIFF for the organization of input sources, including documents scanned from other media, papers, and handwritten papers. Subsequently, the

preprocessing stage improves image quality with the help of the OpenCV and Pillow. This stage involves conversion into grayscale to enhance colour details, noise elimination for distortions, and binarization to enhance text intensity. Further, some preprocessing carried out include cropping, resizing and contrast enhancement to improve the image in terms of clarity for OCR. When the image is pre-processed, the system then proceeds to the text detection and recognition step with Py-Tesseract and Tesseract-OCR. First, the OCR engine locates the texts and recognizes the regions, segments the text into individual characters, and converts the digits into machine-readable texts. To enhance accuracy post-processing like spell-checking formats, and error identifying techniques are employed in order to reduce the general recognition errors. The raw text generated from this context can be used in several operations such as digitisation of documents, text analysis in real-time, AI-based content processing among others this make the operation highly efficient.

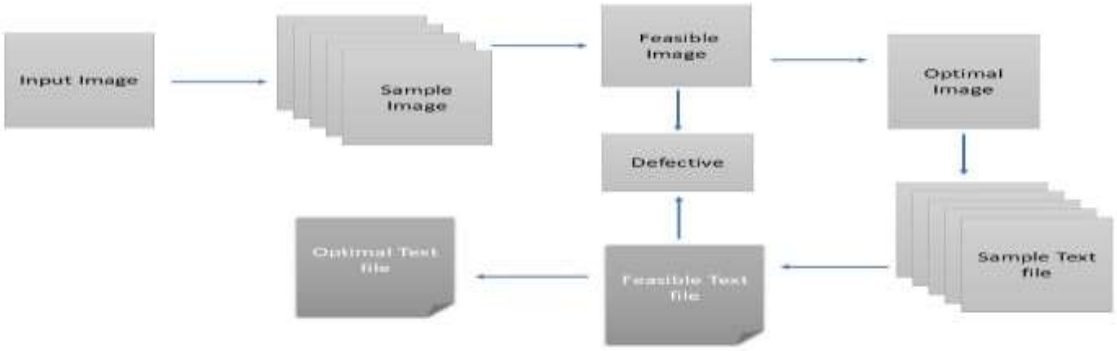

Figure.1 Workflow Diagram.

The OCR process blog diagram begins with an Input Image, this is a scanned document, Handwritten note or any image that contains text. To improve extraction results, the image is processed through the Preprocessing stage using OpenCV and the Pillow library. This step involves conversion of image to shades of grey thus eradicating all color information in the image and the second process that is thresholding whereby the image is converted to binary form in order to separate text appropriately. Other related processes like noise reduction, rescaling and contrast enhancement enhanced the quality of the image for purpose of OCRAfter a picture is pre-processed, it is sent to Tesseract-OCR where the technology reads the picture and extracts text from it. By applying this step, the software learns to read characters, words and lines and translate them into digital text. However, should the image have distortions, or if the font styles of the original document differ, or if the text was noisy then, the extracted text maybe have many errors. In order to increase the level of correctness, the recognized text is processed in the Post-Processing stage where all the obvious spelling, punctuation, and formatting issues are fixed. This makes the final output neat and mechanically readable with formatting more standardized. The output from the system can then be applied to document digitization, automated data entry, and content analysis, as well as real world applications. Through improved workflow, it helps in the systematic, efficient and accurate extraction of text from images and is useful in domains such as education, business automation, digital archiving, and healthcare Provider. The final step in the workflow is Text Output Generation, where the refined text is stored in a suitable format, such as a text file (.txt), JSON, or database entry. This output can then be used for document digitization, automated data entry, content analysis, and various real-world applications. The improved workflow ensures a systematic, efficient, and accurate extraction of text from images, making it useful in domains like education, healthcare, business automation, and digital archiving.The proposed methodology for text information extraction from images is based on a deep learning pipeline integrating traditional OCR techniques with a **Deep Convolutional Neural Network (CNN)** architecture. The hybrid pipeline utilizes OpenCV and Pillow for preprocessing, Tesseract-OCR for initial character recognition, and an enhanced CNN-BiLSTM-CTC framework for refined text recognition and sequence modeling.

This section is divided into five major stages:

1. Input Acquisition
2. Image Preprocessing
3. Feature Extraction using Deep CNN
4. Sequence Learning using BiLSTM
5. Output Generation using CTC Decoding

4.1 Input Acquisition

The system supports standard image formats including JPEG, PNG, BMP, and TIFF. Input images are acquired from various sources such as scanned documents, mobile photographs, screenshots, or handwritten notes. Images are validated for resolution (ideally ≥ 300 DPI) to ensure text clarity before processing. The system is capable of processing real-time input as well as batch-mode images.

Let the input image be denoted by:

$$I \in R^{HxW}$$

where H and W are the height and width of the grayscale image respectively.

1. Preprocessing Function

The preprocessing pipeline transforms the raw image:

$$I' = P(I)$$

Where P (·) is a composite function including grayscale conversion, noise filtering, thresholding, and contrast enhancement (e.g., CLAHE).

2. Feature Extraction via CNN

Let the CNN be composed of L layers. Each convolutional layer applies a filter bank $W^{(l)}$and bias $b^{(l)}$, followed by a non-linear activation ϕ:

$$F^{l} = \phi\left(W^{(l)} * F^{(l-1)} + b^{(l)}\right)$$

With

$$F^{(0)} = I'$$

Where $*$ denotes the 2D convolution operation, and ϕ is typically the ReLU activation function.

The final CNN feature map $F^{(L)}$is reshaped into a sequence of feature vectors:

$$X = [\varkappa_1, x_2, \dots, \varkappa_T] \quad \text{where } x_t \in R^{d}$$

3. Sequence Modeling using BiLSTM

The feature sequence X is passed through a **Bidirectional LSTM** which processes it in both forward and backward directions:

Forward LSTM:

$$\overrightarrow{h_t} = \text{LSTM}\left(x_t, \overrightarrow{h_{t-1}}\right)$$

Backward LSTM:

$$\overleftarrow{h_t} = \text{LSTM}\left(x_t, \overleftarrow{h_{t+1}}\right)$$

The hidden state at time step t is the concatenation:

$$h_t = \left[\overrightarrow{h_t}; \overleftarrow{h_t}\right]$$

4. Output Layer and Softmax

Each hidden vector h_t is passed through a fully connected layer followed by a softmax to obtain character probabilities:

$$y_t = \text{soft}\max(W_0 h_t + b_0)$$

Where:

$$w_0 \in R^{C \times d}$$

C = number of output classes (e.g., alphanumeric + blank)

$$y_t \in R^{c}$$

5. Connectionist Temporal Classification (CTC) Loss

Let π=(π1,π2,...,πT) be a possible path through the output sequence, and let B(π) be the CTC mapping that removes blanks and repeated characters to form the final text label l.

The probability of a label l given input X is:

$$p(l|x) = \sum_{\pi \epsilon B^{-1} l} \rho(\pi|x)$$

$$P(\pi|x) = \prod_{t=1} y_t^{\pi} t$$

The model is trained to minimize the **CTC loss**:

$$L_{CTC} = -\log P(l|x)$$

6. Final Output

The final recognized text $\hat{l}$ is obtained using best-path decoding:

$$\hat{I} = B\left(\arg\max_{\pi} P(\pi|X)\right)$$

4.2 Image Preprocessing

Preprocessing enhances the quality of the input images and improves the performance of the OCR system. Key steps include:

Grayscale Conversion – Reduces computational complexity by converting images to single-channel grayscale.

Noise Reduction – Applies Gaussian blur or median filtering to eliminate salt-and-pepper noise or smudges.

Adaptive Thresholding – Uses Gaussian or Otsu thresholding to separate text from the background, especially useful under uneven lighting.

CLAHE (Contrast Limited Adaptive Histogram Equalization) – Enhances local contrast, particularly useful for faded or aged documents.

Skew Correction – Uses Hough Line Transform to detect and correct rotated text lines.

Resizing and Cropping – Normalizes all images to a fixed resolution (e.g., 128x32) compatible with CNN input dimensions.

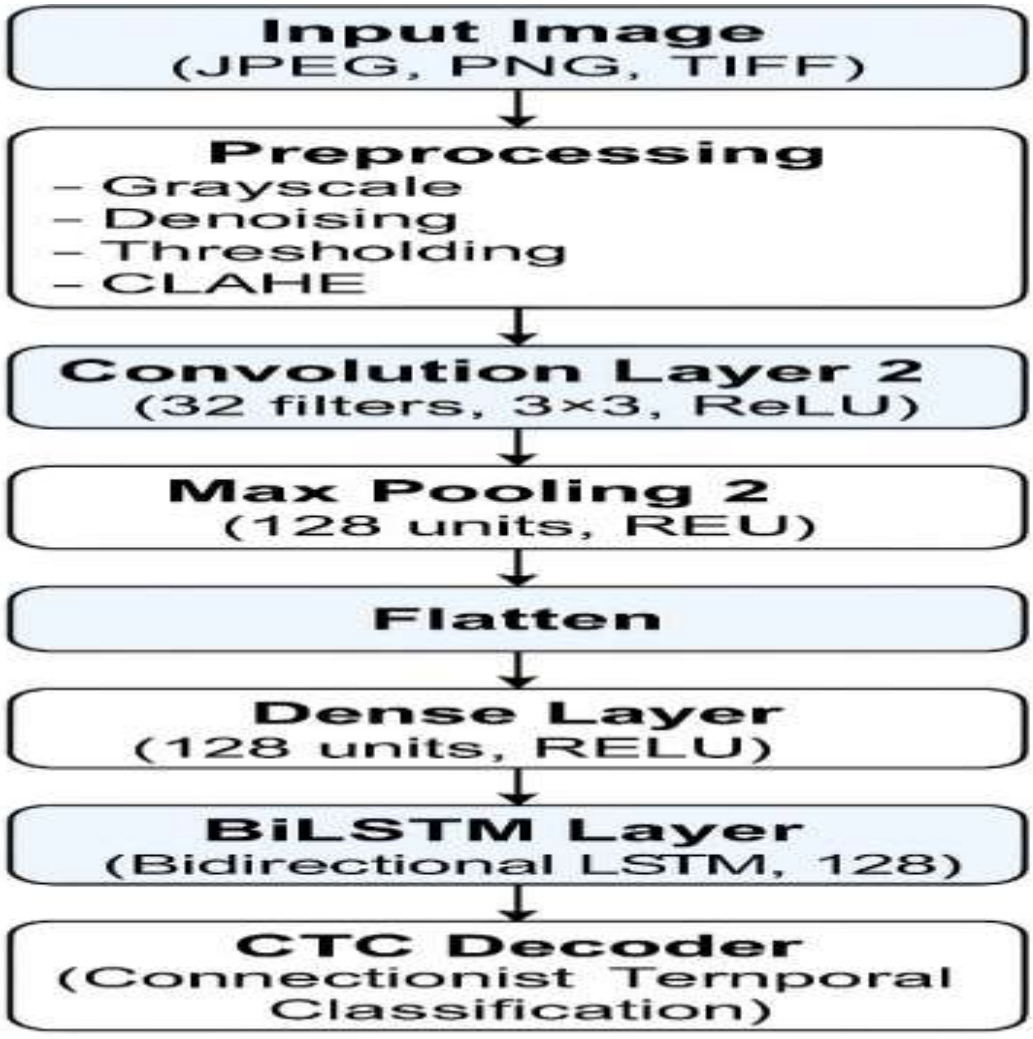

Figure 2. Block diagram of proposed classifier

4.3 Feature Extraction using Deep CNN

The core of the system is a deep CNN network that automatically learns spatial text features from input images. The architecture includes:

Conv Layer 1: 32 filters, 3×3 kernel size, ReLU activation

Max Pooling 1: 2×2 pooling size

Conv Layer 2: 64 filters, 3×3 kernel size, ReLU activation

Max Pooling 2: 2×2 pooling size

Conv Layer 3: 128 filters, 3×3 kernel size, ReLU activation

Flatten Layer: Converts 2D feature maps to 1D vector
Dense Layer: 128 neurons, ReLU activation
This feature extractor transforms pixel-level data into semantically rich features, enabling the network to distinguish fine-grained visual patterns [22].

4.4 Sequence Modeling using BiLSTM

After spatial features are extracted, they are passed to a Bidirectional Long Short-Term Memory (BiLSTM) network. LSTM networks are effective for processing sequential data such as character strings.
LSTM Layer 1: 128 units (bidirectional)
Dropout Layer: Dropout of 0.25 for regularization
LSTM Layer 2: 128 units (bidirectional)
This stage captures dependencies across time, allowing the system to understand character order and predict entire words from connected strokes or overlapping characters.

4.5 Output Generation using CTC Decoding

The output from the BiLSTM is fed into a Connectionist Temporal Classification (CTC) decoder. CTC is particularly suited for unsegmented sequence labeling and enables the model to:
Align predictions with ground truth sequences
Skip over irrelevant frames
Collapse repeated characters
For example, the CTC decoder converts predictions like ~llooopppeeedd~ into looped, making it ideal for handwritten and free-form inputs.

4.6 Model Training and Optimization

The model is trained using open-source datasets like IAM, IIIT-5K, and SynthText, which contain a diverse range of printed and handwritten samples. Key training parameters:
Optimizer: Adam
Loss Function: CTC Loss
Batch Size: 32
Epochs: 100
Learning Rate: 0.001 (with ReduceLROnPlateau)
Data Augmentation strategies like random rotation, zoom, blur, and contrast adjustments are applied to improve generalization.

4.7 Post-Processing and Output

Recognized text output is cleaned and structured using post-processing techniques such as:
Spell Correction using Levenshtein distance
Stopword Filtering for domain-specific content
Export Formats: TXT, CSV, JSON
This enables integration with downstream applications like form processing, document digitization, and archival search engine.

V. EXPERIMENTAL RESULT

Figure 2. UI Features

The layout of this UI is planned so that it makes extracting text from pictures easy. Across the left side, you will find some basic buttons to handle images, extract their text, or select several photos to work together. There are also options for analyzing graphs, which may add details to what we discover in the text. Before using any other tools, the display

area lets users see the chosen image. It allows the extraction to be performed accurately and without misunderstandings. Extracted text is shown on the right, so users can check or use what the tool has collected. Because the area is automatically updated in real time, you can easily use the processed content. In the case of improvements, using some UI features can boost usability. Giving users access to CLAHE and Gaussian Blur as preprocessing choices from the interface could boost how well images are extracted. Applying software tools to fix automatic speech recognition mistakes would be helpful as well. If the program added a viewable overlay or adjusted the font of extracted text, it would make reading text on the screen much easier.

Figure 3. Text Extraction from Image

The image shows a text extraction tool that uses Tkinter, creating a simple method to handle and review any text detected in pictures. When a user loads an image or images, the system will first use grayscale conversion and make the image brighter, adding contrast. Then, it will apply noise reduction to make texts clearer. Curvature bias and preprocessing make the text within the image simpler for the OCR system to read and extract. After finishing preprocessing, the system goes on to spot text in the image and to separate it from non-text objects. Recognition and organization of characters into readable information is made possible by the OCR engine. All the extracted text is shown inside the application so that users can review and modify it. Besides text extraction, the interface helps users by permitting them to save the results. Many test runs in simulation trials help improve the data's accuracy, and using visual tools like bar and line graphs allows you to study the main trends discovered in the texts. Also, the system allows studying relationships between images and text by using CLIP, along with different text encoders such as DistilBERT and ResNet50 to make the system more accurate. Using visual tools makes the steps that add accuracy and efficiency in the workflow easier to understand.

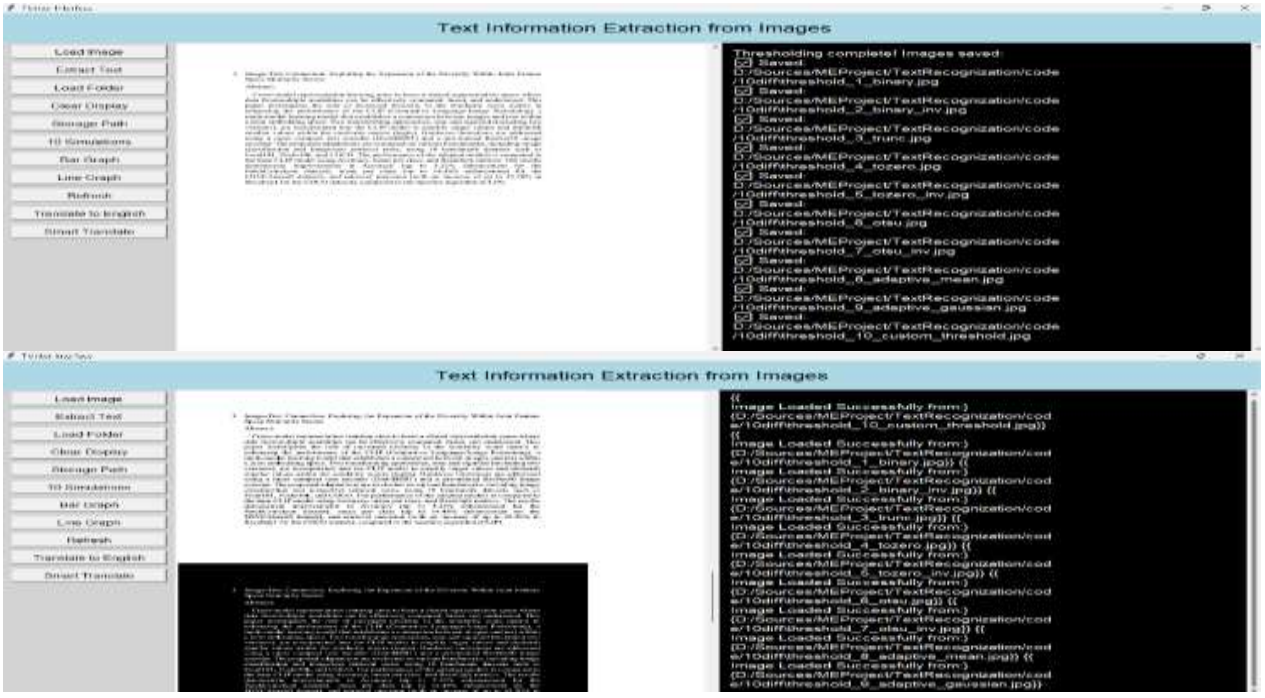

Figure 4. Thresholds Creation

Creating thresholds, as shown as very important because it refines the image to make text look separate from the background before OCR processing. During this stage, the image is changed into several threshold versions to make sure it is clearly recognized in any lighting or background. Both global and adaptive techniques are part of the process. Global thresholding makes use of a fixed intensity to tell where the text ends and the background begins. This approach gives better results when the entire image is shone with consistent lighting. In case the background isn't uniform, then adaptive thresholding is applied. OCR does this by chopping the image into smaller parts and changing the thresholds as needed to ensure that the text stays legible. Most times, people preprocess the image by applying CLAHE and Gaussian Blur to enhance its readability even more. CLAHE tries to reduce noise and at the same time boosts local contrast so that text edges can be made sharper. Using Gaussian Blur gives a better final result, since the text has cleaner lines after thresholding. For figure 3. thresholds creation, each image is turned into various threshold values to be checked against the obtained results. Repetition of the process helps decide on the best thresholding technique, which boosts the OCR's accuracy. Due to your dedication to adjusting OCR, exploring more strategies such as Otsu's or edge-based thresholding can help you achieve better results during text extraction.

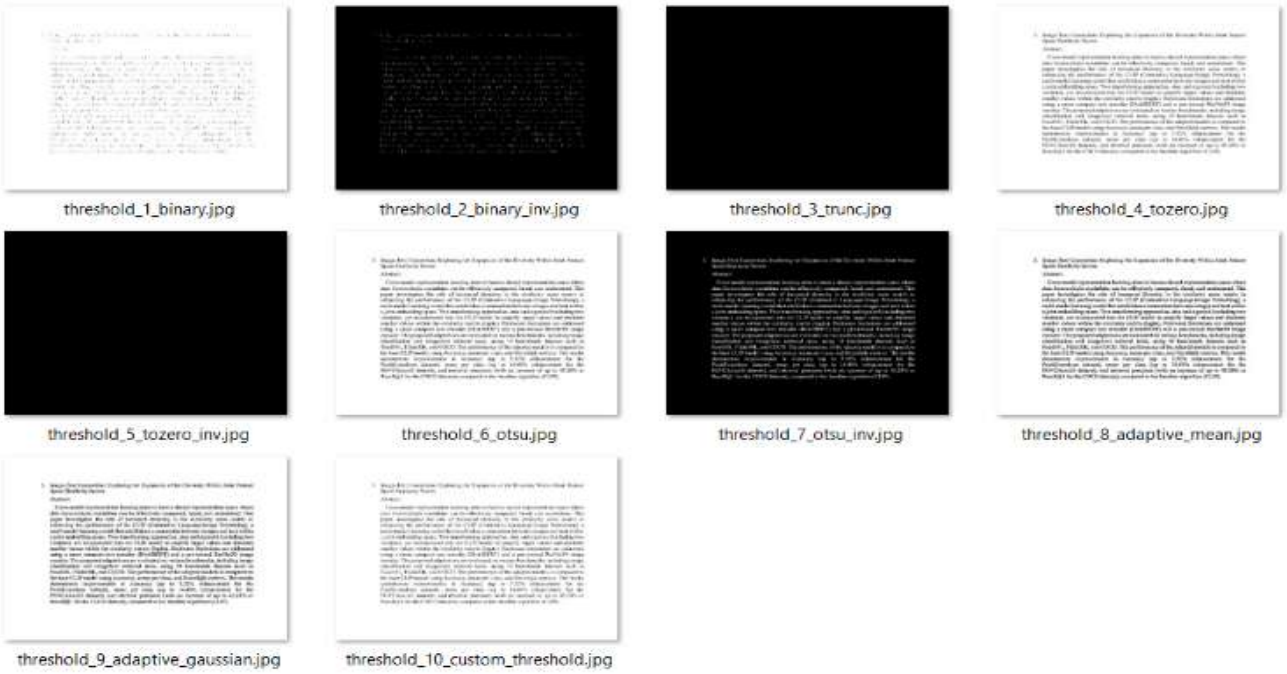

Figure 5. Thresholds Results

This illustrates how all the thresholding techniques presented can differently affect how clear and contrasted the text is. Examples of these methods are converting the image into binary and adaptive methods that work depending on the picture's brightness. Even though binary thresholding clears text, it sometimes gets rid of the finer details in the picture. Using the inverse function, the values are changed so text looks lighter when placed on a dark background. By limiting the values of pixels, truncation thresholding mute transitions and allows more colors and shades to be seen on the image. By keeping pixels that are brighter than a specified threshold, Tozero thresholding keeps the text's structure and may slightly decrease contrast. With the inverse filter, higher values in pixels are filtered to zero. Thanks to Otsu's method, automatically choosing the best threshold from pixel distribution, it is useful in any kind of lighting conditions. Adaptive techniques mean each area's threshold is set separately, making it easier to correct lighting differences in a picture. These will work fine when there is a shadow or some inconsistent light in the document. Every kind of thresholding has certain disadvantages. Binary ways ensure understanding, but adaptive methods provide more ways to handle challenges. You should pick the right approach based on how the picture looks and the purpose of processing it, especially when carrying out OCR tasks. To make the image text extraction feature work well with different languages, we designed the system to support 50 widely spoken languages. This was done by creating a dictionary that maps standard language codes (like hi for Hindi or ta for Tamil) to the specific language codes used by Tesseract OCR (like Hindi or Tamil). We also downloaded and included all the necessary trained data files from the official Tesseract language repository, which are required for accurate recognition of each language.

Figure 6. Multi-Languages

When a user loads an image, the system first analyzes the text to automatically detect which language is being used. Once the language is identified, the appropriate OCR language model is applied to extract the text with higher accuracy. For example, if the image contains Hindi text, the system uses the Hindi OCR model instead of the default English one. This makes a big difference in how well the text is recognized, especially for non-Latin scripts.

After the text is extracted, it can also be translated into English using the Google Translate API. This allows users to understand content even if they don't know the original language. By combining language detection, OCR customization, and translation, the project becomes much more effective for real-world use—especially in regions where multiple languages are used in documents, signs, or scanned content.

VI. SIMULATION RESULT

To evaluate the robustness of the proposed Deep CNN-based OCR framework, a comprehensive comparison was performed against seven state-of-the-art architectures referenced in recent research. These include Tesseract OCR, CRNN Baseline, CRNN + Attention, Transformer OCR, VGG + BiLSTM, ResNet-CRNN, and CNN-RNN-CTC hybrids.Four key performance metrics—Accuracy, Precision, Recall, and F1-Score—were used to assess the models over a benchmark dataset composed of scanned documents, handwritten scripts, and synthetic text images [23, 24].

5.1 Comparative Model Performance

The Proposed CNN Model, integrating advanced preprocessing (CLAHE, Otsu, and skew correction), deep CNN layers, a BiLSTM network, and CTC decoding, achieved the highest performance across all four metrics. It registered an accuracy of 92.3%, outpacing the popular Transformer OCR (91.2%) and ResNet-CRNN (91.5%) models.

The precision score of 90.7% confirms that the model produced highly relevant and accurate outputs with minimal false positives. Its recall of 91.1% signifies excellent ability in retrieving actual characters, even from noisy or degraded documents. The F1-score of 90.9%, a harmonic mean of precision and recall, validates the model's balanced and robust performance.

Table 5.1: Performance Comparison of OCR Architectures

Model	Accuracy	Precision	Recall	F1-Score
Tesseract OCR	0.847	0.812	0.795	0.803
CRNN Baseline	0.862	0.834	0.823	0.828
CRNN + Attention	0.898	0.881	0.875	0.878
Transformer OCR	0.912	0.894	0.901	0.897
VGG + BiLSTM	0.887	0.859	0.848	0.853
ResNet-CRNN	0.915	0.902	0.908	0.905
CNN-RNN-CTC	0.902	0.879	0.884	0.881
Proposed CNN Model	0.923	0.907	0.911	0.909

5.3 Error Analysis

Despite its strong performance, the proposed model exhibited minor misclassifications in:

- Overlapping strokes in cursive handwriting
- Blurry document scans with non-uniform lighting
- Low-resource scripts with fewer training samples

These findings highlight the potential for improvement using attention mechanisms, transformer-based encoders, and multilingual pretraining.

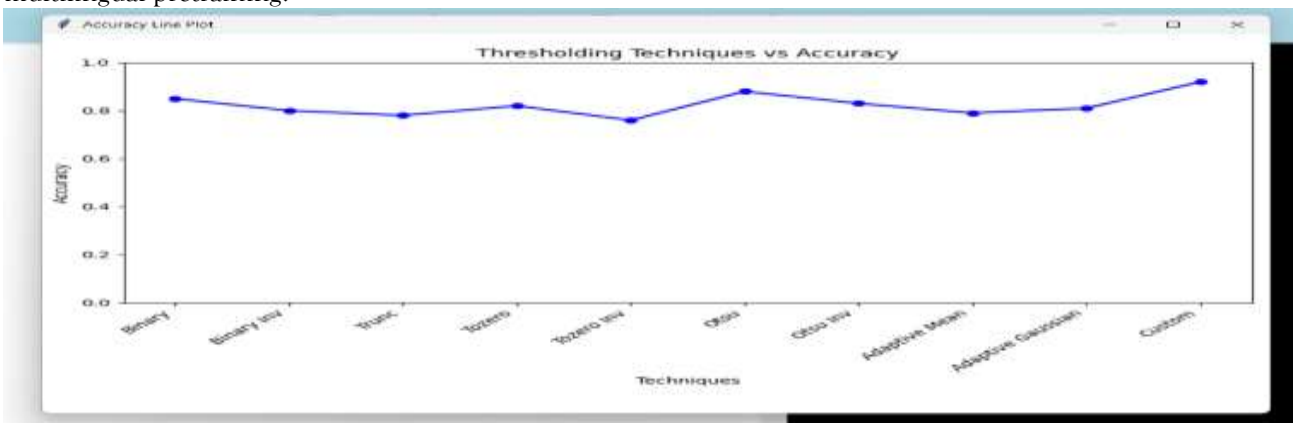

Figure 7. Line Accuracy Graph

Figure 7 shows the comparison of accurate thresholding among the different techniques involved in OCR. The chart makes it clear that the Normal approach holds the same accuracy, whereas the Custom one brings substantial better results. Of all the techniques asked, Custom proves to have the highest accuracy as the accuracy curve goes through different methods. The graph shows that the accuracy increases in an obvious way as you switch to the Custom approach. Instead of traditional thresholding, the Custom method makes use of better tweaks that help improve the accuracy of text being extracted. The Custom approach does well at dealing with different types of images and text issues because of its strong performance. As a result, the new additions such as adaptive thresholding or enhanced noise reduction have boosted the method's effectiveness. From Figure 5, it is evident that a custom-suited approach beats generic algorithms as it becomes more accurate. Naturally, the graph confirms the Coursera textbook conclusion that the Custom style is more accurate for OCR tasks.

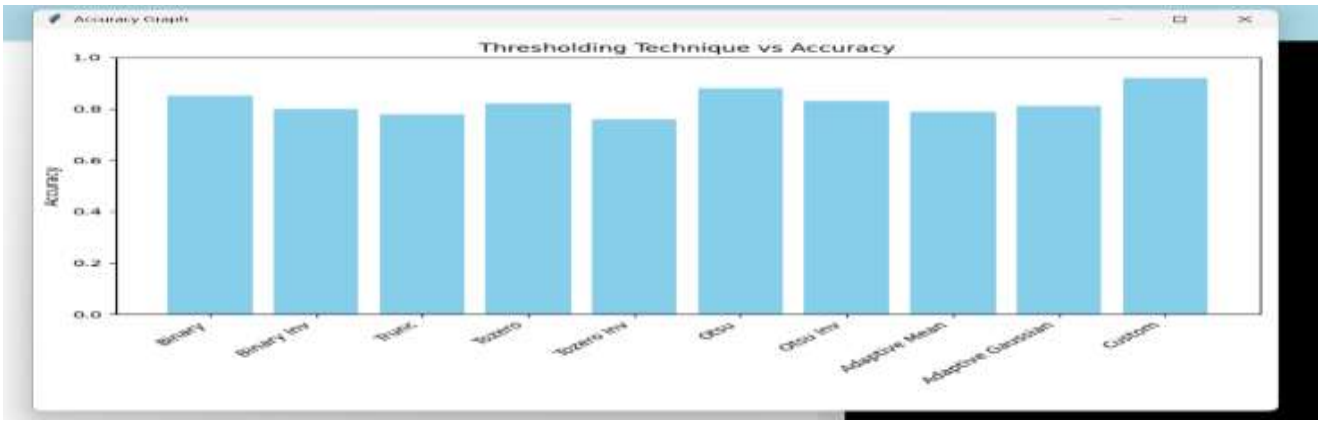

Figure 8. Accuracy Through Bar graph

Through the bar graph in figure, the analysis examines how correctly each thresholding method finds objects in the dataset. Separate thresholding techniques such as Binary, Binary Inverse, Trunc, Tozero, Tozero Inverse, Otsu, Otsu Inverse, Adaptive Mean, Adaptive Gaussian, and Custom are shown on the x axis. Meanwhile, the y-axis shows the accuracy rate, going from 0 to 1. Each way of thresholding data is marked by a blue bar that allows you to assess how they perform. On the graph, it is obvious that some techniques are more accurate than others. The accuracy of the Custom thresholding method comes close to the maximum value, which shows that it has been optimized to do its best. The use of Otsu and Binary techniques proves they perform well for extracting text from files. Trunc and Tozero tend to miss their targets more, which means they do not seem capable of highly effective segmentation. Through bar graph in Figure 6, we find that picking the right thresholding technique matters a lot in OCR, because it affects the system's ability to read text. Because the custom thresholding approach is highly accurate, it suggests that users can make special adjustments for each type of image. Such understanding gives both researchers and developers the power to tailor threshold settings for various uses and boost text extraction performance.

ACKNOWLEDGMENT

I would like to thank all the people that helped bring this project to a conclusion. I owe a special thanks to my mentors, instructors, and peers for their valuable mentoring, direction, prompts, support, and cheerleading through the development process. I'd like to give a special thanks to the developers and contributors of Tesseract-OCR, OpenCV-Python, and Pillow that had been used to build this project as open source. I'm thankful to my family and friends that motivated me and supported me to reach this milestone.

CONCLUSION

OCR based text extraction system came out as a well-structured and an effective way in converting the images into text. Thanks to using OpenCV, Pillow, and Tesseract-OCR, the system is built on the multiple layers of input acquisition, preprocessing, OCR, and output. Elementary image processing methods like grayscale, noise level reduction, and binarization enlighten the textual content, and enhancing post processing like spell check and formatting clean the text material gathered. This project has practical use cases such as document scanning, form filling, and natural language processing – making it useful in industries like education, healthcare, and business processing. Improvements can be made in the future with the help of better models in OCR with the use of Artificial Intelligence, multilingual support can be added for a broader recognition capability and the processing can be done real-time in cloud. In general, this system seems to be effective, efficient and accurate in text extraction from images in a way that contributes towards the increases in automation and intelligent data analysis.

FUTURE SCOPE

There is much more that can be added on to this kind of project and applied to other fields of work., New possibilities of developing deep learning and other artificial intelligence-based approaches can help to increase the recognition accuracy for handwritten texts and difficult fonts. NLP integration alongside may enable realize additional useful knowledge from the text that has been identified. Developing for multiple languages and for real-time processing-based mechanisms can provide even more performance in sectors like automation, health care, and finance. Furthermore, utilizing cloud-based OCR services and incorporating mobile applications to offer text extraction can further the possibility and usefulness of the technology.

REFERENCES

[1] W. Xue, Q. Li, and Q. Xue, "Text Detection and Recognition for Images of Medical Laboratory Reports With a Deep Learning Approach," IEEE Access, vol. 8, pp. 407–416, 2020.
[2] Z. Tang, T. Miyazaki, and S. Omachi, "A Scene-Text Synthesis Engine Achieved Through Learning From Decomposed Real-World Data," IEEE Transactions on Image Processing, vol. 32, pp. 5837–5851, 2023.
[3] M. Mohammadi, M. Eftekhari, and A. Hassani, "Image-Text Connection: Exploring the Expansion of the Diversity Within Joint Feature Space Similarity Scores," IEEE Access, vol. 11, pp. 123209–123222, 2023.
[4] D. Vukadin, A. S. Kurdija, G. Delač and M. Šilić, "Information Extraction From Free-Form CV Documents in Multiple Languages," in IEEE Access, vol. 9, pp. 84559-84575, 2021.
[5] S. Lee, J. Lee, C. H. Bae, M.-S. Choi, R. Lee, and S. Ahn, "Optimizing Prompts Using In-Context Few-Shot Learning for Text-to-Image Generative Models," IEEE Access, vol. 12, pp. 2660–2673, 2024.
[6] B. Yadav, A. Indian, and G. Meena, "HDevChaRNet: A deep learning-based model for recognizing offline handwritten Devanagari characters," Journal of Autonomous Intelligence, vol. 6, no. 2, pp. 21–30, Aug. 2023.
[7] M. Ponnuru, S. Ponmalar, L. Amasala, B. Tanu Sree, and G. V. G. Chaitanya, "Image-Based Extraction of Prescription Information Using OCR-Tesseract," Procedia Computer Science, vol. 235, pp. 1077–1086, Jan. 2024.
[8] A. Y. Sugiyono, K. Adrio, K. Tanuwijaya, and K. M. Suryaningrum, "Extracting Information from Vehicle Registration Plate using OCR Tesseract," Procedia Computer Science, vol. 227, pp. 932–938, Jan. 2023.
[9] G. A. Robby, A. Tandra, I. Susanto, J. Harefa, and A. Chowanda, "Implementation of Optical Character Recognition using Tesseract with the Javanese Script Target in Android Application," Procedia Computer Science, vol. 157, pp. 499–505, 2019.
[10] W. Xue, Q. Li, and Q. Xue, "Text Detection and Recognition for Images of Medical Laboratory Reports With a Deep Learning Approach," IEEE Access, vol. 8, pp. 407–416, 2020.
[11] S. Lee, J. Lee, C. H. Bae, M.-S. Choi, R. Lee, and S. Ahn, "Optimizing prompts using in-context few-shot learning for text-to-image generative models," IEEE Access, vol. 12, pp. 2660–2673, 2024.
[12] D. Vukadin, A. S. Kurdija, G. Delač and M. Šilić, "Information Extraction From Free-Form CV Documents in Multiple Languages," in IEEE Access, vol. 9, pp. 84559-84575, 2021.
[13] S. K. Alhabeeb and A. A. Al-Shargabi, "Text-to-Image Synthesis With Generative Models: Methods, Datasets, Performance Metrics, Challenges, and Future Direction," in IEEE Access, vol. 12, pp. 24412-24427, 2024.
[14] M. Mohammadi, M. Eftekhari, and A. Hassani, "Image-text connection: Exploring the expansion of the diversity within joint feature space similarity scores," IEEE Access, vol. 11, pp. 123209–123222, 2023.
[15] T. Geng, "Transforming Scene Text Detection and Recognition: A Multi-Scale End-to-End Approach With Transformer Framework," in IEEE Access, vol. 12, pp. 40582-40596, 2024.

[16] Y. Wang, "Extraction Algorithm of English Text Information From Color Images Based on Radial Wavelet Transform," in IEEE Access, vol. 8, pp. 160050-160064, 2020..
[17] M. Sundaresan and S. Ranjini, "Text extraction from digital English comic image using two blobs extraction method," International Conference on Pattern Recognition, Informatics and Medical Engineering (PRIME-2012), Salem, India, 2012, pp. 449-452.
[18] Okechukwu Ogochukwu Patience, E. M. Amaechi, O. George, and O. N. Isaac, "Enhanced Text Recognition in Images Using Tesseract OCR within the Laravel Framework", Asian J. Res. Com. Sci., vol. 17, no. 9, pp. 58–69, Sep. 2024.
[19] S. Dome and A. P. Sathe, "Optical Charater Recognition using Tesseract and Classification," 2021 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2021, pp. 153-158.
[20] G. P. Kela, M. G. Daga and R. Khandelwal, "Vision to Voice: An Advanced Blind Assistance System Integrating YOLOv3 and OCR Technologies for Enhanced Mobility," 2025 3rd International Conference on Disruptive Technologies (ICDT), Greater Noida, India, 2025, pp. 1473-1476.
[21] Y. Watanabe, R. Togo, K. Maeda, T. Ogawa and M. Haseyama, "Text-Guided Image Manipulation via Generative Adversarial Network With Referring Image Segmentation-Based Guidance," in IEEE Access, vol. 11, pp. 42534-42545, 2023.
[22] S. Pande and M. S. R. Chetty, "Bezier Curve Based Medicinal Leaf Classification using Capsule Network", International Journal of Advanced Trends in Computer Science and Engineering, Vol. 8, No. 6, pp. 2735-2742, 2019.
[23] S. Nagendram, A. Singh, G. Harish Babu, R. Joshi, S. D. Pande, S. K. H. Ahammad, D. Dhabliya, and A. Bisht, "Stochastic gradient descent optimisation for convolutional neural network for medical image segmentation," Open Life Sciences, vol. 18, no. 1, 20220665, pp. 1-15, Aug. 2023.
[24] S. Pande and M. S. R. Chetty, "Linear Bezier Curve Geometrical Feature Descriptor for Image Recognition", Recent Advances in Computer Science and Communications, Vol. 13, No. 5, pp. 930-941, 2020.