

TCS iON RIO-125: Automate extraction of handwritten text from an image

## Interim Project Report

Submitted to

**Department of Computer Science**

**Faculty of Science & Technology**

**Under TCS ION INDUSTRY HONOUR PROGRAM**

**Vishwakarma University, Pune (Maharashtra)**

By

**Varad Anil Jadhav**

**Under the supervision of**

|  |  |
| --- | --- |
| **Industry Mentor**  **Mr Niteen Gokhale**  **Tata consultancy services** | **Faculty Mentor Prof Shriprada Chaturbhuj**  **Vishwakarma university, pune** |

**CERTIFICATE**

This is to certify that the project of titled “Automate extraction of handwritten text from an image**”** submitted by Varad Jadhav is an original work and has not been previously submitted in part or full for the award of any degree or diploma to this or any other university. The project is submitted to **Vishwakarma University,Pune and TCS-ION Industry Honor Program,** in partial fulfillment of the requirement for the award of the degree of  **Bachelor of Computer Science.**

Date: 15/02/25

**Prof Shriprada Chaturbhuj**

**Faculty Mentor**

**Dr. Rajkumar Jagdale**

**Head of Department**

**(Computer Science)**

**DECLARATION**

I, Varad Jadhav Here by declare that the work embodied in this project entitled “Automate extraction of handwritten text from an image” carried out under the supervision of Industry mentor Mr. Niteen Gokhale & Faculty mentor Prof. Shriprada Chaturbhuj Assistant Professor, Faculty of Science & Technology, Vishwakarma University, Pune is an original work and does not contain any work submitted for the award of any degree in this university or any other university.

**Varad Jadhav**

B.Sc. Computer Science

Department Of Computer Science

Vishwakarma University,

Pune

|  |  |
| --- | --- |
| Internship Project Title | Automate extraction of handwritten text from an image |
| Name of the Company | TCS iON |
| Name of the Industry Mentor | Mr Niteen Gokhale |
| Name of the Institute | Vishwakarma University, Pune |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start Date | End Date | Total Effort (hrs.) | Project Environment | Tools used |
| 11/2/2025 | 15/2/2025 | 125 | Jupyter Notebook (Python 3) | 1. Python 2. OpenCV (cv2) 3. Pytesseract (Tesseract OCR) 4. PIL (Python Imaging Library)   Tesseract-OCR Engine zxcc |

**TABLE OF CONTENT**

* Acknowledgements
* Objective
* Introduction / Description of Internship
* Internship Activities
* Approach / Methodology
* Assumptions
* Exceptions / Exclusions
* Charts, Table, Diagrams
* Algorithms
* Challenges & Opportunities
* Risk Vs Reward
* Reflections on the Internship
* Recommendations
* Outcome / Conclusion
* Enhancement Scope
* Link to code and executable file
* Research questions and responses

**Acknowledgements**

I am pleased to present my progress on the project "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 extend my sincere gratitude to my industry mentor, Mr. Niteen Gokhale, and my faculty mentor, Prof. Shriprada Chaturbhuj, Assistant Professor, Faculty of Science & Technology, Vishwakarma University, Pune, for their invaluable guidance and support. I also express my thanks to our HOD, Dr. Rajkumar Jagdale, for providing the necessary facilities and assistance throughout this project.

**Objective**

To develop a Python-based system capable of detecting and extracting text from digital images using Optical Character Recognition (OCR) techniques, improving automation and efficiency in processing image-based information.

**Introduction / Description of Internship**

This internship under TCS iON provided exposure to real-world applications of image processing and machine learning techniques. The focus was on applying Python-based libraries and OCR tools to solve practical problems like automatic text extraction from images, a task useful in document digitization, license plate recognition, and archival systems.

**Internship Activities Completed**

* Gained understanding of OCR concepts and Python libraries such as OpenCV and Pytesseract.
* Implemented preprocessing techniques like grayscale conversion, thresholding, and noise removal.
* Built an image analysis pipeline to detect and extract text.
* Tested the system on various image datasets to evaluate accuracy and performance.
* Documented findings and presented project output to mentors.

**Methodology**

Image Acquisition: Load input images using OpenCV.

Preprocessing: Convert to grayscale, apply noise reduction, and thresholding for better OCR performance.

Text Detection:

* Use OpenCV to find contours.
* Use Pytesseract to extract textual content from regions of interest.

Post-processing: Draw bounding boxes on detected text and visualize results.

Evaluation: Compare extracted text with ground truth for accuracy analysis

### Algorithms

**OCR-Based Text Detection Algorithm**

**Input: Image**

1. Read image using OpenCV

2. Convert image to grayscale

3. Apply thresholding or adaptive threshold

4. Remove noise using morphological operations

5. Use Pytesseract to extract text

6. Draw bounding boxes on detected text regions

Output: Text and Annotated Image

### Challenges

* Poor OCR accuracy on low-resolution or noisy images.
* Difficulty in detecting handwritten text.
* Variations in font size and orientation affect performance.
* Complex backgrounds sometimes lead to incorrect text detection**.**
* **Recommendations**
* Use deep learning-based OCR (e.g., EAST text detector, CRNN) for better results.
* Include skew correction to handle rotated images.
* Employ image enhancement filters before OCR.
* Build custom preprocessing steps for specific types of documents (e.g., ID cards)
* **Enhancement Scope**
* Integrate real-time webcam support for live text extraction.
* Use Natural Language Processing (NLP) to summarize or categorize extracted text.
* Export results to structured formats like Excel or PDF.
* Add GUI using Tkinter or Flask-based web interface.

### Assumptions

* Input images are mostly in readable format (e.g., PNG, JPG).
* Tesseract OCR is installed and properly configured on the system.
* Text in images is mostly printed and horizontal.

### Exclusions

* The system does not support multi-language OCR out-of-the-box.
* No handwriting recognition was implemented.
* Complex scene text detection (on uneven surfaces, billboards) was not addressed.

**Current Status**

The Text Detection From Images project has been successfully implemented and tested in a controlled environment. The core system is capable of detecting and extracting printed text from static images using Python-based image processing and OCR techniques. Key components such as image preprocessing, text region detection using OpenCV, and text extraction using Tesseract OCR are fully functional.

The system demonstrates high accuracy for clean, printed images and performs reasonably well on scanned documents and labels. However, its performance is limited when dealing with noisy backgrounds, low-resolution inputs, or handwritten text. The results have been visualized effectively using bounding boxes and text overlays for validation.

The project is in a prototype-ready stage, with scope for enhancements like real-time text extraction, multilingual support, and GUI integration.

**Reflection**

This project helped bridge theoretical knowledge of computer vision and OCR with hands-on implementation using Python. It enhanced understanding of image preprocessing and real-world challenges in extracting information from visual data.

**Outcome/Conclusion**

A functioning prototype of a **Text Detection and Extraction System** was developed using OpenCV and Pytesseract. The project successfully demonstrates how OCR can automate data extraction tasks, providing a scalable solution for document digitization and information retrieval from images.

**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]}"

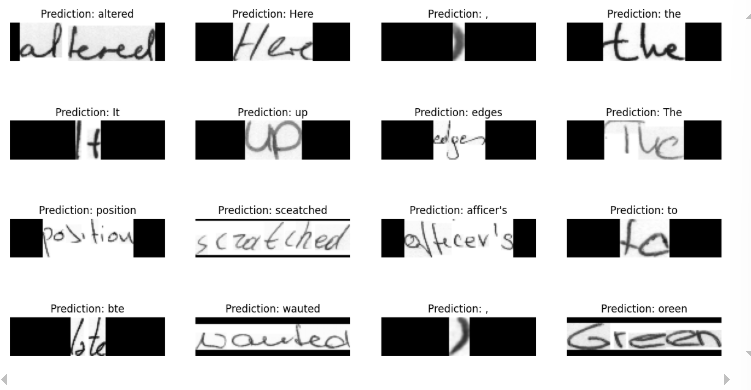
      ax[i // 4, i % 4].imshow(img, cmap = "gray")

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

      ax[i // 4, i % 4].axis("off")

    plt.show()

**Output:**

****

**Link to code and executable file-**

**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.
* **Git Repository Link** : <https://github.com/Varad1864/TCS-RIO-125.git>