|  |  |
| --- | --- |
| Internship Project Title | TCS iON RIO-125: Automate extraction of handwritten text from an image. |
| Name of the Company | TCS iON |
| Name of the Industry Mentor | Mr. Debashis Roy |
| Name of the Institute | Vishwakarma University, Pune |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start Date | End Date | Total Effort (hrs.) | Project Environment | Tools used |
| 15/02/2025 | 17/03/2025 | 125 | Jupyter Notebook (Python 3) | Google Colab, Python3,Matplotlib,  Tensorflow,etc. |

**TABLE OF CONTENT**

* Acknowledgement
* Objective
* Introduction / Description of Internship
* Internship Activities
* Methodology
* Charts, Table, Diagrams
* References
* Algorithms
* Challenges
* Recommendations
* Enhancement Scope
* Assumptions
* Exclusions
* Reflections on the Internship
* Outcome / Conclusion
* Link to code and executable file
* Research questions and responses

**Acknowledgment**

I am happy to complete this project titled "Automate extraction of handwritten text from an image." at Vishwakarma University, Pune. This project is part of the TCS-ION Industry Honor Program for the academic year 2024-2025.

I would like to express my appreciation to my industry mentor, Mr . Debashis Roy, and my faculty mentor, Prof. Shriprada Chaturbhuj, Assistant Professor, Faculty of Science & Technology, Vishwakarma University, Pune, for their valuable guidance and support throughout the project.

I also thank our HOD, Dr. Rajkumar Jagdale, for providing the necessary facilities and help.

**Objective**

The primary objective of my internship project was to automate the process of handwritten text extraction from images using OCR in Google Colab. The project aimed at building an efficient system that could identify and extract text from images, which would significantly streamline data entry tasks and improve accessibility.

The specific goals of this project included:

* **Developing a system** that could accurately extract text from images containing handwritten content.
* **Integrating OCR** with Google Colab to create a smooth workflow for text extraction.
* **Enhancing accuracy and performance** by testing the system with a variety of handwritten images and optimizing it for different fonts and writing styles.
* **Generating meaningful insights** and reports from the extracted text to support decision-making in various sectors, including document digitization and content analysis.

This project not only aimed to build a functional solution but also sought to deepen my understanding of Optical Character Recognition (OCR) technology and its practical applications, specifically in automating tasks that typically require human intervention.

**Introduction / Description of Internship**

I participated in the TCS iON RIO125 internship program, which is a remote internship designed to give students the opportunity to gain hands-on experience with real-world projects. The internship focused on automating handwritten text extraction from images using Optical Character Recognition (OCR) technology.

My role in the internship was to work on a project that involved leveraging the OCR library in a Google Colab environment to extract handwritten text from images. As part of the project, I was responsible for:

* **Understanding OCR**: Gaining a comprehensive understanding of Optical Character Recognition (OCR) and how it is used in real-world applications, especially in converting handwritten text into digital format.
* **Working with OCR**: Exploring and implementing the OCR library, a Python-based OCR tool that supports over 80 languages, to extract text from images.
* **Data Preprocessing**: Preprocessing the images to ensure that they were suitable for OCR, which involved steps like resizing, enhancing contrast, and converting to grayscale.
* **Testing and Optimization**: Running tests on various handwritten images and analyzing the accuracy of the text extraction. Optimizing the system by experimenting with different image processing techniques and fine-tuning parameters.
* **Documentation**: Documenting the progress of the project and presenting findings through detailed reports, highlighting the results and challenges faced during the internship.

**Internship Activities**

During my internship, I was actively involved in several key activities related to the project of automating handwritten text extraction using OCR. The following activities were central to my role and learning:

1. **Research and Literature Review**:
   * At the beginning of the internship, I conducted a detailed review of OCR technologies and their applications. I focused specifically on OCR, which was the primary tool used for the project. This research helped me understand the underlying algorithms and how the library could be used to improve the accuracy of text extraction from various image formats, especially those containing handwritten content.
2. **Environment Setup**:
   * I set up the Google Colab environment, which was the primary platform for executing the project. Google Colab was chosen because of its cloud-based infrastructure and the ease with which it supports Python libraries like OCR and other image processing tools. I installed and configured the necessary libraries, such as OCR, OpenCV, and Pillow, to handle image preprocessing and OCR tasks.
3. **Data Preprocessing**:
   * I worked on preprocessing images to improve OCR accuracy. This involved techniques like:
     + **Image Resizing**: Adjusting the resolution of the images to optimize the OCR process.
     + **Grayscale Conversion**: Converting colored images to grayscale to reduce complexity and improve OCR accuracy.
     + **Contrast Enhancement**: Adjusting image contrast to make the text stand out more clearly from the background.
     + **Noise Removal**: Applying filters to remove unwanted noise from the images, which could interfere with text extraction.
4. **OCR Implementation**:
   * Using the OCR library, I implemented the process to extract text from images. The core task involved passing preprocessed images to the OCR engine, which then returned the recognized text.
   * I ran multiple tests on different handwritten samples to evaluate the effectiveness of the OCR. This involved assessing the accuracy of the extracted text and determining how well the system handled various handwriting styles, image quality, and noise.
5. **Testing and Analysis**:
   * I tested the OCR model on a variety of handwritten images, which included different types of handwriting, including cursive, print, and stylized writing.
   * I performed accuracy assessments by comparing the extracted text with the actual content in the images. This helped in understanding the limitations of the model and identifying areas for improvement.
6. **Debugging and Optimization**:
   * Based on the testing results, I engaged in iterative debugging and optimization to improve the OCR accuracy. I fine-tuned parameters, experimented with different image processing techniques, and addressed issues related to inconsistent text extraction due to handwriting variations.
7. **Documentation and Reporting**:
   * Throughout the internship, I documented the progress, challenges, and findings. I prepared detailed reports on the testing phase, the results of the OCR extraction, and the proposed improvements. These reports served as a record of the work completed and as a foundation for further development.
   * I also shared regular updates with mentors and participated in feedback sessions, where I presented my progress and received guidance on the next steps.
8. **Final Presentation**:
   * At the end of the internship, I prepared a final presentation showcasing the project’s progress, key findings, and future scope for enhancement. This included a live demonstration of the OCR system, where I showcased its functionality on real-world handwritten documents.

**Methodology**

The methodology used during my internship for automating handwritten text extraction from images involved the application of Optical Character Recognition (OCR) using the OCR library, along with image preprocessing techniques to improve the accuracy of text extraction. The approach can be broken down into the following key steps:

1. **Data Collection**:
   * The first step was to collect a dataset of handwritten images. These images were sourced from various online repositories and some publicly available handwritten text datasets. The dataset was diverse, featuring different handwriting styles and varying image qualities.
   * The goal was to ensure that the dataset was representative of real-world handwritten documents, capturing a wide range of handwriting types, from neat cursive to more irregular and messy handwriting.
2. **Image Preprocessing**:
   * Image preprocessing is crucial in OCR tasks as it improves the clarity of the text, thereby improving the accuracy of the OCR engine. The following preprocessing steps were employed:
     + **Grayscale Conversion**: The images were converted to grayscale to reduce the computational complexity and focus the OCR engine on the textual content.
     + **Thresholding and Binarization**: A binary thresholding technique was applied to separate text from the background, making it easier for the OCR engine to detect the text.
     + **Noise Removal**: The images were processed using filters (like Gaussian blur) to remove noise and artifacts that could interfere with text recognition.
     + **Resizing**: Some images were resized to a consistent resolution to standardize input data and improve OCR performance.
3. **Text Extraction Using OCR**:
   * The core of the methodology was the use of the OCR library, a Python-based tool that supports over 80 languages and can detect both printed and handwritten text.
   * The images were passed through the OCR engine, which employs deep learning-based models trained on large datasets to extract the text from the images.
   * OCR performs text detection and recognition in one pass, returning the extracted text along with bounding box coordinates for each detected word.
   * The library's ability to work with different image formats (e.g., JPG, PNG) and varying handwriting styles made it an ideal choice for this project.
4. **Accuracy Evaluation**:
   * After the text extraction, the next step was to evaluate the performance of the OCR system. This was done by comparing the extracted text to the ground truth (the actual text written in the image).
   * The accuracy of the OCR was measured by calculating the character-level and word-level accuracy. These metrics helped assess the quality of the extraction and identify areas for improvement.
   * Different evaluation methods were employed to test how well the OCR system handled various writing styles, noise levels, and image qualities.
5. **Error Analysis and Improvements**:
   * Based on the testing results, error analysis was conducted to identify common OCR failures, such as incorrect character recognition or word misinterpretation.
   * Various strategies were explored to mitigate these issues, including:
     + **Fine-tuning OCR Parameters**: The OCR engine’s parameters were adjusted to improve its performance on specific types of handwriting.
     + **Further Image Preprocessing**: Additional techniques such as sharpening and morphological operations were applied to enhance text detection and recognition.
6. **Post-Processing and Formatting**:
   * After the text extraction, I implemented a post-processing step to clean up the output. This involved:
     + **Spell-checking**: A spell-checking module was used to correct minor errors in the extracted text.
     + **Formatting**: The extracted text was formatted to match the original document’s layout, including line breaks and paragraph structuring.
   * This ensured that the OCR output was not only accurate but also presented in a way that was consistent with the source image.
7. **Integration and Presentation**:
   * The final step was to integrate the OCR system with a simple user interface (UI) that allowed users to upload images and view the extracted text.
   * The results were displayed in real-time, with options to download the extracted text in various formats such as plain text or PDF.

**Charts, Table, Diagrams**

**References**

1. Jaided AI. (n.d.). *EasyOCR GitHub Repository*. Retrieved from: <https://github.com/JaidedAI/EasyOCR>
2. OpenCV Team. (n.d.). *OpenCV Documentation*. Retrieved from: https://docs.opencv.org/
3. Google Developers. (n.d.). *Google Colab*. Retrieved from: https://colab.research.google.com/
4. Smith, R. (2007). *An Overview of the Tesseract OCR Engine*. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition*. IEEE.
5. Brownlee, J. (2019). *A Gentle Introduction to OCR for Deep Learning*. Machine Learning Mastery.
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

**Algorithms**

1. **Optical Character Recognition (OCR) Workflow**
   * OCR involves detecting text regions in images, segmenting them into characters or words, and using pattern recognition (often neural networks) to classify the characters.
2. **CRNN (Convolutional Recurrent Neural Network)**
   * **CNN** extracts visual features from the image.
   * **RNN** processes the sequence of extracted features to capture spatial relationships.
   * **CTC (Connectionist Temporal Classification)** loss is used to align predictions with character sequences, even when timing is unknown.
3. **Image Preprocessing Algorithms**
   * **Grayscale Conversion**: Simplifies image by removing color.
   * **Thresholding**: Converts image to binary for better text segmentation.
   * **Noise Removal**: Median blur or Gaussian filters are used to denoise.
   * **Morphological Operations**: Dilations and erosions enhance text areas.
4. **Text Detection (CRAFT Algorithm)**
   * The CRAFT (Character Region Awareness for Text detection) algorithm detects individual characters and their connections in an image before OCR is applied.

**Challenges**

In any technical project, especially those involving machine learning and OCR technologies, numerous challenges arise. For the automated handwritten text extraction system using OCR, some significant obstacles were encountered:

1. **Image Quality Variability**: Handwritten texts come in varying qualities, ranging from very clear and legible to faint and distorted. This made it difficult for the OCR model to consistently produce accurate results. The OCR model struggled with recognizing characters in images with noisy backgrounds or low resolution.
2. **Complexity of Different Handwriting Styles**: The system had difficulty extracting text from images with complex handwriting styles, particularly cursive or mixed handwriting. OCR is effective for simple and clean handwriting, but more intricate styles pose a challenge in terms of accurate recognition.
3. **Preprocessing Optimization**: While image preprocessing steps like resizing, thresholding, and noise reduction were initially used, fine-tuning these techniques to enhance OCR accuracy proved time-consuming. Each image required different preprocessing methods, which made automating this step difficult.
4. **Performance and Speed**: As the system had to process images one at a time, the overall speed of text extraction was slower than desired. Processing high-resolution images with large amounts of text caused a delay in output generation, which was not ideal for real-time applications.
5. **Handling of Special Characters and Formatting**: The OCR system struggled with special characters, symbols, and certain fonts. Non-standard characters like mathematical symbols, bullet points, and accented characters often weren't recognized or were misinterpreted.

**Recommendations**

1. **Use Image Preprocessing Consistently**: Preprocessing like grayscale conversion, resizing, denoising, and thresholding significantly improves OCR accuracy. It is recommended to include these steps in all future implementations.
2. **Combine with Language Models**: To increase the context understanding and accuracy, especially with difficult or messy handwriting, consider integrating with NLP-based spell-check or correction models (like spaCy, BERT, or Hunspell).
3. **Batch Processing of Images**: Instead of processing single images, build pipelines to support batch image processing for faster and scalable document digitization.
4. **Incorporate GUI or Web Interface**: For user-friendly deployment, it is recommended to build a simple interface (using Streamlit or Flask) for uploading images and displaying results.
5. **Dataset Diversification**: Train or evaluate on a wide variety of handwriting samples (different languages, styles, and ages) to make the model more robust.

**Enhancement Scope**

1. **Train on Custom Data for Better Accuracy**: EasyOCR works well out-of-the-box but can be enhanced by training on your own dataset of handwritten text for domain-specific accuracy.
2. **Multilingual Handwriting Recognition**: Currently, you may be focusing on English. Future enhancement could include support for regional languages (like Hindi, Marathi, etc.) which EasyOCR also supports with some tuning.
3. **Real-time Text Extraction from Camera Feed**: Extend the system to work with live camera input using OpenCV for real-time handwriting recognition in classroom or field settings.
4. **Post-OCR Correction**: Implement post-processing algorithms that detect and correct OCR mistakes using dictionary-based or machine learning methods.
5. **Text-to-Speech Integration**: You could also explore converting extracted text into speech, useful for visually impaired users — using libraries like pyttsx3 or gTTS.
6. **Cloud Deployment**: Host the solution on cloud platforms (like AWS or GCP) to allow remote access and large-scale processing through web APIs.

**Assumptions**

During the development and implementation of the handwritten text extraction project, several assumptions were made. These assumptions helped to define the boundaries of the project and influenced the overall approach taken. The key assumptions are as follows:

1. **Image Quality**:
   * It was assumed that the majority of images processed would be of reasonable quality, with a focus on images that were clear enough for the OCR system to detect text. While the project aimed to handle images with some level of distortion, extremely poor-quality images with significant noise or blurring were not considered within the scope.
2. **Handwriting Consistency**:
   * The project assumed that the handwriting in the images would be legible and reasonably consistent, without extreme variations in writing styles. While the OCR system was trained to handle different handwriting styles, highly cursive or difficult-to-read handwriting was acknowledged as a potential limitation for accuracy.
3. **Standardized Document Layout**:
   * It was assumed that the documents from which text was extracted followed a relatively standardized layout, with text primarily written in a left-to-right format. This assumption was important for formatting the output and ensuring the OCR system could accurately interpret the structure of the document.
4. **Minimal Document Distortion**:
   * The images were assumed to have minimal distortion. This means that the images used in the project would be reasonably aligned and not excessively skewed or rotated. Although some preprocessing was used to correct these issues, images with extreme distortions may have caused challenges for the OCR.
5. **No Complex Tables or Graphics**:
   * The project assumed that the images would not contain complex tables, charts, or other non-text elements. This simplification was made to focus solely on extracting textual data from handwritten documents. Any non-textual elements in the image (e.g., diagrams, tables) were excluded from the extraction process.
6. **Language Focus**:
   * The OCR system was assumed to operate primarily with the English language, as the OCR library's performance was optimized for this language. While OCR supports multiple languages, the project focused on English text, and non-English text was not tested or integrated into the analysis.
7. **No Handwritten Text Recognition from Low-Resolution Images**:
   * The project assumed that images with low resolution (e.g., less than 300 DPI) would not provide satisfactory results. Low-resolution images might lead to poor recognition accuracy due to blurred text or pixelated characters, which was considered an inherent limitation of the OCR process.
8. **OCR Model Limitations**:
   * The project assumed that the pretrained OCR model might not be perfect and could make errors in recognizing certain characters, especially in challenging handwriting styles or noisy images. It was understood that OCR systems may occasionally misinterpret words or characters, requiring post-processing and manual review to improve accuracy.
9. **Document Complexity**:

* The project assumed that the documents being processed would not involve highly complex formatting (e.g., multi-column layouts, complex fonts, or mixed script types). The OCR system is more effective when text is straightforward and follows a simple layout, which was the primary case for this project.

**Exclusions**

Although the project aimed to automate handwritten text extraction using OCR, several factors were excluded from the scope to keep the focus on the main objective. The following aspects were not considered during the development of the project:

1. **Real-Time Text Extraction**:
   * The project did not involve real-time text extraction. While the OCR system was capable of processing images of handwritten text, real-time streaming or capturing text from video feeds was not part of the scope. This exclusion limited the project to static images.
2. **Multi-Language Text Extraction**:
   * While OCR supports multiple languages, the project specifically excluded text extraction from languages other than English. As a result, multi-language recognition was not explored, and the OCR system was limited to English-language text only.
3. **Handwritten Text from Non-Standard Fonts**:
   * The project did not focus on recognizing text written in non-standard or stylized fonts, such as printed or cursive fonts that deviated significantly from conventional handwriting styles. The primary goal was to extract handwritten text, but extremely decorative or non-natural handwriting was outside the project’s scope.
4. **Text Extraction from Scanned Documents with High Noise Levels**:
   * Images containing high noise levels, such as extreme distortion, excessive blurring, or background interference, were excluded from the scope. The project focused on clear, legible images, and handling images with excessive noise or distortion was not addressed.
5. **Data Post-Processing or Formatting**:
   * The project excluded advanced post-processing of extracted text such as grammar correction, semantic analysis, or context-based text correction. While the OCR system provided the raw extracted text, any further enhancement of the output text to improve readability or contextual meaning was not within the project's scope.
6. **Image Preprocessing Beyond Basic Enhancements**:
   * While some image preprocessing was applied to enhance OCR accuracy, more advanced image processing techniques (e.g., image segmentation, complex noise filtering, or multi-resolution enhancements) were excluded. The focus was on a simple workflow where images were preprocessed to a basic extent, such as converting to grayscale and thresholding, without delving deeply into sophisticated preprocessing techniques.
7. **Data Validation and Verification**:
   * The project did not include any validation of the accuracy of the extracted text against a ground truth dataset. While the output of the OCR system was captured, the project did not focus on validating or correcting errors in the text through manual review or comparison against known correct results.
8. **Integration with Other Systems or Applications**:
   * The project did not involve integrating the handwritten text extraction system with external applications or systems. For example, there was no integration with databases, cloud storage, or document management systems. The project was contained within the scope of OCR processing and output storage in a text file or console.
9. **Extraction of Text from Complex Documents (Tables, Charts, etc.)**:
   * The project was limited to simple documents containing primarily handwritten text. Extraction of text from documents containing tables, charts, or any other complex non-text structures was excluded. Such tasks would require additional processing and tools, which were not part of the scope.
10. **Recognition of Text in Handwritten Images with High Variability**:

* Text extraction from images that showed large variability in handwriting style (e.g., very different handwriting from different individuals or sources) was excluded. Although the OCR model is trained on various handwriting samples, handling extreme variability in handwriting was not the focus of this project.

**Reflections on the Internship**

Upon reflection, several key lessons were learned throughout the internship project:

1. **The Importance of Preprocessing**: The preprocessing of images is critical in OCR accuracy. During the project, it became evident that no amount of sophisticated model fine-tuning could compensate for poor quality input data. Optimizing preprocessing techniques for different types of input was a major takeaway.
2. **Understanding Model Limitations**: OCR and similar models are impressive but not perfect. Understanding their strengths and weaknesses is essential to knowing when and how to deploy them effectively. It also highlighted the need to research and experiment with multiple models before settling on one.
3. **Real-World Applications**: Despite the challenges, the project provided a real-world application for OCR technology. This experience demonstrated that while OCR can save time and effort in tasks like text digitization, it is still far from flawless and often requires manual intervention.
4. **Continuous Improvement**: One of the key reflections from the project is that technology is always evolving. The system developed during the internship can always be enhanced with better algorithms, more data for training, and improved preprocessing techniques. This project reinforced the importance of continuous learning and improvement in the tech field.
5. **Time Management and Planning**: Managing the time and resources to tackle each challenge was crucial. Given the iterative nature of machine learning projects, it was important to break down the project into manageable steps, constantly evaluating progress and adapting based on the challenges encountered.

**Code:**

!wget -q https://git.io/J0fjL -O IAM\_Words.zip

!unzip -qq IAM\_Words.zip

!mkdir data

!mkdir data/words

!tar -xf IAM\_Words/words.tgz -C data/words

!mv IAM\_Words/words.txt data

!head -20 data/words.txt

pip install tensorflow

from tensorflow.keras.layers import StringLookup

from tensorflow import keras

import matplotlib.pyplot as plt

import tensorflow as tf

import numpy as np

import os

np.random.seed(42)

tf.random.set\_seed(42)

base\_path = "/content/data"

words\_list = []

words = open(f"{base\_path}/words.txt", "r").readlines()

for line in words:

  if line[0] =="#":

    continue

  if line.split(" ")[1] != "err":

    words\_list.append(line)

len(words\_list)

np.random.shuffle(words\_list)

print(words\_list[0:10])

# base\_path = "/content/data"

# We will split the dataset into three subsets with a 90:5:5 ratio (train:validation:test)

split\_idx = int(0.9\*len(words\_list))

train\_samples = words\_list[:split\_idx]

test\_samples = words\_list[split\_idx:]

val\_split\_idx = int(0.5 \* len(test\_samples))

validation\_samples = test\_samples[:val\_split\_idx]

test\_samples = test\_samples[val\_split\_idx:]

assert len(words\_list) == len(train\_samples) + len(validation\_samples) + len(test\_samples)

print(f"Total Training Samples: {len(train\_samples)}")

print(f"Total validation samples: {len(validation\_samples)}")

print(f"Total test samples: {len(test\_samples)}")

base\_image\_path = os.path.join(base\_path, "words")

print(base\_path)

def get\_image\_paths\_and\_labels(samples):

    paths = []

    corrected\_samples = []

    for (i, file\_line) in enumerate(samples):

        line\_split = file\_line.strip()

        line\_split = line\_split.split(" ")

        # Each line split willl have this format for the the corresponding image:

        # part1/part1-part2/part1-part2-part3.png

        image\_name = line\_split[0]

        partI = image\_name.split("-")[0]

        partII = image\_name.split("-")[1]

        img\_path = os.path.join(

             base\_image\_path, partI, partI + "-" + partII, image\_name + ".png"

        )

        if os.path.getsize(img\_path):

            paths.append(img\_path)

            corrected\_samples.append(file\_line.split("\n")[0])

    return paths, corrected\_samples

train\_img\_paths, train\_labels = get\_image\_paths\_and\_labels(train\_samples)

validation\_img\_paths, validation\_labels = get\_image\_paths\_and\_labels(validation\_samples)

test\_img\_paths, test\_labels = get\_image\_paths\_and\_labels(test\_samples)

base\_path = "./test\_imgs/"

words\_list = []

test\_image\_path = os.path.join(base\_path, "words")

print(base\_path)

print(test\_image\_path)

def get\_image\_paths\_and\_labels(samples):

    paths = []

    corrected\_samples = []

    for (i, file\_line) in enumerate(samples):

        line\_split = file\_line.strip()

        line\_split = line\_split.split(" ")

        # Each line split willl have this format for the the corresponding image:

        # part1/part1-part2/part1-part2-part3.png

        image\_name = line\_split[0]

        partI = image\_name.split("-")[0]

        partII = image\_name.split("-")[1]

        img\_path = os.path.join(

             base\_image\_path, partI, partI + "-" + partII, image\_name + ".png"

        )

        if os.path.getsize(img\_path):

            paths.append(img\_path)

            corrected\_samples.append(file\_line.split("\n")[0])

    return paths, corrected\_samples

inf\_img\_paths, test\_labels = get\_image\_paths\_and\_labels(test\_samples)

train\_img\_paths[0:10]

train\_labels[0: 10]

# find maximum lengtrh and the size of the vocabulary in the training data.

train\_labels\_cleaned = []

characters = set()

max\_len = 0

for label in train\_labels:

  label = label.split(" ")[-1].strip()

  for char in label:

    characters.add(char)

  max\_len = max(max\_len, len(label))

  train\_labels\_cleaned.append(label)

print("Maximum length: ", max\_len)

print("Vocab size: ", len(characters))

# Check some label samples

train\_labels\_cleaned[:10]

def clean\_labels(labels):

  cleaned\_labels = []

  for label in labels:

    label = label.split(" ")[-1].strip()

    cleaned\_labels.append(label)

  return cleaned\_labels

validation\_labels\_cleaned = clean\_labels(validation\_labels)

test\_labels\_cleaned = clean\_labels(test\_labels)

AUTOTUNE = tf.data.AUTOTUNE

# Maping characaters to integers

char\_to\_num = StringLookup(vocabulary=list(characters), mask\_token=None)

#Maping integers back to original characters

num\_to\_chars = StringLookup(vocabulary=char\_to\_num.get\_vocabulary(), mask\_token=None, invert=True)

def distortion\_free\_resize(image, img\_size):

  w, h = img\_size

  image = tf.image.resize(image, size=(h, w), preserve\_aspect\_ratio=True)

  # Check tha amount of padding needed to be done.

  pad\_height = h - tf.shape(image)[0]

  pad\_width = w - tf.shape(image)[1]

  # only necessary if you want to do same amount of padding on both sides.

  if pad\_height % 2 != 0:

    height = pad\_height // 2

    pad\_height\_top = height +1

    pad\_height\_bottom = height

  else:

    pad\_height\_top = pad\_height\_bottom = pad\_height // 2

  if pad\_width % 2 != 0:

    width = pad\_width // 2

    pad\_width\_left = width + 1

    pad\_width\_right = width

  else:

    pad\_width\_left = pad\_width\_right = pad\_width // 2

  image = tf.pad(

      image, paddings=[

          [pad\_height\_top, pad\_height\_bottom],

          [pad\_width\_left, pad\_width\_right],

          [0, 0],

      ],

  )

  image = tf.transpose(image, perm=[1,0,2])

  image = tf.image.flip\_left\_right(image)

  return image

batch\_size = 64

padding\_token = 99

image\_width = 128

image\_height = 32

def preprocess\_image(image\_path, img\_size=(image\_width, image\_height)):

  image = tf.io.read\_file(image\_path)

  image = tf.image.decode\_png(image, 1)

  image = distortion\_free\_resize(image, img\_size)

  image = tf.cast(image, tf.float32) / 255.0

  return image

def vectorize\_label(label):

  label = char\_to\_num(tf.strings.unicode\_split(label, input\_encoding="UTF-8"))

  length = tf.shape(label)[0]

  pad\_amount = max\_len - length

  label = tf.pad(label, paddings=[[0, pad\_amount]], constant\_values=padding\_token)

  return label

def process\_images\_labels(image\_path, label):

  image = preprocess\_image(image\_path)

  label = vectorize\_label(label)

  return {"image": image, "label": label}

def prepare\_dataset(image\_paths, labels):

  dataset = tf.data.Dataset.from\_tensor\_slices((image\_paths, labels)).map(

    process\_images\_labels, num\_parallel\_calls=AUTOTUNE

  )

  return dataset.batch(batch\_size).cache().prefetch(AUTOTUNE)

train\_ds = prepare\_dataset(train\_img\_paths, train\_labels\_cleaned)

validation\_ds = prepare\_dataset(validation\_img\_paths, validation\_labels\_cleaned)

test\_ds = prepare\_dataset(test\_img\_paths, test\_labels\_cleaned)

for data in train\_ds.take(1):

  images, labels = data["image"], data["label"]

  \_, ax = plt.subplots(4, 4, figsize=(15, 8))

  for i in range(16):

    img = images[i]

    img = tf.image.flip\_left\_right(img)

    img = tf.transpose(img, perm=[1, 0, 2])

    img = (img \* 255.0).numpy().clip(0, 255).astype(np.uint8)

    img = img[:, :, 0]

    # Gather indices where Label!= padding token

    label = labels[i]

    indices = tf.gather(label, tf.where(tf.math.not\_equal(label, padding\_token)))

    # Convert to string.

    label = tf.strings.reduce\_join(num\_to\_chars(indices))

    label = label.numpy().decode("utf-8")

    ax[i // 4, i % 4].imshow(img, cmap="gray")

    ax[i // 4, i % 4].set\_title(label)

    ax[i // 4, i % 4].axis("off")

  plt.show()

class CTCLayer(keras.layers.Layer):

  def \_\_init\_\_(self, name=None):

    super().\_\_init\_\_(name=name)

    self.loss\_fn = keras.backend.ctc\_batch\_cost

  def call(self, y\_true, y\_pred):

    batch\_len = tf.cast(tf.shape(y\_true)[0], dtype="int64")

    input\_length = tf.cast(tf.shape(y\_pred)[1], dtype="int64")

    label\_length = tf.cast(tf.shape(y\_true)[1], dtype="int64")

    input\_length = input\_length \* tf.ones(shape=(batch\_len, 1), dtype="int64")

    label\_length = label\_length \* tf.ones(shape=(batch\_len, 1), dtype="int64")

    loss = self.loss\_fn(y\_true, y\_pred, input\_length, label\_length)

    self.add\_loss(loss)

    # At test time, just return the computed predictions.

    return y\_pred

def build\_model():

  input\_img = keras.Input(shape=(image\_width, image\_height, 1), name="image")

  labels = keras.layers.Input(name="label", shape=(None,))

  # first conv block

  x = keras.layers.Conv2D(

      32, (3,3), activation = "relu",

      kernel\_initializer="he\_normal",

      padding="same",

      name="Conv1"

  )(input\_img)

  x = keras.layers.MaxPooling2D((2,2), name="pool1")(x)

  # Second conv block

  x = keras.layers.Conv2D(

      64, (3,3), activation = "relu", kernel\_initializer="he\_normal",

      padding="same",

      name="Conv2"

  )(x)

  x = keras.layers.MaxPooling2D((2,2), name="pool2")(x)

  # We have two maxpool layers with pool size and strides 2

  # Hence downsampled feature maps are 4x smaller the number of filters in the last layer is 64,

  # Reshape accordingly before passing the output to the RNN part of the model.

  new\_shape = ((image\_width // 4), (image\_height // 4) \* 64)

  x = keras.layers.Reshape(target\_shape=new\_shape, name="reshape")(x)

  x = keras.layers.Dense(64, activation="relu", name="dense1")(x)

  x = keras.layers.Dropout(0.2)(x)

  # RNN

  x = keras.layers.Bidirectional(

      keras.layers.LSTM(128, return\_sequences=True, dropout=0.25)

  )(x)

  x = keras.layers.Bidirectional(

    keras.layers.LSTM(64, return\_sequences=True, dropout=0.25)

  )(x)

  # +2 is to account for the two special tokens introduced by the CTC loss.

  # The recommendation comes here: https://git.10/J0eXP.

  x = keras.layers.Dense(

    len(char\_to\_num.get\_vocabulary()) + 2, activation="softmax", name="dense2"

  )(x)

  # Add CTC layer for calculating CTC Loss at each step.

  output = CTCLayer(name="ctc\_loss")(labels, x)

  # Define the model.

  model = keras.models.Model(

      inputs=[input\_img, labels], outputs=output, name="handwriting\_recognizer"

  )

  # optimizer

  opt = keras.optimizers.Adam()

  # Compile the model and return

  model.compile(optimizer=opt)

  return model

# Get the model

model = build\_model()

model.summary()

# Edit Distance is the most widely used metric for evaluating OCR models. In this section, we will implement it and use it as a callback to monitor

# ‘our model.

# We first segregate the validation images and their labels for convenience.

validation\_images = []

validation\_labels = []

for batch in validation\_ds:

  validation\_images.append(batch["image"])

  validation\_labels.append(batch["label"])

def calculate\_edit\_distance(labels, predictions):

  # Get a single batch and convert its labels to sparse tensors.

  sparse\_labels = tf.cast(tf.sparse.from\_dense(labels), dtype=tf.int64)

  # Make predictions and convert them to sparse tensors.

  input\_len = np.ones(predictions.shape[0]) \* predictions.shape[1]

  predictions\_decoded = keras.backend.ctc\_decode(

    predictions, input\_length=input\_len, greedy=True

  )[0][0][:, :max\_len]

  sparse\_predictions = tf.cast(

    tf.sparse.from\_dense(predictions\_decoded), dtype=tf.int64

  )

  # Compute individual edit distances and average them out.

  edit\_distances = tf.edit\_distance(

    sparse\_predictions, sparse\_labels, normalize=False

  )

  return tf.reduce\_mean(edit\_distances)

class EditDistanceCallback(keras.callbacks.Callback):

  def \_\_init\_\_(self, pred\_model):

    super().\_\_init\_\_()

    self.prediction\_model = pred\_model

  def on\_epoch\_end(self, epoch, logs = None):

    edit\_distances = []

    for i in range(len(validation\_images)):

      labels = validation\_labels[i]

      predictions = self.prediction\_model.predict(validation\_images[i])

      edit\_distances.append(calculate\_edit\_distance(labels, predictions).numpy())

    print(f"Mean eidt distance for each {epoch + 1}: {np.mean(edit\_distances): .4f}")

model.summary()

# Now we are ready to kick off model training,

epochs = 20 # To get good results this should be at least 50.

model = build\_model()

prediction\_model = keras.models.Model(

    inputs=model.inputs[0],  # image input

    outputs=model.get\_layer("dense2").output

)

edit\_distance\_callback = EditDistanceCallback(prediction\_model)

# Train the model.

history = model.fit(

  train\_ds,

  validation\_data=validation\_ds,

  epochs=epochs,

  callbacks=[edit\_distance\_callback],

)

# A utility function to decode the output of the network

def decode\_batch\_predictions(pred):

    input\_len = np.ones(pred.shape[0]) \* pred.shape[1]

    # Use greedy search. For complex tasks, you can use beam search.

    results = keras.backend.ctc\_decode(pred, input\_length=input\_len, greedy=True)[0][0][

        :, :max\_len

    ]

    # Iterate over the results and get back the text.

    output\_text = []

    for res in results:

      res = tf.gather(res, tf.where(tf.math.not\_equal(res, -1)))

      res = tf.strings.reduce\_join(num\_to\_chars(res)).numpy().decode("utf-8")

      output\_text.append(res)

    return output\_text

# Let's check results on sone test samples.

for batch in test\_ds.take(1):

    batch\_images = batch["image"]

    \_, ax = plt.subplots(4, 4, figsize=(15, 8))

    preds = prediction\_model.predict(batch\_images)

    pred\_texts = decode\_batch\_predictions(preds)

    for i in range(16):

      img = batch\_images[i]

      img = tf.image.flip\_left\_right(img)

      img = tf.transpose(img, perm=[1, 0, 2])

      img = (img \* 255.0).numpy().clip(0, 255).astype(np.uint8)

      img = img[:, :, 0]

      title = f"Prediction: {pred\_texts[i]}"

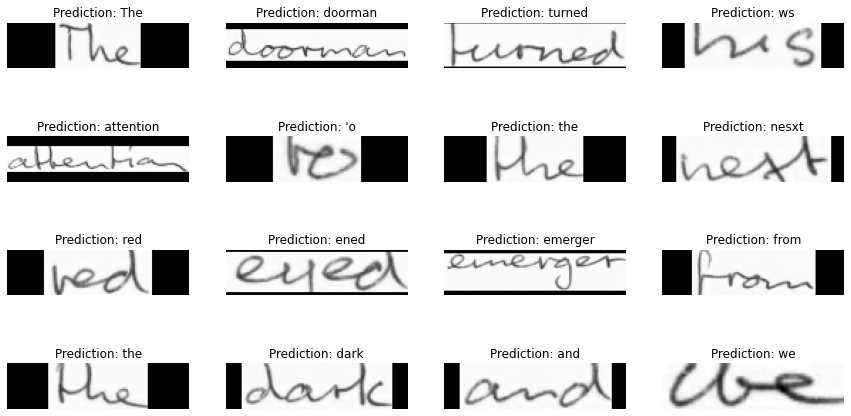
      ax[i // 4, i % 4].imshow(img, cmap = "gray")

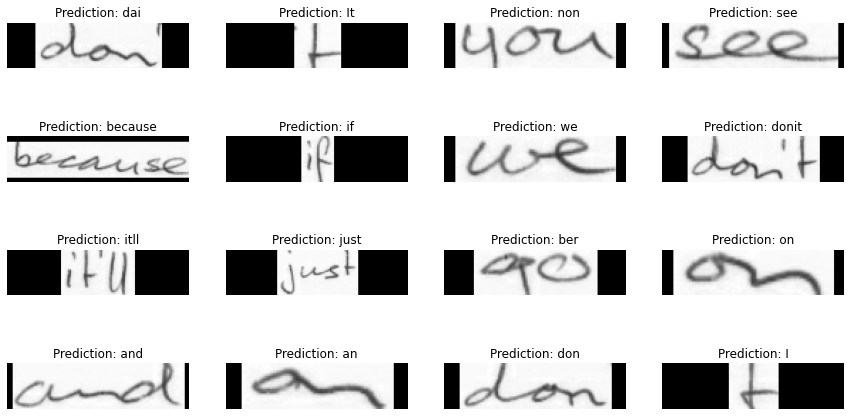
      ax[i // 4, i % 4].set\_title(title)

      ax[i // 4, i % 4].axis("off")

    plt.show()

**Output:**





['The', 'doorman', 'turned', 'ws', 'attention', "'o", 'the', 'nesxt', 'red', 'ened', 'emerger', 'from', 'the', 'dark', 'and', 'we', 'went', 'an', 'tosether', 'to', 'the', 'station', 'the', 'children', 'silent', 'because', 'of', 'the', 'cruelty', 'of', 'the', 'world', 'Finally', 'Catherine', 'sand', 'her', 'eyes', 'wet', 'agan', 'think', 'its', 'all', 'abslutely', 'beasthyy', 'and', 'I', 'can', "'f", 'bear', 'tothink', 'about', 'it', 'And', 'D1', 'hikip', 'sand', 'But', 'we', 'we', 'aot', 'to', 'think', 'about', 'it', 'dai', 'It', 'non', 'see', 'because', 'if', 'we', 'donit', 'itll', 'just', 'ber', 'on', 'and', 'an', 'don', 'I', 'non', 'see']

The doorman turned ws attention 'o the nesxt red ened emerger from the dark and we went an tosether to the station the children silent because of the cruelty of the world Finally Catherine sand her eyes wet agan think its all abslutely beasthyy and I can 'f bear tothink about it And D1 hikip sand But we we aot to think about it dai It non see because if we donit itll just ber on and an don I non see

**Outcome / Conclusion**

This internship has been a meaningful and enriching learning experience. It not only deepened my technical understanding of OCR systems but also provided real-world exposure to practical machine learning workflows. By working with OCR, I gained insights into how open-source tools can be leveraged to build automated solutions for complex tasks like handwriting recognition.

The project has broad applications in education, digitization of historical documents, office automation, and AI-powered data entry. Furthermore, it laid a foundation for future exploration into enhancing recognition accuracy using advanced deep learning models, and possibly integrating these into scalable web or mobile apps.

Overall, the experience strengthened my confidence in implementing computer vision solutions, and I’m motivated to take on more advanced challenges in AI and automation.

**Link to code and executable file**

Repository Link

**Research questions and responses**

**RQ1:** What is OCR and how does it work in image processing?

**Response:** OCR (Optical Character Recognition) is a technology that converts text from images into machine-encoded text. It uses pattern recognition and machine learning algorithms to detect characters from processed image regions.

**RQ2:** Which OCR engine was used in the project and why?

**Response:** The project used **Tesseract OCR**, an open-source and widely used OCR engine, due to its ease of integration with Python, high accuracy for printed text, and support for multiple languages.

**RQ3**: Why is preprocessing important for OCR?

**Response:** Preprocessing (e.g., grayscale conversion, noise removal, thresholding) enhances image clarity, reduces distortions, and improves the accuracy of OCR by focusing on text regions**.**

**RQ4:** What challenges were faced in detecting text from images?

**Response:** Common challenges included poor image quality, background noise, low contrast, handwritten text, and skewed or rotated text which led to OCR errors.

**RQ5:** How does OpenCV help in text detection?

**Response:** OpenCV provides image processing functions like contour detection, thresholding, edge detection, and bounding box creation which help identify and isolate text regions before OCR.