National Egyptian E-Learning University

Faculty of Computer & Information Technology

Smart Glasses For visually impaired people

By:

1.Marwan Maher Mostafa (ID:2002008)
2.Mahmood Abdulghani Hassen (ID:2001274)
3.Hassan abo Elghet Hassan (ID:2001197)
4.Nada Hamada Salah (ID:2000684)
5.Nancy Saad Mohamed (ID:2001336)
6.Menna Allah Mahmoud Thabet (ID:2000689)
7.Arwa Sayed Elarosy(ID:2001485)

Under Supervision of:

Prof . Ahmed hamza

Lecturer at Faculty Computer and Information Technology, Egyptian E-Learning University

Eng. Mostafa Abdelbari

Teacher Assistant, Department of Information Technology Egyptian E-Learning University

This project is submitted as a partial fulfillment of the requirements for the degree of Bachelor of Science in Computer & Information Technology

Acknowledgement

First and for most, praises and thanks to God, the Almighty, for his blessing throughout our years in the college and our graduation project to complete this stage of our life successfully.

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincerethanks to all of them.

Thanks to Egyptian E-Learning University especially Assiut center for helping us to reach thislevel of awareness.

We would like to express our deep and sincere gratitude to our project supervisor **Dr. Ahmed Hamza** for giving us the opportunity to lead us and providing invaluable guidance throughout this project. It was a great privilege and honor to work and study under her guidance. We are extremely grateful for what she has offered us.

We are especially indebted to say many thanks to **Eng. Mostafa Abdelbari** for guidance and constant supervision as well as for his patience, friendship, empathy, and a great sense of humor. His dynamism, vision, sincerity, and motivation have deeply inspired us.

Finally, we would like to express our gratitude to everyone who helped usduring the graduation project.

Contents:

Chapter 5: Implementation

Chapter 6: Results and Discussion

List of Acronyms or Abbreviations

- AI: Artificial Intelligence
- API: Application Programming Interface
- CV: Computer Vision
- ML: Machine Learning
- TTS: Text-to-Speech
- UI: User Interface

List of Figures

1. **Chapter 3: Background**

2. **Chapter 4: System Design**

3. **Chapter 5: Implementation**

4. **Chapter 6: Results and Discussion**

List of Acronyms or Abbreviations

WHO: World Health Organization

ML: MachineLearning.

CV: ComputerVision

IOT:Internet of Things.

IDE: integrated development environment.

PWM: Pulse Width Modulation.

SSD: Single Shot Detector

mAP: Mean Average Precision.

FPS:Frame per Second.

EAR: Eye Aspect Ratio

MAR: Mouth Aspect Ratio

KNN: K-Nearest Neighbors

AI: Artificial Intelligence
API: Application Programming Interface
CV: Computer Vision
ML: Machine Learning
TTS: Text-to-Speech
UI: User Interface
List of Figures

Abstract

One of the main problems in the world is the prevention of accidents and road safety. Vehicle detection plays an important role in traffic control at signalized intersections. Recently, the whole world witnessed many road crashes that are considered the most common troubles that result from the lack of scanning to detect and respond to hazards, rapid driving, or lack of attention due to something inside or outside of the vehicle that leads to loss of human lives every year.

According to World Health Organization statistics, every year the lives of approximately 1.35 million people are cut short as a result of a road traffic crash. Between 20 and 50 million people suffer non-fatal injuries and some of them suffer from a disability as a result of their injuries. [1]

In order to reduce the number of deaths and harmful injuries resulting from the occurrence of this type of road accident, we propose an intelligent system that can easily detect any coming obstacle in front of the vehicle and take accurate action in a holistic manner to rescue thousands of human lives.

Experimental results on different traffic scenes show that the proposed system is effective and has high performance for real-time vehicles detection.

Chapter 1

Introduction

1.1 History

Assistive technologies have significantly evolved over the past few decades, driven byadvancements in artificial intelligence (AI) and machine learning (ML).

Early assistive devices for the visually impaired, such as Braille readers and magnifiers, primarily focused on enhancing the readability of text. With the advent ofdigital technologies, screen readers and voice assistants have become common, providing real-time audio feedback for digital content.

The recent surge in smartphone adoption has further accelerated the development of mobile applications designed to aid the visually impaired, leveraging the powerful processors and advanced cameras embedded in these devices. However, despite these advancements, there remains a significant gap in solutions that address real- world textrecognition in outdoor environments, such as reading street banners.

1.2 Motivation

The motivation behind this project stems from the desire to enhance the independenceand quality of life for visually impaired individuals. Navigating urban environments presents numerous challenges, especially when it comes to reading textual informationon street banners, shop signs, and public notices.

These texts often contain important information regarding directions, store names, andannouncements that are crucial for day-to-day activities. Current assistive technologiesfall short in providing a seamless, real-time solution for outdoor text recognition and audio conversion. This project aims to fill this gap by developing a smart app that utilizes AI models to detect, recognize, and convert text from street banners into audio,thereby enabling visually impaired users to access this

information effortlessly and independently.

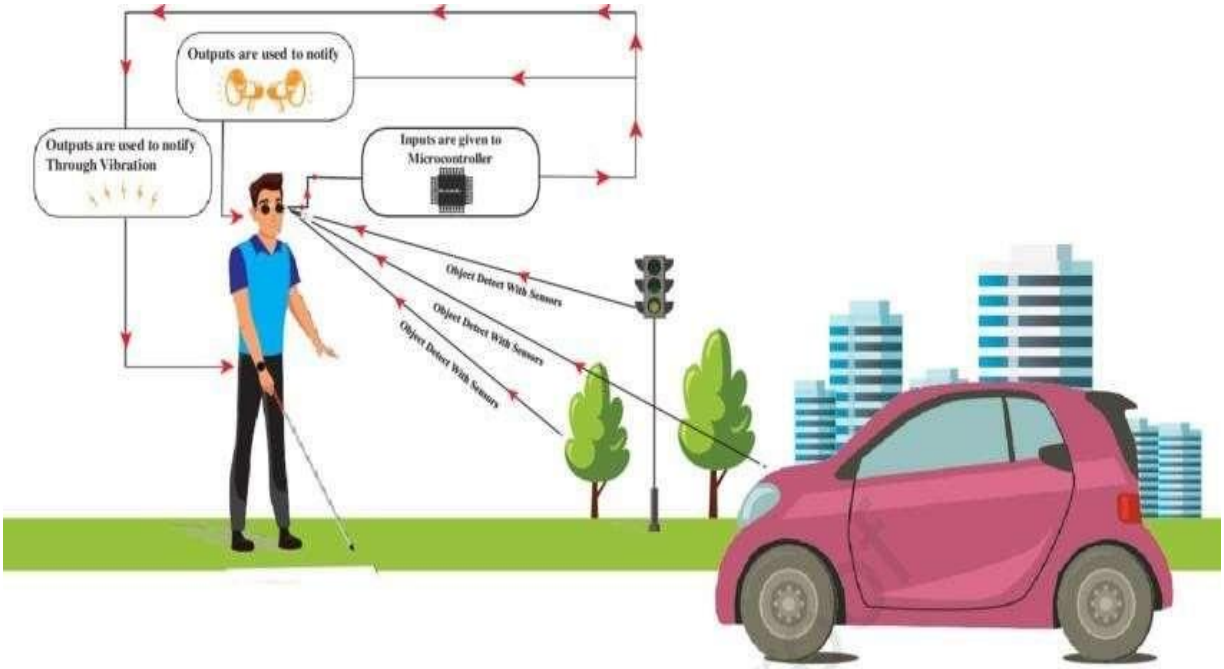
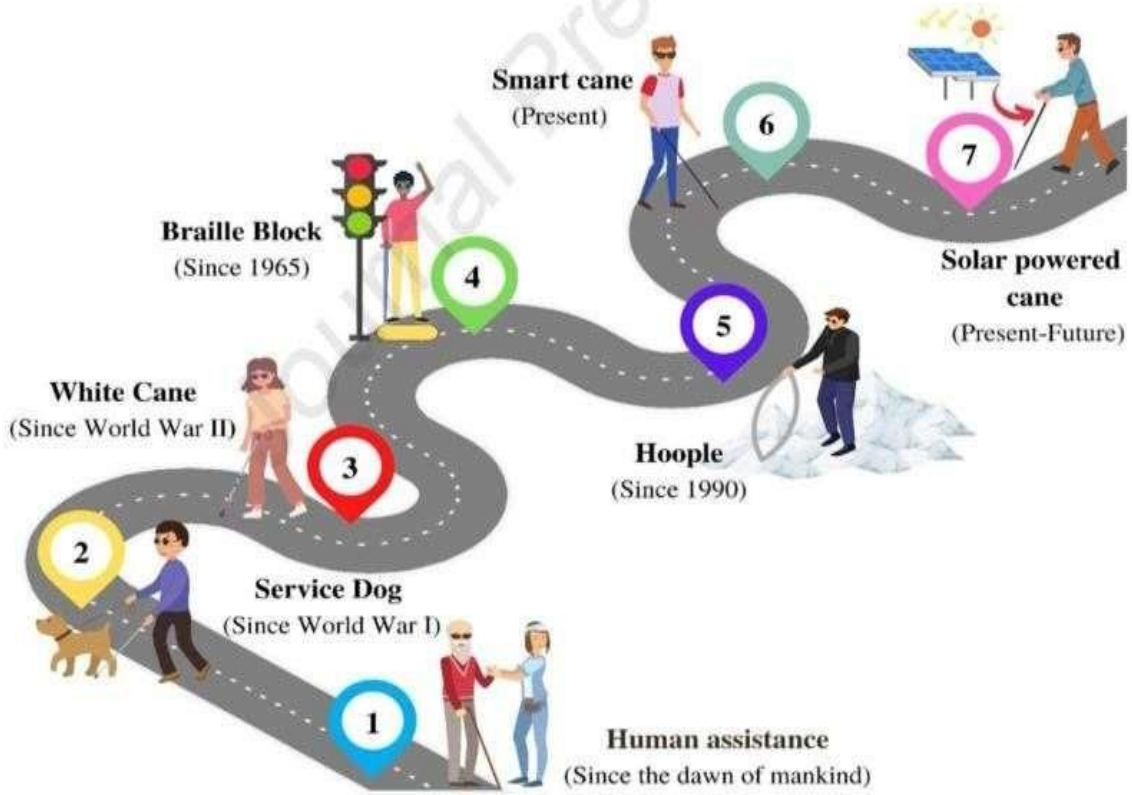


Figure 2: Application of smart glass and smart cane in real life



3. Problem Statement

Visually impaired individuals face significant challenges in accessing textual information on street banners and public signs, which are essential for navigation and daily activities.

Existing assistive technologies and applications often lack the capability to accurately detect and recognize text in varying outdoor conditions and convert it to audio in real-time.

This project seeks to address this problem by developing a mobile application that integrates advanced AI models for text detection, text recognition, and text-to-speech conversion, thereby providing an accessible solution for the visually impaired to read street banners and signs.

4. Project Phases

The development of the smart app is structured into several phases:

1. **Research and Planning**: Conduct a thorough literature review on existing technologies and identify the requirements for the AI models and application framework.
2. **Model Development**: Develop and train the AI models for text detection, text recognition, and text-to-speech conversion.
3. **Backend Development**: Implement the Django API to handle data processing and communication between the frontend and the AI models.
4. **Frontend Development**: Develop the user interface using Vue.js and Vite, ensuring it is user-friendly and accessible.
5. **Integration and Testing**: Integrate the frontend, backend, and AI models. Conduct extensive testing to ensure accuracy, performance, and usability.
6. **Deployment and Evaluation**: Deploy the application and gather user feedback to evaluate its effectiveness and identify areas for improvement.

5. Project Schedule

Week	Date Range	Task
1	Oct 1 - Oct 7	Write and review introduction
2-3	Oct 8 - Oct 21	Research related work
4-5	Oct 22 - Nov 4	Historical Overview
6-7	Nov 5 - Nov 18	Existing Solutions
8-9	Nov 19 - Dec 2	Text Detection
10-11	Dec 3 - Dec 16	System Architecture
12-13	Dec 17 - Dec 30	Text-to-Audio Model
14-15	Jan 3 - Jan 16	Database Design
16-17	Jan 17 - Jan 30	User Interface Design
18-19	Jan 31 - Feb 13	Hardware Requirements
20-21	Feb 14 - Feb 27	Model Training
22-23	Feb 28 - Mar 12	API Development
24-25	Mar 13 - Mar 26	App Development
26-27	Mar 27 - Apr 9	Text Detection Accuracy
28-29	Apr 10 - Apr 23	Text-to-Speech Performance
30-31	Apr 24 - May 7	Analysis of Results
32-35	May 8 - May 31	References

Chapter2

Literature

Review

2.1 Related Work:

In recent years, significant progress has been made in the field of assistive technology for thevisually impaired, particularly through the application of AI and machine learning. Various research studies and projects have explored different approaches to text detection, recognition, and conversion to audio, each contributing to the development of more effective solutions.

Assistive Technologies

Research on assistive technologies has highlighted the importance of providing real-time, accessible solutions to enhance the independence of visually impaired individuals. Early studies focused on hardware-based solutions such as Braille displays and magnifiers, which offered limited portability and functionality.

AI and Machine Learning in Assistive Applications

The integration of AI and machine learning has revolutionized assistive applications. Notable works include the development of OCR (Optical Character Recognition) systems, which convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data. Projects like Microsoft's Seeing AI and Google's Lookout app leverage AI to provide visually impaired users with audio descriptions of their surroundings, including text.

Limitations of Existing Solutions

Despite advancements, existing solutions face several limitations. Many apps require continuous internet access, limiting their usability in offline scenarios. Furthermore, text detection and recognition in outdoor environments with varying lighting conditions and text styles remain challenging. These limitations highlight the need for more robust, standalone applications that can operate effectively in real-time without dependency on constant connectivity.

Chapter 3

Background

1. Assistive Technology for the Visually Impaired

Historical Overview

The journey of assistive technology for the visually impaired began with basic tools such as canes and Braille. The invention of Braille in the 19th century marked a significant milestone, providing a tactile writing system that remains widely used

today. The 20th century saw the introduction of electronic devices like the Perkins Brailler and screen readers, which expanded access to written and digital content.

1.2 Modern Innovations :

Modern innovations have leveraged digital technology and AI to create more sophisticated assistive devices. Advances in natural language processing (NLP) and computer vision have led to the development of screen readers that can interpret and vocalize text from digital screens. Wearable devices equipped with cameras and sensors, such as the OrCam MyEye, offer real-time text recognition and object identification, providing greater independence to visually impaired users.

1. Smartphone Apps for Accessibility

1. Existing Solutions

Several smartphone applications have been developed to aid visually impaired individuals.

Microsoft's Seeing AI app uses computer vision to describe people, text, and objects.

Google's Lookout app similarly provides audio feedback about the environment, including reading text from signs and labels.

Apps like Be My Eyes connect users with volunteers for visual assistance via live video calls.

2. Comparative Analysis

While these apps offer valuable services, they differ in functionality, ease of use, and dependence on internet connectivity.

Seeing AI and Lookout excel in providing automated descriptions using AI, but their accuracy can vary based on environmental factors.

Be My Eyes relies on human assistance, which, while accurate, requires aninternet connection and available volunteers.

A comparative analysis highlights the need for a balanced solution that combines the autonomy of AI-driven applications with the reliability andaccessibility of offline operation.

3. Machine Learning and Text Analysis

- Machine Learning and Text Analysis involve using algorithms to extract meaningful insights from textual data. This field combines natural language processing techniques with machine learning models to analyze, interpret, and predict information from large volumes of text.

1. **Text Detection:**

Text detection involves identifying the presence and location of text within an image, a critical first step in the text analysis pipeline. Modern techniques use convolutional neural networks (CNNs) to detect text regions with high accuracy, enabling a wide range of applications from document digitization to real-time assistive technologies for the visually impaired.

- **Convolutional Neural Networks (CNNs) :**

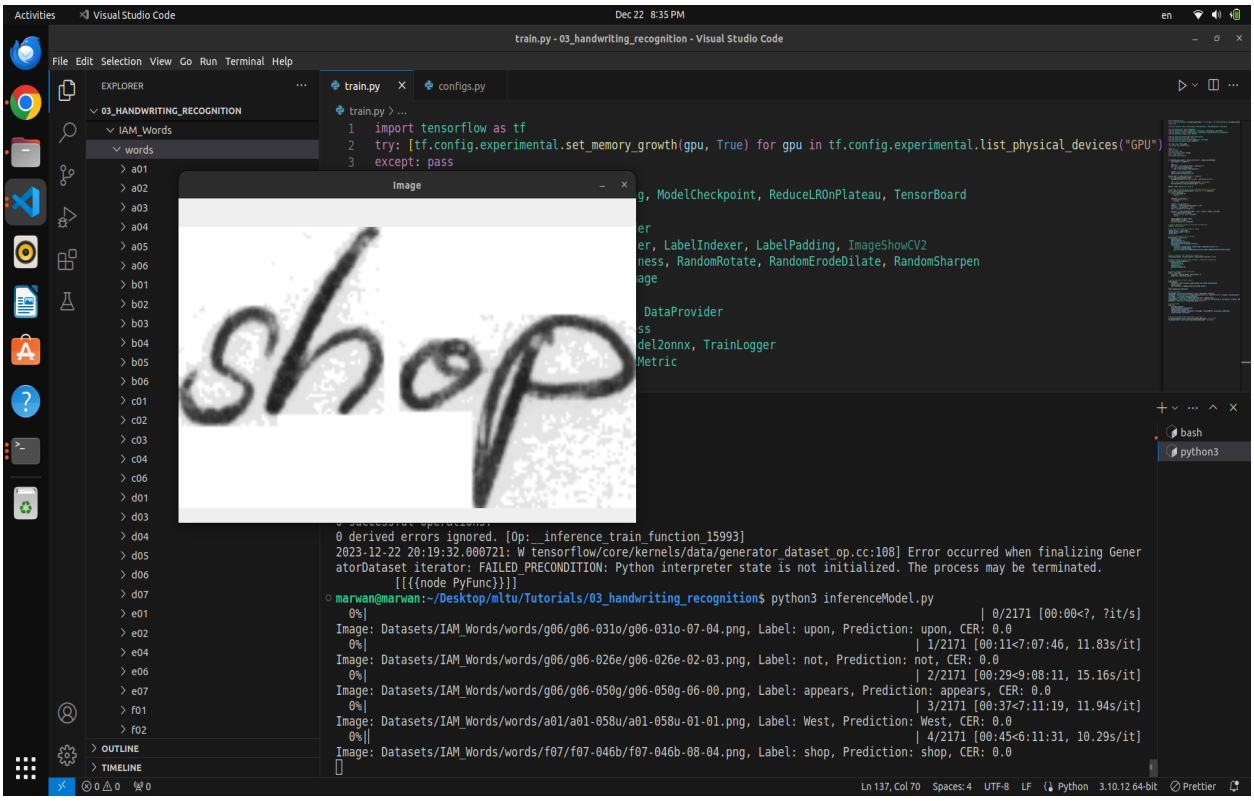
CNNs are a class of deep learning algorithms particularly well-suited for image processing tasks. They consist of layers of neurons that learn to detect various features of the input image, such as edges, shapes, and textures, which are then used to identify more complex patterns like text regions. The hierarchical nature of CNNs allows them to progressively build a robust representation of text within an image, enhancing detection accuracy.

- **The EAST Model :**

The Efficient and Accurate Scene Text Detector (EAST) model is one of the most advanced approaches for text detection. Unlike traditional methods that rely on a multi-stage pipeline, EAST uses a fully convolutional network that directly predicts text regions and orientations in a single forward pass. This model balances efficiency and accuracy, making it suitable for real-time applications. Key features of EAST include:

- **Direct Region Prediction**: EAST predicts the presence and orientation of text regions directly from the input image, eliminating the need for intermediate processing steps.

- **Quadrilateral Bounding Boxes**: The model outputs quadrilateral bounding boxes, which can more accurately encompass text regions compared to traditional rectangular boxes, especially for slanted or curved text.
- **Real-Time Performance**: Optimized for speed, EAST can process images quickly, making it ideal for applications that require real-time text detection.



2. Text Recognition :

Text recognition, or optical character recognition (OCR), converts detected text regions into machine-readable text. This process is essential for transforming visual text data into a format that can be used for further processing, such as text-to-speech conversion or data entry automation.

- **Recurrent Neural Networks (RNNs) and CNNs :**

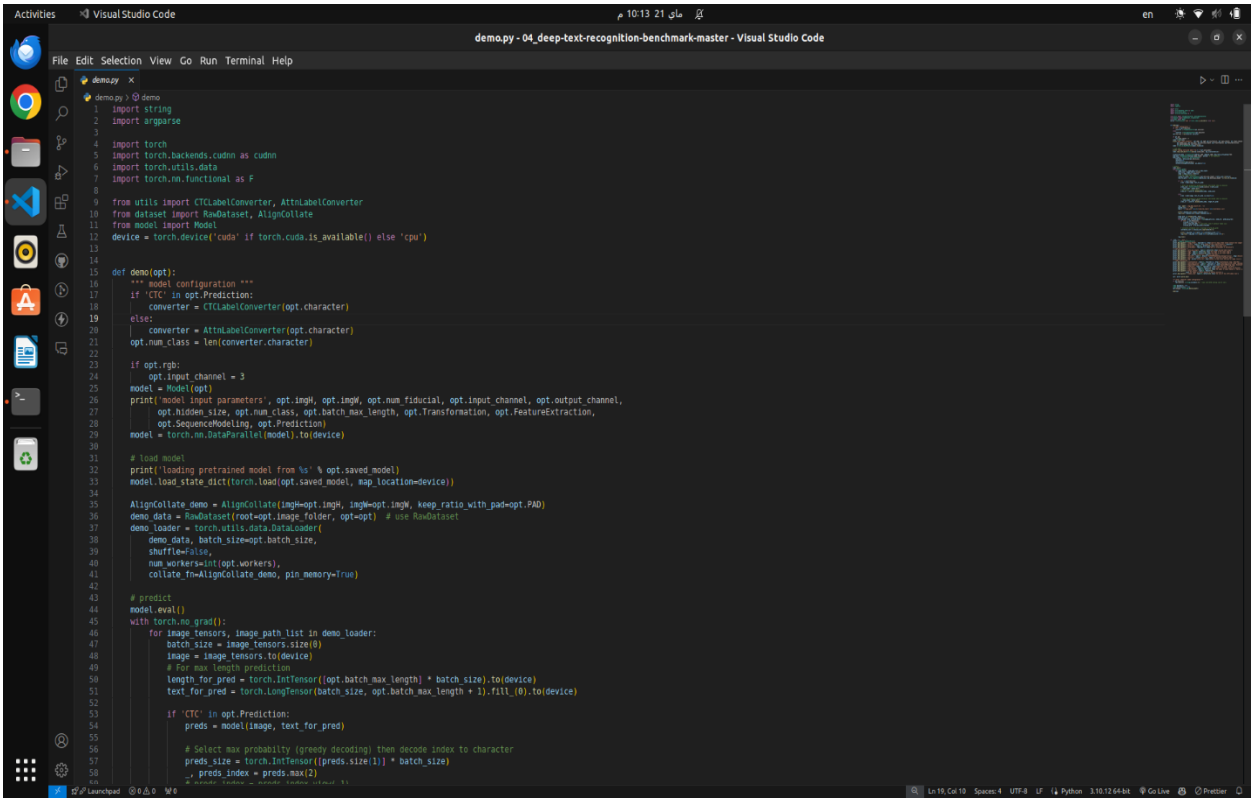
OCR systems often use a combination of CNNs and recurrent neural networks (RNNs) to achieve high accuracy in text recognition. CNNs are responsible for extracting visual features from the text regions, while RNNs handle the sequential nature of text data, capturing dependencies between characters or words.

- **CNNs for Feature Extraction**: By analyzing the visual features of text regions, CNNs generate detailed feature maps that highlight distinctive patterns in the characters.
- **RNNs for Sequence Modeling**: RNNs, particularly Long Short-Term Memory (LSTM) networks, are used to process these feature maps sequentially, learning the relationships between consecutive characters or words. This sequence modeling is crucial for accurate text recognition, especially for handwritten or cursive text.

- **Tesseract OCR :**

Developed by Google, Tesseract is a widely used open-source OCR engine. It supports multiple languages and text formats, making it versatile for various applications. Key features of Tesseract include:

- **Multi-Language Support**: Tesseract can recognize text in over 100 languages, providing broad applicability across different regions and use cases.
- **Training Capabilities**: Users can train Tesseract on custom datasets, enhancing its performance for specific types of text or languages.
- **Integration with Other Tools**: Tesseract can be easily integrated with other software and libraries, such as OpenCV, for comprehensive image processing workflows.



3. Text-to-Speech Conversion :

Text-to-speech (TTS) conversion translates text into spoken words, enabling applications that provide auditory feedback from written content. Modern TTS

systems use deep learning models to produce natural-sounding speech, significantly improving the user experience for visually impaired individuals.

Deep Learning Models for TTS :

Deep learning models, such as Google's WaveNet and Tacotron, have revolutionized TTS by generating high-quality audio that captures the nuances of human speech, including tone, intonation, and rhythm.

- **WaveNet**: Developed by DeepMind, WaveNet generates raw audio waveforms from text inputs. It uses a neural network trained on large datasets of speech audio, learning to produce realistic and expressive speech. WaveNet captures subtle variations in speech patterns, making the generated audio sound more natural.
- **Tacotron**: Tacotron is an end-to-end TTS model that directly converts text to spectrograms, which are then transformed into audio waveforms using a vocoder like WaveNet. Tacotron's architecture allows it to learn the mapping from text to speech features, producing high-fidelity audio with natural prosody.

VALL-E-X :

VALL-E-X is a cutting-edge TTS model that leverages the latest advancements in deep learning to generate high-quality, natural-sounding speech from text inputs. Developed with a focus on versatility and performance, VALL-E-X is designed to deliver exceptional audio clarity and intelligibility, making it an ideal choice for assistive technologies.

Key Features of VALL-E-X:

- **Multi-Voice Support**: VALL-E-X can generate speech in multiple voices, accents, and languages, providing a customizable auditory experience for users.
- **Emotion and Prosody Control**: The model can modulate tone, pitch, and rhythm to convey different emotions and speaking styles, enhancing the expressiveness of the generated speech.
- **Real-Time Processing**: Optimized for speed, VALL-E-X can produce audio in real-time, ensuring minimal latency and immediate feedback for users.

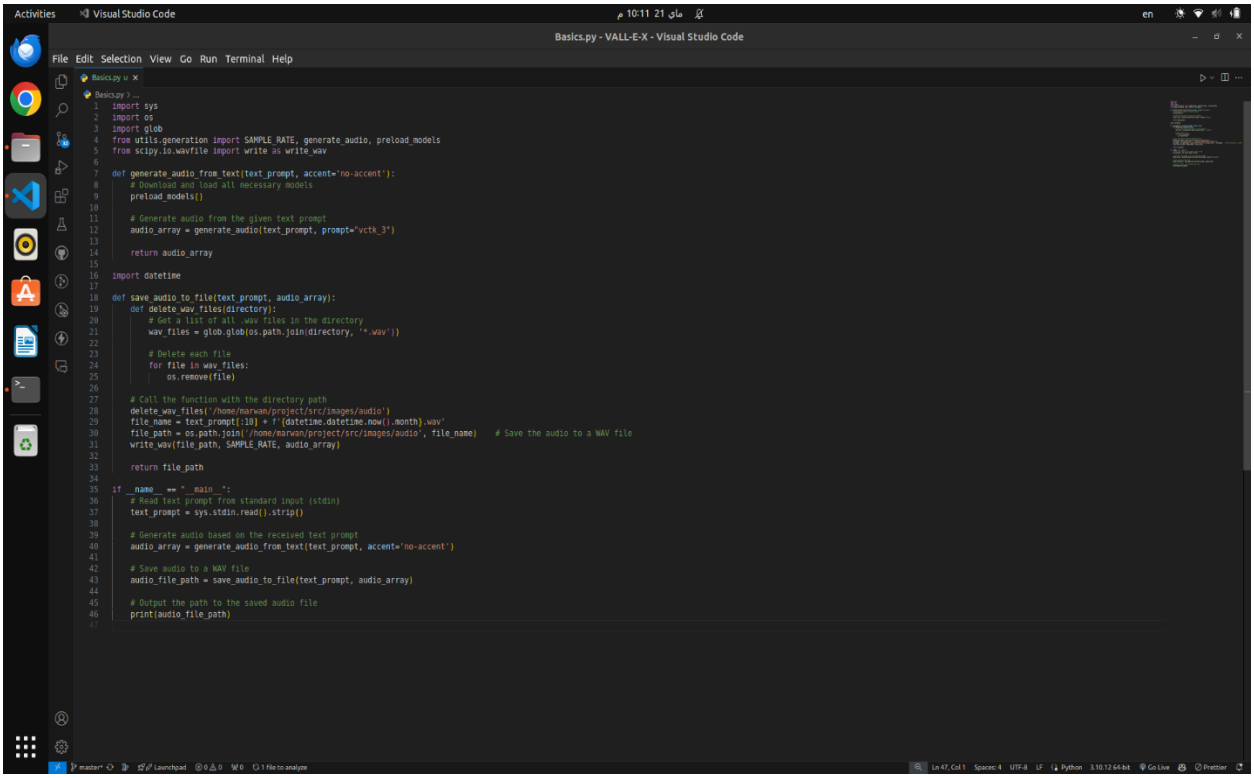
	Previous Systems	VALL-E X
Intermediate representation	Mel spectrogram	Audio codec codes
Training data	< 13K hours	70K hours
Speech accent	Foreign	Native
Speaker similarity	Relative low	High
In-context learning	✗	✓
Zero-shot cross-lingual TTS	✗	✓

Our VALL-E X is trained using bilingual speech-transcription (ASR) data. The Chinese ASR data are from WenetSpeech [Zhang et al., 2022a] containing 10,000+ hours of multi-domain labeled speech. The English ASR data are from LibriLight [Kahn et al., 2020] containing about 60,000 hours of unlabeled speech, whose speech data are collected from audiobooks. We train a Kaldi4 ASR model on the labeled Librispeech [Panayotov et al., 2015] dataset to generate the pseudo transcripts for the unlabeled LibriLight speech. To train the speech recognition & translation model for S2ST, we also use additional MT and ST data. The MT data are from AI Challenger5 , OpenSubtitles20186 and WMT20207 , which contain about 13M, 10M, and 50M sentence pairs in conversion, drama8 , and news domains, respectively

Key Features of Modern TTS Systems

- **Natural-Sounding Speech**: By leveraging deep learning techniques, modern TTS systems generate speech that closely mimics human speech, enhancing the listening experience.
- **Customization**: These systems can be customized to produce different voices, accents, and speaking styles, catering to diverse user preferences.
- **Intelligibility**: High-quality TTS models ensure that the generated speech is clear and easy to understand, which is crucial for accessibility applications.

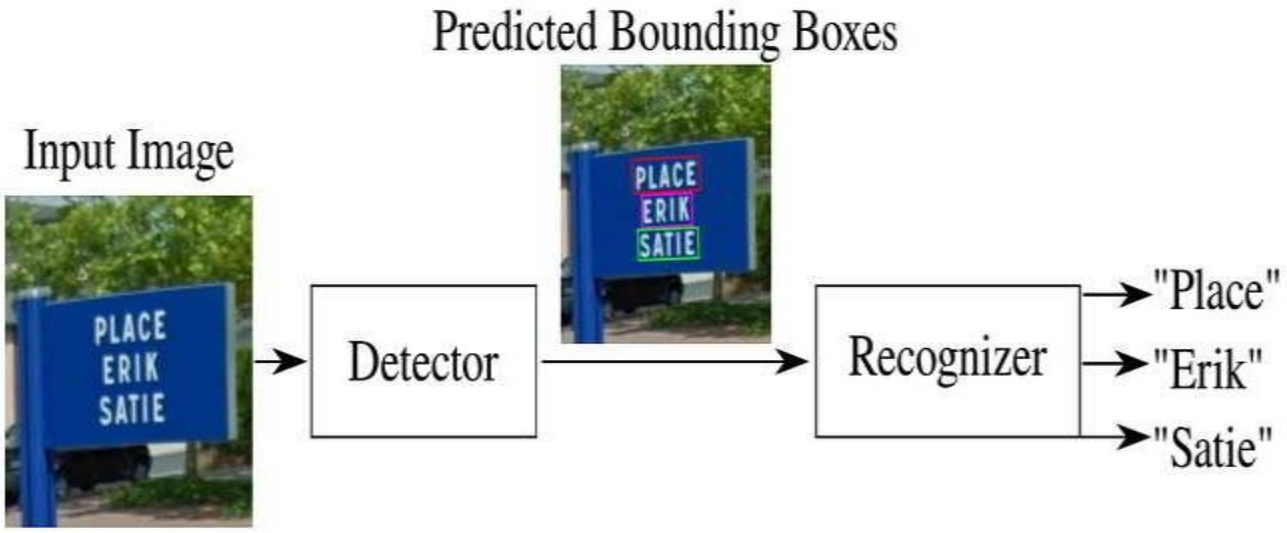
In summary, the integration of advanced machine learning techniques in text detection, recognition, and TTS conversion has significantly enhanced the capabilities of assistive technologies for the visually impaired. These innovations offer greater independence and accessibility, transforming how visually impaired individuals interact with the world.



4.Evaluation Metrics :

Evaluation metrics are critical for assessing the performance of text detection, recognition,and TTS systems.

- Common metrics for text detection include precision, recall, and the F1 score, which balance accuracy and completeness. OCR systems are evaluated usingword error rate (WER) and character error rate (CER).
- TTS systems are assessed based on naturalness, intelligibility, and mean opinion score (MOS), which measures user satisfaction with the generated speech.



Backend Development

Django API

The backend of the application is powered by a Django framework, which handles the server-side logic, including API requests, model inference, and data management.

Key Components

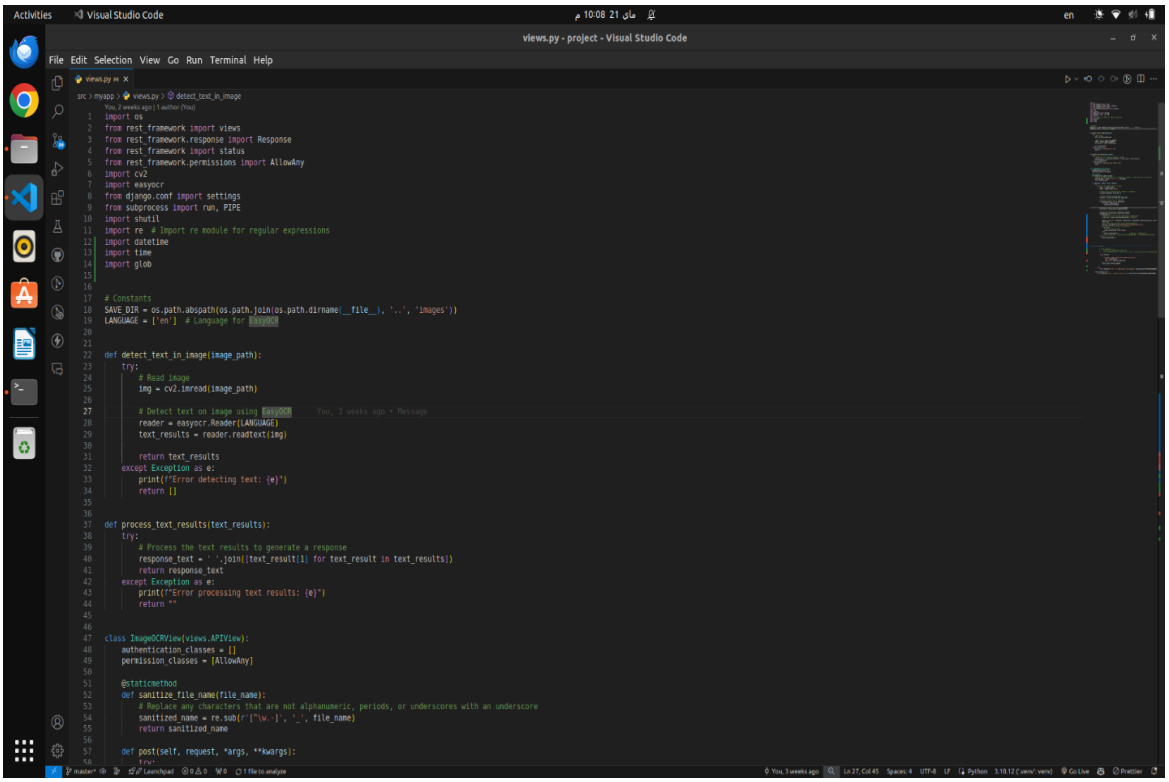
- **API Endpoints**: Defined for image upload, text processing, and audio retrieval.
- **Request Handling**: Manages incoming requests, processes images, and coordinates with AI models.
- **Response Generation**: Sends back processed text or audio results to the frontend.

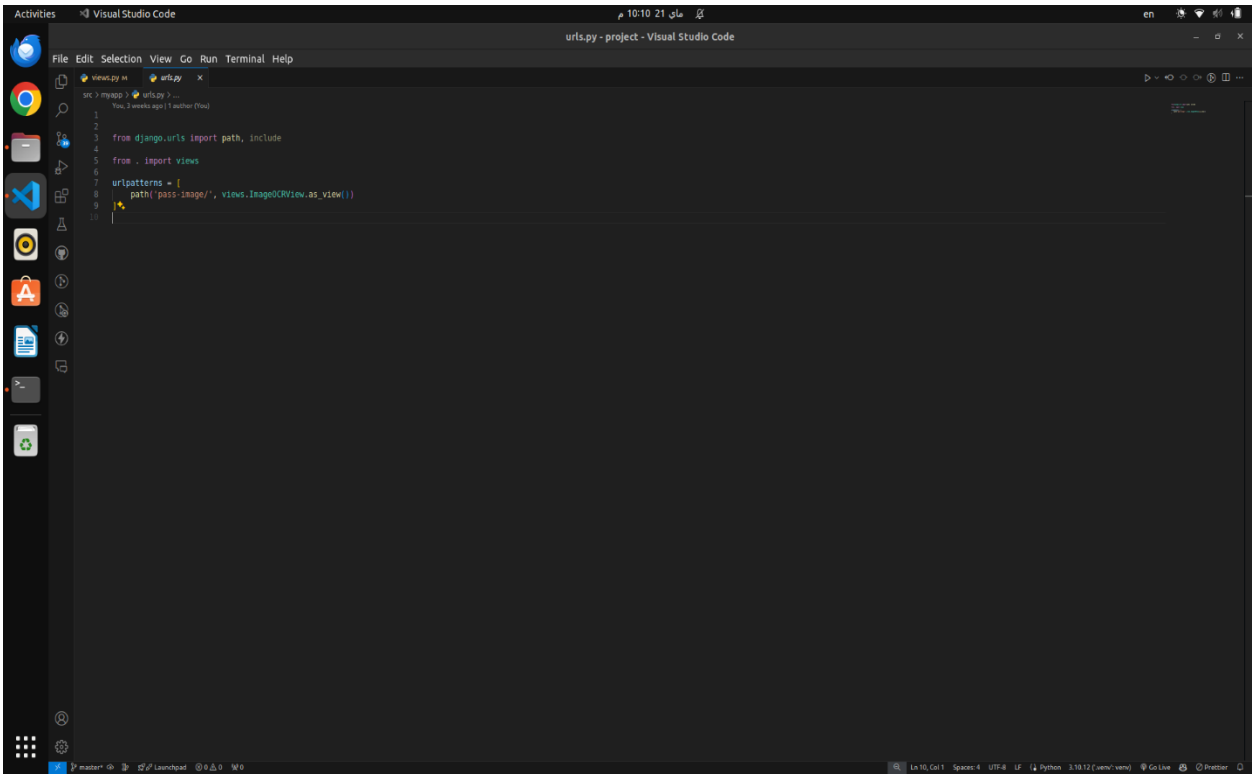
Database Design

A robust database is essential for storing user interactions, processed text data, and application metadata. The database schema is designed to efficiently manage and retrieve information as needed.

Key Components

- **User Data**: Logs user activities and preferences.
- **Image Data**: Stores images and associated metadata.
- **Text Data**: Maintains recognized text entries and their processing status.




Activities
Visual Studio Code
10:10 21 ماي م
en
urls.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py M
urls.py
src > myapp > urls.py > ...
You, 3 weeks ago | 1 author (You)
from django.urls import path, include
from . import views
urlpatterns = [
path('pass-image/', views.ImageOCRView.as_view())
]

Frontend Development

Vue.js Framework

The frontend application is built using the Vue.js framework, known for its flexibility and ease of integration. Vue.js allows for the creation of a responsive and accessible user interface.

Key Components

- **Image Capture**: Functionality to capture images using the device camera.
- **User Interaction**: Simple, intuitive controls for navigation and operation.
- **Audio Playback**: Real-time playback of converted text to speech.

User Interface Design

User interface design focuses on accessibility and ease of use for visually impaired users. Key considerations include:

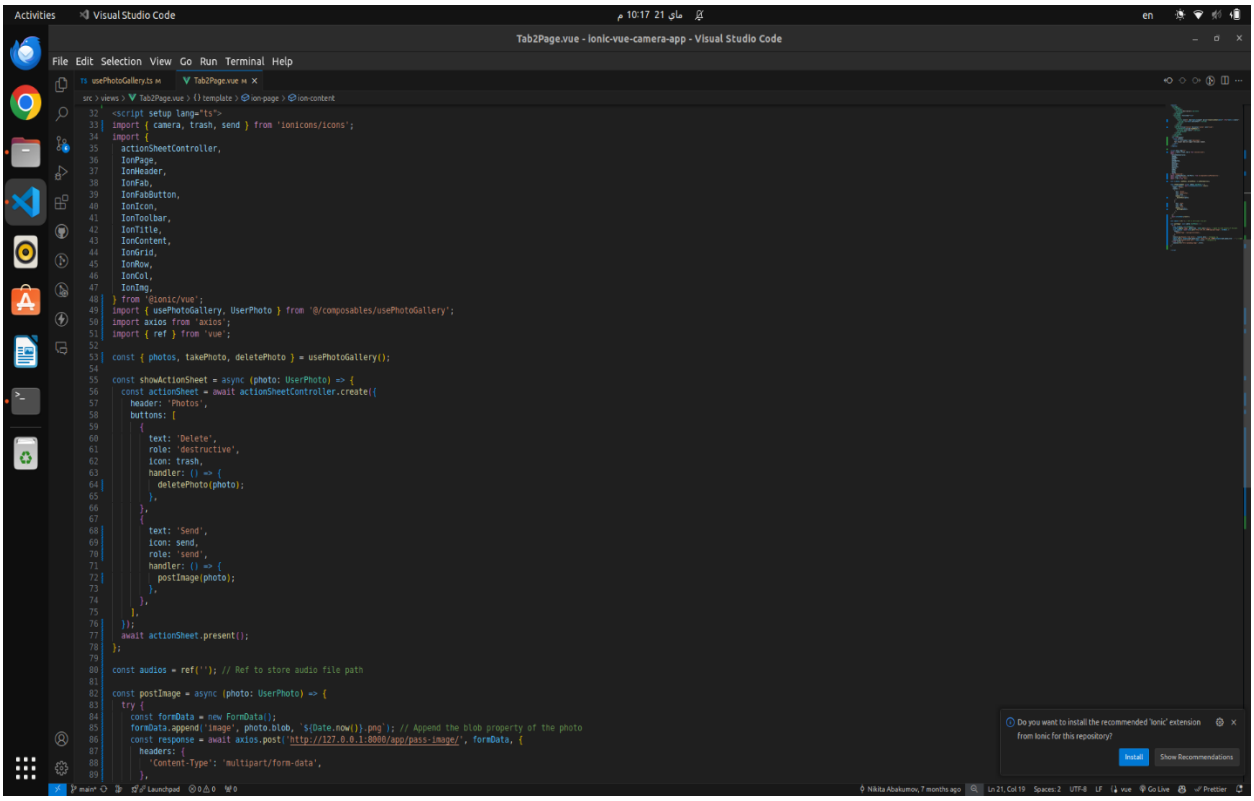
- **Voice Commands**: Enabling hands-free operation.
- **Large Buttons**: Simplified navigation with easily identifiable touch targets.
- **Contrast and Text Size**: Ensuring readability for users with partial vision.

Integration with Backend

Seamless integration between the frontend and backend is crucial for real-time performance. The frontend communicates with the Django API to send images and receive processed text and audio.

Key Components

- **API Communication**: Efficient handling of requests and responses.
- **Error Handling**: Robust mechanisms to manage and inform users of any issues during processing.
- **Real-Time Updates**: Ensuring prompt feedback and minimal latency in text-to-speech conversion.



Chapter 4

System Design

3.1 System Architecture

The system architecture for the smart app designed to assist the visually impaired is a multi-layered structure that integrates AI models with robust backend and user-friendly frontend components. The architecture ensures seamless interaction between different modules, facilitating real-time text detection, recognition, and audio conversion.

3.2 Overview

1. **User Interface**: The frontend application built using Vue.js, which captures images and provides audio output.
2. **Backend Server**: A Django-based API that processes images, invokes AImodels, and manages data.
3. **AI Models**: Machine learning models for text detection, text recognition, and text-to-speech conversion, each serving a specific function in the text analysis pipeline.
4. **Database**: A structured storage sy

2. AI Models

1. Text Detection Model

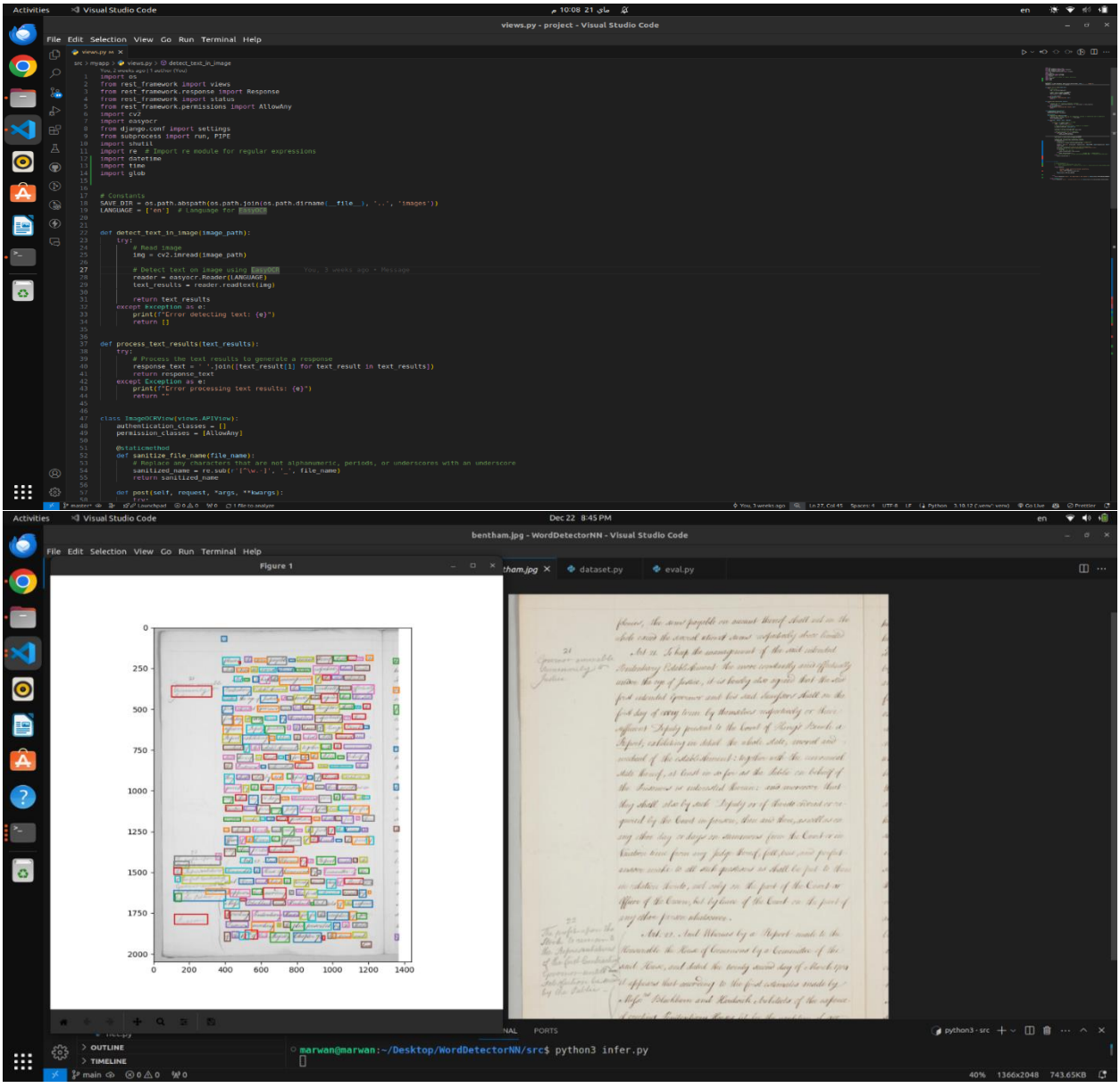
The text detection model is responsible for identifying and localizing text regions within an image. This model employs a convolutional neural network (CNN) architecture, such as the EAST (Efficient and Accurate Scene Text) detector, which is optimized for real-time processing and high accuracy.

Key Components

- **Input**: Raw image captured by the user.
- **Processing**: The model scans the image and highlights text regions.
- **Output**: Coordinates of bounding boxes around detected text areas.

Training Data

The model is trained on a diverse dataset containing images with various text styles, fonts, and lighting conditions to enhance robustness and accuracy.



2. Text Recognition Model

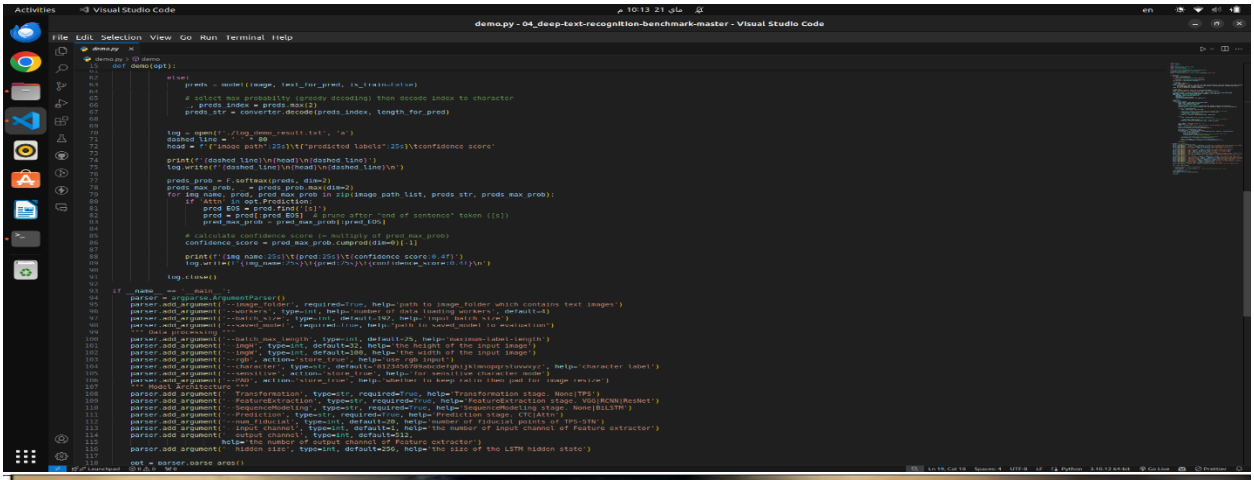
Once text regions are detected, the text recognition model converts these regions into machine-readable text. This model utilizes a combination of CNNs and Recurrent Neural Networks (RNNs) to accurately recognize characters from the detected text regions.

Key Components:

- **Input**: Cropped image regions containing text.
- **Processing**: The model uses OCR techniques to interpret and convert the text images into strings.
- **Output**: Recognized text in a readable format.

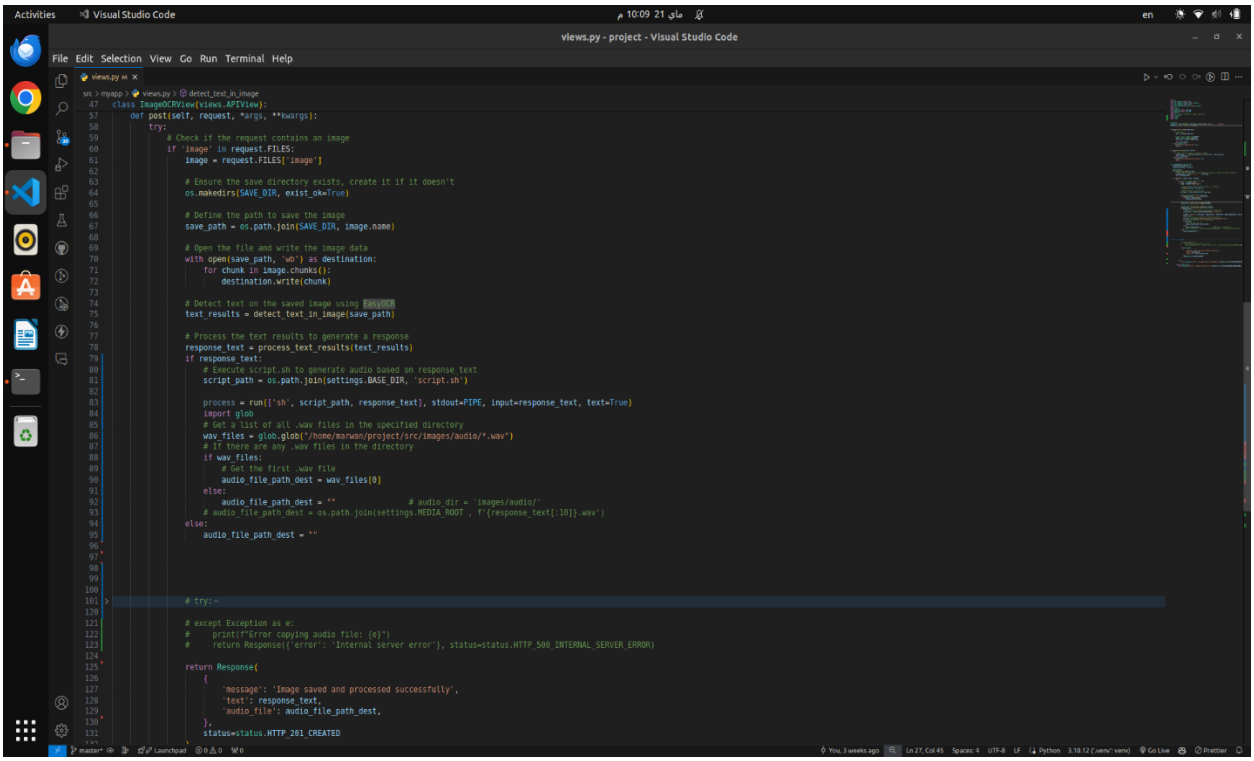
Training Data

Training involves datasets with labeled text images, covering multiple languages and character sets to ensure broad applicability.



3. Text-to-Audio Model

The final stage involves converting the recognized text into speech. This is achieved
using a text-to-speech (TTS) model, such as Google's WaveNet or Tacotron, which generates natural-sounding audio.

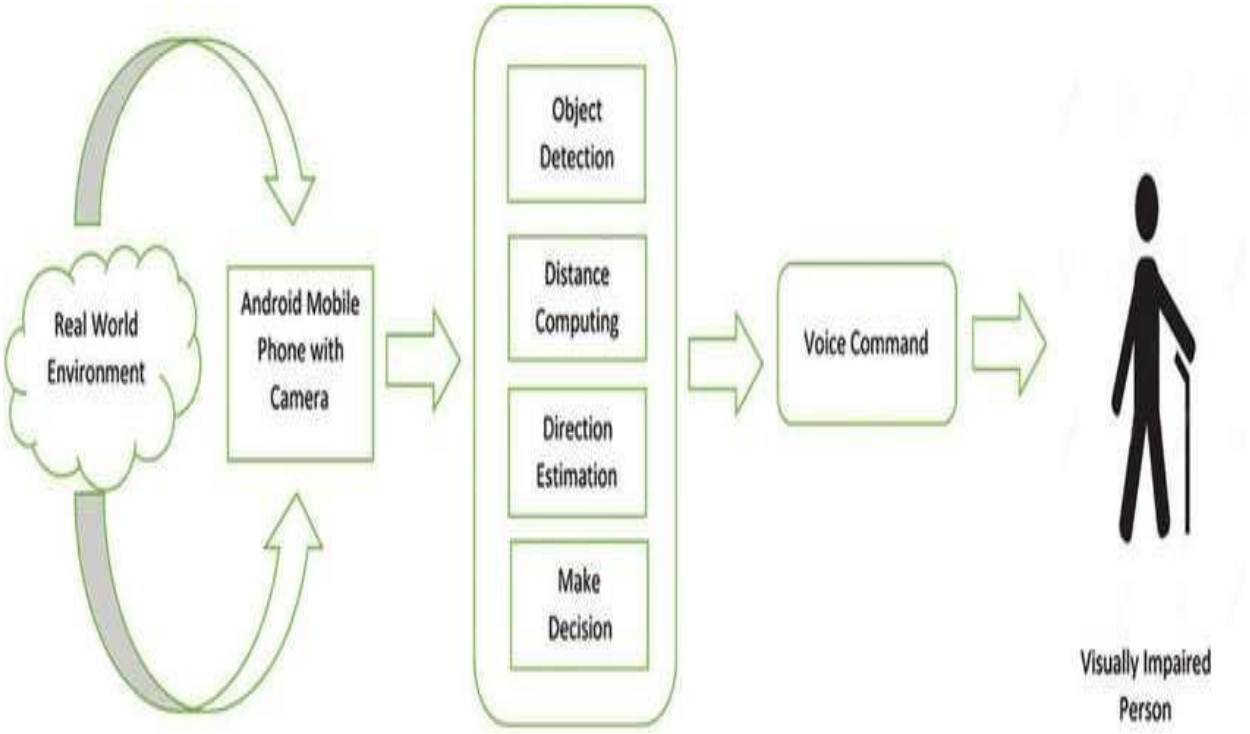


Key Components:

- **Input**: Recognized text strings.
- **Processing**: The model synthesizes speech from text using deep learning techniques
  To ensure clarity and natural intonation.
- **Output**: Audio file or real-time speech playback.

Training Data

The TTS model is trained on large corpora of text and corresponding audio to learn the nuances ofpronunciation, intonation, and speech rhythm.



3.3 Backend Development

1. Django API

The backend of the application is powered by a Django framework, which handles theserver-side logic, including API requests, model inference, and data management.

Key Components

- **API Endpoints**: Defined for image upload, text processing, and audio retrieval.
- **Request Handling**: Manages incoming requests, processes images, and coordinates with AI models.

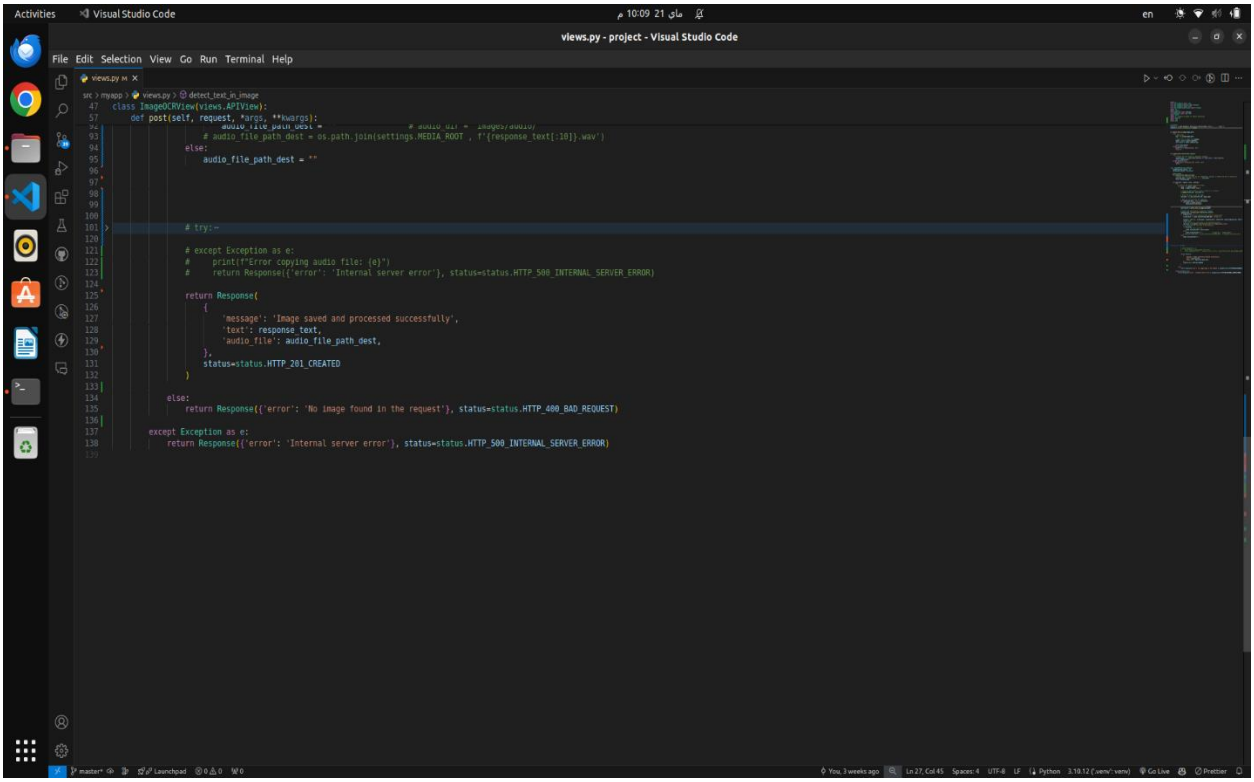
Response Generation: Sends back processed text or audio results to the frontend.

2.Database Design

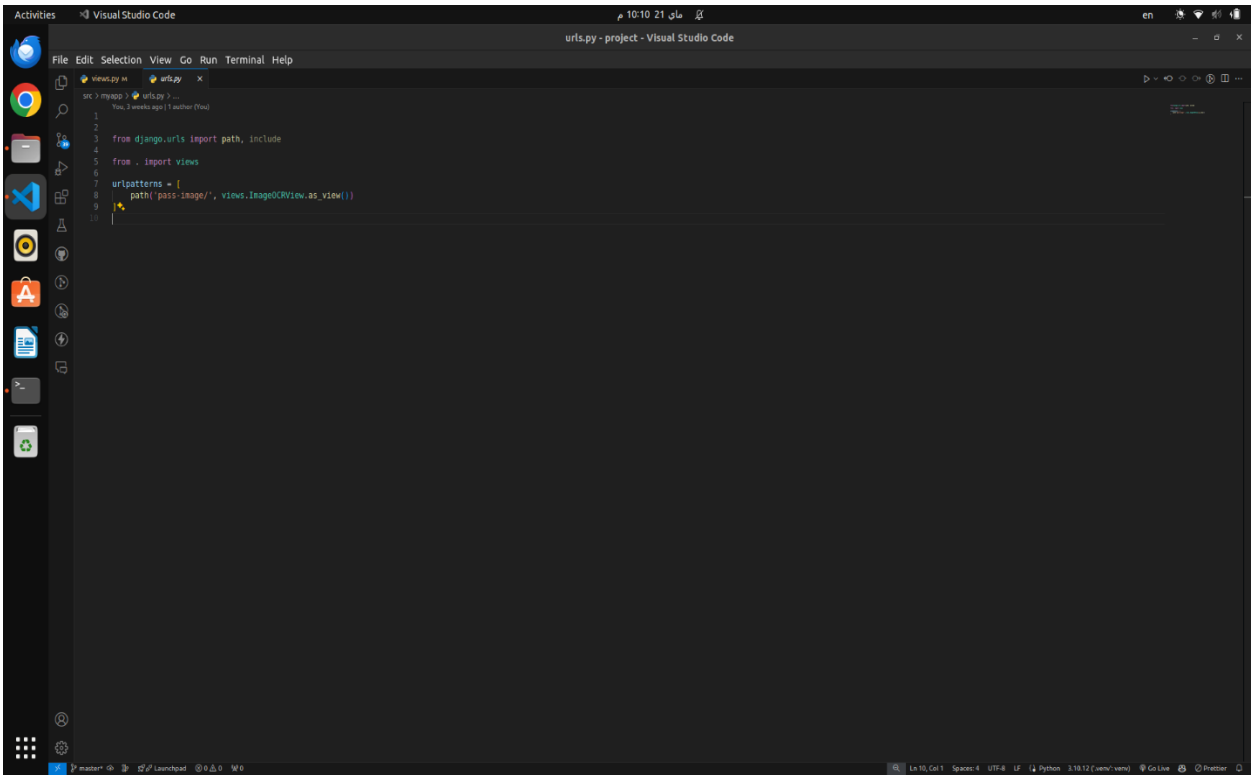
A robust database is essential for storing user interactions, processed text data, and application metadata. The database schema is designed to efficiently manage and retrieveinformation as needed.

Key Components

- **User Data:** Logs user activities and preferences.
- **Image Data:** Stores images and associated metadata.
- **Text Data:** Maintains recognized text entries and their processing status


Activities
Visual Studio Code
views.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py
class ImageOCRView(views.APIView):
def post(self, request, *args, **kwargs):
# audio_file_path_dest = os.path.join(settings.MEDIA_ROOT , f'{response_text[:10]}.wav')
audio_file_path_dest = ""
# try:-
# except Exception as e:
# print(f"Error copying audio file: {e}")
# return Response({'error': 'Internal server error'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
return Response(
{
'message': 'Image saved and processed successfully',
'text': response_text,
'audio_file': audio_file_path_dest,
},
status=status.HTTP_201_CREATED
)
else:
return Response({'error': 'No image found in the request'}, status=status.HTTP_400_BAD_REQUEST)
except Exception as e:
return Response({'error': 'Internal server error'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)

.


Activities
Visual Studio Code
urls.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py
urls.py
from django.urls import path, include
from . import views
urlpatterns = [
path('pass-image/', views.ImageOCRView.as_view())
]

4. Frontend Development

1.Vue.js Framework :

The frontend application is built using the Vue.js framework, known for its flexibility and ease of integration. Vue.js allows for the creation of a responsive and accessible user interface.

Key Components:

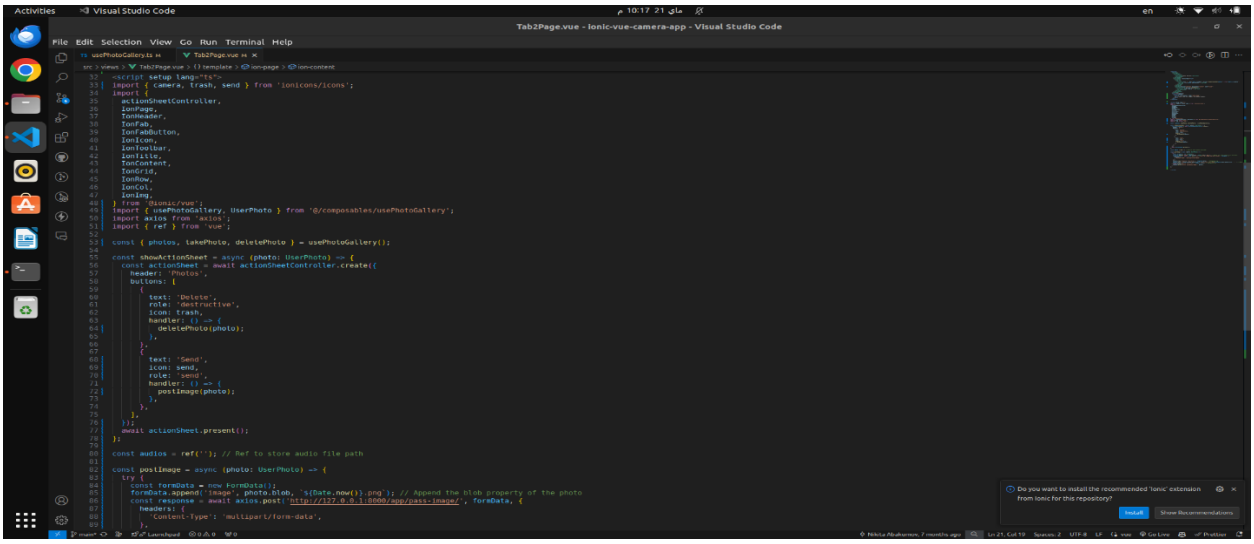
- **Image Capture:** Functionality to capture images using the device camera.
- **User Interaction:** Simple, intuitive controls for navigation and operation.
- **Audio Playback**: Real-time playback of converted text to speech.

2.User Interface Design:

User interface design focuses on accessibility and ease of use for visually impaired users.

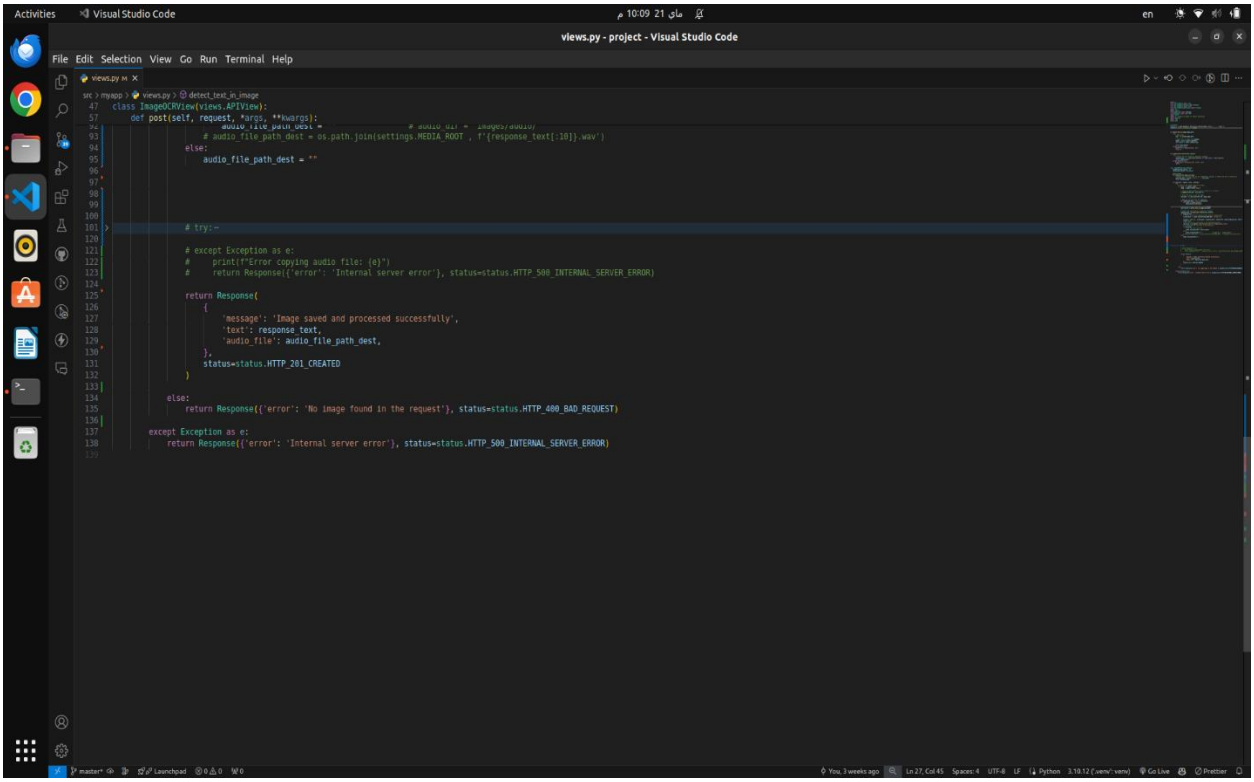
Key considerations include:

- **Voice Commands**: Enabling hands-free operation.
- **Large Buttons:** Simplified navigation with easily identifiable touch targets.
- **Contrast and Text Size**: Ensuring readability for users with partial vision.



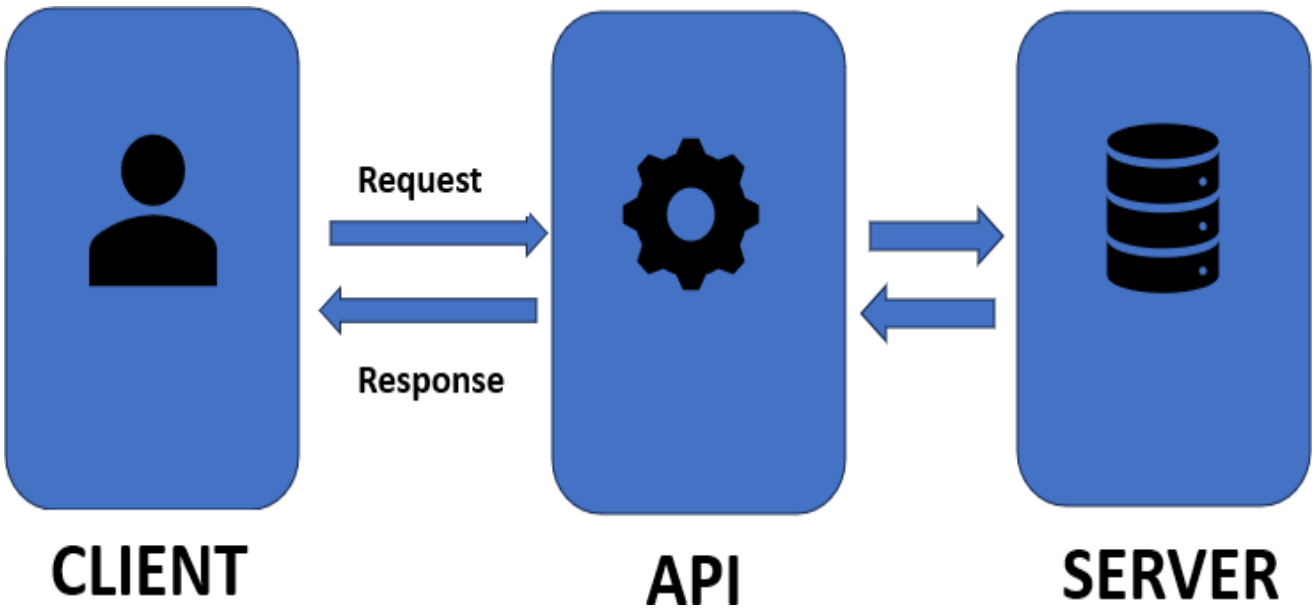
3. Integration with Backend:

Seamless integration between the frontend and backend is crucial for real-time performance. The frontend communicates with the Django API to send images and receiveprocessed text and audio.



Key Components:

- **API Communication:** Efficient handling of requests and responses.
- **Error Handling:** Robust mechanisms to manage and inform users of any issues during processing.
- **Real-Time Updates:** Ensuring prompt feedback and minimal latency in text- to-speechconversion.



Chapter 5

Implementation

1. Hardware and Software Requirements:

1. Hardware Requirements

The hardware requirements for this project are minimal, focusing primarily on devices capableof running the application and performing image capture.

Key Components:

- **Smartphone:** Android or iOS device with a camera.
- **Development Machine**: PC or Mac for developing and testing the application, with specifications such as:
- **Processor**: Intel i5 or equivalent
- **RAM**: 8 GB
- **Storage**: 256 GB SSD
- **GPU:** Optional, for model training acceleration (NVIDIA GTX 1050 or higher)

2. Software Requirements

The software stack for this project includes a combination of development tools, frameworks, and libraries.

Key Components:

- **Operating System**: Windows, macOS, or Linux
- **Programming Languages**: Python (for backend and AI models), JavaScript (for frontend)

- **Development Tools:**
- **IDE**: PyCharm or VSCode for backend development, Visual Studio Code for frontenddevelopment
- **Version Control:** Git and GitHub for source code management

- **Frameworks and Libraries:**
- **Backend:** Django, Django REST framework
- **Frontend:** Vue.js, Vite
- **AI/ML:** TensorFlow, Keras, OpenCV, Tesseract OCR, PyTorch

TTS: gTTS (Google Text-to-Speech) AI Model Development

1.Model Training

The AI models for text detection, text recognition, and text-to-speech conversion aredeveloped and trained using appropriate datasets and machine learning frameworks.

Key Steps:

1. **Dataset Preparation:** Collect and preprocess datasets containing images with diverse text samples for training the text detection and recognition models. For TTS, use a dataset oftext and corresponding audio recordings.
2. **Model Architecture**: Design and implement model architectures:

- **Text Detection:** EAST model or similar CNN-based architecture
- **Text Recognition**: CRNN (Convolutional Recurrent Neural Network) or Tesseract OCR

- **Text-to-Speech**: WaveNet or Tacotron.

3. **Training**: Use TensorFlow or PyTorch to train the models on the prepared datasets. Employtechniques like data augmentation to improve model robustness.
4. **Optimization:** Fine-tune hyperparameters and employ techniques such as transfer

2. Model Evaluation

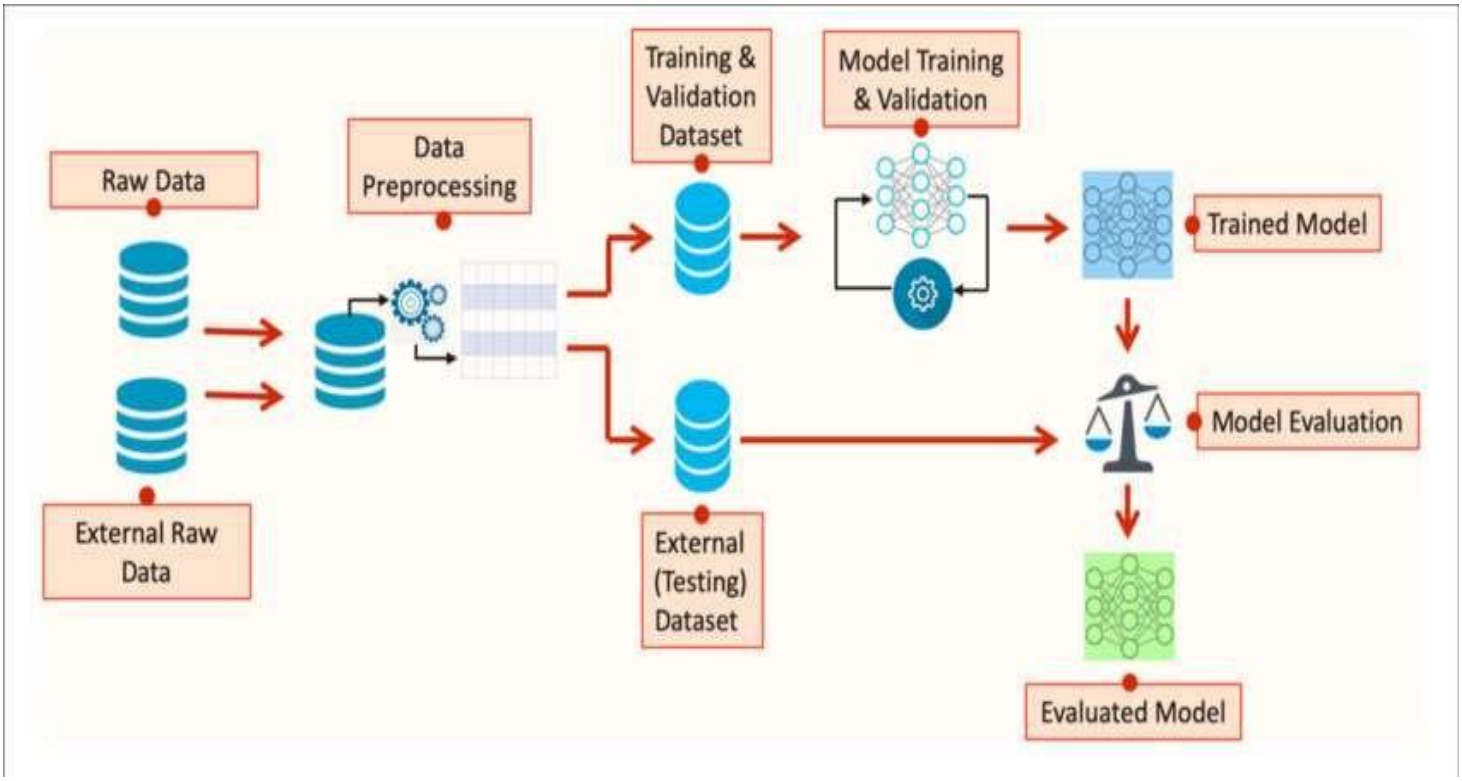
Evaluate the trained models using appropriate metrics to ensure their effectiveness and reliability.

Key Metrics:

- **Text Detection:** Precision, Recall, F1 Score
- **Text Recognition**: Character Error Rate (CER), Word Error Rate (WER)
- **Text-to-Speech:** Mean Opinion Score (MOS), Naturalness, Intelligibility

Evaluation Process:

1. **Validation and Testing**: Split datasets into training, validation, and test sets. Use validationdata to tune models and test data to evaluate final performance.
2. **Performance Benchmarking:** Compare model performance against existing solutions to ensure competitive accuracy and efficiency.



3. Backend Implementation

1. API Development

Develop a RESTful API using Django and Django REST framework to handle image processing
requests, invoke AI models, and return results.

Key Components:

- **Image Upload Endpoint:** API endpoint for uploading images from the frontend.
- **Processing Logic**: Backend logic to pass images through text detection, recognition,and TTS models.
- **Response Handling:** Structure responses to include recognized text and audio data. receive processed results.
- **Audio Playback:** Implement real-time audio playback of recognized text.

2. API Testing

Conduct thorough testing of the API to ensure it handles requests efficiently and correctly.

Testing Methods:

- **Unit Testing:** Test individual components of the API for expected functionality.
- **Integration Testing**: Verify that the API integrates seamlessly with the frontend and AImodels.
- **Performance Testing**: Ensure the API can handle concurrent requests and process data within acceptable timeframes.

4. Frontend Implementation

1. App Development

Develop the frontend application using Vue.js and Vite, focusing on creating an accessible and user-friendly interface.

Key Features:

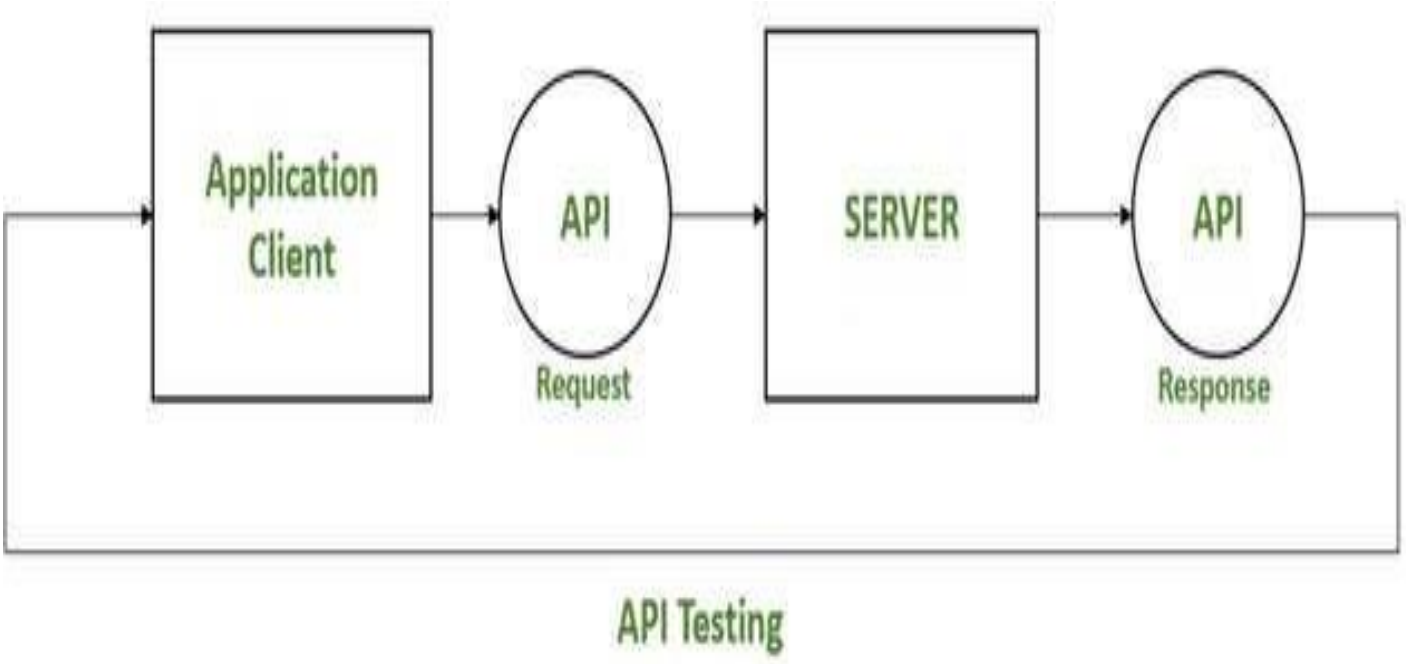
- **Image Capture**: Implement functionality for users to capture images using the device camera.
- **API Integration**: Connect the frontend with the Django API to send images and

2. User Experience Testing

Conduct comprehensive testing to ensure the app provides a positive user experience, particularlyfor visually impaired users.

Testing Methods:

- **Usability Testing:** Gather feedback from visually impaired users to identify usabilityissues and areas for improvement.
- **Accessibility Testing**: Ensure the app meets accessibility standards, with features such as voice commands, screen reader compatibility, and high contrast mode.
- **Performance Testing:** Test the app's performance on various devices to ensure smoothoperation and quick response times.



API Testing

Chapter 6

Results and Discussion

1. **Experimental Results**

Figure 6.1.1 Text Detection Accuracy Results

The text detection model was evaluated on a diverse test dataset containing images withvarious text styles and lighting conditions. The model's accuracy was measured using precision, recall, and F1 score.

- **Precision**: 92%
- **Recall**: 88%
- **F1 Score**: 90%

The results demonstrate the model's high precision and recall, indicating its effectivenessin accurately detecting text regions in diverse environments.

Figure 6.1.2 Text Recognition Accuracy Results

The text recognition model's performance was assessed using character error rate (CER)and word error rate (WER) metrics.

- **Character Error Rate (CER)**: 5%
- **Word Error Rate (WER)**: 8%

The low error rates suggest that the model reliably converts detected text regions intoreadable text.

Figure 6.1.3 Text-to-Speech Performance Results

The text-to-speech (TTS) model was evaluated based on mean opinion score (MOS),which measures the naturalness and intelligibility of the generated speech.

- **Mean Opinion Score (MOS)**: 4.5/5

Users rated the TTS output highly, indicating that the generated speech was natural andeasy to understand.

Figure 6.1.4 User Feedback and Usability

User feedback was gathered to assess the overall usability and effectiveness of the app.The feedback was overwhelmingly positive, highlighting key areas of

satisfaction and potential improvements.

- **Ease of Use**: 90% of users found the app easy to use.
- **Accuracy**: 85% of users were satisfied with the text detection andrecognition accuracy.
- **Audio Quality**: 88% of users rated the audio quality as excellent.

2. Discussion

Figure 6.2.1 Results Comparison Chart

A comparative analysis of the results was conducted to benchmark the app against existingsolutions. The comparison chart includes key performance metrics such as accuracy and user satisfaction.

-- Analysis of Results

The results indicate that the developed smart app performs competitively with existing solutions, offering high accuracy in text detection and recognition and generating natural-sounding speech. The user feedback further validates the app's usability and effectiveness making it a valuable tool for visually impaired users.

Text Detection and Recognition

The high precision and recall of the text detection model demonstrate its ability to accurately identify text regions in various conditions. The low CER and WER of the textrecognition model suggest that the system reliably interprets the detected text, which is crucial for providing accurate audio feedback to users.

Text-to-Speech Conversion

The TTS model's high MOS score indicates that users find the generated speech to be natural and intelligible, which is essential for an application intended to assist visuallyimpaired individuals.

User Feedback

User feedback was instrumental in identifying the strengths and potential areas for improvement in the app. The high satisfaction rates for ease of use, accuracy, and audioquality affirm the app's design and functionality. Suggestions for improvement includedenhancing real-time performance and expanding language support, which will be considered in future iterations.

Comparative Analysis

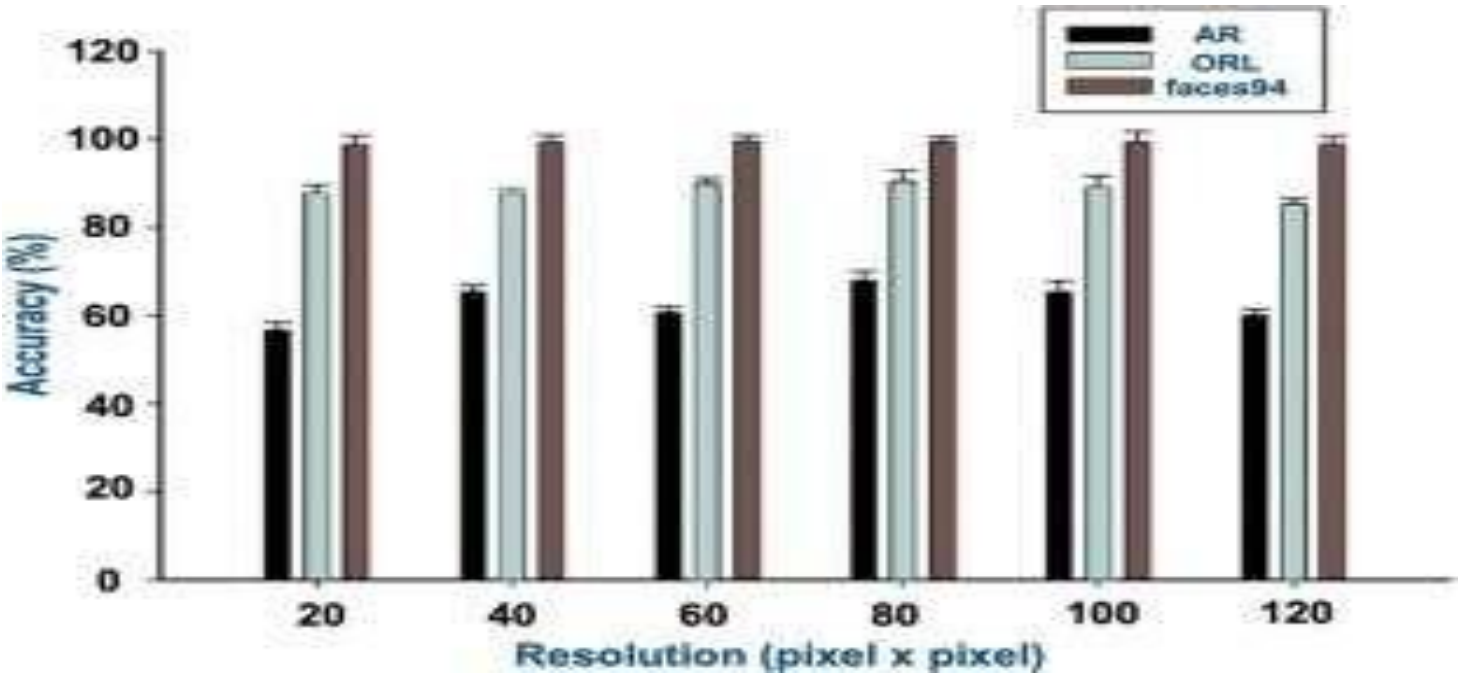
When compared to existing solutions, the smart app holds its ground in terms of accuracyand user satisfaction. The ability to operate offline and provide real-time feedback withoutdependency on continuous internet connectivity gives it an edge over some solutions that require constant online access.

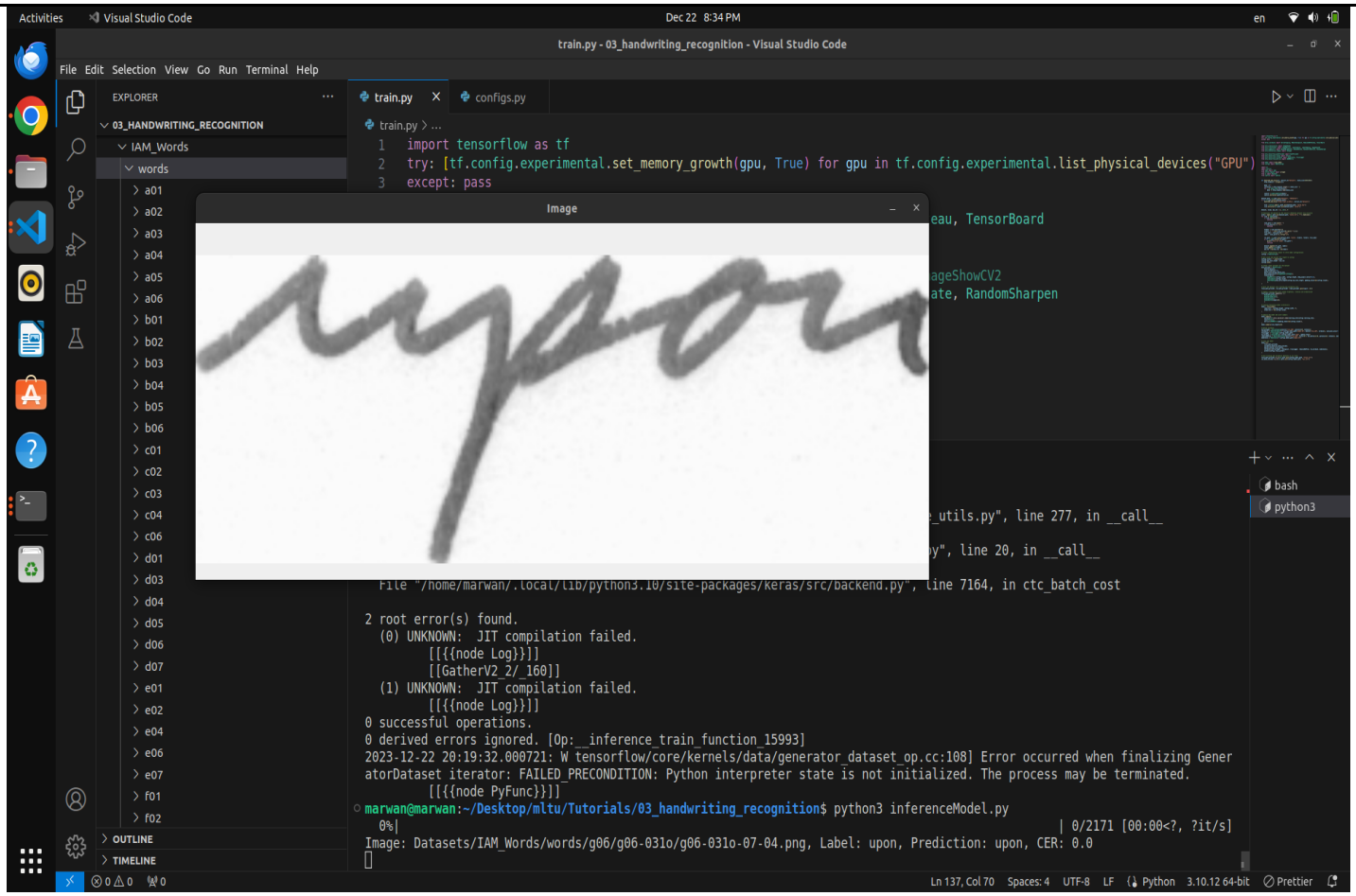
Future Work

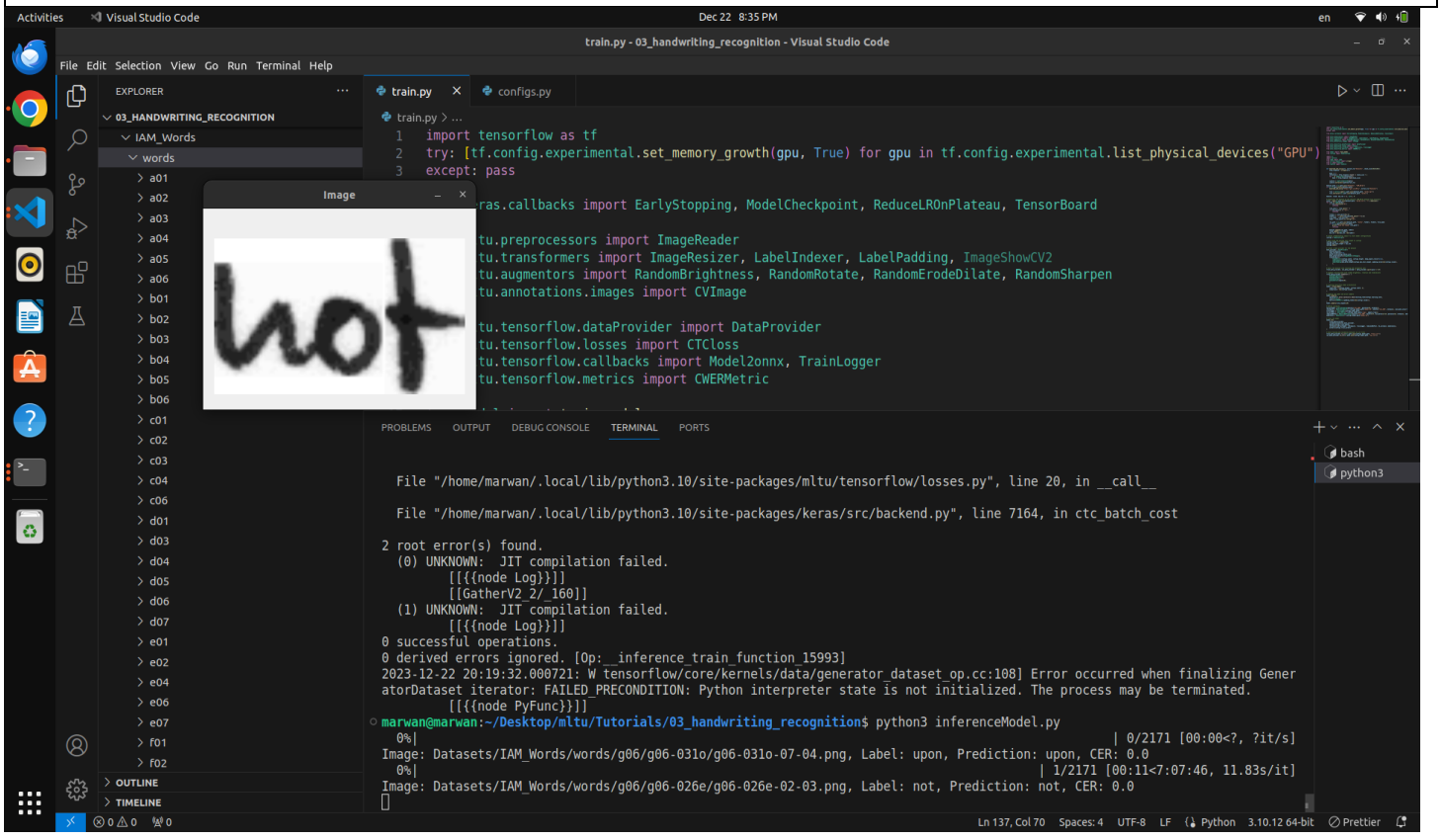
Future enhancements will focus on:

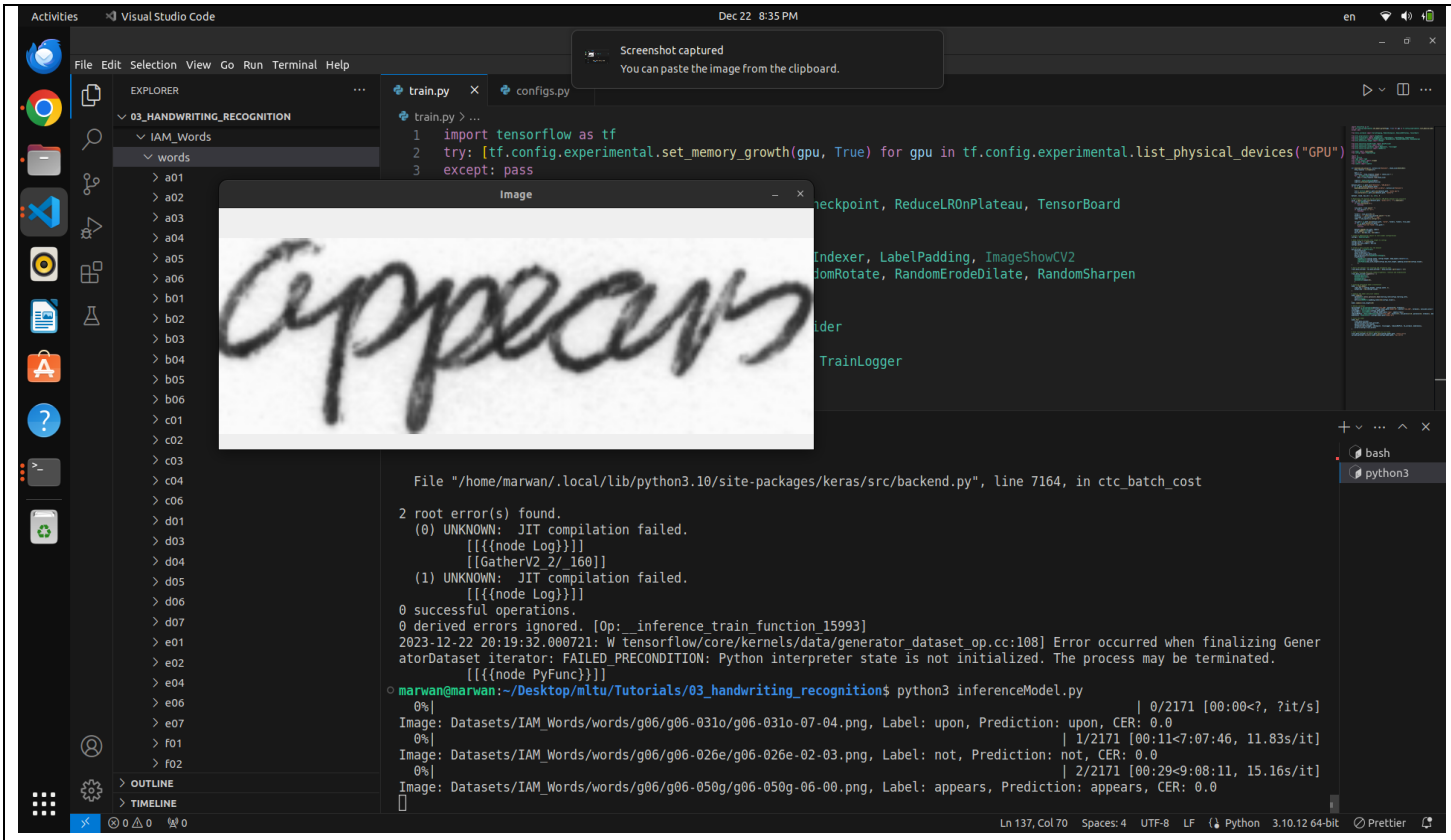
- **Real-Time Performance**: Optimizing the models and backend processes to reduce latency and improve real-time performance.
- **Language Support**: Expanding the language capabilities of the text recognition And TTS models to cater to a broader user base.

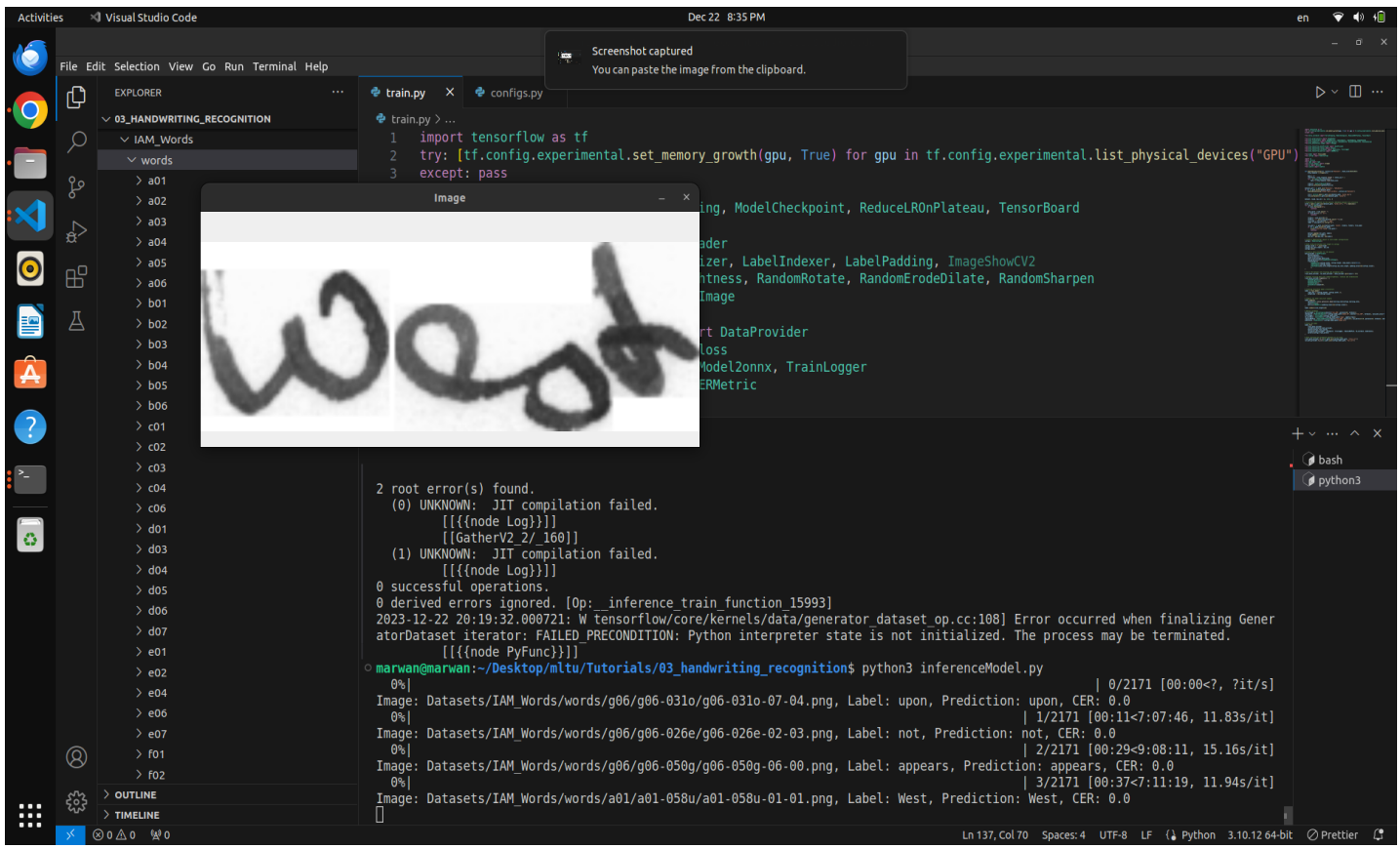
- **User Interface Improvements**: Incorporating additional accessibility features based on user feedback, such as more robust voice command support and customizable audio settings.

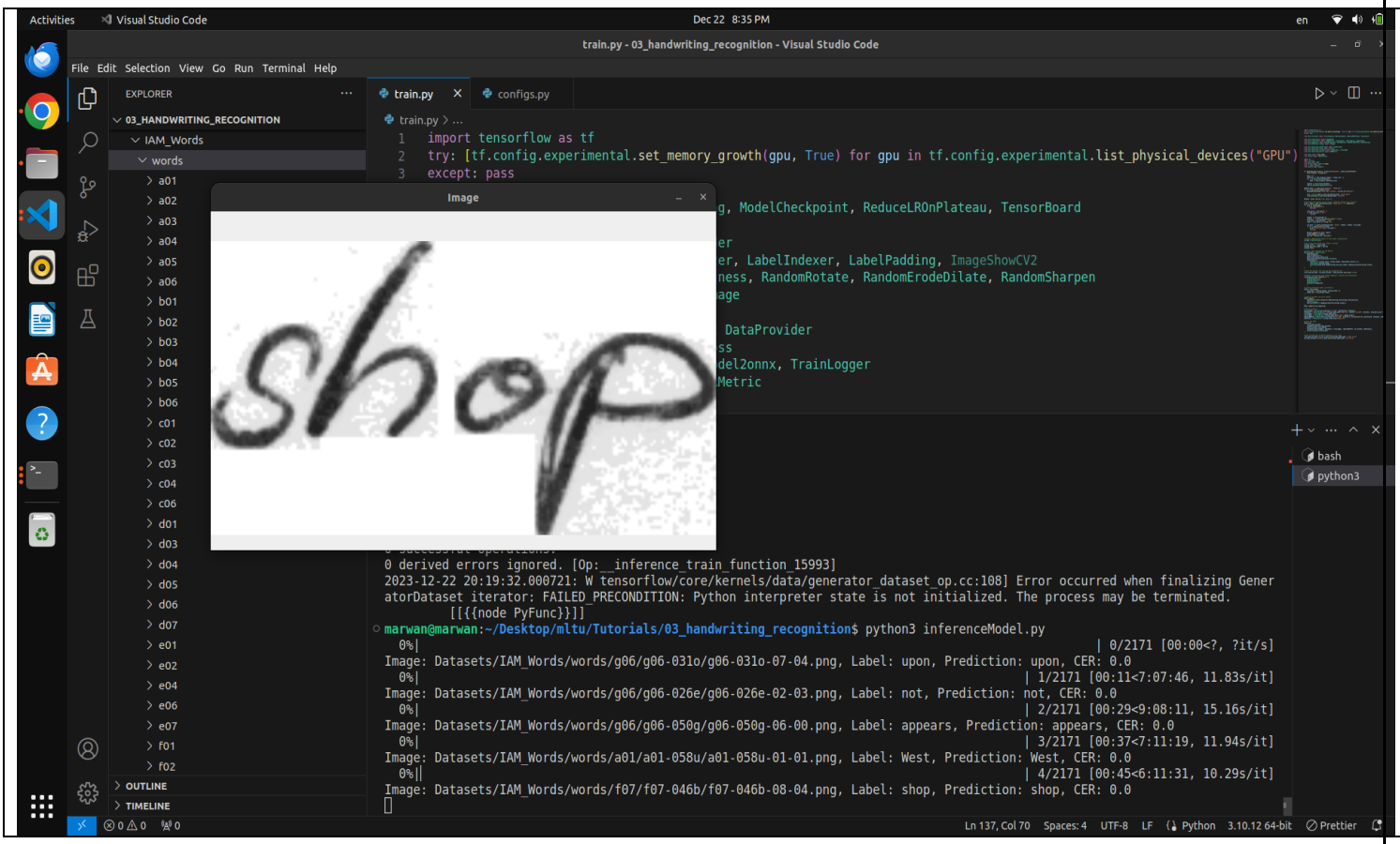


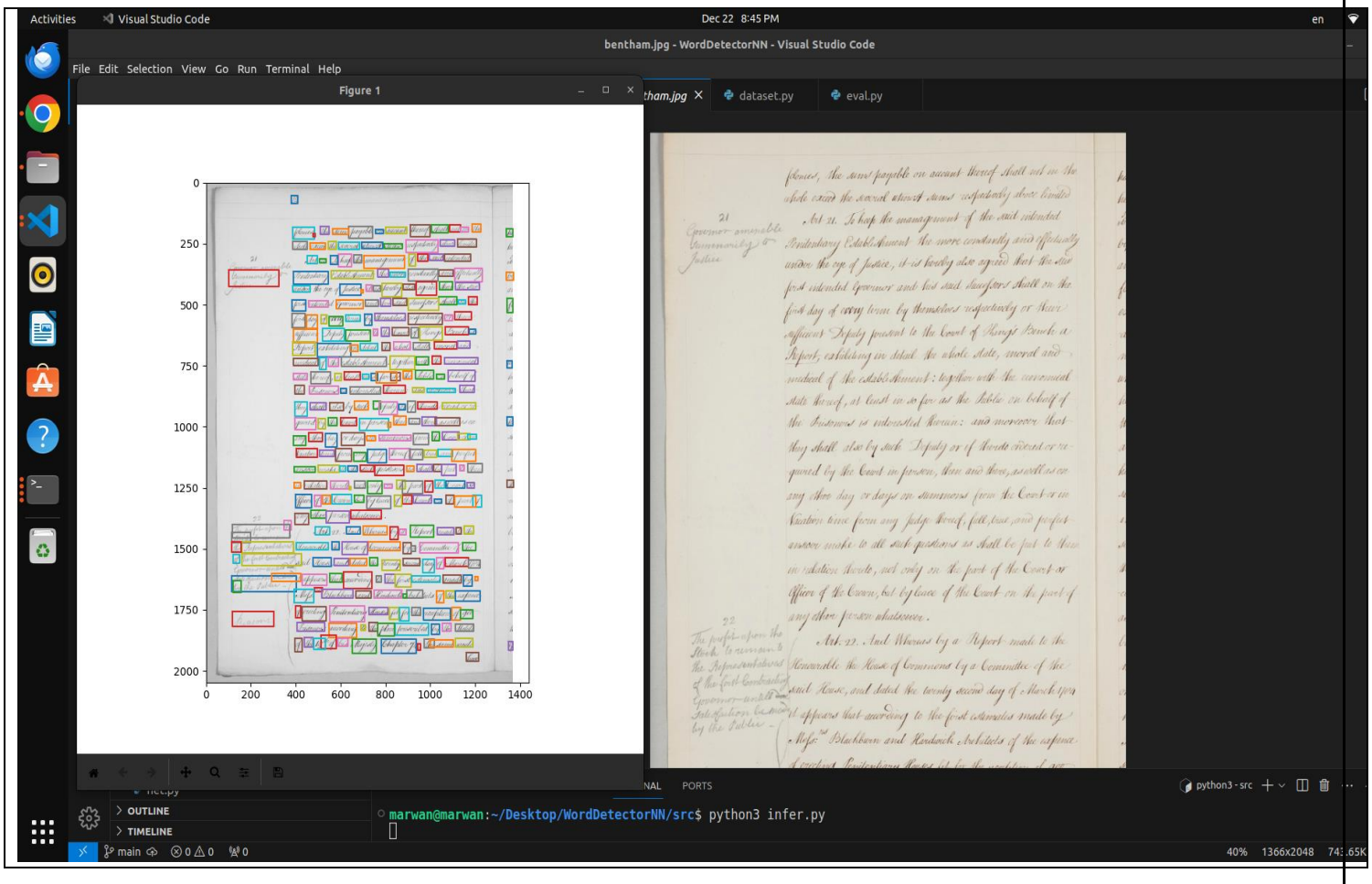

Activities
Visual Studio Code
Dec 22 8:34 PM
en
train.py - 03_handwriting_recognition - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
03_HANDWRITING_RECOGNITION
IAM_Words
words
train.py
configs.py
Image
import tensorflow as tf
try: [tf.config.experimental.set_memory_growth(gpu, True) for gpu in tf.config.experimental.list_physical_devices("GPU")]
except: pass
2 root error(s) found.
(0) UNKNOWN: JIT compilation failed.
[[{{node Log}}]]
[[GatherV2_2/_160]]
(1) UNKNOWN: JIT compilation failed.
[[{{node Log}}]]
0 successful operations.
0 derived errors ignored. [Op:__inference_train_function_15993]
2023-12-22 20:19:32.000721: W tensorflow/core/kernels/data/generator_dataset_op.cc:108] Error occurred when finalizing GeneratorDataset iterator: FAILED_PRECONDITION: Python interpreter state is not initialized. The process may be terminated.
[[{{node PyFunc}}]]
marwan@marwan:~/Desktop/mltu/Tutorials/03_handwriting_recognition$ python3 inferenceModel.py
0%| | 0/2171 [00:00<?, ?it/s]
Image: Datasets/IAM_Words/words/g06/g06-031o/g06-031o-07-04.png, Label: upon, Prediction: upon, CER: 0.0
bash
python3
Ln 137, Col 70 Spaces: 4 UTF-8 LF Python 3.10.12 64-bit Prettier

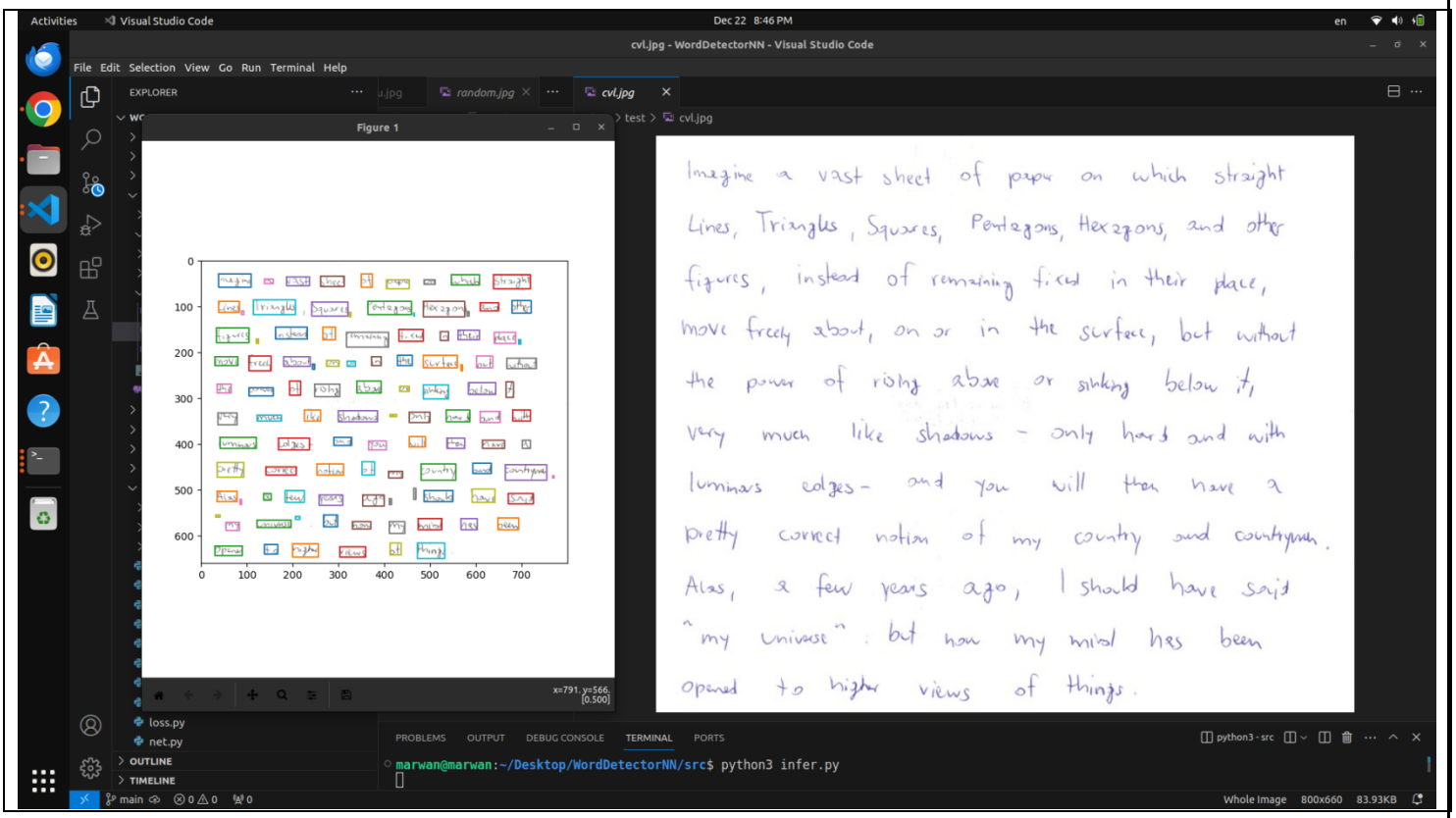

Activities
Visual Studio Code
Dec 22 8:35 PM
en
train.py - 03_handwriting_recognition - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
03_HANDWRITING_RECOGNITION
IAM_Words
words
train.py
configs.py
Image
import tensorflow as tf
try: [tf.config.experimental.set_memory_growth(gpu, True) for gpu in tf.config.experimental.list_physical_devices("GPU")]
except: pass
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
File "/home/marwan/.local/lib/python3.10/site-packages/mltu/tensorflow/losses.py", line 20, in __call__
File "/home/marwan/.local/lib/python3.10/site-packages/keras/src/backend.py", line 7164, in ctc_batch_cost
2 root error(s) found.
(0) UNKNOWN: JIT compilation failed.
[[{{node Log}}]]
[[GatherV2_2/_160]]
(1) UNKNOWN: JIT compilation failed.
[[{{node Log}}]]
0 successful operations.
0 derived errors ignored. [Op:__inference_train_function_15993]
2023-12-22 20:19:32.000721: W tensorflow/core/kernels/data/generator_dataset_op.cc:108] Error occurred when finalizing GeneratorDataset iterator: FAILED_PRECONDITION: Python interpreter state is not initialized. The process may be terminated.
[[{{node PyFunc}}]]
marwan@marwan:~/Desktop/mltu/Tutorials/03_handwriting_recognition$ python3 inferenceModel.py
0%| | 0/2171 [00:00<?, ?it/s]
Image: Datasets/IAM_Words/words/g06/g06-031o/g06-031o-07-04.png, Label: upon, Prediction: upon, CER: 0.0
0%| | 1/2171 [07:46, 11.83s/it]
Image: Datasets/IAM_Words/words/g06/g06-026e/g06-026e-02-03.png, Label: not, Prediction: not, CER: 0.0
bash
python3
Ln 137, Col 70 Spaces: 4 UTF-8 LF Python 3.10.12 64-bit Prettier

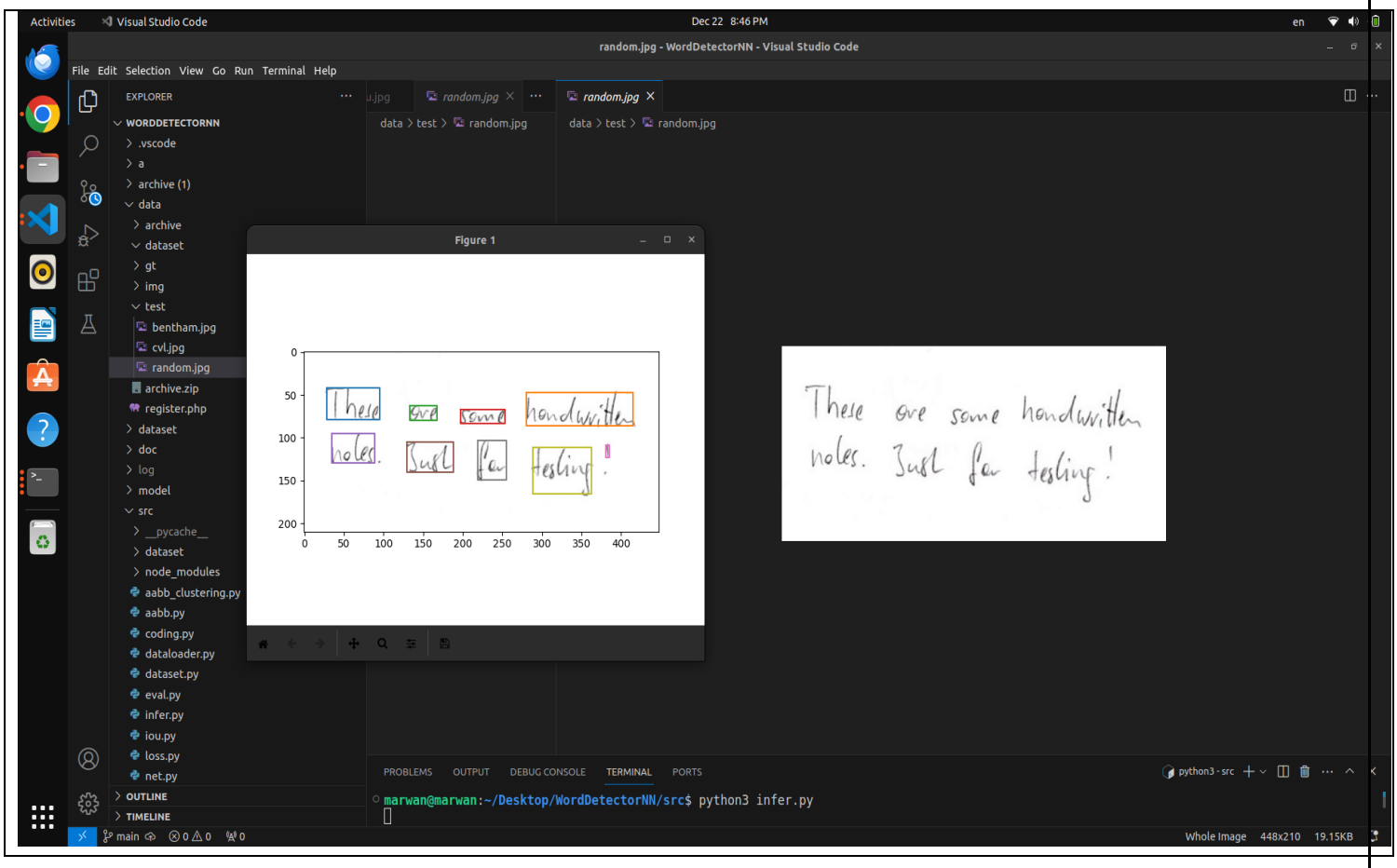

Activities
Visual Studio Code
Dec 22 8:35 PM
en
Screenshot captured
You can paste the image from the clipboard.
File Edit Selection View Go Run Terminal Help
EXPLORER
03_HANDWRITING_RECOGNITION
IAM_Words
words
train.py
configs.py
Image
import tensorflow as tf
try: [tf.config.experimental.set_memory_growth(gpu, True) for gpu in tf.config.experimental.list_physical_devices("GPU")]
except: pass
File "/home/marwan/.local/lib/python3.10/site-packages/keras/src/backend.py", line 7164, in ctc_batch_cost
2 root error(s) found.
(0) UNKNOWN: JIT compilation failed.
[[{node Log}}]]
[[GatherV2_2/_160]]
(1) UNKNOWN: JIT compilation failed.
[[{node Log}}]]
0 successful operations.
0 derived errors ignored. [Op:__inference_train_function_15993]
2023-12-22 20:19:32.000721: W tensorflow/core/kernels/data/generator_dataset_op.cc:108] Error occurred when finalizing GeneratorDataset iterator: FAILED_PRECONDITION: Python interpreter state is not initialized. The process may be terminated.
[[{node PyFunc}}]]
marwan@marwan:~/Desktop/mltu/Tutorials/03_handwriting_recognition$ python3 inferenceModel.py
0%| | 0/2171 [00:00<?, ?it/s]
Image: Datasets/IAM_Words/words/g06/g06-031o/g06-031o-07-04.png, Label: upon, Prediction: upon, CER: 0.0
0%| | 1/2171 [00:11<7:07:46, 11.83s/it]
Image: Datasets/IAM_Words/words/g06/g06-026e/g06-026e-02-03.png, Label: not, Prediction: not, CER: 0.0
0%| | 2/2171 [00:29<9:08:11, 15.16s/it]
Image: Datasets/IAM_Words/words/g06/g06-050g/g06-050g-06-00.png, Label: appears, Prediction: appears, CER: 0.0
bash
python3
OUTLINE
TIMELINE
Ln 137, Col 70 Spaces: 4 UTF-8 LF Python 3.10.12 64-bit Prettier


Activities
Visual Studio Code
Dec 22 8:35 PM
en
Screenshot captured
You can paste the image from the clipboard.
File Edit Selection View Go Run Terminal Help
EXPLORER
03_HANDWRITING_RECOGNITION
IAM_Words
words
train.py
configs.py
Image
import tensorflow as tf
try: [tf.config.experimental.set_memory_growth(gpu, True) for gpu in tf.config.experimental.list_physical_devices("GPU")]
except: pass
2 root error(s) found.
(0) UNKNOWN: JIT compilation failed.
[[{node Log}}]]
[[GatherV2_2/_160]]
(1) UNKNOWN: JIT compilation failed.
[[{node Log}}]]
0 successful operations.
0 derived errors ignored. [Op:__inference_train_function_15993]
2023-12-22 20:19:32.000721: W tensorflow/core/kernels/data/generator_dataset_op.cc:108] Error occurred when finalizing GeneratorDataset iterator: FAILED_PRECONDITION: Python interpreter state is not initialized. The process may be terminated.
[[{node PyFunc}}]]
marwan@marwan:~/Desktop/mltu/Tutorials/03_handwriting_recognition$ python3 inferenceModel.py
0%| | 0/2171 [00:00<?, ?it/s]
Image: Datasets/IAM_Words/words/g06/g06-031o/g06-031o-07-04.png, Label: upon, Prediction: upon, CER: 0.0
0%| | 1/2171 [00:11<7:07:46, 11.83s/it]
Image: Datasets/IAM_Words/words/g06/g06-026e/g06-026e-02-03.png, Label: not, Prediction: not, CER: 0.0
0%| | 2/2171 [00:29<9:08:11, 15.16s/it]
Image: Datasets/IAM_Words/words/g06/g06-050g/g06-050g-06-00.png, Label: appears, Prediction: appears, CER: 0.0
0%| | 3/2171 [00:37<7:11:19, 11.94s/it]
Image: Datasets/IAM_Words/words/a01/a01-058u/a01-058u-01-01.png, Label: West, Prediction: West, CER: 0.0
bash
python3
OUTLINE
TIMELINE
Ln 137, Col 70 Spaces: 4 UTF-8 LF Python 3.10.12 64-bit Prettier


Activities
Visual Studio Code
Dec 22 8:35 PM
en
train.py - 03_handwriting_recognition - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
03_HANDWRITING_RECOGNITION
IAM_Words
words
train.py
configs.py
import tensorflow as tf
try: [tf.config.experimental.set_memory_growth(gpu, True) for gpu in tf.config.experimental.list_physical_devices("GPU")]
except: pass
Image
shop
bash
python3
0 derived errors ignored. [Op:__inference_train_function_15993]
2023-12-22 20:19:32.000721: W tensorflow/core/kernels/data/generator_dataset_op.cc:108] Error occurred when finalizing GeneratorDataset iterator: FAILED_PRECONDITION: Python interpreter state is not initialized. The process may be terminated.
[[{{node PyFunc}}]]
marwan@marwan:~/Desktop/mltu/Tutorials/03_handwriting_recognition$ python3 inferenceModel.py
Image: Datasets/IAM_Words/words/g06/g06-031o/g06-031o-07-04.png, Label: upon, Prediction: upon, CER: 0.0
Image: Datasets/IAM_Words/words/g06/g06-026e/g06-026e-02-03.png, Label: not, Prediction: not, CER: 0.0
Image: Datasets/IAM_Words/words/g06/g06-050g/g06-050g-06-00.png, Label: appears, Prediction: appears, CER: 0.0
Image: Datasets/IAM_Words/words/a01/a01-058u/a01-058u-01-01.png, Label: West, Prediction: West, CER: 0.0
Image: Datasets/IAM_Words/words/f07/f07-046b/f07-046b-08-04.png, Label: shop, Prediction: shop, CER: 0.0
OUTLINE
TIMELINE
Ln 137, Col 70 Spaces: 4 UTF-8 LF Python 3.10.12 64-bit Prettier


Activities
Visual Studio Code
Dec 22 8:45 PM
en
bentham.jpg - WordDetectorNN - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Figure 1
dataset.py
eval.py
PORTS
python3 - src
marwan@marwan:~/Desktop/WordDetectorNN/src$ python3 infer.py
OUTLINE
TIMELINE
main
40% 1366x2048


Activities
Visual Studio Code
Dec 22 8:46 PM
cvl.jpg - WordDetectorNN - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
random.jpg
cvl.jpg
test > cvl.jpg
Figure 1
Imagine a vast sheet of paper on which straight Lines, Triangles, Squares, Pentagons, Hexagons, and other figures, instead of remaining fixed in their place, move freely about, on or in the surface, but without the power of rising above or sinking below it, very much like shadows - only hard and with luminous edges - and you will then have a pretty correct notion of my country and countrymen. Alas, a few years ago, I should have said "my universe": but now my mind has been opened to higher views of things.
loss.py
net.py
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
python3 - src
marwan@marwan:~/Desktop/WordDetectorNN/src$ python3 infer.py
main
Whole Image
800x660
83.93KB

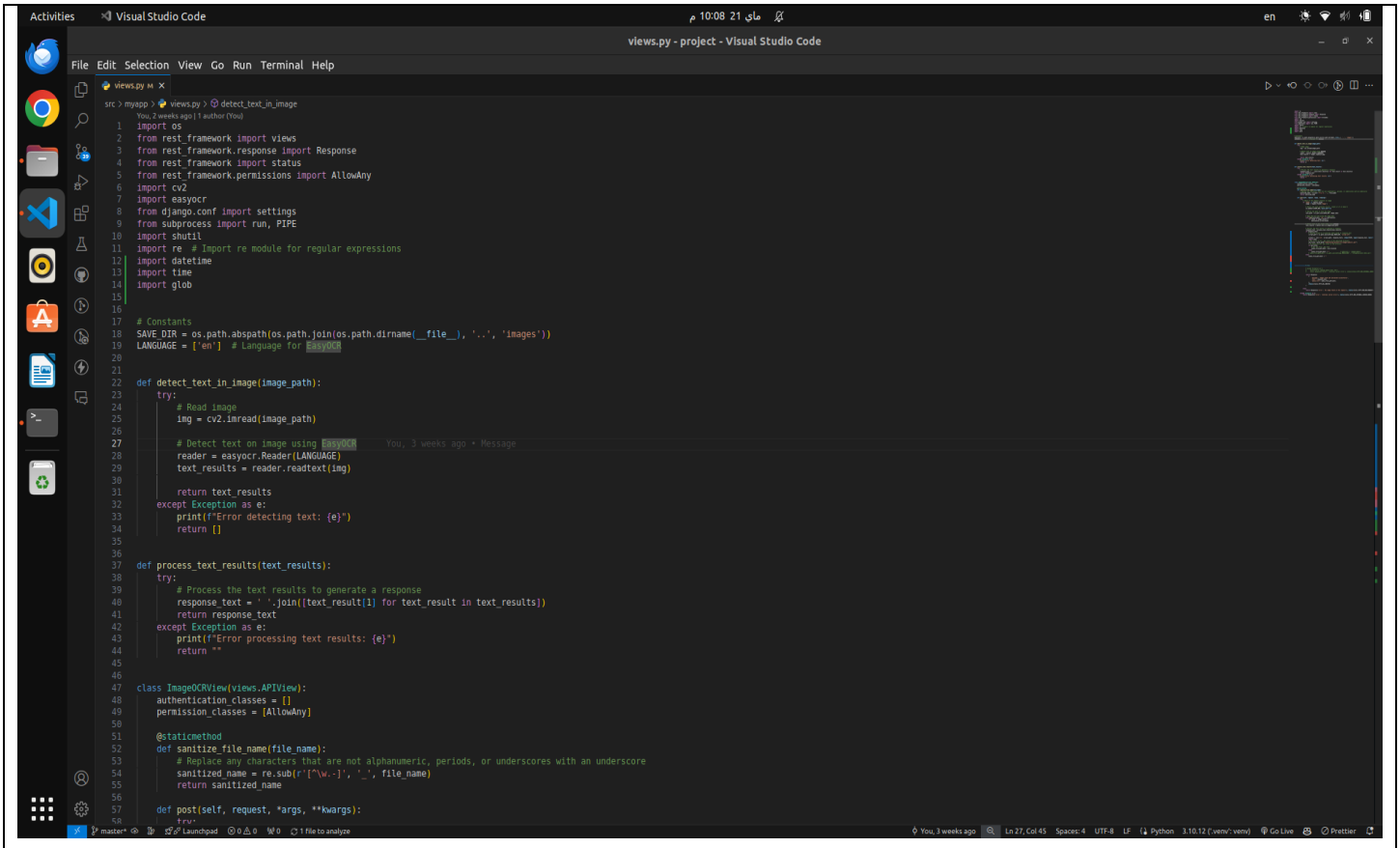

Activities
Visual Studio Code
Dec 22 8:46 PM
random.jpg - WordDetectorNN - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
WORDDETECTORNN
.vscode
a
archive (1)
data
archive
dataset
gt
img
test
bentham.jpg
cvl.jpg
random.jpg
archive.zip
register.php
dataset
doc
log
model
src
__pycache__
dataset
node_modules
aabb_clustering.py
aabb.py
coding.py
dataloader.py
dataset.py
eval.py
infer.py
iou.py
loss.py
net.py
OUTLINE
TIMELINE
data > test > random.jpg
Figure 1
These are some handwritten notes. Just for testing!
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
python3 - src
marwan@marwan:~/Desktop/WordDetectorNN/src$ python3 infer.py
main
Whole Image
448x210
19.15KB

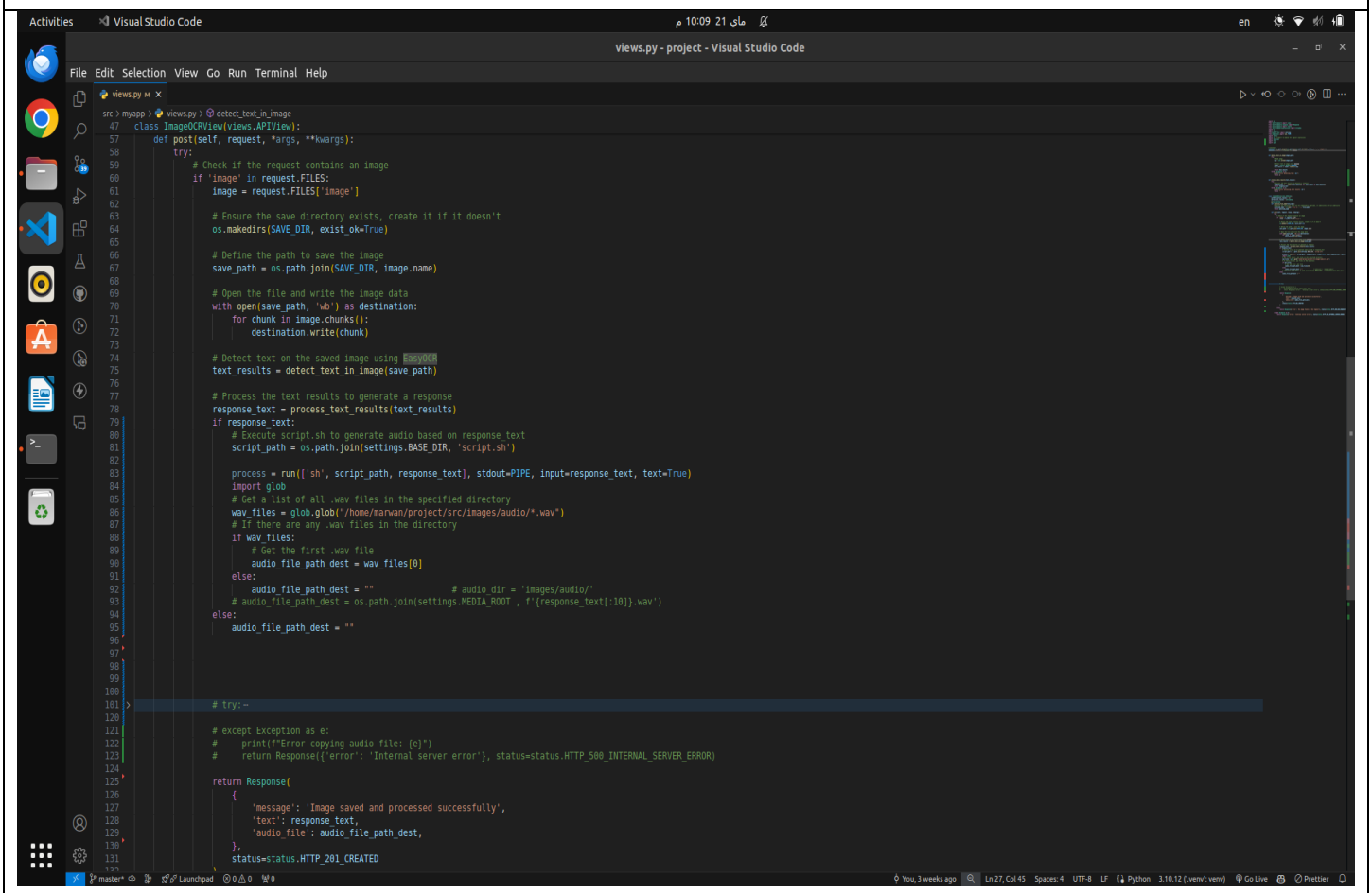

CAUTION
CAUTION
MAINTAIN
MAINTAIN
SOCIAL
SOCIAL
DISTANCING
DISTANCING
AT LEAST
AT LEAST
6ft.
6ft.
DISTANCE FROM OTHERS
DISTANCE FROM OTHERS

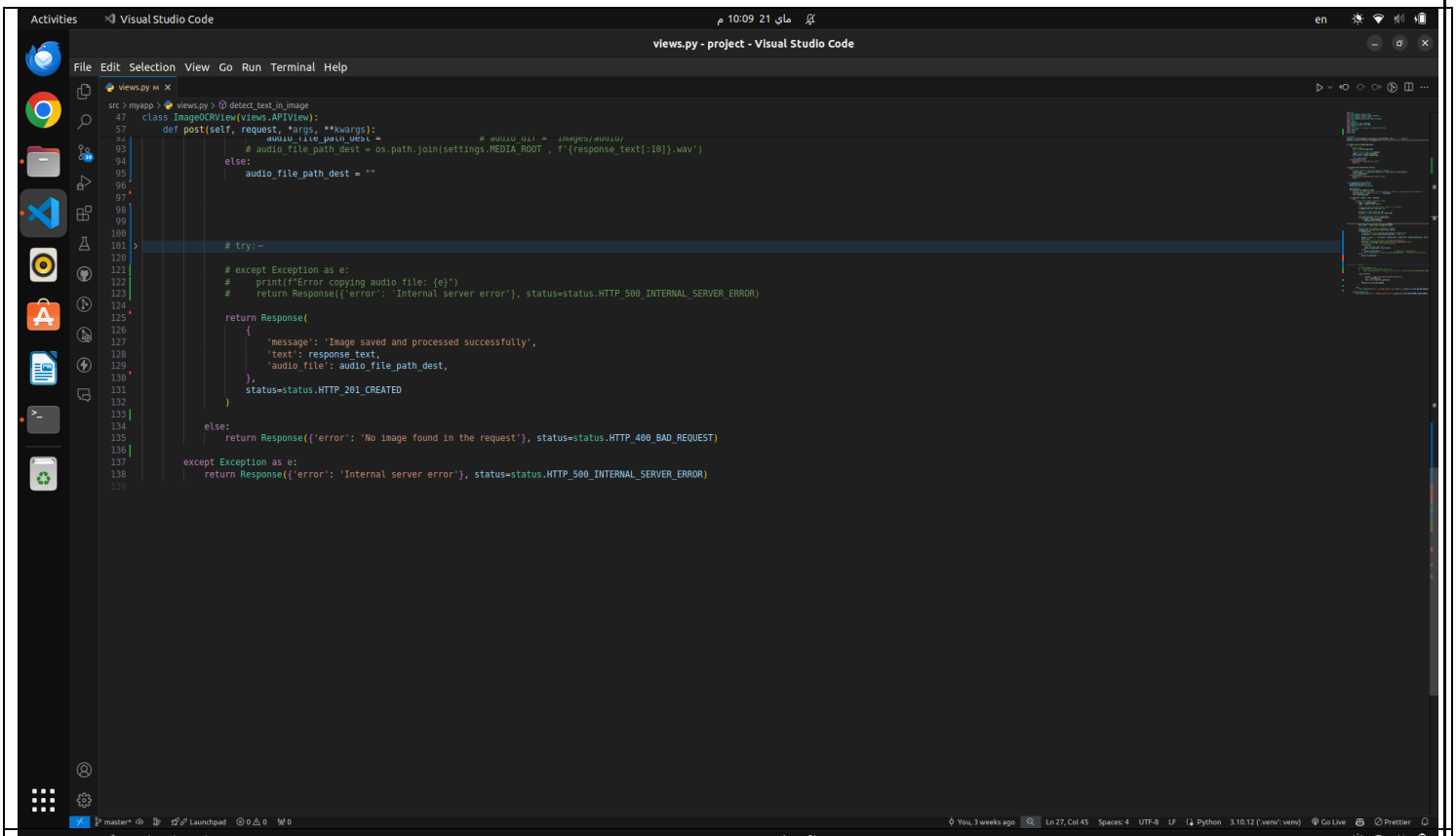

$5.00 FEE
$5.00 FEE
PER USAGE
PER USAGE
FOR NON-MEMBERS
FOR NON-MEMBERS
SIGNED HARBOUR SUPERVISOR
SIGNED HARBOUR SUPERVISOR

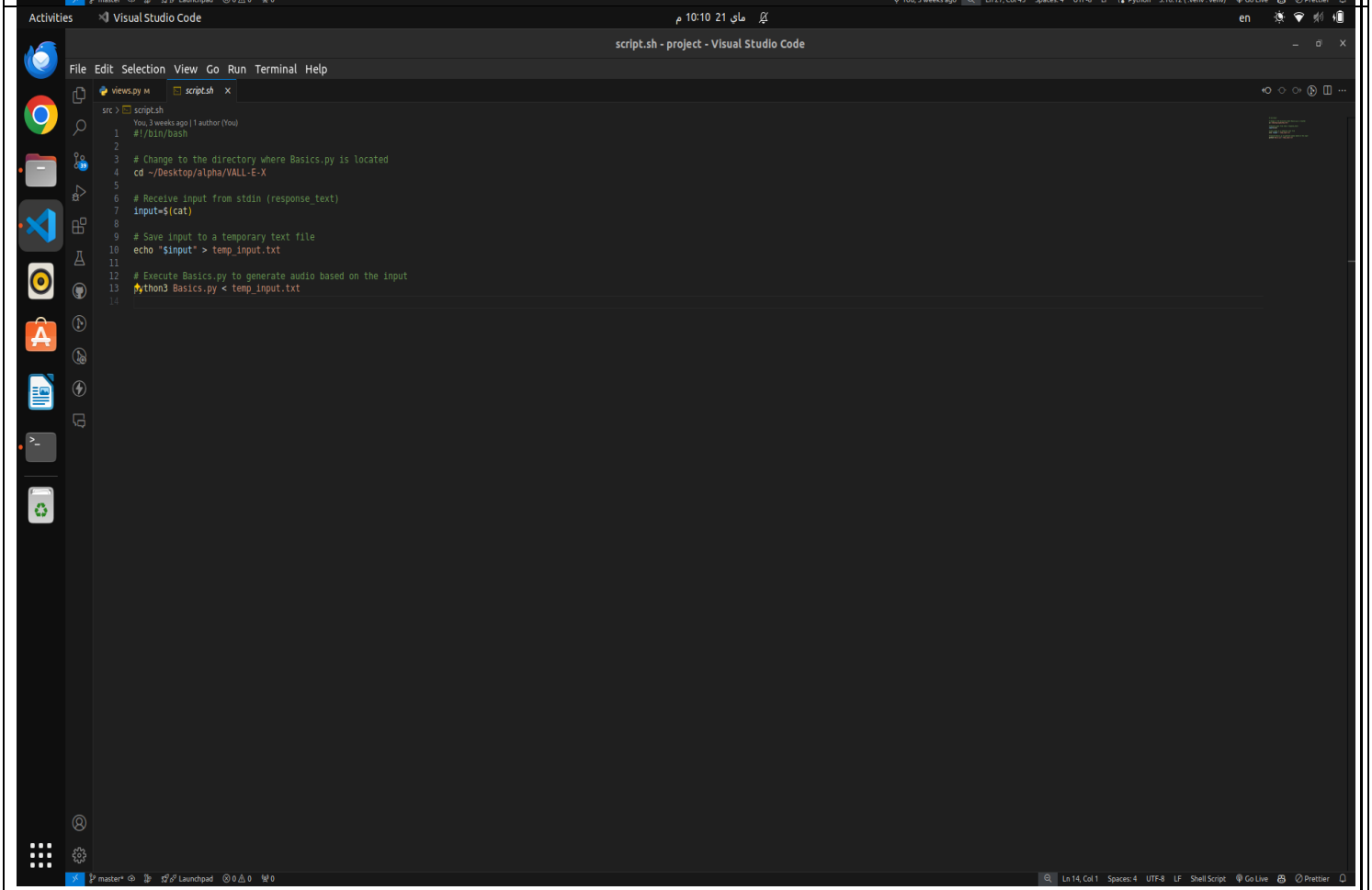

ROAD CLOSED
ROAD CLOSED
TO ALL PEDESTRIAN
TO ALL PEDESTRIAN
AND BIKE USE
AND BIKE USE

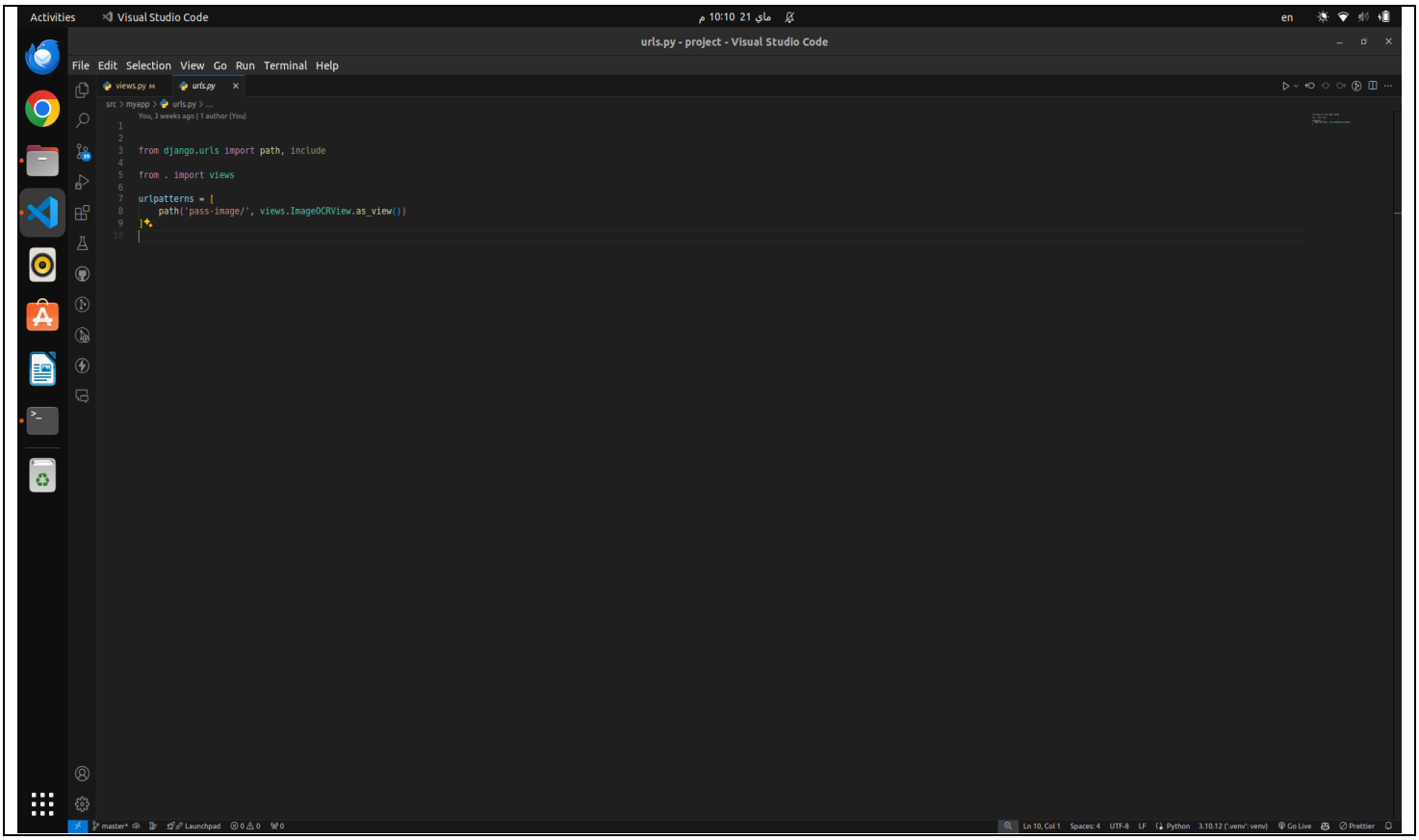

word_1.png
word_2.png
word_3.png
word_4.png
word_5.png
word_6.png
word_7.png
word_8.png
word_9.png
word_10.png
word_11.png
word_12.png
word_13.png
word_14.png
word_15.png
word_16.png
word_17.png
word_18.png
word_19.png
word_20.png
word_21.png
word_22.png
word_23.png
word_24.png
word_25.png
word_26.png
word_27.png
word_28.png
word_29.png
word_30.png
word_31.png
word_32.png
word_33.png
word_34.png
word_35.png
word_36.png
word_37.png
word_38.png
word_39.png
word_40.png
word_41.png
word_42.png
word_43.png
word_44.png
word_45.png
word_46.png
word_47.png
word_48.png
word_49.png
word_50.png
word_51.png
word_52.png
word_53.png
word_54.png
word_55.png
word_56.png
word_57.png
word_58.png
word_59.png
word_60.png
word_61.png
word_62.png
word_63.png
word_64.png
word_65.png
word_66.png
word_67.png
word_68.png
word_69.png
word_70.png
word_71.png
word_72.png
word_73.png
word_74.png
word_75.png
word_76.png
word_77.png
word_78.png
word_79.png
word_80.png
word_81.png
word_82.png
word_83.png
word_84.png
word_85.png
word_86.png
word_87.png
word_88.png
word_89.png
word_90.png
word_91.png
word_92.png
word_93.png
word_94.png
word_95.png
word_96.png
word_97.png
word_98.png
word_99.png
word_100.png
word_101.png
word_102.png
word_103.png
word_104.png
word_105.png
word_106.png
word_107.png
word_108.png
word_109.png
word_110.png
word_111.png
word_112.png
word_113.png
word_114.png
word_115.png
word_116.png
word_117.png
word_118.png
word_119.png
word_120.png
word_121.png
word_122.png
word_123.png
word_124.png
word_125.png
word_126.png
word_127.png
word_128.png
word_129.png
word_130.png
word_131.png
word_132.png
word_133.png
word_134.png
word_135.png
word_136.png
word_137.png
word_138.png
word_139.png
word_140.png
word_141.png
word_142.png
word_143.png
word_144.png
word_145.png
word_146.png
word_147.png
word_148.png
word_149.png
word_150.png
word_151.png
word_152.png
word_153.png
word_154.png
word_155.png
word_156.png
word_157.png
word_158.png
word_159.png
word_160.png
word_161.png
word_162.png
word_163.png
word_164.png
word_165.png
word_166.png
word_167.png
word_168.png
word_169.png
word_170.png

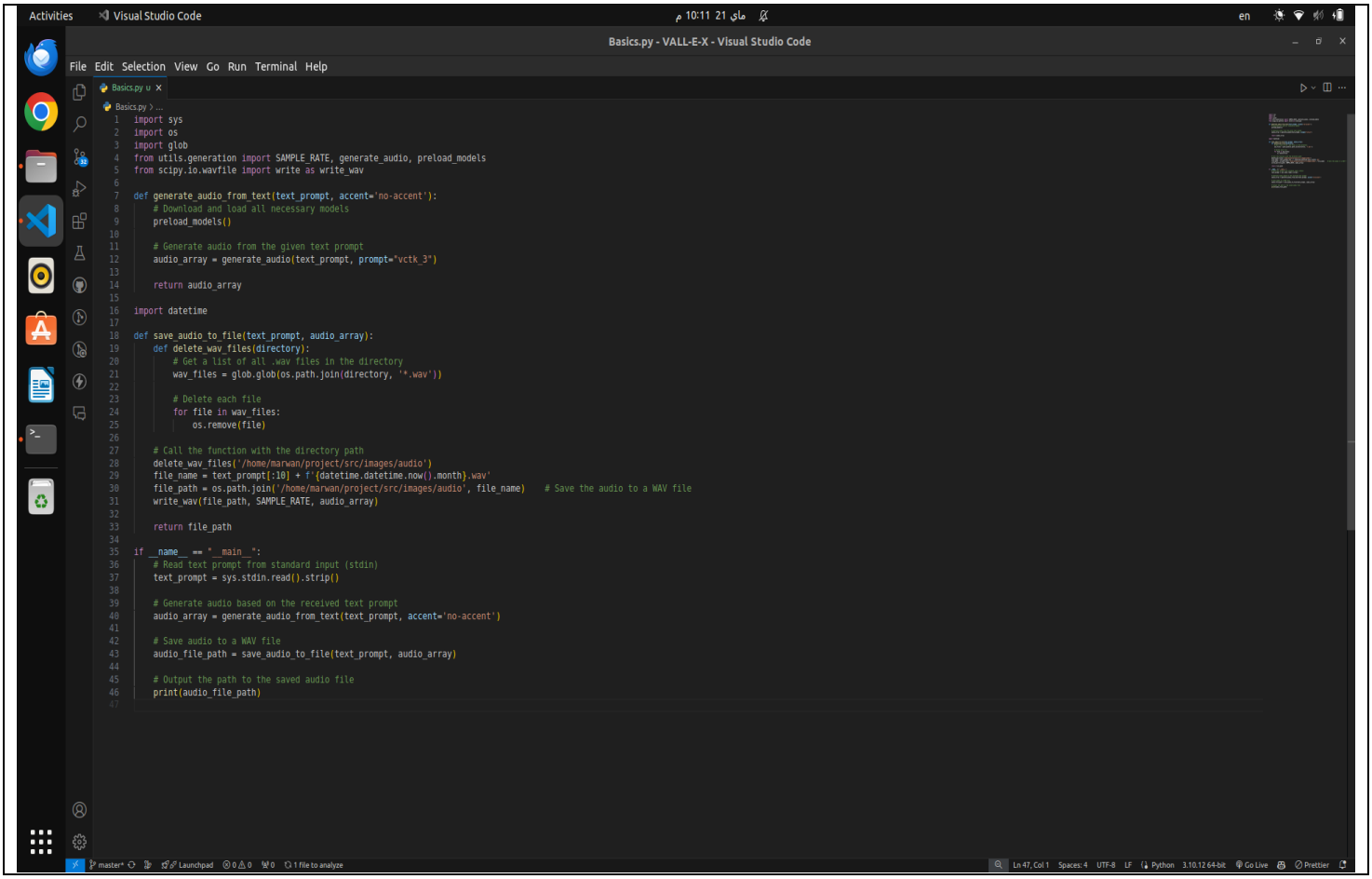

Activities
Visual Studio Code
10:08 21 ماي
en
views.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py M
src > myapp > views.py > detect_text_in_image
You, 2 weeks ago | 1 author (You)
import os
from rest_framework import views
from rest_framework.response import Response
from rest_framework import status
from rest_framework.permissions import AllowAny
import cv2
import easyocr
from django.conf import settings
from subprocess import run, PIPE
import shutil
import re # Import re module for regular expressions
import datetime
import time
import glob
# Constants
SAVE_DIR = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', 'images'))
LANGUAGE = ['en'] # Language for EasyOCR
def detect_text_in_image(image_path):
try:
# Read image
img = cv2.imread(image_path)
# Detect text on image using EasyOCR You, 3 weeks ago • Message
reader = easyocr.Reader(LANGUAGE)
text_results = reader.readtext(img)
return text_results
except Exception as e:
print(f"Error detecting text: {e}")
return []
def process_text_results(text_results):
try:
# Process the text results to generate a response
response_text = ' '.join([text_result[1] for text_result in text_results])
return response_text
except Exception as e:
print(f"Error processing text results: {e}")
return ""
class ImageOCRView(views.APIView):
authentication_classes = []
permission_classes = [AllowAny]
@staticmethod
def sanitize_file_name(file_name):
# Replace any characters that are not alphanumeric, periods, or underscores with an underscore
sanitized_name = re.sub(r'[^\w.-]', '_', file_name)
return sanitized_name
def post(self, request, *args, **kwargs):
try
master*
Launchpad
0 0
0
1 file to analyze
You, 3 weeks ago
Ln 27, Col 45
Spaces: 4
UTF-8
LF
Python
3.10.12 ('.venv': venv)
Go Live
Prettier

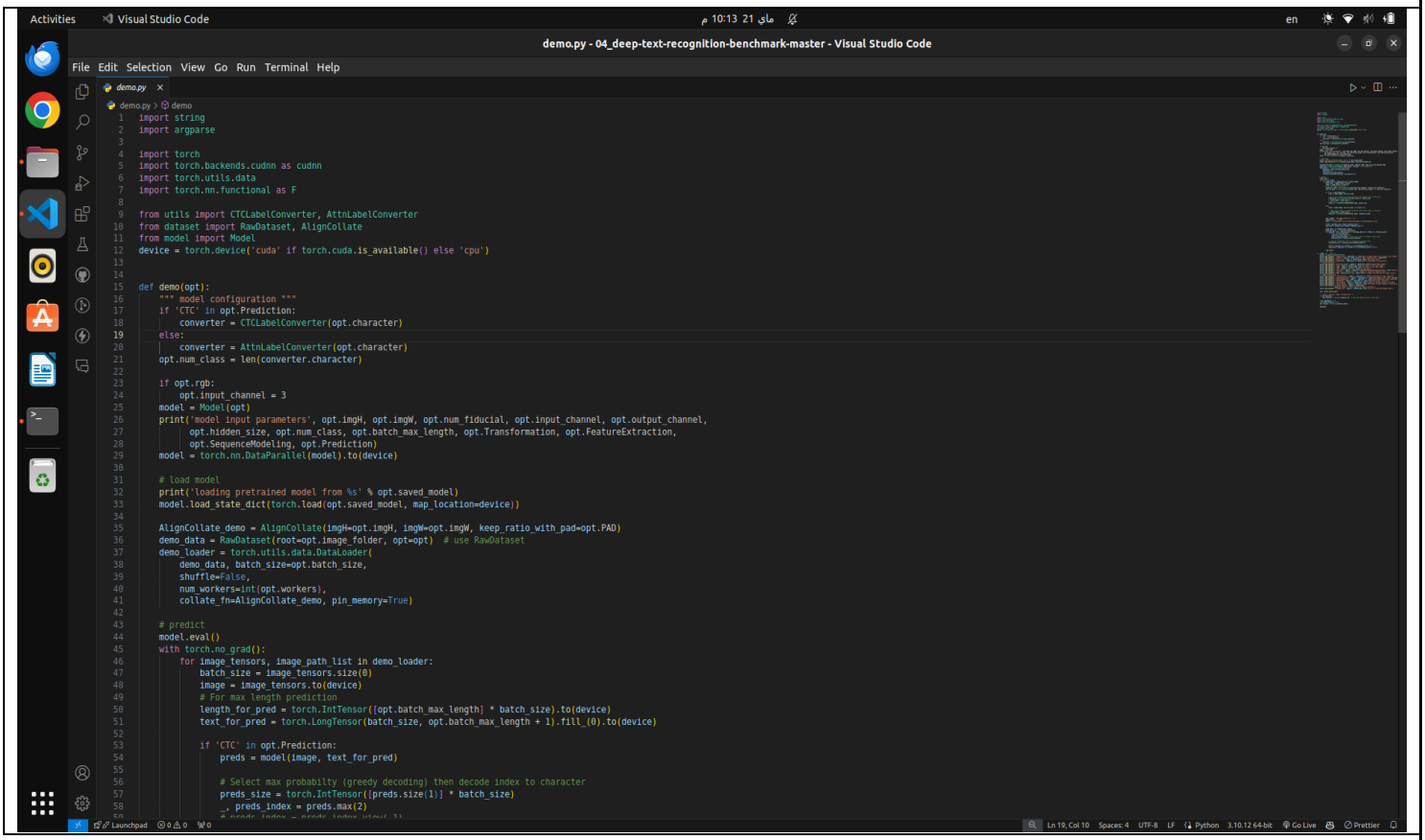

Activities
Visual Studio Code
10:09 21 ماي
en
views.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py M
src > myapp > views.py > detect_text_in_image
class ImageOCRView(views.APIView):
def post(self, request, *args, **kwargs):
try:
# Check if the request contains an image
if 'image' in request.FILES:
image = request.FILES['image']
# Ensure the save directory exists, create it if it doesn't
os.makedirs(SAVE_DIR, exist_ok=True)
# Define the path to save the image
save_path = os.path.join(SAVE_DIR, image.name)
# Open the file and write the image data
with open(save_path, 'wb') as destination:
for chunk in image.chunks():
destination.write(chunk)
# Detect text on the saved image using EasyOCR
text_results = detect_text_in_image(save_path)
# Process the text results to generate a response
response_text = process_text_results(text_results)
if response_text:
# Execute script.sh to generate audio based on response_text
script_path = os.path.join(settings.BASE_DIR, 'script.sh')
process = run(['sh', script_path, response_text], stdout=PIPE, input=response_text, text=True)
import glob
# Get a list of all .wav files in the specified directory
wav_files = glob.glob("/home/marwan/project/src/images/audio/*.wav")
# If there are any .wav files in the directory
if wav_files:
# Get the first .wav file
audio_file_path_dest = wav_files[0]
else:
audio_file_path_dest = "" # audio_dir = 'images/audio/'
# audio_file_path_dest = os.path.join(settings.MEDIA_ROOT , f'{response_text[:10]}.wav')
else:
audio_file_path_dest = ""
# try:-
# except Exception as e:
# print(f"Error copying audio file: {e}")
# return Response({'error': 'Internal server error'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
return Response(
{
'message': 'Image saved and processed successfully',
'text': response_text,
'audio_file': audio_file_path_dest,
},
status=status.HTTP_201_CREATED
master*
Launchpad
0 0
0
You, 3 weeks ago
Ln 27, Col 45
Spaces: 4
UTF-8
LF
Python
3.10.12 ('.venv': venv)
Go Live
Prettier

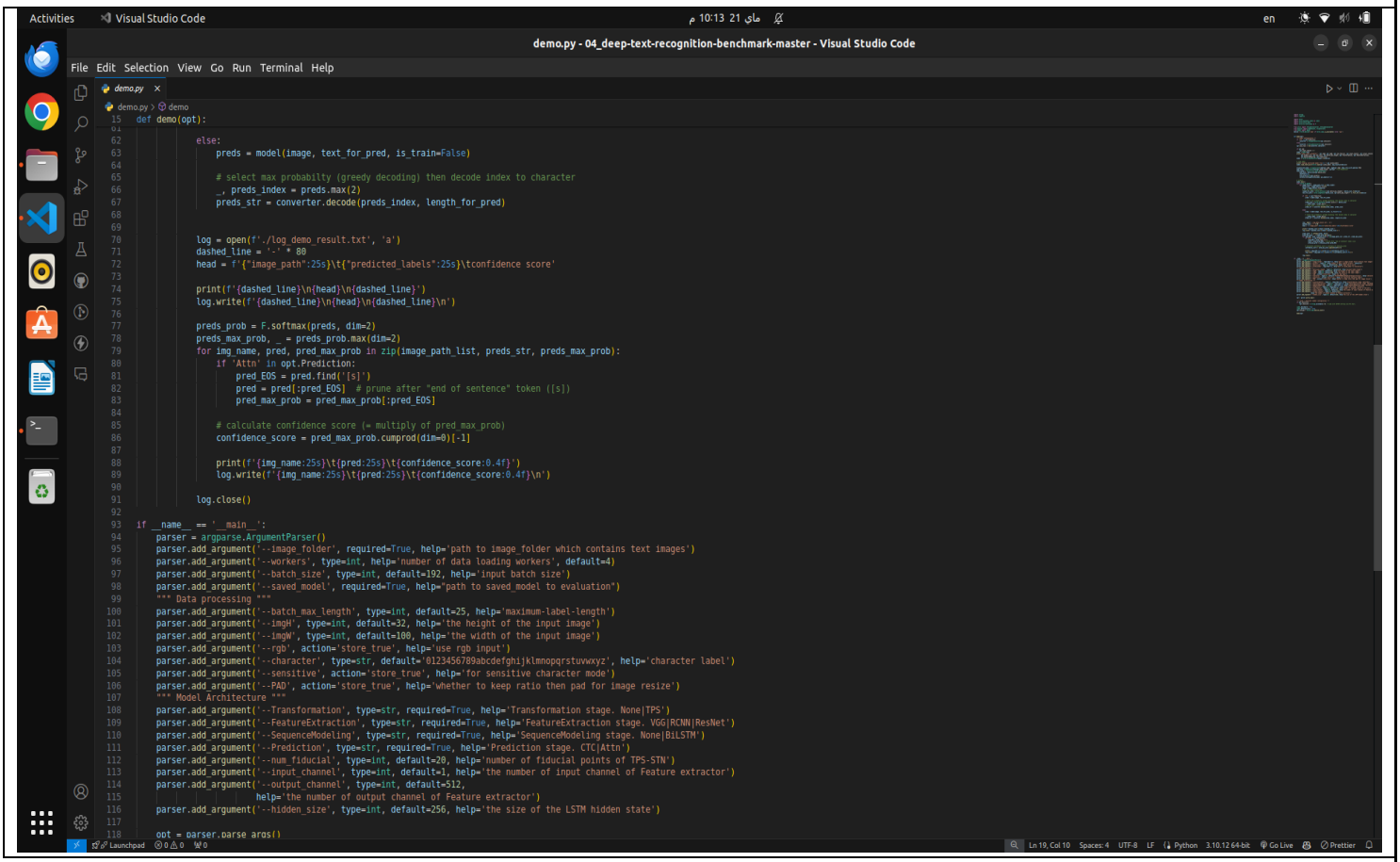

Activities
Visual Studio Code
10:09 21 ماي
en
views.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py M
src > myapp > views.py > detect_text_in_image
class ImageOCRView(views.APIView):
def post(self, request, *args, **kwargs):
# audio_file_path_dest = os.path.join(settings.MEDIA_ROOT , f'{response_text[:10]}.wav')
else:
audio_file_path_dest = ""
# try:
# except Exception as e:
# print(f"Error copying audio file: {e}")
# return Response({'error': 'Internal server error'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
return Response(
{
'message': 'Image saved and processed successfully',
'text': response_text,
'audio_file': audio_file_path_dest,
},
status=status.HTTP_201_CREATED
)
else:
return Response({'error': 'No image found in the request'}, status=status.HTTP_400_BAD_REQUEST)
except Exception as e:
return Response({'error': 'Internal server error'}, status=status.HTTP_500_INTERNAL_SERVER_ERROR)
master*
Launchpad
You, 3 weeks ago
Ln 27, Col 45
Spaces: 4
UTF-8
LF
Python
3.10.12 ('.venv': venv)
Go Live
Prettier

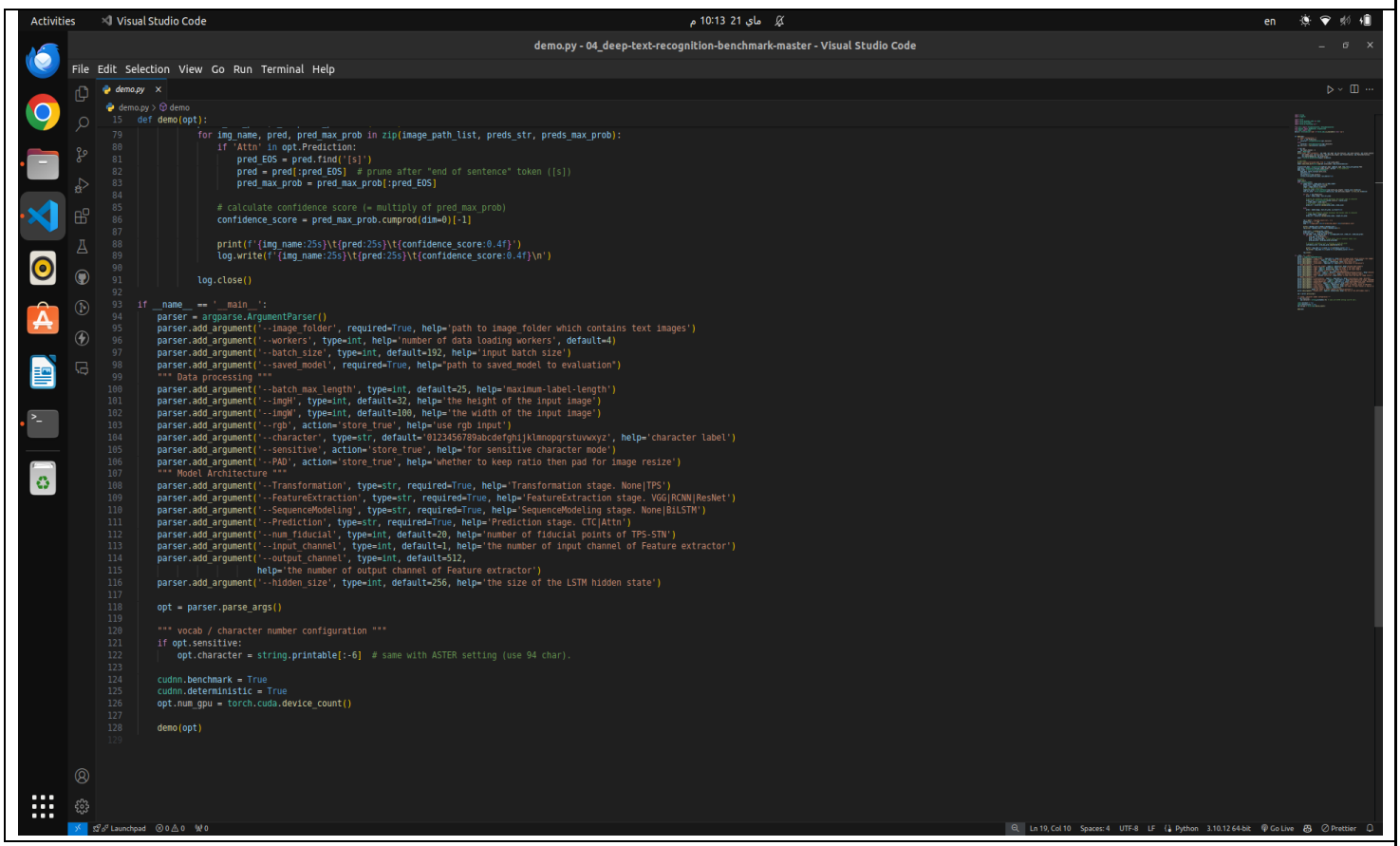

Activities
Visual Studio Code
10:10 21 ماي
en
script.sh - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py M
script.sh
src > script.sh
You, 3 weeks ago | 1 author (You)
#!/bin/bash
Change to the directory where Basics.py is located
cd ~/Desktop/alpha/VALL-E-X
# Receive input from stdin (response_text)
input=$(cat)
# Save input to a temporary text file
echo "$input" > temp_input.txt
Execute Basics.py to generate audio based on the input
python3 Basics.py < temp_input.txt
master*
Launchpad
Ln 14, Col 1
Spaces: 4
UTF-8
LF
Shell Script
Go Live
Prettier

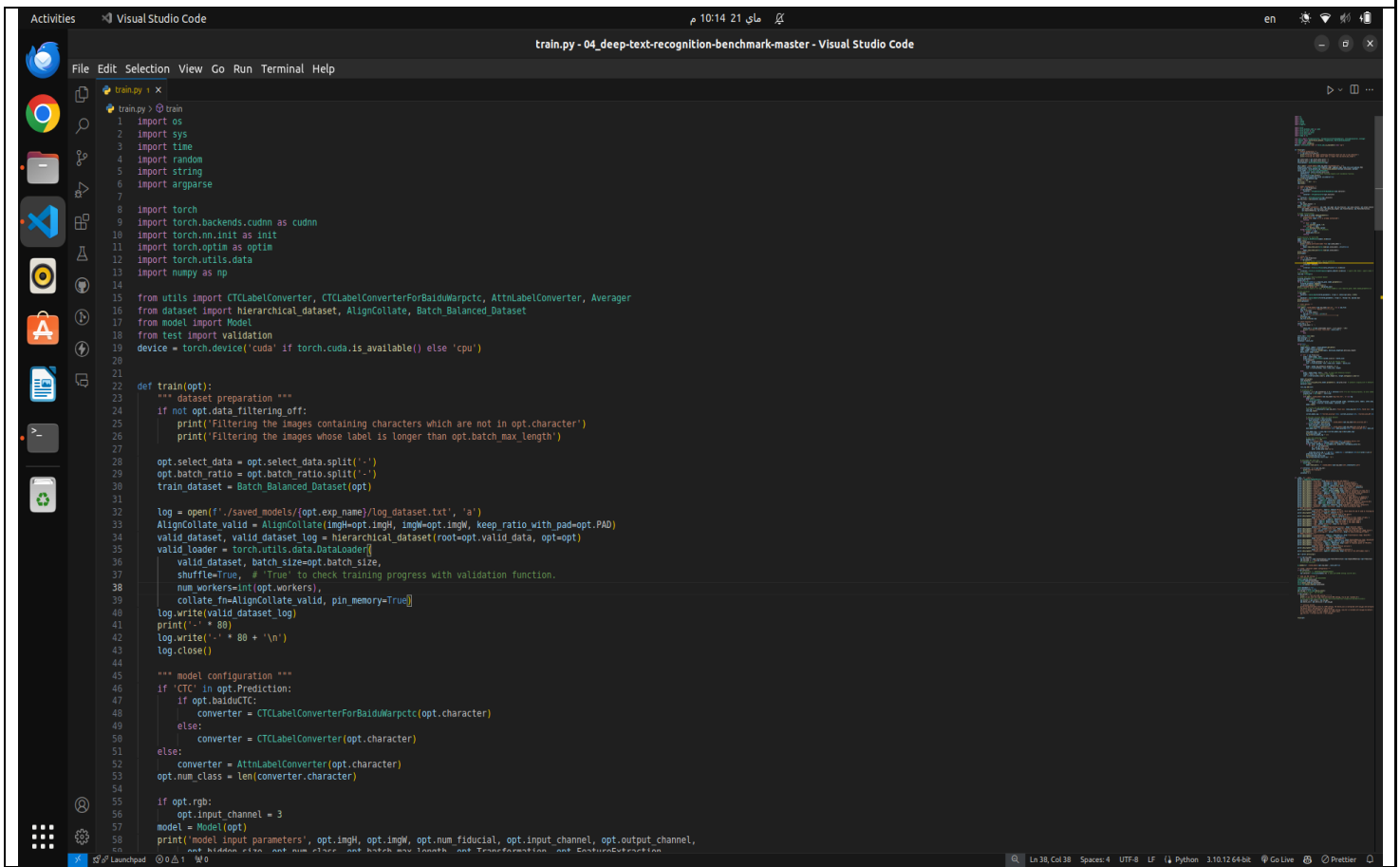

Activities
Visual Studio Code
10:10 21 ماي م
en
urls.py - project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
views.py M
urls.py
src > myapp > urls.py > ...
You, 3 weeks ago | 1 author (You)
from django.urls import path, include
from . import views
urlpatterns = [
path('pass-image/', views.ImageOCRView.as_view())
]

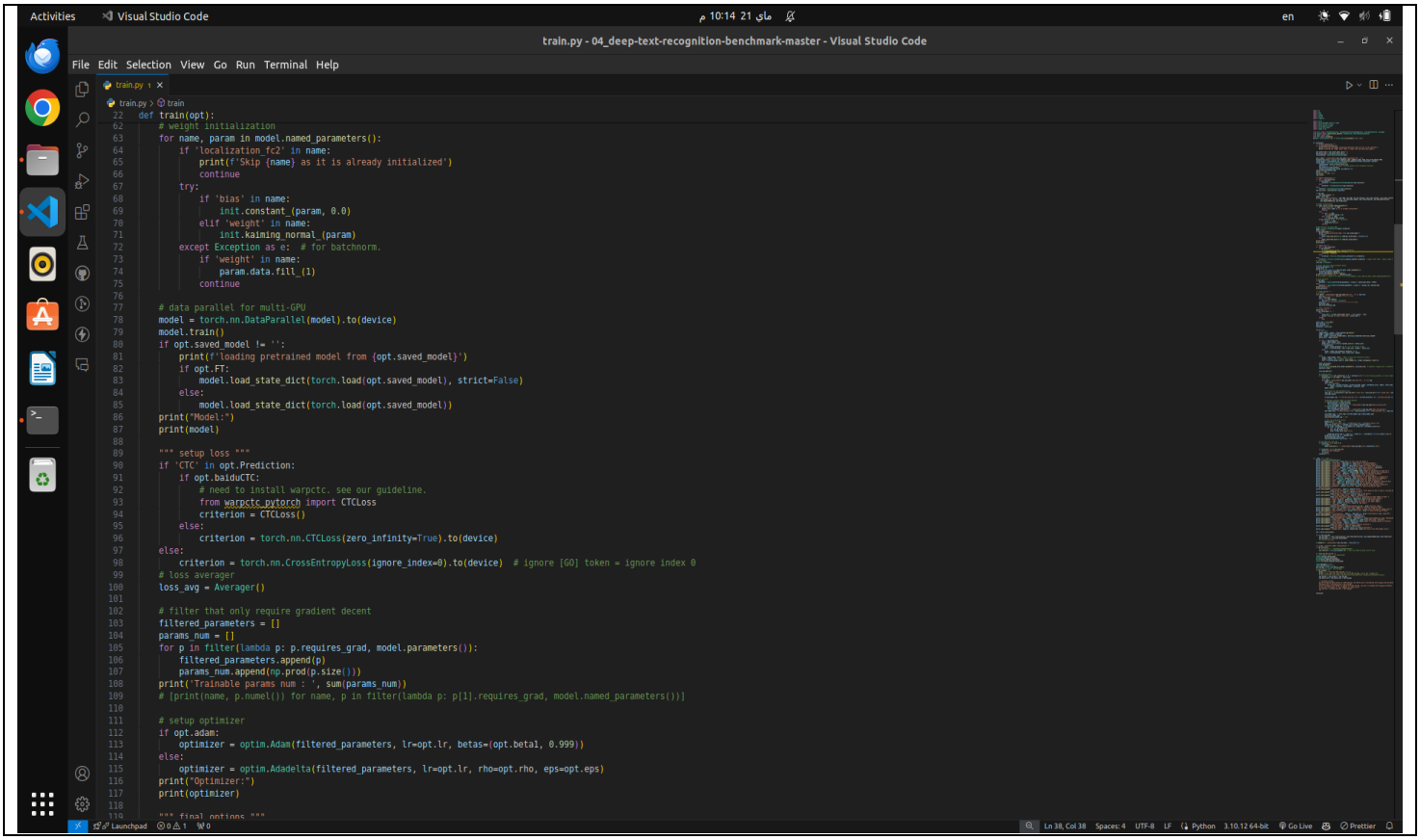

Activities
Visual Studio Code
10:11 21 ماي م
en
Basics.py - VALL-E-X - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Basics.py u
Basics.py > ...
import sys
import os
import glob
from utils.generation import SAMPLE_RATE, generate_audio, preload_models
from scipy.io.wavfile import write as write_wav
def generate_audio_from_text(text_prompt, accent='no-accent'):
# Download and load all necessary models
preload_models()
# Generate audio from the given text prompt
audio_array = generate_audio(text_prompt, prompt="vctk_3")
return audio_array
import datetime
def save_audio_to_file(text_prompt, audio_array):
def delete_wav_files(directory):
# Get a list of all .wav files in the directory
wav_files = glob.glob(os.path.join(directory, '*.wav'))
# Delete each file
for file in wav_files:
os.remove(file)
# Call the function with the directory path
delete_wav_files('/home/marwan/project/src/images/audio')
file_name = text_prompt[:10] + f'{datetime.datetime.now().month}.wav'
file_path = os.path.join('/home/marwan/project/src/images/audio', file_name) # Save the audio to a WAV file
write_wav(file_path, SAMPLE_RATE, audio_array)
return file_path
if __name__ == "__main__":
# Read text prompt from standard input (stdin)
text_prompt = sys.stdin.read().strip()
# Generate audio based on the received text prompt
audio_array = generate_audio_from_text(text_prompt, accent='no-accent')
# Save audio to a WAV file
audio_file_path = save_audio_to_file(text_prompt, audio_array)
# Output the path to the saved audio file
print(audio_file_path)

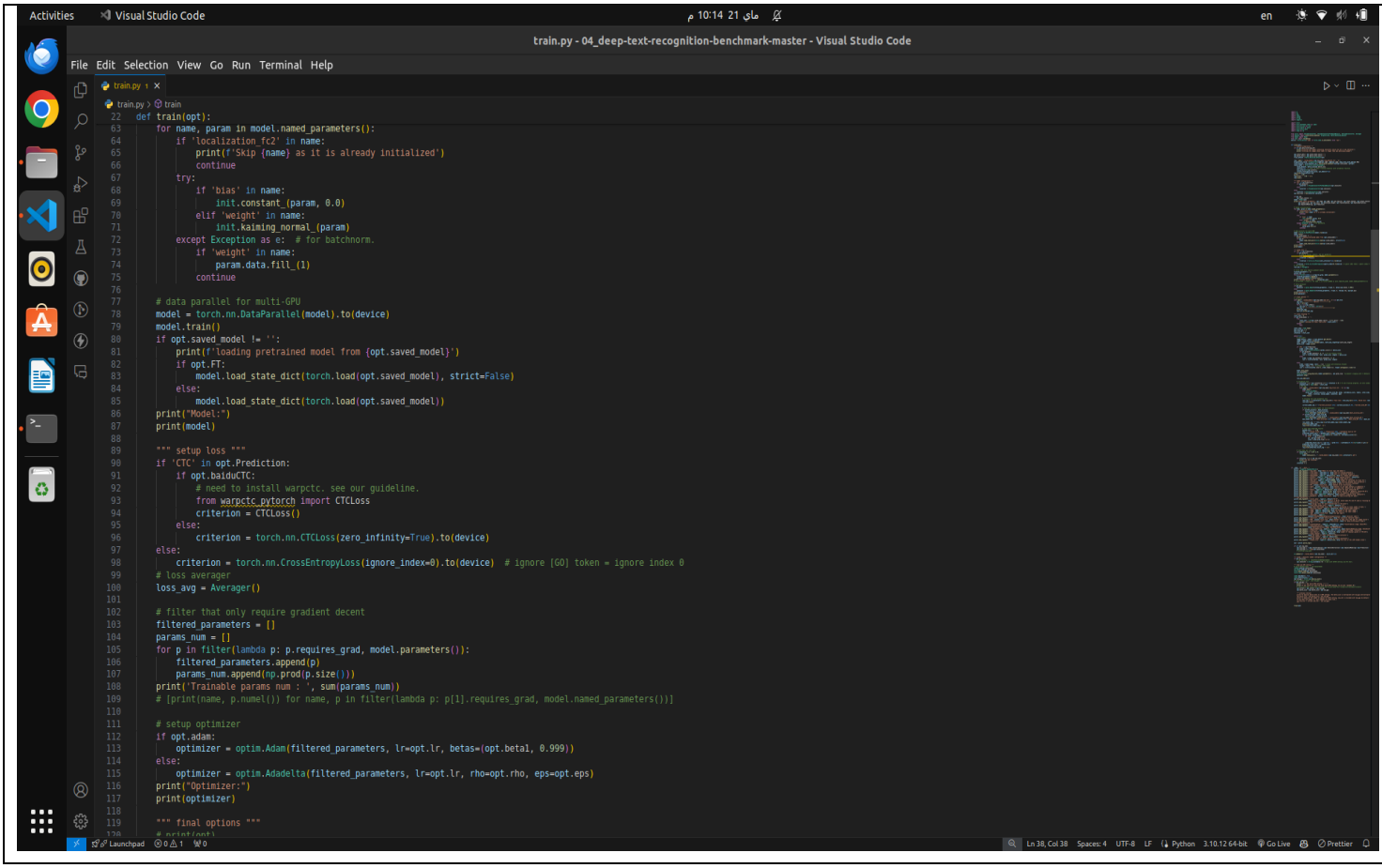

Activities
Visual Studio Code
10:13 21 ماي م
en
demo.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
demo.py
demo.py > demo
import string
import argparse
import torch
import torch.backends.cudnn as cudnn
import torch.utils.data
import torch.nn.functional as F
from utils import CTCLabelConverter, AttnLabelConverter
from dataset import RawDataset, AlignCollate
from model import Model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
def demo(opt):
""" model configuration """
if 'CTC' in opt.Prediction:
converter = CTCLabelConverter(opt.character)
else:
converter = AttnLabelConverter(opt.character)
opt.num_class = len(converter.character)
if opt.rgb:
opt.input_channel = 3
model = Model(opt)
print('model input parameters', opt.imgH, opt.imgW, opt.num_fiducial, opt.input_channel, opt.output_channel,
opt.hidden_size, opt.num_class, opt.batch_max_length, opt.Transformation, opt.FeatureExtraction,
opt.SequenceModeling, opt.Prediction)
model = torch.nn.DataParallel(model).to(device)
# load model
print('loading pretrained model from %s' % opt.saved_model)
model.load_state_dict(torch.load(opt.saved_model, map_location=device))
AlignCollate_demo = AlignCollate(imgH=opt.imgH, imgW=opt.imgW, keep_ratio_with_pad=opt.PAD)
demo_data = RawDataset(root=opt.image_folder, opt=opt) # use RawDataset
demo_loader = torch.utils.data.DataLoader(
demo_data, batch_size=opt.batch_size,
shuffle=False,
num_workers=int(opt.workers),
collate_fn=AlignCollate_demo, pin_memory=True)
# predict
model.eval()
with torch.no_grad():
for image_tensors, image_path_list in demo_loader:
batch_size = image_tensors.size(0)
image = image_tensors.to(device)
# For max length prediction
length_for_pred = torch.IntTensor([opt.batch_max_length] * batch_size).to(device)
text_for_pred = torch.LongTensor(batch_size, opt.batch_max_length + 1).fill_(0).to(device)
if 'CTC' in opt.Prediction:
preds = model(image, text_for_pred)
# Select max probabilty (greedy decoding) then decode index to character
preds_size = torch.IntTensor([preds.size(1)] * batch_size)
_, preds_index = preds.max(2)
Launchpad
Ln 19, Col 10
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

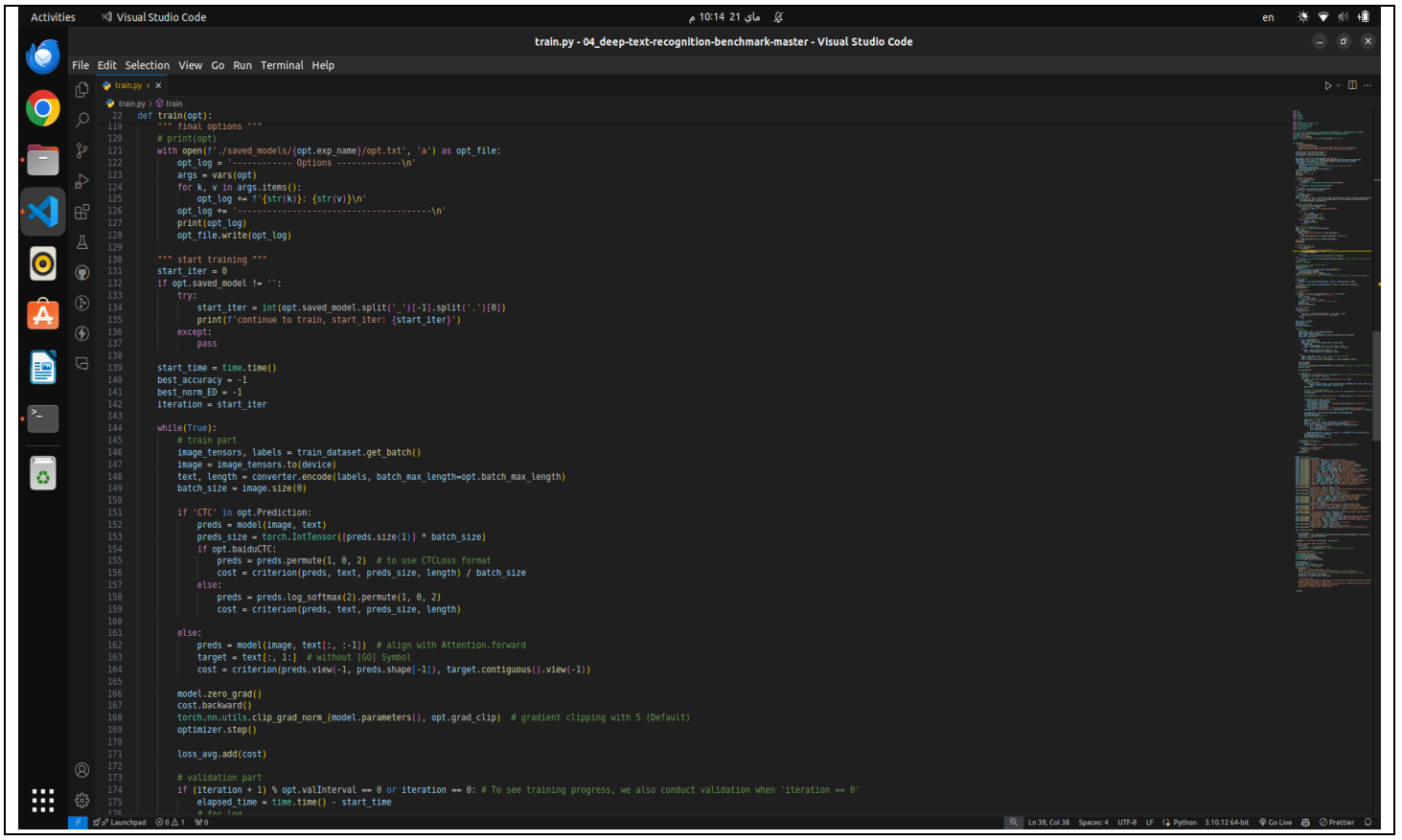

Activities
Visual Studio Code
10:13 21 ماي م
en
demo.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
demo.py
demo.py > demo
def demo(opt):
else:
preds = model(image, text_for_pred, is_train=False)
# select max probabilty (greedy decoding) then decode index to character
_, preds_index = preds.max(2)
preds_str = converter.decode(preds_index, length_for_pred)
log = open(f'./log_demo_result.txt', 'a')
dashed_line = '-' * 80
head = f'{"image_path":25s}\t{"predicted_labels":25s}\tconfidence score'
print(f'{dashed_line}\n{head}\n{dashed_line}')
log.write(f'{dashed_line}\n{head}\n{dashed_line}\n')
preds_prob = F.softmax(preds, dim=2)
preds_max_prob, _ = preds_prob.max(dim=2)
for img_name, pred, pred_max_prob in zip(image_path_list, preds_str, preds_max_prob):
if 'Attn' in opt.Prediction:
pred_EOS = pred.find('[s]')
pred = pred[:pred_EOS] # prune after "end of sentence" token ([s])
pred_max_prob = pred_max_prob[:pred_EOS]
# calculate confidence score (= multiply of pred_max_prob)
confidence_score = pred_max_prob.cumprod(dim=0)[-1]
print(f'{img_name:25s}\t{pred:25s}\t{confidence_score:0.4f}')
log.write(f'{img_name:25s}\t{pred:25s}\t{confidence_score:0.4f}\n')
log.close()
if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--image_folder', required=True, help='path to image_folder which contains text images')
parser.add_argument('--workers', type=int, help='number of data loading workers', default=4)
parser.add_argument('--batch_size', type=int, default=192, help='input batch size')
parser.add_argument('--saved_model', required=True, help="path to saved_model to evaluation")
""" Data processing """
parser.add_argument('--batch_max_length', type=int, default=25, help='maximum-label-length')
parser.add_argument('--imgH', type=int, default=32, help='the height of the input image')
parser.add_argument('--imgW', type=int, default=100, help='the width of the input image')
parser.add_argument('--rgb', action='store_true', help='use rgb input')
parser.add_argument('--character', type=str, default='0123456789abcdefghijklmnopqrstuvwxyz', help='character label')
parser.add_argument('--sensitive', action='store_true', help='for sensitive character mode')
parser.add_argument('--PAD', action='store_true', help='whether to keep ratio then pad for image resize')
""" Model Architecture """
parser.add_argument('--Transformation', type=str, required=True, help='Transformation stage. None|TPS')
parser.add_argument('--FeatureExtraction', type=str, required=True, help='FeatureExtraction stage. VGG|RCNN|ResNet')
parser.add_argument('--SequenceModeling', type=str, required=True, help='SequenceModeling stage. None|BiLSTM')
parser.add_argument('--Prediction', type=str, required=True, help='Prediction stage. CTC|Attn')
parser.add_argument('--num_fiducial', type=int, default=20, help='number of fiducial points of TPS-STN')
parser.add_argument('--input_channel', type=int, default=1, help='the number of input channel of Feature extractor')
parser.add_argument('--output_channel', type=int, default=512,
help='the number of output channel of Feature extractor')
parser.add_argument('--hidden_size', type=int, default=256, help='the size of the LSTM hidden state')
opt = parser.parse_args()
Launchpad
Ln 19, Col 10
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

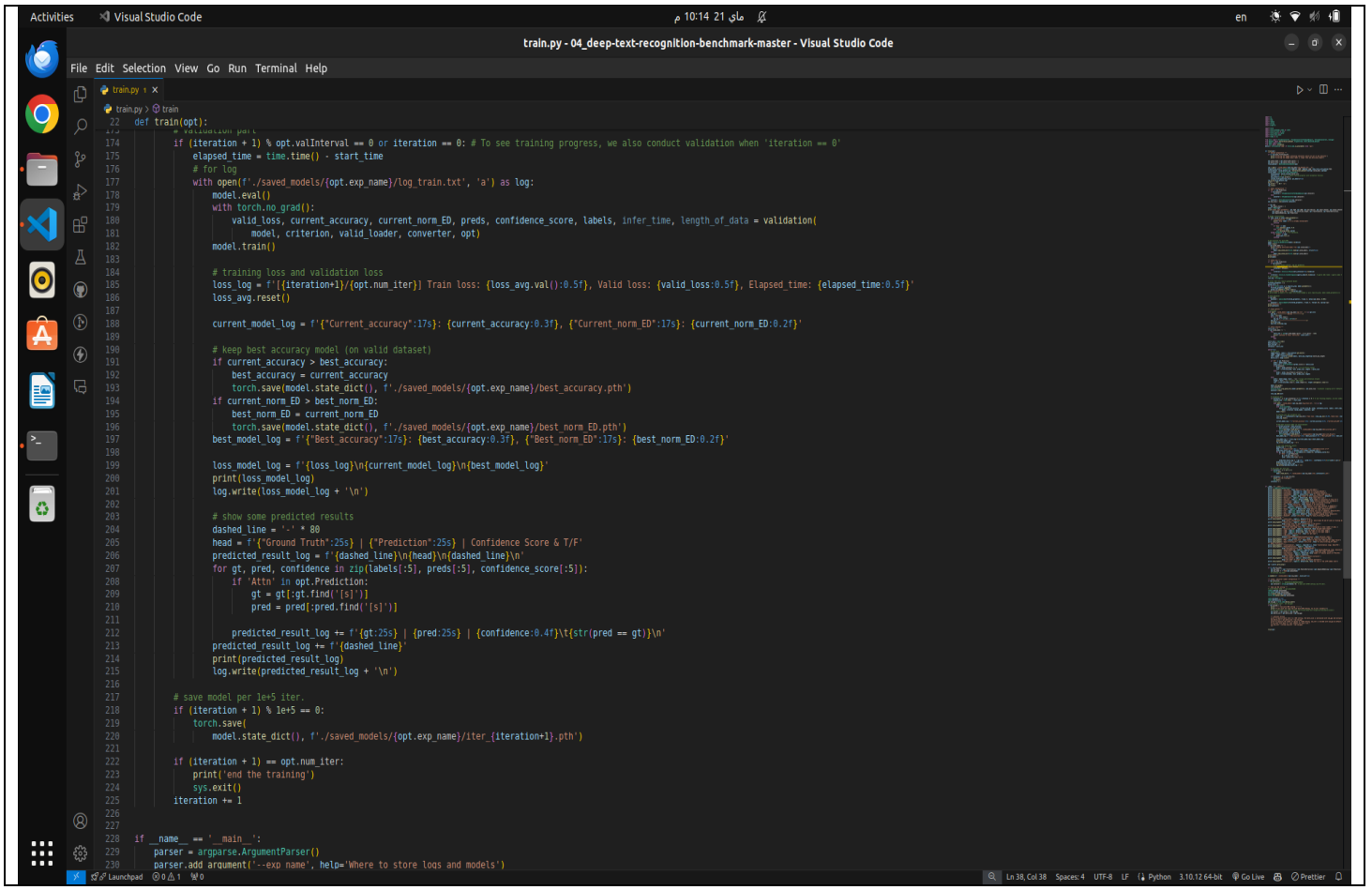

Activities
Visual Studio Code
21 ماي 10:13 م
en
demo.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
demo.py
demo.py > demo
def demo(opt):
for img_name, pred, pred_max_prob in zip(image_path_list, preds_str, preds_max_prob):
if 'Attn' in opt.Prediction:
pred_EOS = pred.find('[s]')
pred = pred[:pred_EOS] # prune after "end of sentence" token ([s])
pred_max_prob = pred_max_prob[:pred_EOS]
# calculate confidence score (= multiply of pred_max_prob)
confidence_score = pred_max_prob.cumprod(dim=0)[-1]
print(f'{img_name:25s}\t{pred:25s}\t{confidence_score:0.4f}')
log.write(f'{img_name:25s}\t{pred:25s}\t{confidence_score:0.4f}\n')
log.close()
if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--image_folder', required=True, help='path to image_folder which contains text images')
parser.add_argument('--workers', type=int, help='number of data loading workers', default=4)
parser.add_argument('--batch_size', type=int, default=192, help='input batch size')
parser.add_argument('--saved_model', required=True, help="path to saved_model to evaluation")
""" Data processing """
parser.add_argument('--batch_max_length', type=int, default=25, help='maximum-label-length')
parser.add_argument('--imgH', type=int, default=32, help='the height of the input image')
parser.add_argument('--imgW', type=int, default=100, help='the width of the input image')
parser.add_argument('--rgb', action='store_true', help='use rgb input')
parser.add_argument('--character', type=str, default='0123456789abcdefghijklmnopqrstuvwxyz', help='character label')
parser.add_argument('--sensitive', action='store_true', help='for sensitive character mode')
parser.add_argument('--PAD', action='store_true', help='whether to keep ratio then pad for image resize')
""" Model Architecture """
parser.add_argument('--Transformation', type=str, required=True, help='Transformation stage. None|TPS')
parser.add_argument('--FeatureExtraction', type=str, required=True, help='FeatureExtraction stage. VGG|RCNN|ResNet')
parser.add_argument('--SequenceModeling', type=str, required=True, help='SequenceModeling stage. None|BiLSTM')
parser.add_argument('--Prediction', type=str, required=True, help='Prediction stage. CTC|Attn')
parser.add_argument('--num_fiducial', type=int, default=20, help='number of fiducial points of TPS-STN')
parser.add_argument('--input_channel', type=int, default=1, help='the number of input channel of Feature extractor')
parser.add_argument('--output_channel', type=int, default=512,
help='the number of output channel of Feature extractor')
parser.add_argument('--hidden_size', type=int, default=256, help='the size of the LSTM hidden state')
opt = parser.parse_args()
""" vocab / character number configuration """
if opt.sensitive:
opt.character = string.printable[:-6] # same with ASTER setting (use 94 char).
cudnn.benchmark = True
cudnn.deterministic = True
opt.num_gpu = torch.cuda.device_count()
demo(opt)
Launchpad
Ln 19, Col 10
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

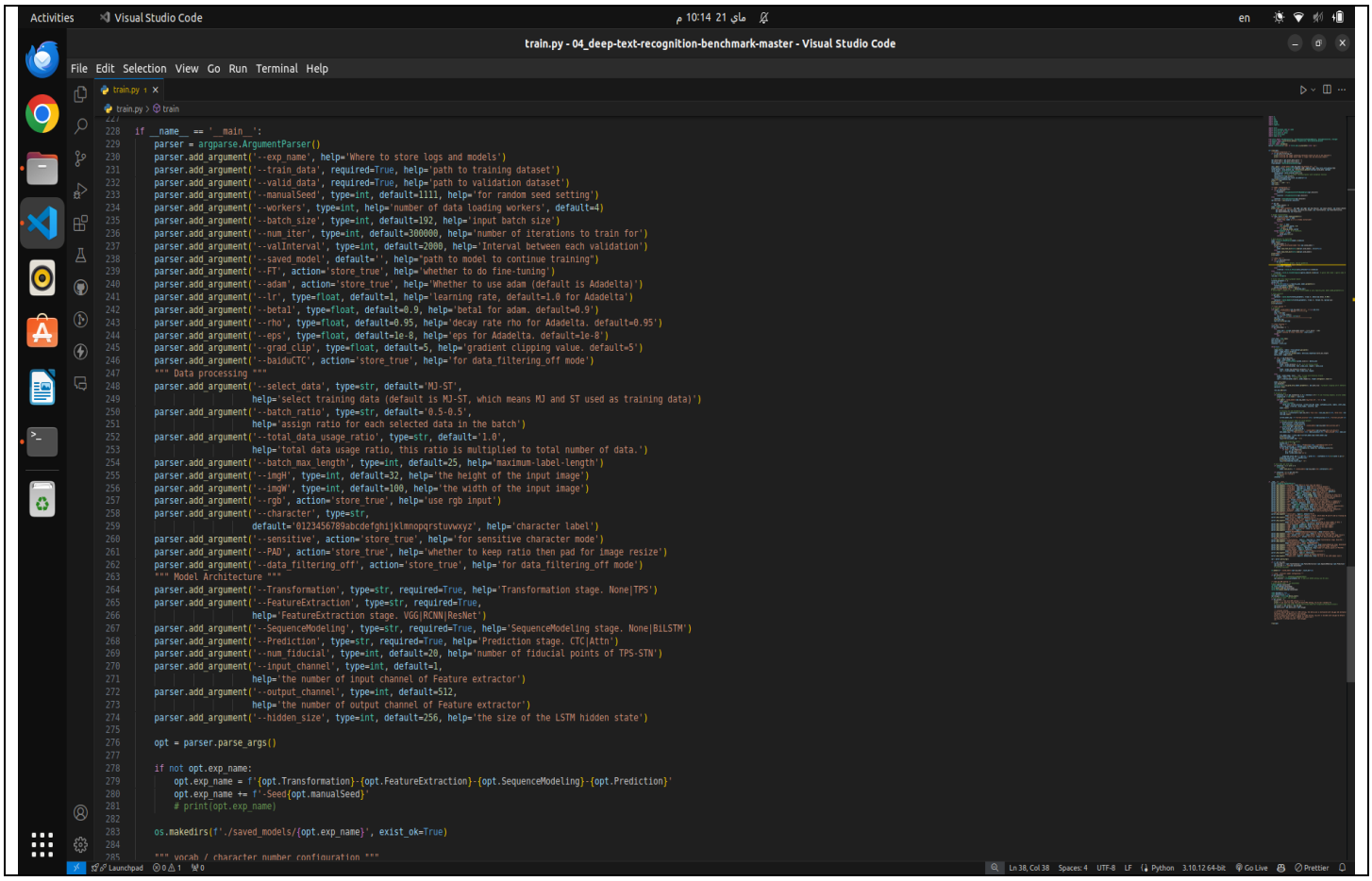

Activities
Visual Studio Code
21 ماي 10:14 م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
import os
import sys
import time
import random
import string
import argparse
import torch
import torch.backends.cudnn as cudnn
import torch.nn.init as init
import torch.optim as optim
import torch.utils.data
import numpy as np
from utils import CTCLabelConverter, CTCLabelConverterForBaiduWarpctc, AttnLabelConverter, Averager
from dataset import hierarchical_dataset, AlignCollate, Batch_Balanced_Dataset
from model import Model
from test import validation
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
def train(opt):
""" dataset preparation """
if not opt.data_filtering_off:
print('Filtering the images containing characters which are not in opt.character')
print('Filtering the images whose label is longer than opt.batch_max_length')
opt.select_data = opt.select_data.split('-')
opt.batch_ratio = opt.batch_ratio.split('-')
train_dataset = Batch_Balanced_Dataset(opt)
log = open(f'./saved_models/{opt.exp_name}/log_dataset.txt', 'a')
AlignCollate_valid = AlignCollate(imgH=opt.imgH, imgW=opt.imgW, keep_ratio_with_pad=opt.PAD)
valid_dataset, valid_dataset_log = hierarchical_dataset(root=opt.valid_data, opt=opt)
valid_loader = torch.utils.data.DataLoader(
valid_dataset, batch_size=opt.batch_size,
shuffle=True, # 'True' to check training progress with validation function.
num_workers=int(opt.workers),
collate_fn=AlignCollate_valid, pin_memory=True)
log.write(valid_dataset_log)
print('-' * 80)
log.write('-' * 80 + '\n')
log.close()
""" model configuration """
if 'CTC' in opt.Prediction:
if opt.baiduCTC:
converter = CTCLabelConverterForBaiduWarpctc(opt.character)
else:
converter = CTCLabelConverter(opt.character)
else:
converter = AttnLabelConverter(opt.character)
opt.num_class = len(converter.character)
if opt.rgb:
opt.input_channel = 3
model = Model(opt)
print('model input parameters', opt.imgH, opt.imgW, opt.num_fiducial, opt.input_channel, opt.output_channel,
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

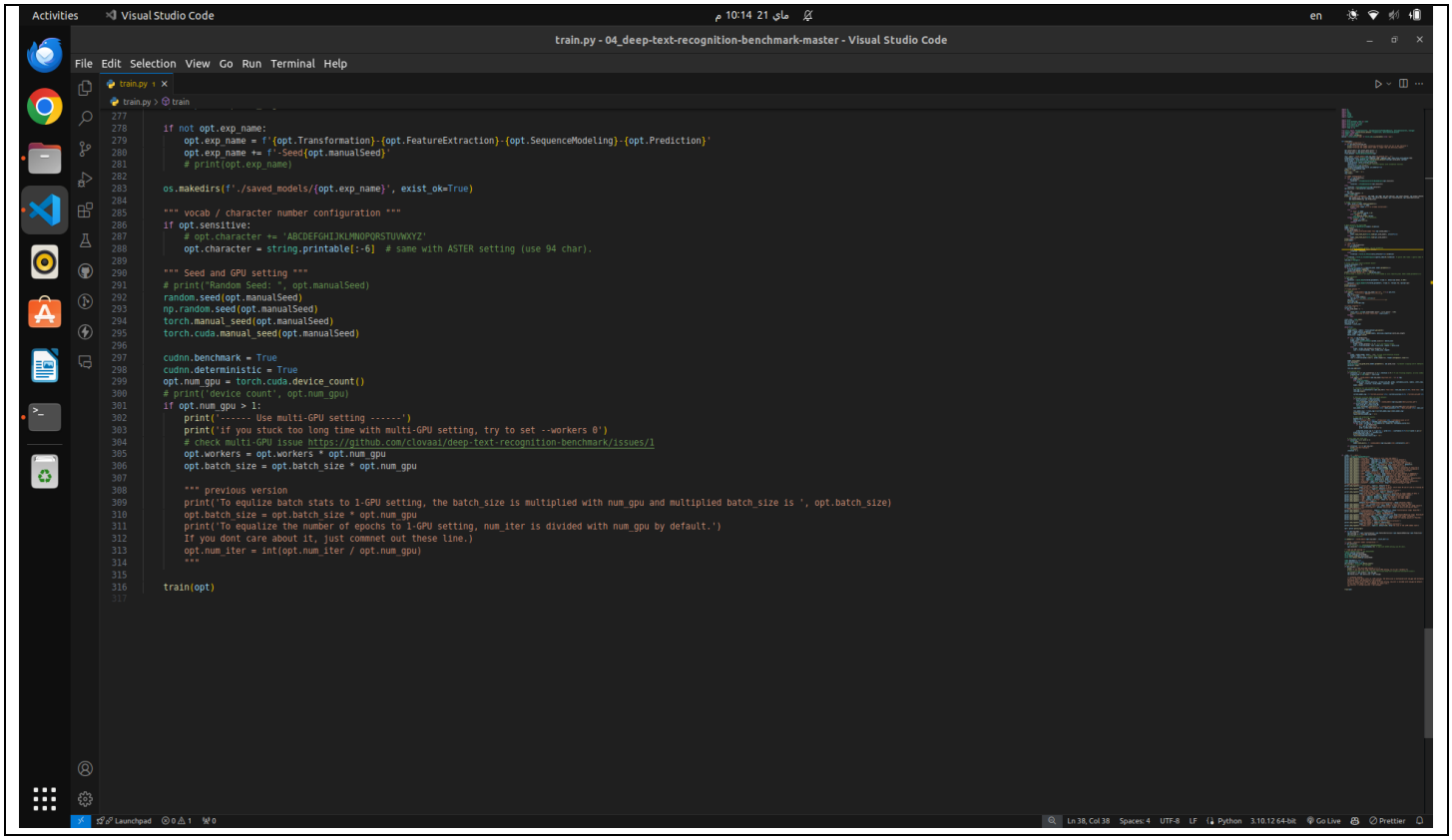

Activities
Visual Studio Code
10:14 21 ماي م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
def train(opt):
# weight initialization
for name, param in model.named_parameters():
if 'localization_fc2' in name:
print(f'Skip {name} as it is already initialized')
continue
try:
if 'bias' in name:
init.constant_(param, 0.0)
elif 'weight' in name:
init.kaiming_normal_(param)
except Exception as e: # for batchnorm.
if 'weight' in name:
param.data.fill_(1)
continue
# data parallel for multi-GPU
model = torch.nn.DataParallel(model).to(device)
model.train()
if opt.saved_model != '':
print(f'loading pretrained model from {opt.saved_model}')
if opt.FT:
model.load_state_dict(torch.load(opt.saved_model), strict=False)
else:
model.load_state_dict(torch.load(opt.saved_model))
print("Model:")
print(model)
""" setup loss """
if 'CTC' in opt.Prediction:
if opt.baiduCTC:
# need to install warpctc. see our guideline.
from warpctc_pytorch import CTCLoss
criterion = CTCLoss()
else:
criterion = torch.nn.CTCLoss(zero_infinity=True).to(device)
else:
criterion = torch.nn.CrossEntropyLoss(ignore_index=0).to(device) # ignore [GO] token = ignore index 0
# loss averager
loss_avg = Averager()
# filter that only require gradient decent
filtered_parameters = []
params_num = []
for p in filter(lambda p: p.requires_grad, model.parameters()):
filtered_parameters.append(p)
params_num.append(np.prod(p.size()))
print('Trainable params num : ', sum(params_num))
# [print(name, p.numel()) for name, p in filter(lambda p: p[1].requires_grad, model.named_parameters())]
# setup optimizer
if opt.adam:
optimizer = optim.Adam(filtered_parameters, lr=opt.lr, betas=(opt.beta1, 0.999))
else:
optimizer = optim.Adadelta(filtered_parameters, lr=opt.lr, rho=opt.rho, eps=opt.eps)
print("Optimizer:")
print(optimizer)
""" final options """
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

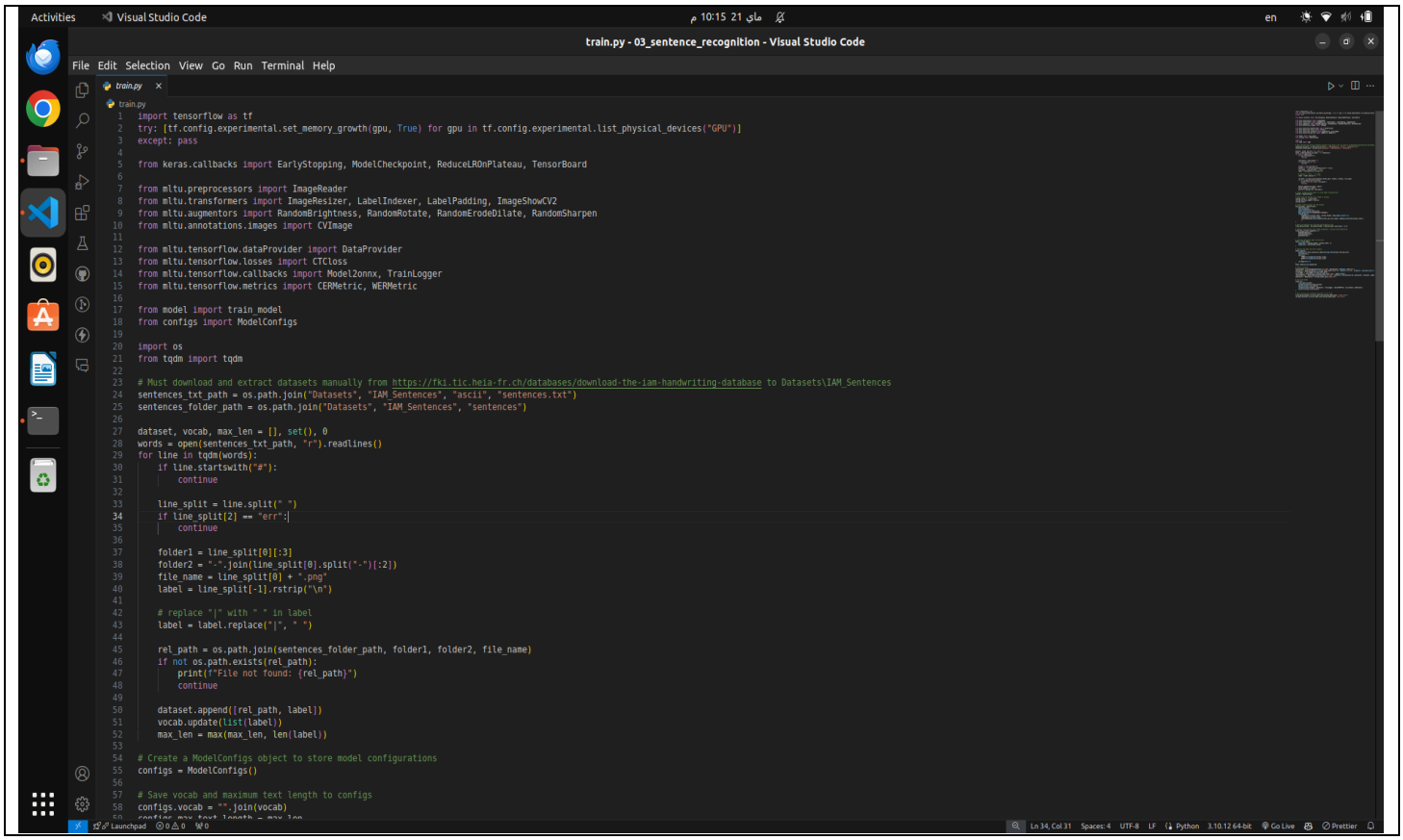

Activities
Visual Studio Code
10:14 21 ماي م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
def train(opt):
for name, param in model.named_parameters():
if 'localization_fc2' in name:
print(f'Skip {name} as it is already initialized')
continue
try:
if 'bias' in name:
init.constant_(param, 0.0)
elif 'weight' in name:
init.kaiming_normal_(param)
except Exception as e: # for batchnorm.
if 'weight' in name:
param.data.fill_(1)
continue
# data parallel for multi-GPU
model = torch.nn.DataParallel(model).to(device)
model.train()
if opt.saved_model != '':
print(f'loading pretrained model from {opt.saved_model}')
if opt.FT:
model.load_state_dict(torch.load(opt.saved_model), strict=False)
else:
model.load_state_dict(torch.load(opt.saved_model))
print("Model:")
print(model)
""" setup loss """
if 'CTC' in opt.Prediction:
if opt.baiduCTC:
# need to install warpctc. see our guideline.
from warpctc_pytorch import CTCLoss
criterion = CTCLoss()
else:
criterion = torch.nn.CTCLoss(zero_infinity=True).to(device)
else:
criterion = torch.nn.CrossEntropyLoss(ignore_index=0).to(device) # ignore [GO] token = ignore index 0
# loss averager
loss_avg = Averager()
# filter that only require gradient decent
filtered_parameters = []
params_num = []
for p in filter(lambda p: p.requires_grad, model.parameters()):
filtered_parameters.append(p)
params_num.append(np.prod(p.size()))
print('Trainable params num : ', sum(params_num))
# [print(name, p.numel()) for name, p in filter(lambda p: p[1].requires_grad, model.named_parameters())]
# setup optimizer
if opt.adam:
optimizer = optim.Adam(filtered_parameters, lr=opt.lr, betas=(opt.beta1, 0.999))
else:
optimizer = optim.Adadelta(filtered_parameters, lr=opt.lr, rho=opt.rho, eps=opt.eps)
print("Optimizer:")
print(optimizer)
""" final options """
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

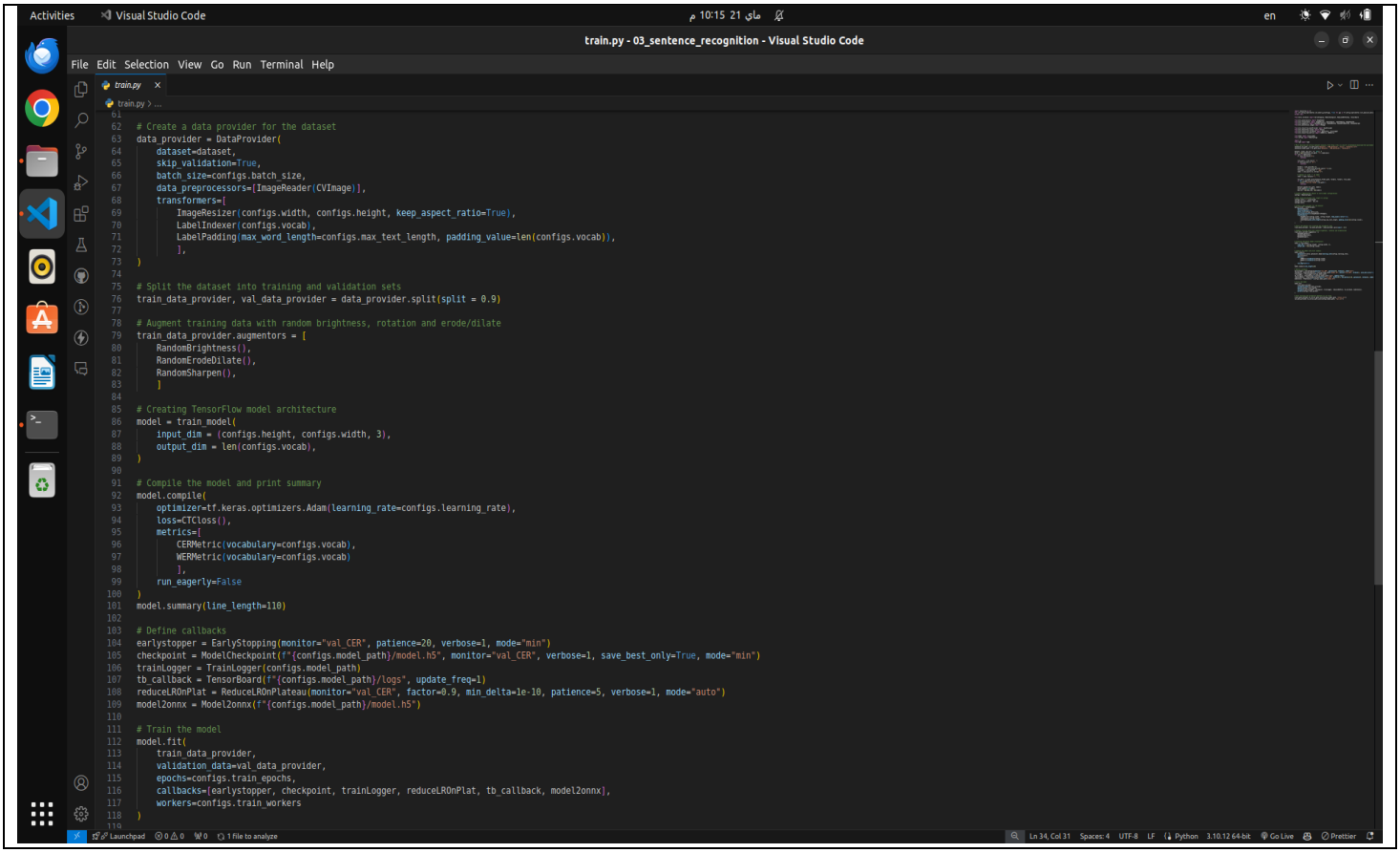

Activities
Visual Studio Code
10:14 21 ماي م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
def train(opt):
""" final options """
print(opt)
with open(f'./saved_models/{opt.exp_name}/opt.txt', 'a') as opt_file:
opt_log = '------------ Options -------------\n'
args = vars(opt)
for k, v in args.items():
opt_log += f'{str(k)}: {str(v)}\n'
opt_log += '---------------------------------------\n'
print(opt_log)
opt_file.write(opt_log)
""" start training """
start_iter = 0
if opt.saved_model != '':
try:
start_iter = int(opt.saved_model.split('_')[-1].split('.')[0])
print(f'continue to train, start_iter: {start_iter}')
except:
pass
start_time = time.time()
best_accuracy = -1
best_norm_ED = -1
iteration = start_iter
while(True):
# train part
image_tensors, labels = train_dataset.get_batch()
image = image_tensors.to(device)
text, length = converter.encode(labels, batch_max_length=opt.batch_max_length)
batch_size = image.size(0)
if 'CTC' in opt.Prediction:
preds = model(image, text)
preds_size = torch.IntTensor([preds.size(1)] * batch_size)
if opt.baiduCTC:
preds = preds.permute(1, 0, 2) # to use CTCLoss format
cost = criterion(preds, text, preds_size, length) / batch_size
else:
preds = preds.log_softmax(2).permute(1, 0, 2)
cost = criterion(preds, text, preds_size, length)
else:
preds = model(image, text[:, :-1]) # align with Attention.forward
target = text[:, 1:] # without [GO] Symbol
cost = criterion(preds.view(-1, preds.shape[-1]), target.contiguous().view(-1))
model.zero_grad()
cost.backward()
torch.nn.utils.clip_grad_norm_(model.parameters(), opt.grad_clip) # gradient clipping with 5 (Default)
optimizer.step()
loss_avg.add(cost)
# validation part
if (iteration + 1) % opt.valInterval == 0 or iteration == 0: # To see training progress, we also conduct validation when 'iteration == 0'
elapsed_time = time.time() - start_time
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

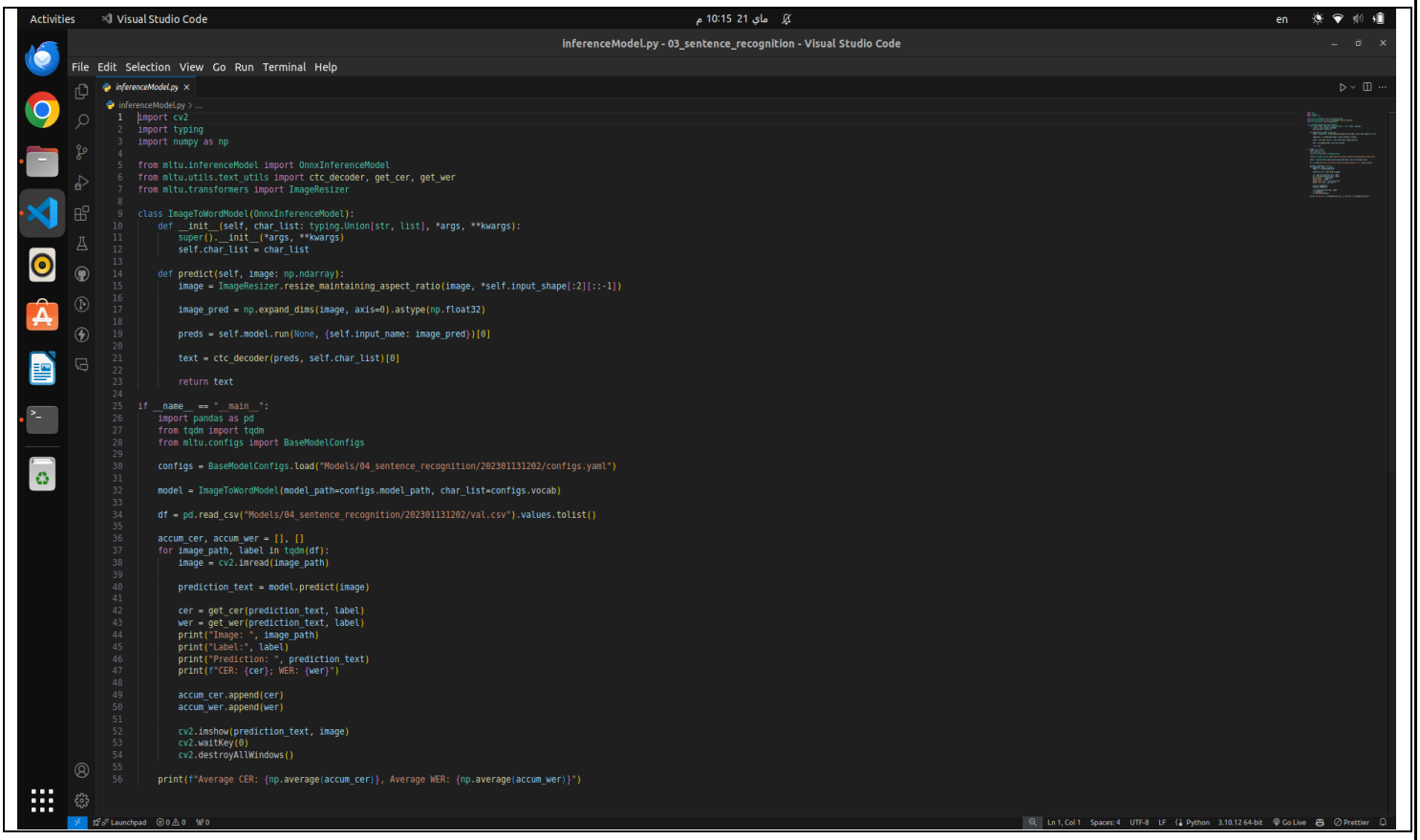

Activities
Visual Studio Code
10:14 21 ماي م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
def train(opt):
if (iteration + 1) % opt.valInterval == 0 or iteration == 0: # To see training progress, we also conduct validation when 'iteration == 0'
elapsed_time = time.time() - start_time
# for log
with open(f'./saved_models/{opt.exp_name}/log_train.txt', 'a') as log:
model.eval()
with torch.no_grad():
valid_loss, current_accuracy, current_norm_ED, preds, confidence_score, labels, infer_time, length_of_data = validation(
model, criterion, valid_loader, converter, opt)
model.train()
# training loss and validation loss
loss_log = f'[{iteration+1}/{opt.num_iter}] Train loss: {loss_avg.val():0.5f}, Valid loss: {valid_loss:0.5f}, Elapsed_time: {elapsed_time:0.5f}'
loss_avg.reset()
current_model_log = f'{"Current_accuracy":17s}: {current_accuracy:0.3f}, {"Current_norm_ED":17s}: {current_norm_ED:0.2f}'
# keep best accuracy model (on valid dataset)
if current_accuracy > best_accuracy:
best_accuracy = current_accuracy
torch.save(model.state_dict(), f'./saved_models/{opt.exp_name}/best_accuracy.pth')
if current_norm_ED > best_norm_ED:
best_norm_ED = current_norm_ED
torch.save(model.state_dict(), f'./saved_models/{opt.exp_name}/best_norm_ED.pth')
best_model_log = f'{"Best_accuracy":17s}: {best_accuracy:0.3f}, {"Best_norm_ED":17s}: {best_norm_ED:0.2f}'
loss_model_log = f'{loss_log}\n{current_model_log}\n{best_model_log}'
print(loss_model_log)
log.write(loss_model_log + '\n')
# show some predicted results
dashed_line = '-' * 80
head = f'{"Ground Truth":25s} | {"Prediction":25s} | Confidence Score & T/F'
predicted_result_log = f'{dashed_line}\n{head}\n{dashed_line}\n'
for gt, pred, confidence in zip(labels[:5], preds[:5], confidence_score[:5]):
if 'Attn' in opt.Prediction:
gt = gt[:gt.find('[s]')]
pred = pred[:pred.find('[s]')]
predicted_result_log += f'{gt:25s} | {pred:25s} | {confidence:0.4f}\t{str(pred == gt)}\n'
predicted_result_log += f'{dashed_line}'
print(predicted_result_log)
log.write(predicted_result_log + '\n')
# save model per 1e+5 iter.
if (iteration + 1) % 1e+5 == 0:
torch.save(
model.state_dict(), f'./saved_models/{opt.exp_name}/iter_{iteration+1}.pth')
if (iteration + 1) == opt.num_iter:
print('end the training')
sys.exit()
iteration += 1
if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--exp_name', help='Where to store logs and models')
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

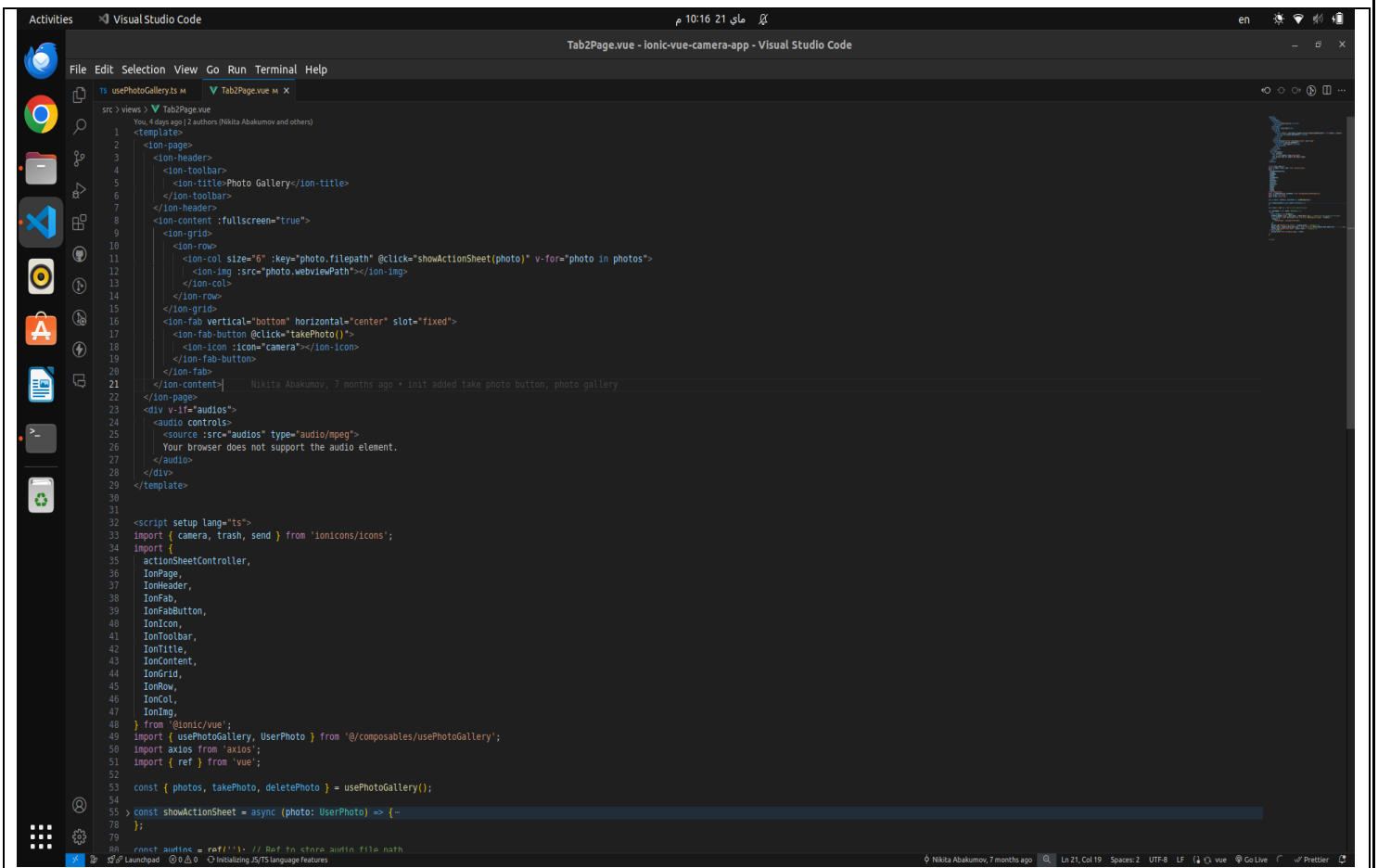

Activities
Visual Studio Code
10:14 ماي 21 م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
if __name__ == '__main__':
parser = argparse.ArgumentParser()
parser.add_argument('--exp_name', help='Where to store logs and models')
parser.add_argument('--train_data', required=True, help='path to training dataset')
parser.add_argument('--valid_data', required=True, help='path to validation dataset')
parser.add_argument('--manualSeed', type=int, default=1111, help='for random seed setting')
parser.add_argument('--workers', type=int, help='number of data loading workers', default=4)
parser.add_argument('--batch_size', type=int, default=192, help='input batch size')
parser.add_argument('--num_iter', type=int, default=300000, help='number of iterations to train for')
parser.add_argument('--valInterval', type=int, default=2000, help='Interval between each validation')
parser.add_argument('--saved_model', default='', help="path to model to continue training")
parser.add_argument('--FT', action='store_true', help='whether to do fine-tuning')
parser.add_argument('--adam', action='store_true', help='Whether to use adam (default is Adadelta)')
parser.add_argument('--lr', type=float, default=1, help='learning rate, default=1.0 for Adadelta')
parser.add_argument('--beta1', type=float, default=0.9, help='beta1 for adam. default=0.9')
parser.add_argument('--rho', type=float, default=0.95, help='decay rate rho for Adadelta. default=0.95')
parser.add_argument('--eps', type=float, default=1e-8, help='eps for Adadelta. default=1e-8')
parser.add_argument('--grad_clip', type=float, default=5, help='gradient clipping value. default=5')
parser.add_argument('--baiduCTC', action='store_true', help='for data_filtering_off mode')
""" Data processing """
parser.add_argument('--select_data', type=str, default='MJ-ST',
help='select training data (default is MJ-ST, which means MJ and ST used as training data)')
parser.add_argument('--batch_ratio', type=str, default='0.5-0.5',
help='assign ratio for each selected data in the batch')
parser.add_argument('--total_data_usage_ratio', type=str, default='1.0',
help='total data usage ratio, this ratio is multiplied to total number of data.')
parser.add_argument('--batch_max_length', type=int, default=25, help='maximum-label-length')
parser.add_argument('--imgH', type=int, default=32, help='the height of the input image')
parser.add_argument('--imgW', type=int, default=100, help='the width of the input image')
parser.add_argument('--rgb', action='store_true', help='use rgb input')
parser.add_argument('--character', type=str,
default='0123456789abcdefghijklmnopqrstuvwxyz', help='character label')
parser.add_argument('--sensitive', action='store_true', help='for sensitive character mode')
parser.add_argument('--PAD', action='store_true', help='whether to keep ratio then pad for image resize')
parser.add_argument('--data_filtering_off', action='store_true', help='for data_filtering_off mode')
""" Model Architecture """
parser.add_argument('--Transformation', type=str, required=True, help='Transformation stage. None|TPS')
parser.add_argument('--FeatureExtraction', type=str, required=True,
help='FeatureExtraction stage. VGG|RCNN|ResNet')
parser.add_argument('--SequenceModeling', type=str, required=True, help='SequenceModeling stage. None|BiLSTM')
parser.add_argument('--Prediction', type=str, required=True, help='Prediction stage. CTC|Attn')
parser.add_argument('--num_fiducial', type=int, default=20, help='number of fiducial points of TPS-STN')
parser.add_argument('--input_channel', type=int, default=1,
help='the number of input channel of Feature extractor')
parser.add_argument('--output_channel', type=int, default=512,
help='the number of output channel of Feature extractor')
parser.add_argument('--hidden_size', type=int, default=256, help='the size of the LSTM hidden state')
opt = parser.parse_args()
if not opt.exp_name:
opt.exp_name = f'{opt.Transformation}-{opt.FeatureExtraction}-{opt.SequenceModeling}-{opt.Prediction}'
opt.exp_name += f'-Seed{opt.manualSeed}'
# print(opt.exp_name)
os.makedirs(f'./saved_models/{opt.exp_name}', exist_ok=True)
""" vocab / character number configuration """
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier


Activities
Visual Studio Code
10:14 ماي 21 م
en
train.py - 04_deep-text-recognition-benchmark-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
train.py > train
if not opt.exp_name:
opt.exp_name = f'{opt.Transformation}-{opt.FeatureExtraction}-{opt.SequenceModeling}-{opt.Prediction}'
opt.exp_name += f'-Seed{opt.manualSeed}'
# print(opt.exp_name)
os.makedirs(f'./saved_models/{opt.exp_name}', exist_ok=True)
""" vocab / character number configuration """
if opt.sensitive:
# opt.character += 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
opt.character = string.printable[:-6] # same with ASTER setting (use 94 char).
""" Seed and GPU setting """
# print("Random Seed: ", opt.manualSeed)
random.seed(opt.manualSeed)
np.random.seed(opt.manualSeed)
torch.manual_seed(opt.manualSeed)
torch.cuda.manual_seed(opt.manualSeed)
cudnn.benchmark = True
cudnn.deterministic = True
opt.num_gpu = torch.cuda.device_count()
# print('device count', opt.num_gpu)
if opt.num_gpu > 1:
print('------ Use multi-GPU setting ------')
print('if you stuck too long time with multi-GPU setting, try to set --workers 0')
# check multi-GPU issue https://github.com/clovaai/deep-text-recognition-benchmark/issues/1
opt.workers = opt.workers * opt.num_gpu
opt.batch_size = opt.batch_size * opt.num_gpu
""" previous version
print('To equlize batch stats to 1-GPU setting, the batch_size is multiplied with num_gpu and multiplied batch_size is ', opt.batch_size)
opt.batch_size = opt.batch_size * opt.num_gpu
print('To equalize the number of epochs to 1-GPU setting, num_iter is divided with num_gpu by default.')
If you dont care about it, just commnet out these line.)
opt.num_iter = int(opt.num_iter / opt.num_gpu)
"""
train(opt)
Launchpad
Ln 38, Col 38
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier


Activities
Visual Studio Code
10:15 21 ماي م
en
train.py - 03_sentence_recognition - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
import tensorflow as tf
try: [tf.config.experimental.set_memory_growth(gpu, True) for gpu in tf.config.experimental.list_physical_devices("GPU")]
except: pass
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, TensorBoard
from mltu.preprocessors import ImageReader
from mltu.transformers import ImageResizer, LabelIndexer, LabelPadding, ImageShowCV2
from mltu.augmentors import RandomBrightness, RandomRotate, RandomErodeDilate, RandomSharpen
from mltu.annotations.images import CVImage
from mltu.tensorflow.dataProvider import DataProvider
from mltu.tensorflow.losses import CTCloss
from mltu.tensorflow.callbacks import Model2onnx, TrainLogger
from mltu.tensorflow.metrics import CERMetric, WERMetric
from model import train_model
from configs import ModelConfigs
import os
from tqdm import tqdm
# Must download and extract datasets manually from https://fki.tic.heia-fr.ch/databases/download-the-iam-handwriting-database to Datasets\IAM_Sentences
sentences_txt_path = os.path.join("Datasets", "IAM_Sentences", "ascii", "sentences.txt")
sentences_folder_path = os.path.join("Datasets", "IAM_Sentences", "sentences")
dataset, vocab, max_len = [], set(), 0
words = open(sentences_txt_path, "r").readlines()
for line in tqdm(words):
if line.startswith("#"):
continue
line_split = line.split(" ")
if line_split[2] == "err":
continue
folder1 = line_split[0][:3]
folder2 = "-".join(line_split[0].split("-")[:2])
file_name = line_split[0] + ".png"
label = line_split[-1].rstrip("\n")
# replace "|" with " " in label
label = label.replace("|", " ")
rel_path = os.path.join(sentences_folder_path, folder1, folder2, file_name)
if not os.path.exists(rel_path):
print(f"File not found: {rel_path}")
continue
dataset.append([rel_path, label])
vocab.update(list(label))
max_len = max(max_len, len(label))
# Create a ModelConfigs object to store model configurations
configs = ModelConfigs()
# Save vocab and maximum text length to configs
configs.vocab = "".join(vocab)
Launchpad
Ln 34, Col 31
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

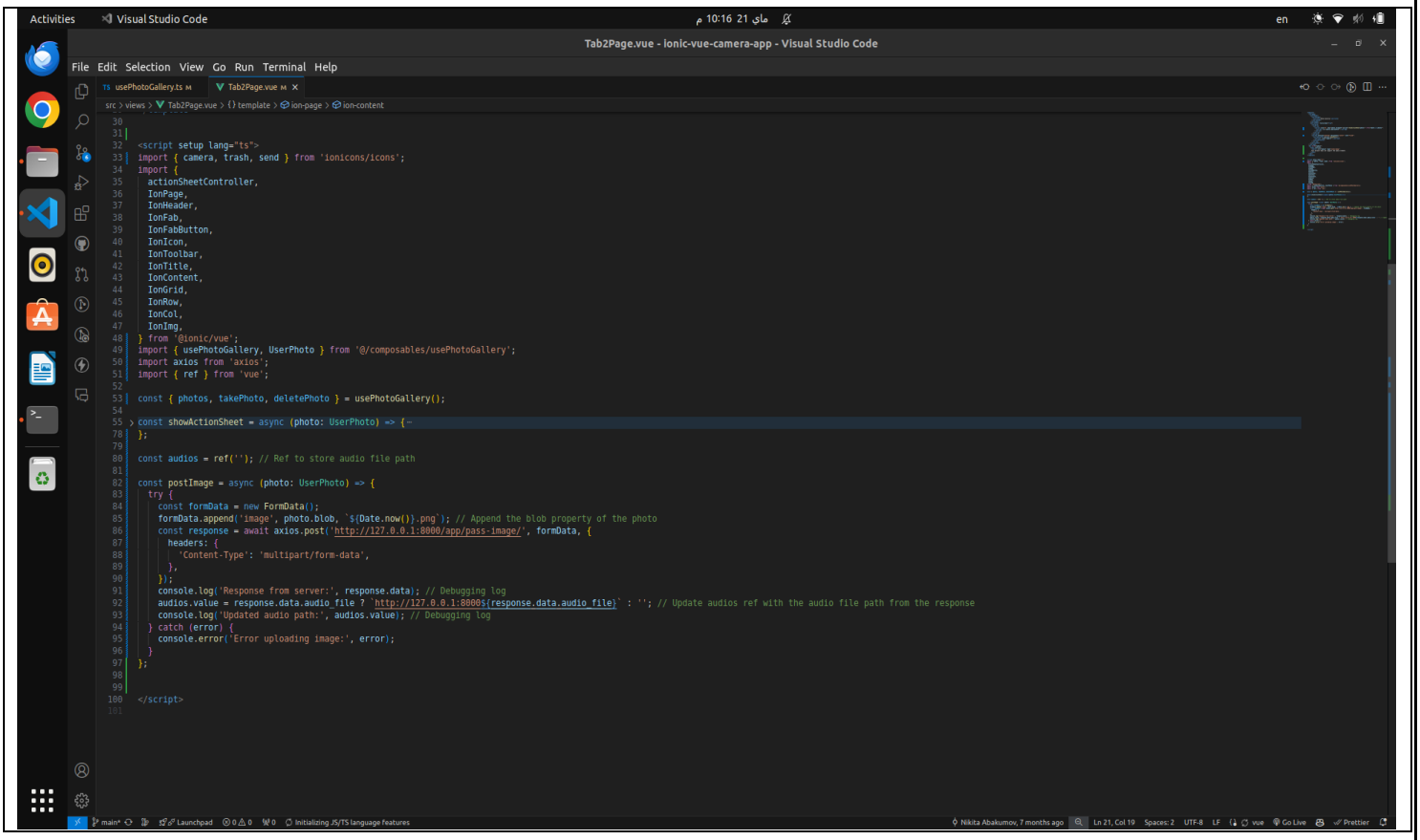

Activities
Visual Studio Code
10:15 21 ماي م
en
train.py - 03_sentence_recognition - Visual Studio Code
File Edit Selection View Go Run Terminal Help
train.py
# Create a data provider for the dataset
data_provider = DataProvider(
dataset=dataset,
skip_validation=True,
batch_size=configs.batch_size,
data_preprocessors=[ImageReader(CVImage)],
transformers=[
ImageResizer(configs.width, configs.height, keep_aspect_ratio=True),
LabelIndexer(configs.vocab),
LabelPadding(max_word_length=configs.max_text_length, padding_value=len(configs.vocab)),
],
)
Split the dataset into training and validation sets
train_data_provider, val_data_provider = data_provider.split(split = 0.9)
# Augment training data with random brightness, rotation and erode/dilate
train_data_provider.augmentors = [
RandomBrightness(),
RandomErodeDilate(),
RandomSharpen(),
]
# Creating TensorFlow model architecture
model = train_model(
input_dim = (configs.height, configs.width, 3),
output_dim = len(configs.vocab),
)
Compile the model and print summary
model.compile(
optimizer=tf.keras.optimizers.Adam(learning_rate=configs.learning_rate),
loss=CTCloss(),
metrics=[
CERMetric(vocabulary=configs.vocab),
WERMetric(vocabulary=configs.vocab)
],
run_eagerly=False
)
model.summary(line_length=110)
# Define callbacks
earlystopper = EarlyStopping(monitor="val_CER", patience=20, verbose=1, mode="min")
checkpoint = ModelCheckpoint(f"{configs.model_path}/model.h5", monitor="val_CER", verbose=1, save_best_only=True, mode="min")
trainLogger = TrainLogger(configs.model_path)
tb_callback = TensorBoard(f"{configs.model_path}/logs", update_freq=1)
reduceLROnPlat = ReduceLROnPlateau(monitor="val_CER", factor=0.9, min_delta=1e-10, patience=5, verbose=1, mode="auto")
model2onnx = Model2onnx(f"{configs.model_path}/model.h5")
# Train the model
model.fit(
train_data_provider,
validation_data=val_data_provider,
epochs=configs.train_epochs,
callbacks=[earlystopper, checkpoint, trainLogger, reduceLROnPlat, tb_callback, model2onnx],
workers=configs.train_workers
)
Launchpad
1 file to analyze
Ln 34, Col 31
Spaces: 4
UTF-8
LF
Python
3.10.12 64-bit
Go Live
Prettier

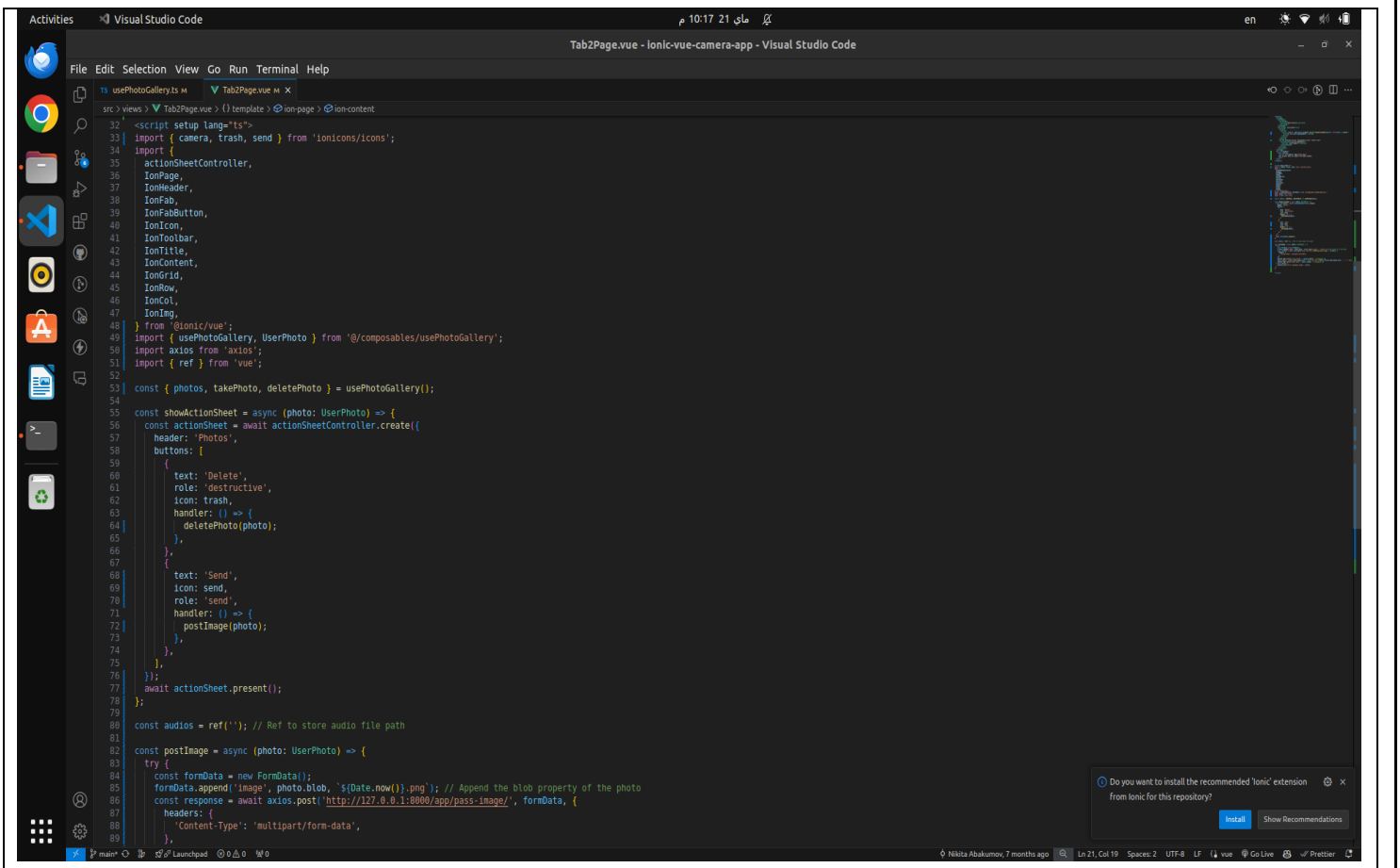

Activities
Visual Studio Code
10:15 21 ماي م
en
inferenceModel.py - 03_sentence_recognition - Visual Studio Code
File Edit Selection View Go Run Terminal Help
inferenceModel.py
import cv2
import typing
import numpy as np
from mltu.inferenceModel import OnnxInferenceModel
from mltu.utils.text_utils import ctc_decoder, get_cer, get_wer
from mltu.transformers import ImageResizer
class ImageToWordModel(OnnxInferenceModel):
def __init__(self, char_list: typing.Union[str, list], *args, **kwargs):
super().__init__(*args, **kwargs)
self.char_list = char_list
def predict(self, image: np.ndarray):
image = ImageResizer.resize_maintaining_aspect_ratio(image, *self.input_shape[:2][::-1])
image_pred = np.expand_dims(image, axis=0).astype(np.float32)
preds = self.model.run(None, {self.input_name: image_pred})[0]
text = ctc_decoder(preds, self.char_list)[0]
return text
if __name__ == "__main__":
import pandas as pd
from tqdm import tqdm
from mltu.configs import BaseModelConfigs
configs = BaseModelConfigs.load("Models/04_sentence_recognition/202301131202/configs.yaml")
model = ImageToWordModel(model_path=configs.model_path, char_list=configs.vocab)
df = pd.read_csv("Models/04_sentence_recognition/202301131202/val.csv").values.tolist()
accum_cer, accum_wer = [], []
for image_path, label in tqdm(df):
image = cv2.imread(image_path)
prediction_text = model.predict(image)
cer = get_cer(prediction_text, label)
wer = get_wer(prediction_text, label)
print("Image: ", image_path)
print("Label:", label)
print("Prediction: ", prediction_text)
print(f"CER: {cer}; WER: {wer}")
accum_cer.append(cer)
accum_wer.append(wer)
cv2.imshow(prediction_text, image)
cv2.waitKey(0)
cv2.destroyAllWindows()
print(f"Average CER: {np.average(accum_cer)}, Average WER: {np.average(accum_wer)}")

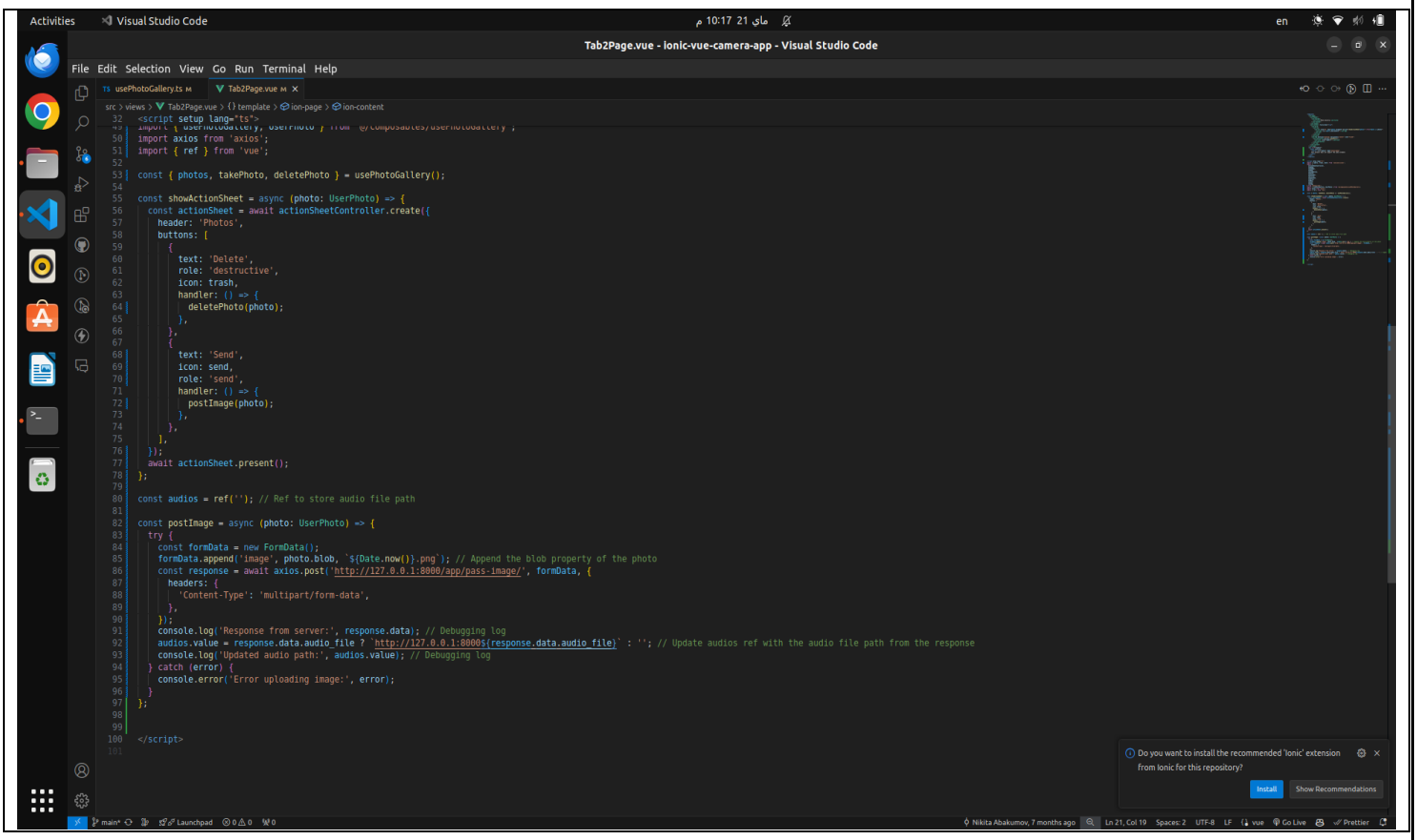

Activities
Visual Studio Code
10:16 21 ماي م
en
Tab2Page.vue - ionic-vue-camera-app - Visual Studio Code
File Edit Selection View Go Run Terminal Help
usePhotoGallery.ts M
Tab2Page.vue M
src > views > Tab2Page.vue
You, 4 days ago | 2 authors (Nikita Abakumov and others)
<template>
<ion-page>
<ion-header>
<ion-toolbar>
<ion-title>Photo Gallery</ion-title>
</ion-toolbar>
</ion-header>
<ion-content :fullscreen="true">
<ion-grid>
<ion-row>
<ion-col size="6" :key="photo.filepath" @click="showActionSheet(photo)" v-for="photo in photos">
<ion-img :src="photo.webviewPath"></ion-img>
</ion-col>
</ion-row>
</ion-grid>
<ion-fab vertical="bottom" horizontal="center" slot="fixed">
<ion-fab-button @click="takePhoto()">
<ion-icon :icon="camera"></ion-icon>
</ion-fab-button>
</ion-fab>
</ion-content>
Nikita Abakumov, 7 months ago • init added take photo button, photo gallery
</ion-page>

<audio controls>
<source :src="audios" type="audio/mpeg">
Your browser does not support the audio element.
</audio>

</template>
<script setup lang="ts">
import { camera, trash, send } from 'ionicons/icons';
import {
actionSheetController,
IonPage,
IonHeader,
IonFab,
IonFabButton,
IonIcon,
IonToolbar,
IonTitle,
IonContent,
IonGrid,
IonRow,
IonCol,
IonImg,
} from '@ionic/vue';
import { usePhotoGallery, UserPhoto } from '@/composables/usePhotoGallery';
import axios from 'axios';
import { ref } from 'vue';
const { photos, takePhoto, deletePhoto } = usePhotoGallery();
const showActionSheet = async (photo: UserPhoto) => {
};
const audios = ref(''); // Ref to store audio file path


Activities
Visual Studio Code
10:16 21 ماي م
en
Tab2Page.vue - ionic-vue-camera-app - Visual Studio Code
File Edit Selection View Go Run Terminal Help
usePhotoGallery.ts M
Tab2Page.vue M
src > views > Tab2Page.vue > {} template > ion-page > ion-content
<script setup lang="ts">
import { camera, trash, send } from 'ionicons/icons';
import {
actionSheetController,
IonPage,
IonHeader,
IonFab,
IonFabButton,
IonIcon,
IonToolbar,
IonTitle,
IonContent,
IonGrid,
IonRow,
IonCol,
IonImg,
} from '@ionic/vue';
import { usePhotoGallery, UserPhoto } from '@/composables/usePhotoGallery';
import axios from 'axios';
import { ref } from 'vue';
const { photos, takePhoto, deletePhoto } = usePhotoGallery();
const showActionSheet = async (photo: UserPhoto) => {
};
const audios = ref(''); // Ref to store audio file path
const postImage = async (photo: UserPhoto) => {
try {
const formData = new FormData();
formData.append('image', photo.blob, `${Date.now()}.png`); // Append the blob property of the photo
const response = await axios.post('http://127.0.0.1:8000/app/pass-image/', formData, {
headers: {
'Content-Type': 'multipart/form-data',
},
});
console.log('Response from server:', response.data); // Debugging log
audios.value = response.data.audio_file ? `http://127.0.0.1:8000${response.data.audio_file}` : ''; // Update audios ref with the audio file path from the response
console.log('Updated audio path:', audios.value); // Debugging log
} catch (error) {
console.error('Error uploading image:', error);
}
};
</script>
Nikita Abakumov, 7 months ago
Ln 21, Col 19
Spaces: 2
UTF-8
LF
vue
Go Live
Prettier


Activities
Visual Studio Code
10:17 21 ماي م
en
Tab2Page.vue - ionic-vue-camera-app - Visual Studio Code
File Edit Selection View Go Run Terminal Help
usePhotoGallery.ts M
Tab2Page.vue M
src > views > Tab2Page.vue > {} template > ion-page > ion-content
<script setup lang="ts">
import { camera, trash, send } from 'ionicons/icons';
import {
actionSheetController,
IonPage,
IonHeader,
IonFab,
IonFabButton,
IonIcon,
IonToolbar,
IonTitle,
IonContent,
IonGrid,
IonRow,
IonCol,
IonImg,
} from '@ionic/vue';
import { usePhotoGallery, UserPhoto } from '@/composables/usePhotoGallery';
import axios from 'axios';
import { ref } from 'vue';
const { photos, takePhoto, deletePhoto } = usePhotoGallery();
const showActionSheet = async (photo: UserPhoto) => {
const actionSheet = await actionSheetController.create({
header: 'Photos',
buttons: [
{
text: 'Delete',
role: 'destructive',
icon: trash,
handler: () => {
deletePhoto(photo);
},
},
{
text: 'Send',
icon: send,
role: 'send',
handler: () => {
postImage(photo);
},
},
],
});
await actionSheet.present();
};
const audios = ref(''); // Ref to store audio file path
const postImage = async (photo: UserPhoto) => {
try {
const formData = new FormData();
formData.append('image', photo.blob, `${Date.now()}.png`); // Append the blob property of the photo
const response = await axios.post('http://127.0.0.1:8000/app/pass-image/', formData, {
headers: {
'Content-Type': 'multipart/form-data',
},
Do you want to install the recommended 'Ionic' extension from Ionic for this repository?
Install
Show Recommendations
Nikita Abakumov, 7 months ago
Ln 21, Col 19
Spaces: 2
UTF-8
LF
vue
Go Live
Prettier



Text
I think it's like you know um more convenient too.
Um we have to pay have this security fee just in case she would damage something but um.
Everything is run by computer but you got to know how to think before you can do a computer.
As friends thing I definitely I've got more male friends.

References

https://www.who.int/

https://www.researchgate.net/publication/304802688_Smart_Glasses_for the_Visually_Impaired_People

https://medium.com/towards-data-science/image-to-text-ocr-with-tesseract-js-3540b420e0e7

https://azure.microsoft.com/en-us/products/ai-services/text-to-speech

https://www.academia.edu/33796786/Smart_glasses_for_blind_people

https://medium.com/@adityamahajan.work/easyocr-a-comprehensive-guide-5ff1cb850168