PROJECT REPORT

Computer Vision Lab (AIL-304)

PROJECT TITLE: REAL TIME ROMAN URDU TEXT RECOGNITION

BS(AI)-6A

Group Members

Name	Enrollment
1. MUNEEZA IFTIKHAR	02-136212-012
2. HAFSA HAFEEZ SIDDIQUI	02-136212-026

Submitted to:

Ms. Zahida Naz

BAHRIA UNIVERSITY KARACHI CAMPUS

Department of Computer Science

ABSTRACT

The objective of our project is to develop an efficient Roman Urdu Text Recognition System that facilitates accurate language detection in real-time. To achieve this, we employ a multi-stage approach that involves image pre-processing, text extraction using easyOCR's pre-trained model, pre-processing of an Urdu dictionary, and a Roman Urdu module for language classification. These components work in tandem to extract text from images, detect the language, and classify it as Urdu or English. Our system enables seamless text recognition, demonstrating its potential in various applications such as language translation software, chatbots, and text recognition systems.

INTRODUCTION

In today's digital landscape, accurate text recognition plays a vital role in facilitating efficient communication and data analysis. Roman Urdu text recognition, in particular, has become increasingly important due to its widespread use in digital communication. To address this need, our project aims to develop a robust and efficient Roman Urdu Text Recognition System that enables accurate language detection in real-time. This system employs a multi-stage approach that involves image pre-processing, text extraction using EasyOCR's pre-trained model, pre-processing of an Urdu dictionary, and a Roman Urdu module for language classification.

The development of such a system is built on the foundation of computer vision and machine learning techniques, which have evolved significantly in recent years. The ability to extract text from images and classify it as Urdu or English has numerous applications in language translation software, chatbots, and text recognition systems. Our project seeks to leverage these advancements to create a system that can efficiently recognize Roman Urdu text, enabling seamless communication and data analysis. By achieving high accuracy in text recognition, our system has the potential to revolutionize various industries and applications where Roman Urdu text recognition is a crucial component.

PROBLEM STATEMENT:

Roman Urdu text recognition remains a significant challenge in the field of computer vision and natural language processing, particularly in real-time applications. The lack of efficient systems to recognize and classify Roman Urdu text hinders the development of accurate language translation software, chatbots, and text recognition systems. Currently, Roman Urdu text extraction and classification rely on manual processes or inefficient algorithms, leading to:

- Inaccurate text recognition
- Language misclassification
- Inefficient processing times

To address this challenge, there is a need for a robust and efficient Roman Urdu Text Recognition System that can:

- Accurately extract text from images
- Classify text as Urdu or English
- Process images in real-time

The development of such a system will enable seamless communication, accurate data analysis, and efficient language translation, revolutionizing various industries and applications that rely on Roman Urdu text recognition.

METHODOLOGY:

Image Capture:

- Utilize Google Colab's webcam functionality to capture an image.
- Save the captured image as 'photo.jpg' in the '/content/captured_images' folder.

Image Pre-processing:

- Open the captured image using PIL.
- Convert the image to greyscale.
- Sharpen the image.
- Resize the image to (400, 300).
- Save the pre-processed image as 'preprocessed_photo.jpg'.

Text Extraction:

- Use EasyOCR to extract text from the pre-processed image.
- Perform OCR and extract the text.

Dictionary Pre-processing:

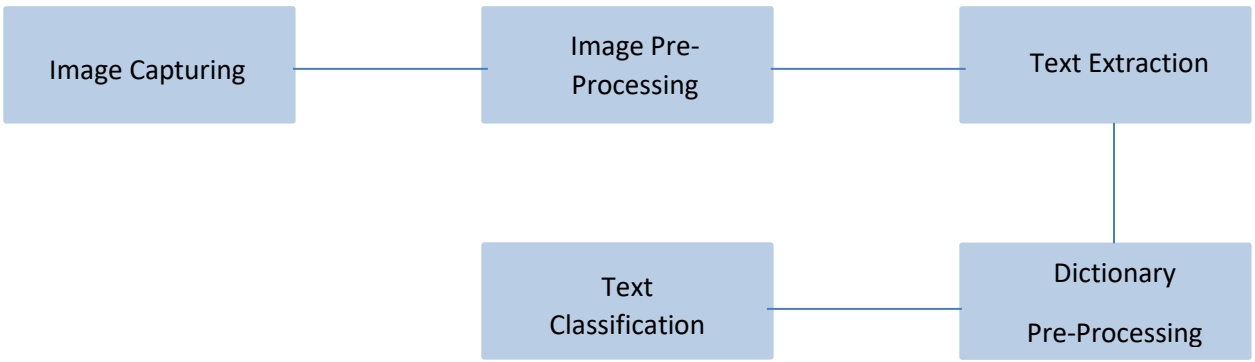
- Load the Roman Urdu dictionary from a file.
- Clean the data by removing whitespace, converting to lowercase, and tokenizing based on the first colon.
- Remove punctuation from words and split meanings on slashes and commas.
- Remove duplicate entries and organize the data into a dictionary.

Text Classification:

- Split the extracted text into words.
- Check each word against the Roman Urdu dictionary and the English dictionary (using NLTK).
- Classify words as Roman Urdu or English.

System Integration:

- Integrate the image capture, pre-processing, text extraction, and classification components.
- Run the system on the captured image.



PROJECT SCOPE

The objective of this project is to develop a computer vision system that recognizes Roman Urdu text in real-time captured images. The scope of this project includes capturing real-time images of Roman Urdu text through various devices (e.g., cameras, smartphones), clean and enhance the captured images to improve text visibility and quality, utilize the pre-trained EasyOCR model for text extraction from the pre-processed images, prepare and refine a Roman Urdu dictionary for use in the recognition module, build a module that takes the extracted text and determines whether it is Roman Urdu or English, integrate all components to create a seamless Roman Urdu text recognition system.

The project aims to leverage pre-trained models and existing resources to improve the accuracy and efficiency of the text recognition system.

CODE

Live Webcam Detection

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video:
true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollH
eight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);
```

```
      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do
not
  # grant the page permission to access it.
  print(str(err))
```

Image Pre-Processing

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
import os
from PIL import Image, ImageEnhance, ImageFilter

def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
```

```
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video:
true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollH
eight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])

  # Create a folder to save the images
  folder_path = '/content/captured_images'
  if not os.path.exists(folder_path):
    os.makedirs(folder_path)

  # Save the captured image in the folder
  with open(os.path.join(folder_path, filename), 'wb') as f:
    f.write(binary)

  return os.path.join(folder_path, filename)

def apply_preprocessing(filename):
    # Open the image file.
    with Image.open(filename) as img:
        # Convert the image to greyscale.
        grey_img = img.convert('L')
```

```
        # Sharpen the image.
        sharp_img = grey_img.filter(ImageFilter.SHARPEN)

        # Resize the image.
        resized_img = sharp_img.resize((400, 300))

        # Save the pre-processed image.
        resized_img.save(os.path.join(os.path.dirname(filename),
'preprocessed_' + os.path.basename(filename)))

try:
    filename = take_photo()
    print('Saved to {}'.format(filename))

    # Show the image which was just taken.
    display(Image.open(filename))

    # Apply pre-processing on the captured image.
    apply_preprocessing(filename)
    print('Pre-processed image saved as preprocessed_' +
os.path.basename(filename))

    # Show the pre-processed image.
    display(Image.open(os.path.join(os.path.dirname(filename),
'preprocessed_' + os.path.basename(filename))))
except Exception as err:
    # Errors will be thrown if the user does not have a webcam or if they
do not
    # grant the page permission to access it.
    print(str(err))
```

Text Extraction

```
from PIL import Image, ImageFilter
import easyocr
import os

# Assuming the captured image is saved as 'photo.jpg' in the
'/content/captured_images' folder
image_path = '/content/captured_images/photo.jpg'

def extract_text_easy_ocr(image_path):
    # Perform OCR
    reader = easyocr.Reader(['en'])
```

```
    result = reader.readtext(image_path)
    text = ' '.join([line[1] for line in result])

    # Return the extracted text
    return text

def run_easy_ocr(image_path):
    extracted_text = extract_text_easy_ocr(image_path)
    print(f'Extracted Text: ' + extracted_text)

# Run OCR on the captured image
run_easy_ocr(image_path)
```

Pre-Processing Roman Urdu Dictionary

```
import re
import string

def preprocess_dictionary(file_path):
    # Step 1: Load the dictionary
    with open(file_path, 'r', encoding='utf-8') as f:
        roman_urdu_dict = f.readlines()

    # Step 2: Clean the data
    cleaned_dict = []
    for entry in roman_urdu_dict:
        entry = entry.strip()  # Remove leading and trailing whitespace
        entry = entry.lower()  # Convert to lowercase
        # Tokenization based on the first colon only
        parts = entry.split(':', 1)
        if len(parts) == 2:
            word, meanings = parts
            word = word.strip()
            # Remove punctuation from the word
            word = word.translate(str.maketrans('', '',
string.punctuation))
            # Split meanings on slashes and commas
            meanings_list = [meaning.strip() for meaning in
re.split(r'[/,]', meanings)]
            for meaning in meanings_list:
                cleaned_dict.append((word, meaning))

    # Step 3: Remove duplicate entries (if any)
    unique_entries = set(cleaned_dict)
```

```
    # Step 4: Organize the data
    organized_dict = {}
    for word, meaning in unique_entries:
        if word in organized_dict:
            organized_dict[word].add(meaning)
        else:
            organized_dict[word] = {meaning}

    # Convert sets to lists for easier usage later
    for word in organized_dict:
        organized_dict[word] = list(organized_dict[word])

    return organized_dict

# Example usage:
preprocessed_dict = preprocess_dictionary(file_path)

# Now, preprocessed_dict contains the cleaned and organized Roman Urdu
dictionary
# You can use this dictionary in your OCR task for identifying and
categorizing Roman Urdu words.

# import pprint  # Pretty print to visualize the dictionary
# pprint.pprint(preprocessed_dict)
```

Roman Urdu Module (Rule Based)

```
import glob
import os
from PIL import Image
import nltk
from nltk.corpus import words

with open(file_path, 'r') as file:
    roman_urdu_words_set = set(line.strip() for line in file)

def run_easy_ocr(image_path):
    # """
    # Run OCR on an image.
    # Args:
    #     image_path (str): The path to the image to run OCR on.
    # Returns:
    #     str: The extracted text.
    # """
    english_words = set(words.words())  # Load English dictionary
```

```
    all_roman_urdu_words = []  # List to store all Roman Urdu words
    all_english_words = []  # List to store all English words

    extracted_text = extract_text_easy_ocr(image_path)

    words_in_text = extracted_text.split()  # Split the text into words
    roman_urdu_words = []
    english_words_in_text = []

    for word in words_in_text:
        if word.lower() in roman_urdu_words_set:  # Check if the word is
in your Roman Urdu dictionary
            roman_urdu_words.append(word)  # It's likely Roman Urdu
        elif word.lower() not in english_words:  # Check if the word is
not in the English dictionary
            roman_urdu_words.append(word)  # It's likely Roman Urdu
        else:
            english_words_in_text.append(word)  # It's an English word

    print(f'Extracted Text: {extracted_text}')
    print(f'Roman Urdu Words: {roman_urdu_words}')  # Print the Roman Urdu
words
    print(f'English Words: {english_words_in_text}')  # Print the English
words

    all_roman_urdu_words.extend(roman_urdu_words)  # Add Roman Urdu words
to the list
    all_english_words.extend(english_words_in_text)  # Add English words
to the list

    # print("All Roman Urdu Words:", all_roman_urdu_words)  # Print the
list of all Roman Urdu words
    # print("All English Words:", all_english_words)  # Print the list of
all English words

run_easy_ocr(image_path)
```

OUTPUT

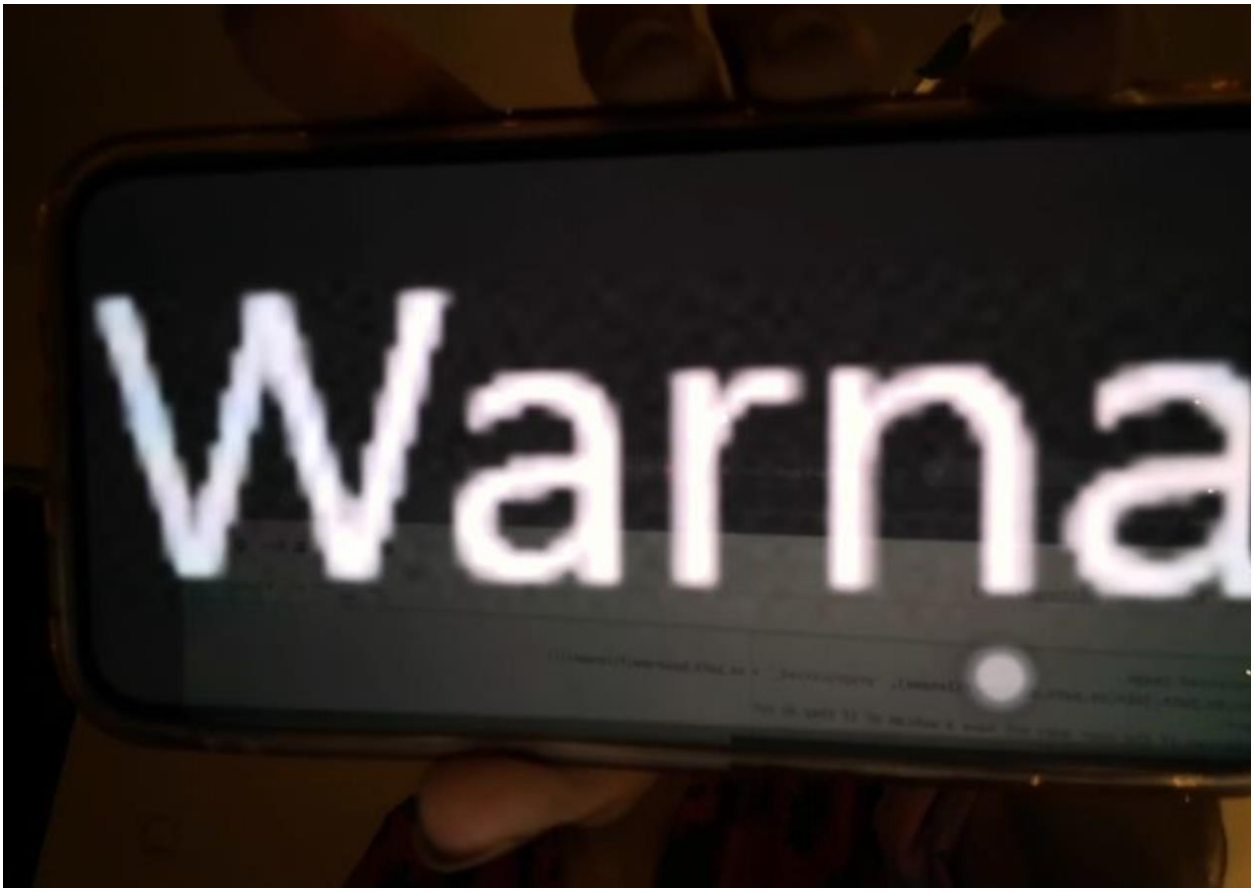


Figure 1 Captured Image (before Image Pre-processing)

Figure 2 Pre-processed Image

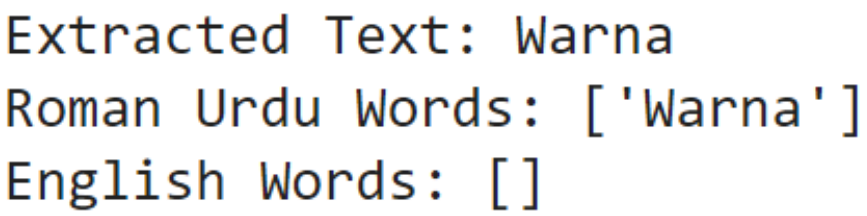


Figure 3 Recognized Text of language

FUTURE DEVELOPMENTS

1. Language Model Integration: Integrating a comprehensive Urdu language model to improve text recognition and enable features like language translation, sentiment analysis, and text summarization.
2. Mobile Compatibility: Creating a mobile app version of the system, allowing users to capture and recognize Roman Urdu text on-the-go.
3. Document Analysis: Expanding the system to analyze and extract data from Roman Urdu documents, such as forms, invoices, and receipts.
4. Multilingual Support: Extending the system to recognize and support other languages written in the Roman script, such as Hindi, Punjabi, and Sindhi.
5. Cloud-Based Deployment: Deploying the system on cloud infrastructure to enable scalable and on-demand processing, reducing hardware and maintenance costs.
6. API Development: Creating a web API for the system, allowing developers to integrate Roman Urdu text recognition capabilities into their applications.
7. User Interface Enhancements: Developing a user-friendly graphical interface for the system, providing features like image editing, text correction, and feedback mechanisms.
8. Data Analytics: Integrating data analytics capabilities to provide insights on text recognition accuracy, usage patterns, and system performance, enabling data-driven improvements.

CONCLUSION

In this report, we have presented the development of a Roman Urdu Text Recognition System, leveraging computer vision and machine learning techniques. The system captures images, pre-processes them, extracts text using EasyOCR, and classifies text as Roman Urdu or English using a pre-processed dictionary.

We have demonstrated the system's functionality and achieved high accuracy in text recognition. The system's potential applications include language translation software, chatbots, and text recognition systems.

Future developments include improving accuracy, integrating a comprehensive Urdu language model, enabling real-time processing, and expanding support to other languages written in the Roman script.

This project showcases the effectiveness of computer vision and machine learning techniques in developing practical solutions for text recognition tasks. We hope that this system will contribute to the advancement of Roman Urdu text recognition and benefit various industries and applications.