

**HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY OF INFORMATION TECHNOLOGY**



****

**FINAL TERM PROJECT**

**Topic:**

**BUILDING CONSTRUCTION OF APPLICATION FOR TEXT HANDWRITING RECOGNITION**

**Course name:** Digital Image Processing

**Lecturer name**: Assoc. Prof. Hoang Van Dung

**List of members:**

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Student name** | **Contribution (%)** |
| 22110059 | Ngô Tường Phát | 100% |
| 22110053 | Nguyễn Hà My | 100% |
| 20110423 | Nguyễn Thanh Minh Triết | 100% |

*Thu Duc City, 5/2025*

# Acknowledgment

*First and foremost, our group would like to express our sincere gratitude to Mr. Hoang Van Dung, who has enthusiastically guided and imparted invaluable knowledge throughout the duration of the Digital Image Processing course. His dedication and passion have not only helped us grasp many crucial and important concepts of the subject but also ignited our enthusiasm and love for studying, researching, and applying the knowledge from this course to real-world problems.*

*The knowledge and experience we have gained from this course will undoubtedly serve as a solid foundation, a robust base not only for our future career paths but also for many other aspects of life, where plain knowledge and theory alone are insufficient to address real challenges.*

*From the bottom of our hearts, we, Group 14, sincerely thank you, Mr. Hoang Van Dung!​​*

**FEEDBACK OF SUPERVISOR**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

**.............................................................................................................................................**

# Table of Contents

[**Acknowledgment 1**](#_heading=h.1fob9te)

[**Table of Contents 3**](#_heading=h.xvjgbmpeu23z)

[**List of figures 4**](#_heading=h.nxmpkcp6dyji)

[**List of tables 4**](#_heading=h.hl5rjxvxc14j)

[**CHAPTER 1. INTRODUCTION 5**](#_heading=)

[1.1 Reason for choosing the topic 5](#_heading=h.2et92p0)

[1.2 Project goals 5](#_heading=h.tyjcwt)

[1.3 Project mission 5](#_heading=h.3dy6vkm)

[**CHAPTER 2. BACKGROUND KNOWLEDGES 6**](#_heading=h.1t3h5sf)

[2.1 Implementation 6](#_heading=h.4d34og8)

[2.1.1 Tools 6](#_heading=h.c1vsj4hmezoo)

[2.1.2 Technologies 6](#_heading=h.zg0foyvdioh5)

[2.1.3 Testing method and techniques 6](#_heading=h.atu8v0ilpo6z)

[2.2 Using Black Box Testing 6](#_heading=h.2s8eyo1)

[2.2.1 General Test Scenarios: 6](#_heading=h.kxh82bpyx9x)

[2.2.2 Test Scenarios For A Window: 6](#_heading=h.wxtiim1h644z)

[2.2.3 Test Scenarios For Image Upload Functionality: 7](#_heading=h.gj7n81rp6eop)

[2.3 Convolutional recurrent neural network (CRNN) 7](#_heading=h.17dp8vu)

[2.3.1. Convolution Layers: 8](#_heading=h.sndiwhyqzspy)

[2.3.2. Recurrent Layer: 8](#_heading=h.b73stg4hp0pk)

[2.3.3. Transcription Layer: 9](#_heading=h.i5xj76rabwci)

[2.4 CTC Loss 9](#_heading=h.26in1rg)

[**CHAPTER 3. DESIGN SYSTEM 10**](#_heading=h.lnxbz9)

[3.1 Train Model 10](#_heading=h.35nkun2)

[3.2 Model Building 11](#_heading=h.3j2qqm3)

[3.3 Image Processing 13](#_heading=h.2xcytpi)

[3.3.1. Convert images to uniform brightness: 13](#_heading=h.uq6uq1xl9yng)

[3.3.2. Separating background and text: 13](#_heading=h.celofsip5uge)

[3.3.3. Noise removal: 14](#_heading=h.eh4zz3u59jec)

[3.3.4. Convert the image to the appropriate format: 14](#_heading=h.elnm28vz3jth)

[3.3.5. Resize the image: 14](#_heading=h.38zcxczgymve)

[3.3 Crop text line 21](#_heading=h.1pxezwc)

[3.4 Vietnamese OCR 23](#_heading=h.iyhw4m21h01x)

[**CHAPTER 4. IMPLEMENTATION & TESTING 25**](#_heading=h.2p2csry)

[4.1 Implementation 25](#_heading=h.147n2zr)

[4.2 Demo 25](#_heading=h.wbq7enb8uart)

[**CHAPTER 5. CONCLUSION 29**](#_heading=h.1hmsyys)

[5.1 Achievements 29](#_heading=h.20ptz2w7ab7t)

[5.2 Advantages 29](#_heading=h.2grqrue)

[5.3 Disadvantages 29](#_heading=h.3fwokq0)

[5.4 Future Development 29](#_heading=h.1v1yuxt)

[**ASSIGNMENT TABLE 30**](#_heading=h.4f1mdlm)

[**REFERENCES 31**](#_heading=h.nmf14n)

# List of figures

[Figure 1. Convolutional Recurrent Neural Network(CRNN) 7](#_heading=h.3rdcrjn)

[Figure 2. Train labels 9](#_heading=h.1ksv4uv)

[Figure 3. Show the model 9](#_heading=h.44sinio)

[Figure 4. Epochs 10](#_heading=h.2jxsxqh)

[Figure 5. Result of training 10](#_heading=h.z337ya)

[Figure 6. Model weights 10](#_heading=h.1y810tw)

[Figure 7. Model building 11](#_heading=h.4i7ojhp)

[Figure 8. File process\_image.py(1) 13](#_heading=h.1ci93xb)

[Figure 9. File process\_image.py(2) 1](#_heading=h.3whwml4)5

[Figure 10. File process\_image.py(3) 1](#_heading=h.2bn6wsx)6

[Figure 11. File process\_image.py(4) 1](#_heading=h.qsh70q)9

[Figure 12. File process\_image.py(5)](#_heading=h.3as4poj) 20

[Figure 13. Segmentation of text line 21](#_heading=h.49x2ik5)

[Figure 14. Vietnamese OCR](#_heading=h.3o7alnk) 23

[Figure 15.](#_heading=h.23ckvvd) [Test cases](#_heading=h.3o7alnk)25

[Figure 16. User interface](#_heading=h.ihv636) 25

[Figure 17. User interface (1)](#_heading=h.ihv636) 26

[Figure 18. User interface (2)](#_heading=h.ihv636) 26

[Figure 19. User interface (3)](#_heading=h.ihv636) 26

[Figure 20. User interface (4)](#_heading=h.ihv636) 27

[Figure 21. User interface (5)](#_heading=h.ihv636) 27

[Figure 22. User interface (6)](#_heading=h.ihv636) 27

# List of tables

Table 1. Accuracy Table28

Table 2. Assignment Table [29](#_heading=h.1ksv4uv)

# CHAPTER 1. INTRODUCTION

## 1.1 Reason for choosing the topic

Nowadays, information technology is rapidly advancing and becoming increasingly sophisticated. Its applications are deeply integrated into everyday life, making it an essential part of the modern, civilized world.

One notable application is in supporting users in the healthcare and insurance claim process. For example, when a doctor writes a prescription by hand, it must later be entered into the hospital’s online system so that the patient can submit it to their insurer for reimbursement after treatment.

The traditional process requires significant time and effort, as it relies on individuals manually reading and inputting handwritten prescriptions, often resulting in low accuracy and efficiency.

By applying information technology to management tasks, these operations can be streamlined, becoming faster, more accurate, and more convenient. Addressing this issue is critical to improving service quality and efficiency.

## 1.2 Project goals

* Accuracy: The application must ensure high accuracy in identifying Vietnamese words to avoid potentially confusing situations of words.
* Processing speed: The application needs to have a stable processing speed to respond promptly to users' use.
* Classification feature: The application needs to be able to classify Vietnamese words information by type to help users easily understand necessary information.
* Friendly interface: The application needs to have a friendly interface that is easy to user are not concern about technology

## 1.3 Project mission

* The technology used is to train a dataset of images of Vietnamese writing to identify and recognize Vietnamese writing.
* Limit:
* Some letters cannot be recognized
* Some letters have similar shapes and are easily confused
* Some images are blurry and hard to recognize

# CHAPTER 2. BACKGROUND KNOWLEDGES

## 2.1 Implementation

### **2.1.1 Tools**

* **Visual Studio Code** for coding.

### **2.1.2 Technologies**

* **CRNN, CTC Loss, Image Processing** are used for the entire project.
* **Python** is the main programming language.
* Use **tensorflow** to train model
* Use **Streamlit** to build interface for app
* Others : **opencv**, **numpy**, **matplotlib**, **vietocr**, **pyperclip**.

### **2.1.3 Testing method and techniques**

**Black box:**

* Black box testing is a method of software testing done without knowing the internal structure of the software, the way that the tester checks the system as a black box, with no way of seeing. inside of the box.
* It is also known as the data direction test or in/out direction test.
* The tester should build groups of input values that will fulfill all of the functional requirements of the program.
* The testers' approach to the system is to not use any knowledge of the programming structure inside the system, see the system as a complete structure, can not interfere inside.

## 2.2 Using Black Box Testing

### **2.2.1 General Test Scenarios:**

- Input fields should be checked for the max field value. Input values greater than the specified max limit should not be accepted or stored in the database.

### **2.2.2 Test Scenarios For A Window:**

- Check if the window is resizable.

### **2.2.3 Test Scenarios For Image Upload Functionality:**

- Check image upload and change functionality.

- Check image upload functionality with file types other than images (For Example, txt, doc, pdf, exe, etc.). A proper error message should be displayed.

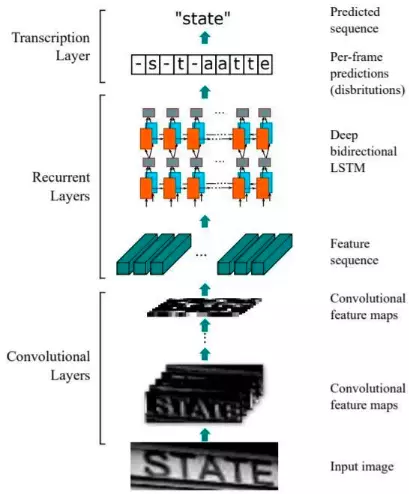
- Check duplicate name image upload.

## 2.3 Convolutional recurrent neural network (CRNN)

Convolutional Recurrent Neural Network is an upgraded neural network structure in deep learning, used for problems related to natural language processing and character recognition.

In CRNN architecture, layered CNNs are used to extract specific input images, while layered RNNs are used to process string characters generated by CNN layers. This allows the CRNN model to automatically learn specific images and systems between characters in a character string without the need for the user's print box.

The convolutional or convolutional layer in the CNN model performs the extraction of specific image inputs by performing analyzes between filters and image inputs. Pooling layers are used to reduce the size of the output matrix after going through the convolutional layer.



*Figure 1. Convolutional Recurrent Neural Network(CRNN)*

### **2.3.1. Convolution Layers:**

With input images passed through the CNN layer to help extract special images, the output of the CNN layer is a feature map.

### **2.3.2. Recurrent Layer:**

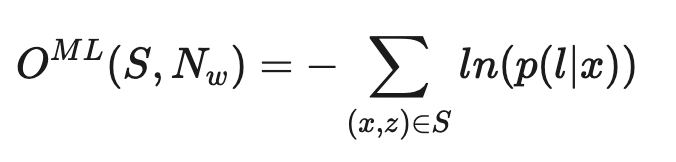
The output from the CNN layer is fed as sequences into recurrent layers. Recurrent layers include deep two-dimensional LSTM (long short-term memory) networks. RNN is capable of capturing contextual information very well within a sequence. The output from the RNN layer will include performance values for each label corresponding to each input characteristic (input feature).

### **2.3.3. Transcription Layer:**

The last component is the transcription layer. It uses Connectionist temporal classification (CTC) to predict the output for each step. We will take an example to better understand how to use CTC loss.

The output of the RNN will be a unique representation of the text (character by character). Suppose we have an input image with the text "good", the output of the RNN is ['g', 'g','o','o','o','d','d', 'd'] . To predict the output text, we can attract similar characters, then the predicted word will be "god" and the result will be completely wrong so the text input. CTC solves this problem by adding spaces between consecutive similar characters. And when decoding, CTC will get rid of those character spaces and give "good" output.

## 2.4 CTC Loss



CTC Loss is used to calculate the error between the model's expected output and the actual output (ground truth).

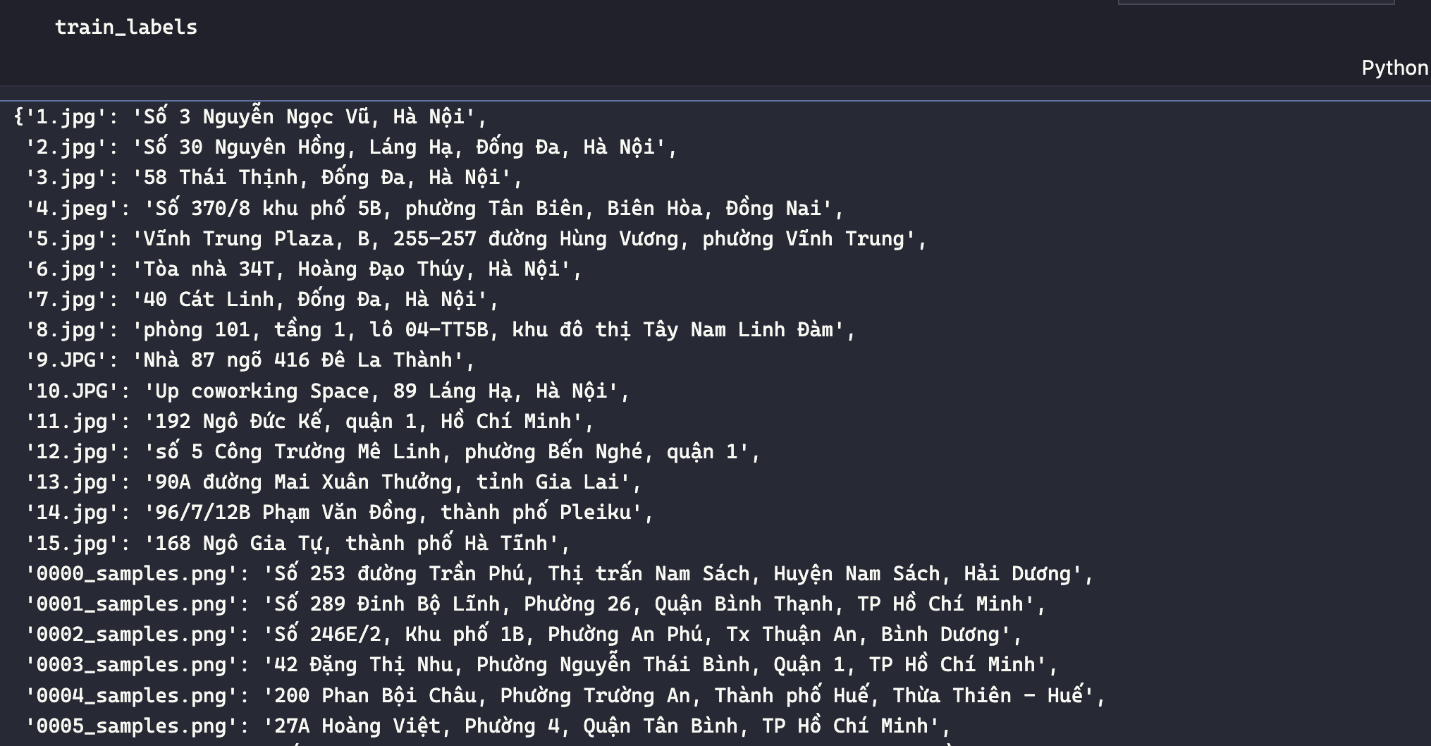
CTC Loss calculates the error by using a matrix combination (alignment matrix) to map the model's expected character string beginnings to the actual character string beginnings. The error is then calculated by comparing this matrix combination with the beginning of the string character.

# CHAPTER 3. DESIGN SYSTEM

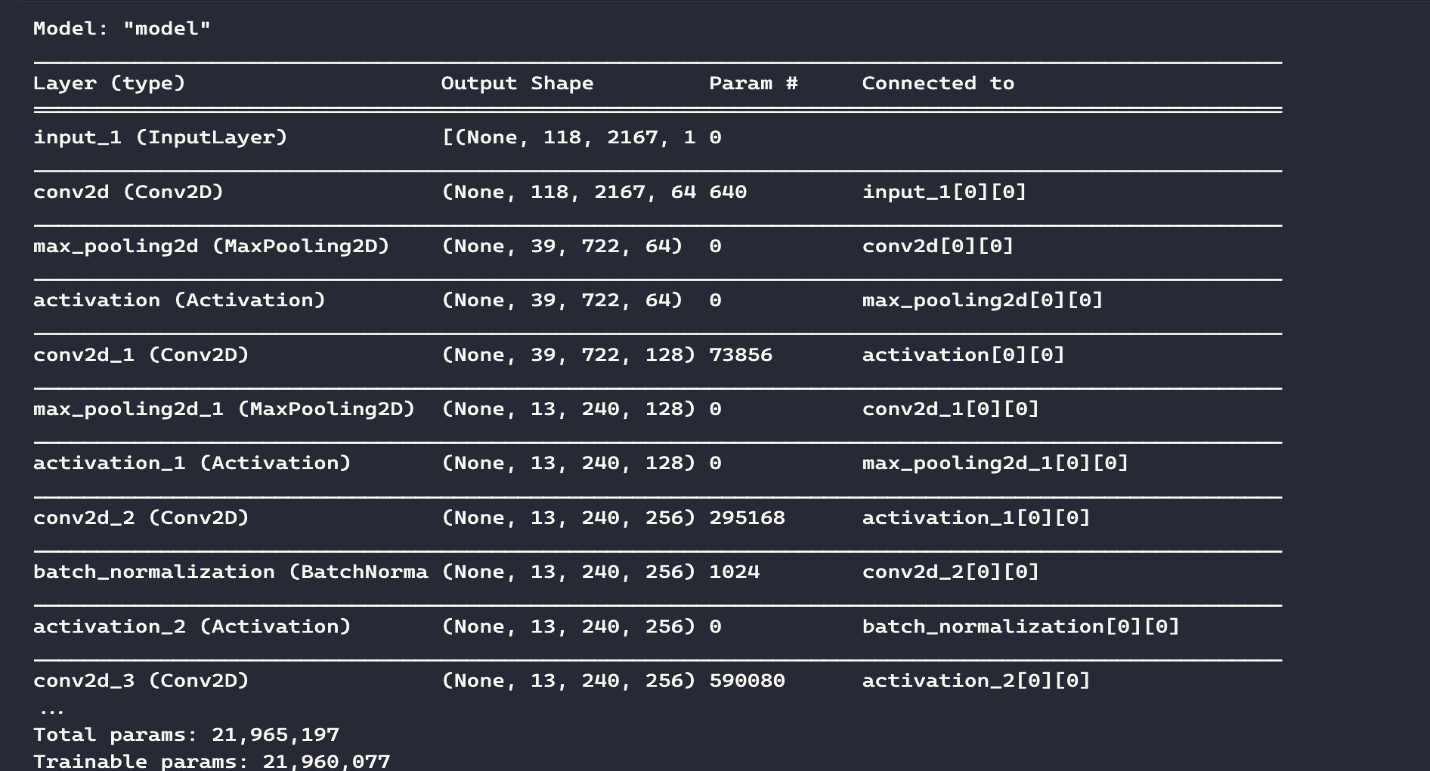
## 3.1 Train Model

Use Google Colab to train. Google Colab offers a number of free computational resources to users, including CPUs, GPUs, and TPUs, making model training faster and more efficient. In addition, users can also share their notebooks with others and work together on the same notebook.

Open notebook to train model:

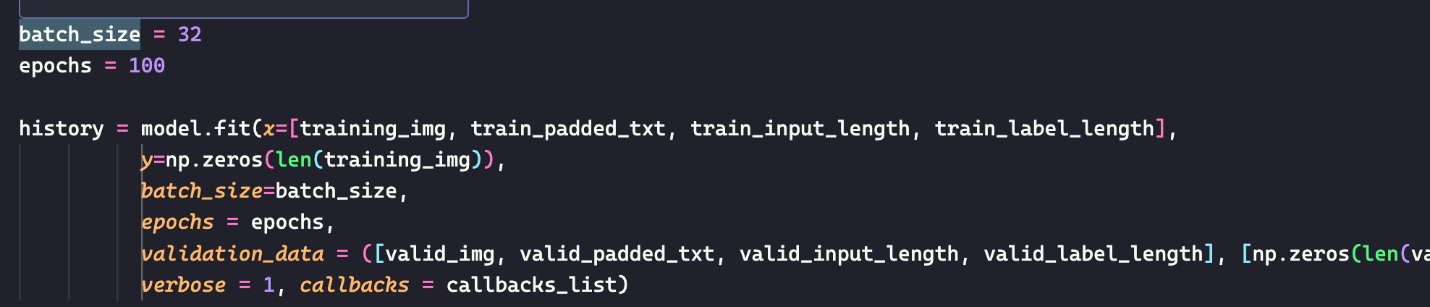


*Figure 2. Train labels*



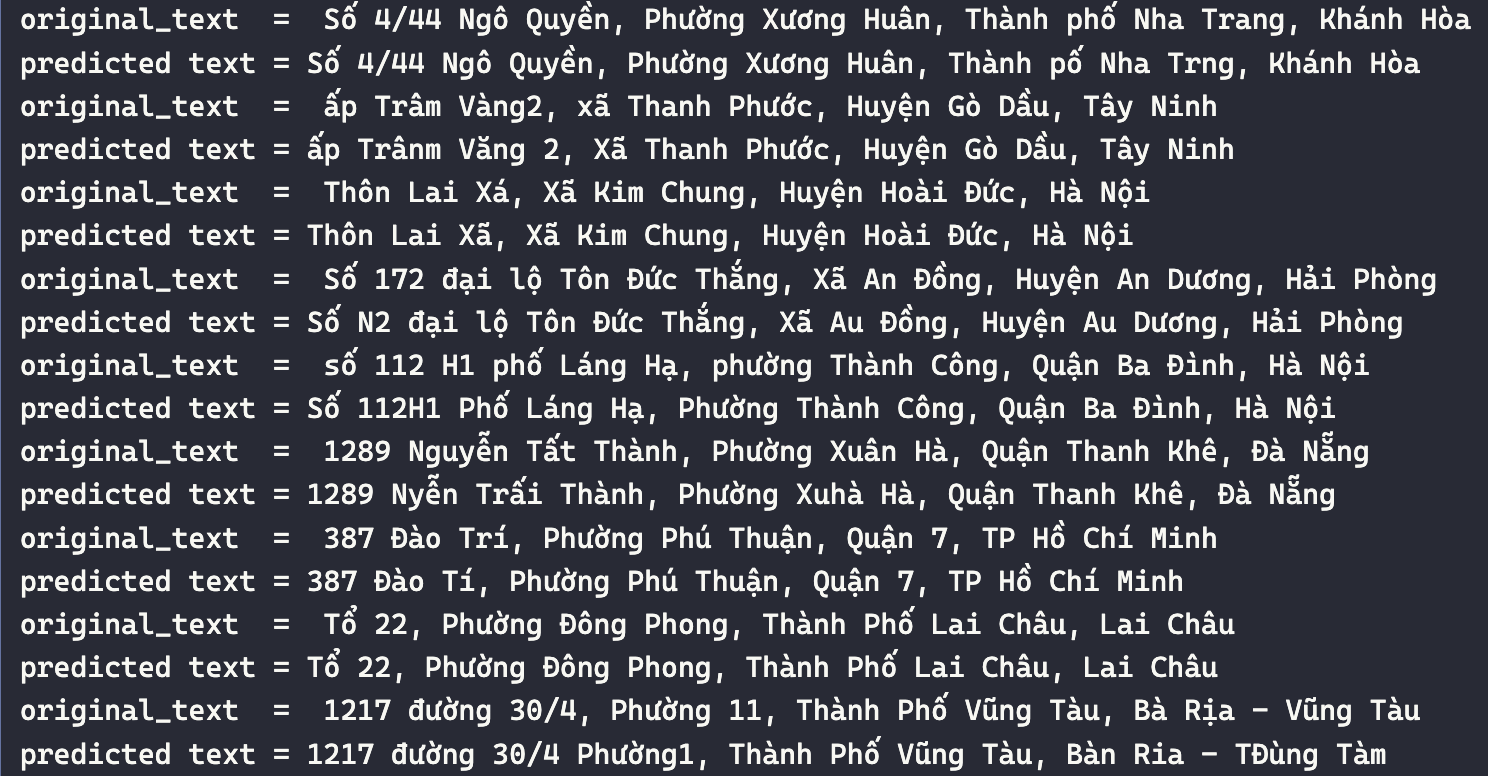
*Figure 3. Show the model*

We train the data with size 32 and number of epochs are 100.



*Figure 4. Epochs*

The result:



*Figure 5. Result of training*

## 3.2 Model Building

Base on the model file :



*Figure 6. Model weights*

We build the model :

A screenshot of a computer program

AI-generated content may be incorrect.

*Figure 7. Model building*

**Imports:** The necessary libraries are imported, including TensorFlow and Keras.

**Model Architecture:**

Input Layer: The input layer is defined with a shape of (118, 2167, 1), indicating an image with dimensions 118x2167 and a single channel (grayscale).

**Convolutional Blocks:**

Several convolutional blocks are defined, each consisting of a convolutional layer followed by activation functions (ReLU) and in some cases, batch normalization.

These blocks extract features from the input image.

Pooling Layers: Max-pooling layers are applied to reduce spatial dimensions.

**Lambda Layer:** A lambda layer is used to remove the first dimension (batch size) from the output tensor and reshape it.

**Bidirectional LSTM Layers:**

Two Bidirectional Long Short-Term Memory (LSTM) layers with 512 units each are defined. These layers capture temporal dependencies in the data.

Dropout regularization with a rate of 0.2 is applied to prevent overfitting.

**Output Layer**: A dense layer with a softmax activation function is used to output probabilities for each character. The number of units is 140+1, where 140 represents the number of characters in the output vocabulary and 1 is added for the blank symbol.

**Model Compilation:** Model Loading: Pre-trained weights for the model are loaded from the file ./data/model\_checkpoint\_weights.hdf5.

## 3.3 Image Processing

Preprocessing images before recognition Some methods of processing input images when recognizing writing through images:

### **3.3.1. Convert images to uniform brightness:**

In some cases, the input image has uneven brightness, affecting recognition ability. Convert images to uniform brightness by applying algorithms such as Histogram Equalization, Adaptive Histogram Equalization...

### **3.3.2.** **Separating background and text:**

Separating background and text helps eliminate unnecessary details and makes the forum identification process more accurate. Common methods for separating background and text include Threshold and Morphology operations.

### **3.3.3. Noise removal:**

For images with noise, noise removal makes the recognition process more accurate. Common methods for noise removal include Median Filtering and Gaussian Filtering.

### **3.3.4. Convert the image to the appropriate format:**

After processing the image, we need to convert the image to the appropriate format for use in recognition work. Popular formats include JPG, PNG.

### **3.3.5. Resize the image:**

To reduce processing time and ensure recognition on devices with different size screens, we should resize the image to match the previous size when identifying server-side code in addition to the client-side code without the need to learn a completely different language.

**process\_multi(segments) Function:**

A screen shot of a computer code

AI-generated content may be incorrect.

*Figure 8. File process\_image.py(1)*

**Input:**

**segments**: It is a list of image segments.

**Variables Initialization:**

**copy\_valid:** An empty list initialized to store processed images.

**size:** A variable initialized to keep track of the number of processed images.

**valid\_imgs:** An empty list to store the processed images.

**Image Processing Loop:**

The function iterates over each image segment in the segments list.

For each segment, it calls a function process\_image\_mul(img) to process the image. The processed image is then appended to the valid\_imgs list.

It also increments the size counter.

**Optional Visualization (Commented Out):**

There are some commented-out lines that seem to be used for visualization using matplotlib, likely to show the processed images. However, they are not executed in the current code.

**Copying List:**

After processing all images, the function creates a copy of the valid\_imgs list and assigns it to copy\_valid. This is done to preserve the original list since it will be converted to a numpy array in the next step.

**Conversion to Numpy Array:**

The valid\_imgs list is converted into a numpy array using np.array(valid\_imgs). This creates a numpy array containing all the processed images.

**Return:**

The function returns three values: the numpy array of processed images (valid\_imgs), a copy of the list of processed images (copy\_valid), and the size of the list (size). This allows the caller to access the processed images, the copy of the processed images, and the number of processed images.

**process\_image\_mul(cv2\_img) Function:**

A computer screen shot of text

AI-generated content may be incorrect.

*Figure 9. File process\_image.py(2)*

**Input:**

cv2\_img: This appears to be an image represented as a NumPy array, likely loaded using OpenCV's cv2.imread() function.

**Grayscale Conversion:**

The input image is converted from color (BGR) to grayscale using cv2.cvtColor().

**Bilateral Filtering:**

Bilateral filtering is applied to the grayscale image using cv2.bilateralFilter(). This is a noise-reducing smoothing filter that preserves edges.

**Adaptive Thresholding:**

Adaptive thresholding is performed on the filtered image using cv2.adaptiveThreshold(). It converts the grayscale image into a binary image, enhances the contrast and makes it easier to segment objects from the background.

**Padding:**

The padding\_image() function is called to pad the image to a predefined size (height = 118, width = 2167). This likely pads the image with zeros to achieve the desired dimensions.

**Resizing and Padding:**

The image is resized to have a height of 118 pixels while maintaining the aspect ratio. Any additional width beyond 2167 pixels is padded with the median value of the image.

**Dilation:**

Dilation operation is applied to the image using cv2.dilate() with a 1x1 kernel to enhance the features and thicken the characters.

**Normalization:**

The image is normalized by dividing by 255, which brings pixel values to the range [0, 1].

**Output:**

The processed image is returned as a NumPy array with an additional singleton dimension (axis=2), making it suitable for feeding into neural networks.

**process\_image(img\_file) Function:**

A computer screen shot of text

AI-generated content may be incorrect.

*Figure 10. File process\_image.py(3)*

**Input:**

img\_file: It appears to be an image file represented as a NumPy array, likely loaded using OpenCV's cv2.imread() function.

**Image Conversion:**

The image is converted from color (BGR) to grayscale using cv2.cvtColor().

**Resizing and Padding:**

The image is resized to have a height of 118 pixels while maintaining the aspect ratio. Any additional width beyond 2122 pixels is padded with the median value of the image.

**Gaussian Blurring:**

Gaussian blur is applied to the image using cv2.GaussianBlur(). This helps to smooth the image and reduce noise.

**Adaptive Thresholding:**

Adaptive thresholding is performed on the blurred image using cv2.adaptiveThreshold(). This converts the grayscale image into a binary image, enhances the contrast and makes it easier to segment objects from the background.

**Normalization:**

The image is normalized by dividing by 255, which brings pixel values to the range [0, 1].

**Output:**

The processed image is returned as a NumPy array with an additional singleton dimension (axis=2), making it suitable for feeding into neural networks.

**load\_original\_img(path) Function:**

**Input:**

path: Path to the image file.

Image Loading:

The function loads the image using cv2.imread() given the file path.

**Output:**

Returns the loaded image as a NumPy array.

**crop\_image(image, width, height) Function:**

A computer code with text

AI-generated content may be incorrect.

*Figure 11. File process\_image.py(4)*

**Input:**

**image:** The input image to be cropped, represented as a NumPy array.

**width:** The desired width of the cropped image.

**height:** The desired height of the cropped image.

**Image Dimensions Extraction:**

The height and width of the input image are extracted using image.shape[:2].

**Condition Check:**

A condition is checked to ensure that the dimensions of the input image are greater than the desired dimensions for cropping (width and height).

**Cropping:**

If the image is larger than the specified dimensions, the cropping process proceeds:

Calculate the starting pixel indices (startx and starty) for the crop. These are calculated by subtracting half of the desired width and height from the center of the image.

Use NumPy array slicing to extract the cropped region from the original image based on the calculated starting indices and the specified width and height.

If the image is smaller than the specified dimensions, no cropping is performed, and the original image is returned.

**Return:**

The function returns the cropped image if cropping is performed, otherwise, it returns the original image.

**erosion\_dilation\_image(image, kernel\_size, isErosion) Function:**

A computer screen shot of text

AI-generated content may be incorrect.

*Figure 12. File process\_image.py(5)*

**Input:**

**image:** The input image on which erosion or dilation operation will be applied.

**Kernel\_size:** The size of the square-shaped kernel used for erosion or dilation.

**isErosion:** A boolean flag indicating whether erosion operation should be performed (True) or dilation operation (False).

**Kernel Creation:**

A square-shaped kernel of size kernel\_size is created using NumPy's np.ones() function. This kernel will be used for erosion or dilation.

**Erosion or Dilation Operation:**

If isErosion is True, erosion operation is performed on the input image using cv2.erode() function. Erosion removes pixels at the boundaries of objects in the image, which helps in shrinking the objects.

If isErosion is False, dilation operation is performed on the input image using cv2.dilate() function. Dilation adds pixels to the boundaries of objects in the image, which helps in expanding the objects.

**Return:**

The resulting image after erosion or dilation operation is returned.

## 3.3 Crop text line

A computer screen shot of a program code

AI-generated content may be incorrect.

*Figure 13. Segmentation of text line*

**Function Definition: segmentation\_text\_line(image):**

**Input:** image - The input image to be segmented.

**Output:** Returns a tuple (img\_clone, segments) where:

**img\_clone:** A copy of the input image with bounding boxes drawn around the segmented text lines.

**segments:** A list containing the segmented text lines.

**Preprocessing:**

The input image is copied to img\_clone visualization for purposes.

The input image is converted to grayscale using cv2.cvtColor() and then blurred using cv2.GaussianBlur() to reduce noise.

**Thresholding:**

Adaptive thresholding is applied to the blurred image using cv2.adaptiveThreshold() to create a binary image (thresh). This helps in separating foreground (text) from the background.

**Erosion:**

Erosion operation is applied to the binary image (thresh) using cv2.erode() to reduce the thickness of the text regions.

**Dilation:**

Dilation operation is applied to the eroded image (ero) using cv2.dilate() to merge nearby text regions.

**Contour Detection:**

Contours are detected in the dilated image using cv2.findContours().

**Contour Sorting and Filtering:**

The contours are sorted based on their y-coordinate.

Each contour's bounding box is calculated, and contours are filtered based on minimum width and height thresholds (min\_width and min\_height).

Bounding boxes of filtered contours are drawn on img\_clone, and the corresponding text regions are extracted and added to the segments list.

**Return:**

The function returns the image with drawn bounding boxes (img\_clone) and the segmented text lines (segments).

## 3.4 Vietnamese OCR

A computer screen shot of a program code

AI-generated content may be incorrect.

*Figure 14. Vietnamese OCR*

**Definitions:**

Definitions of various constants and lists:

**NO\_PREDICTS**: Number of predictions.

**OFFSET:** Offset value used for prediction.

**char\_list:** List of characters used for decoding predictions.

**prediction\_ocr(valid\_img):**

This function takes a single image (valid\_img) as input.

It predicts the output using the CRNN model (crnn\_model.model.predict()).

The predicted output is decoded using Connectionist Temporal Classification (CTC) decoding (tf.keras.backend.ctc\_decode()).

The decoded predictions are converted into characters using the char\_list.

The function returns the predicted text.

**prediction\_ocr\_multi(valid\_img, SIZE):**

This function takes multiple images (valid\_img) and their size (SIZE) as input.

It predicts the output for multiple images using the CRNN model.

Decoding and character conversion are performed similarly to prediction\_ocr().

The function returns a string containing all the predicted texts separated by newline characters.

**Explanation:**

These functions are used to predict text from input images using a CRNN model.

The CRNN model has been trained separately and is imported from the crnn\_model module.

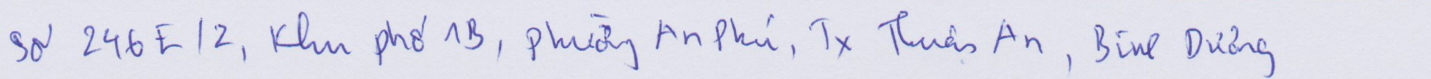
The char\_list is used for decoding the predicted output into readable text.

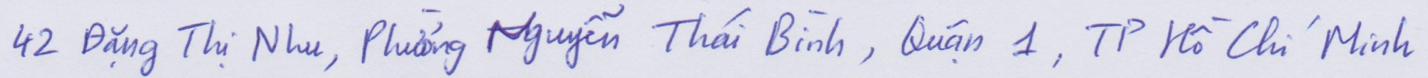
The prediction functions handle single or multiple images and return the predicted texts.

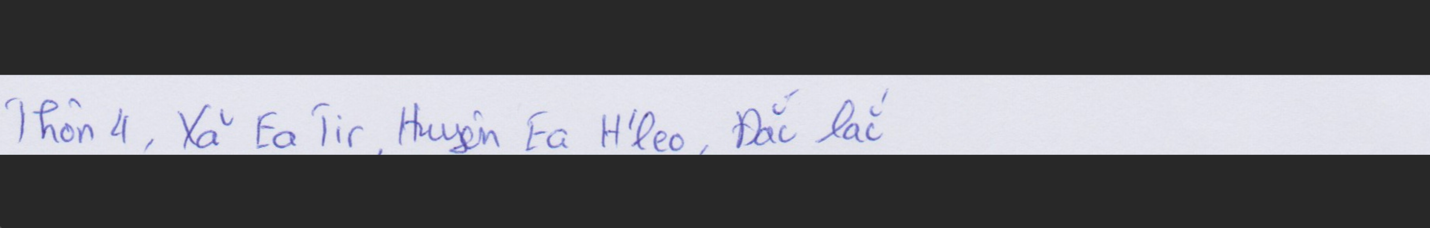
# CHAPTER 4. IMPLEMENTATION & TESTING

## 4.1 Implementation

We use the dataset that we train to check the accuracy of training data.



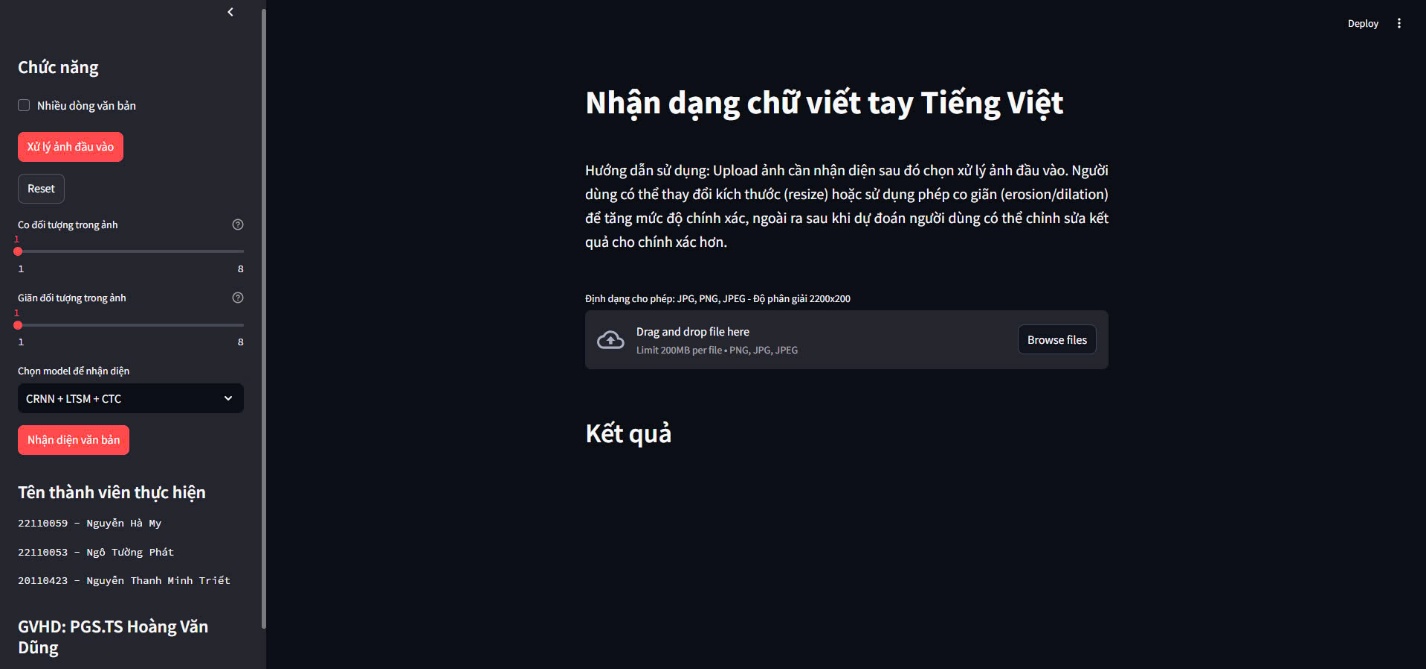




*Figure 15. Test cases*

## 4.2 Demo

* + **User Interface:** The user interface will show the streamlit site to show the app. It has:
* The function to input the file
* The button to process the image before detection
* The button to choose the model to detect

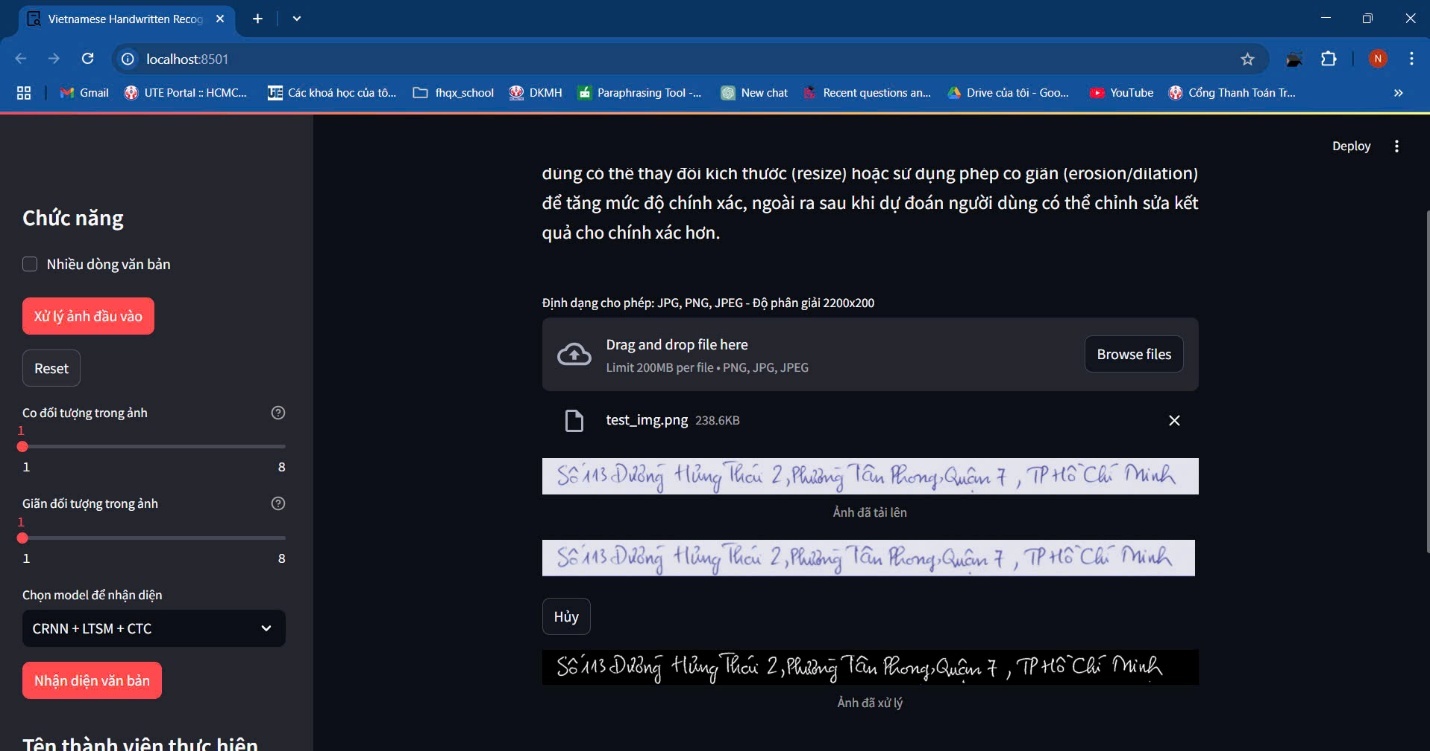


*Figure 16. User interface*

* + **Detect image:**
* **Browse image**: Click or drag to put input file

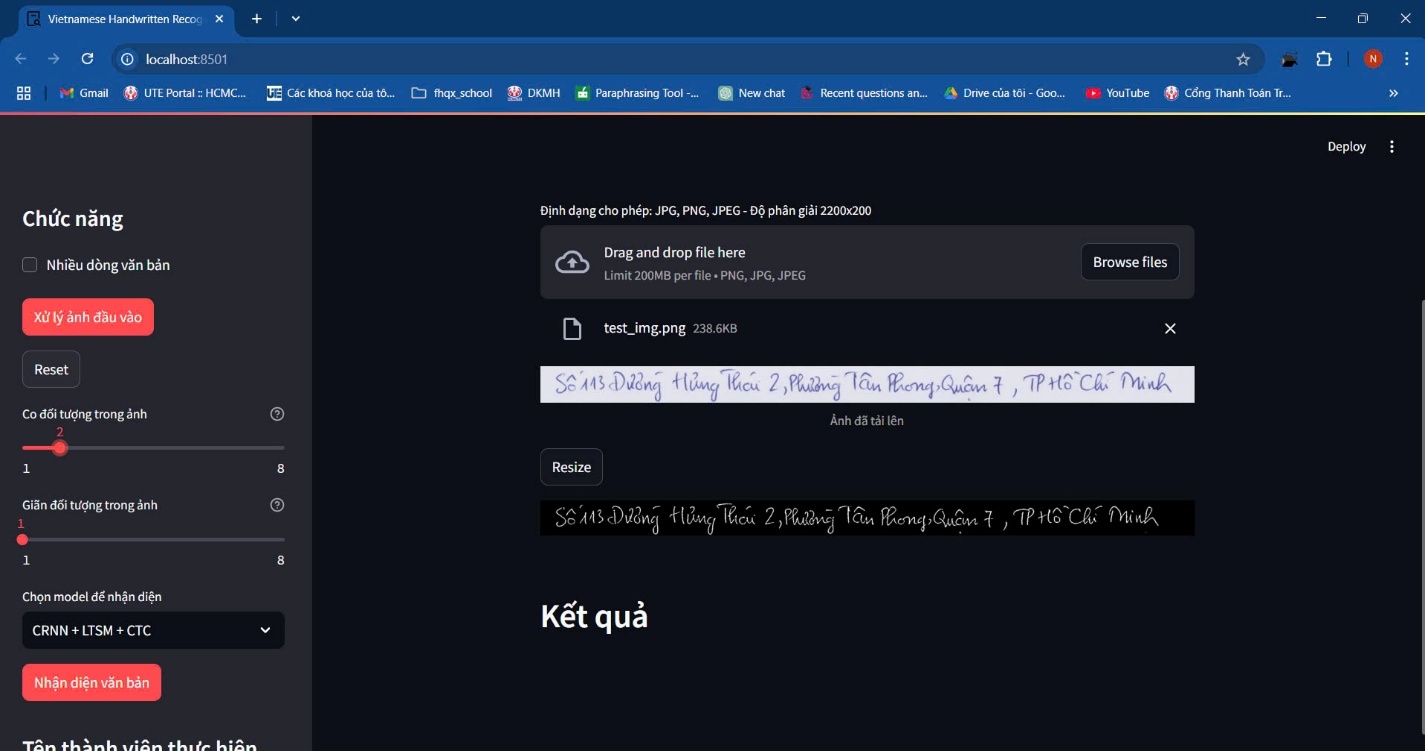


*Figure 17. User interface(1)*



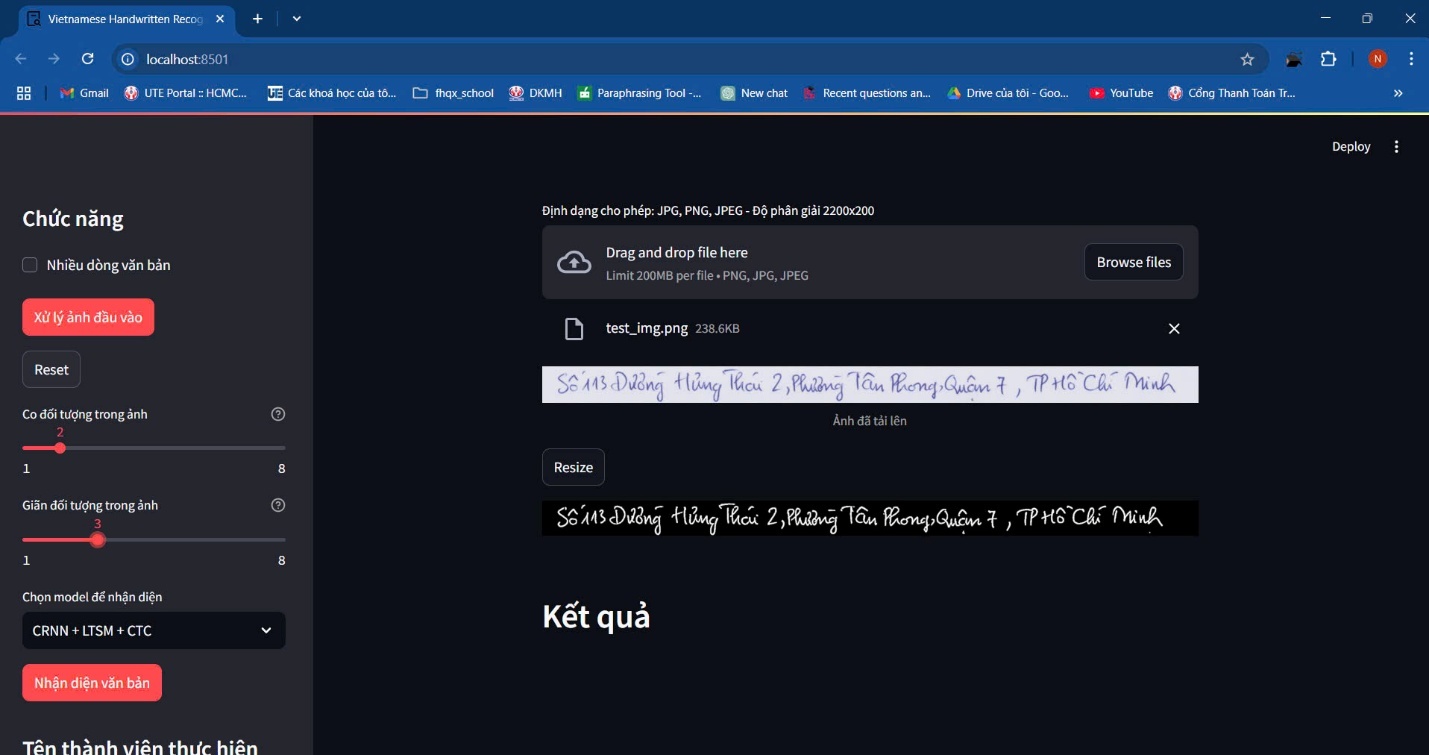
*Figure 18. User interface(2)*

* **Process input image:** Click process input images



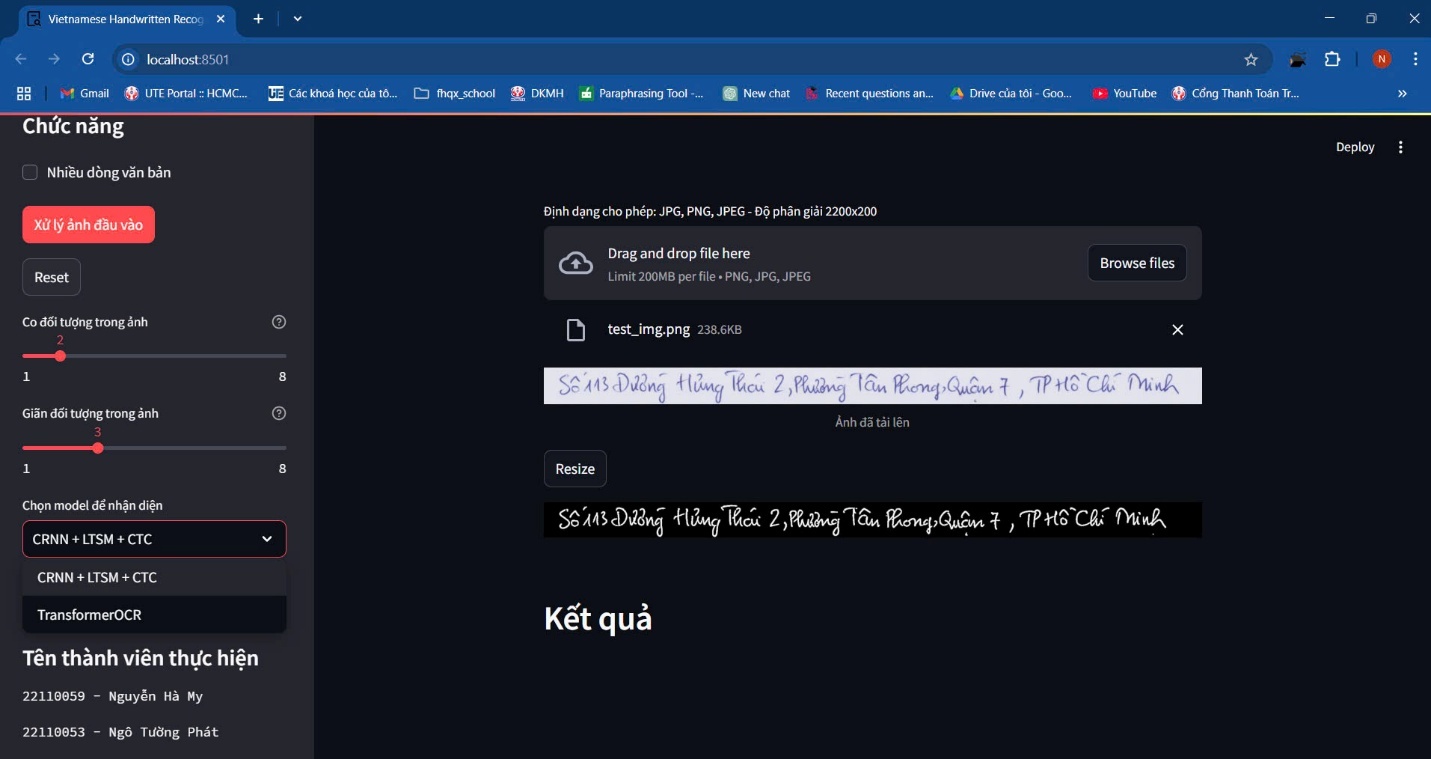
*Figure 19. User interface (3)*

* **Stretch the object:** We can use stretch object

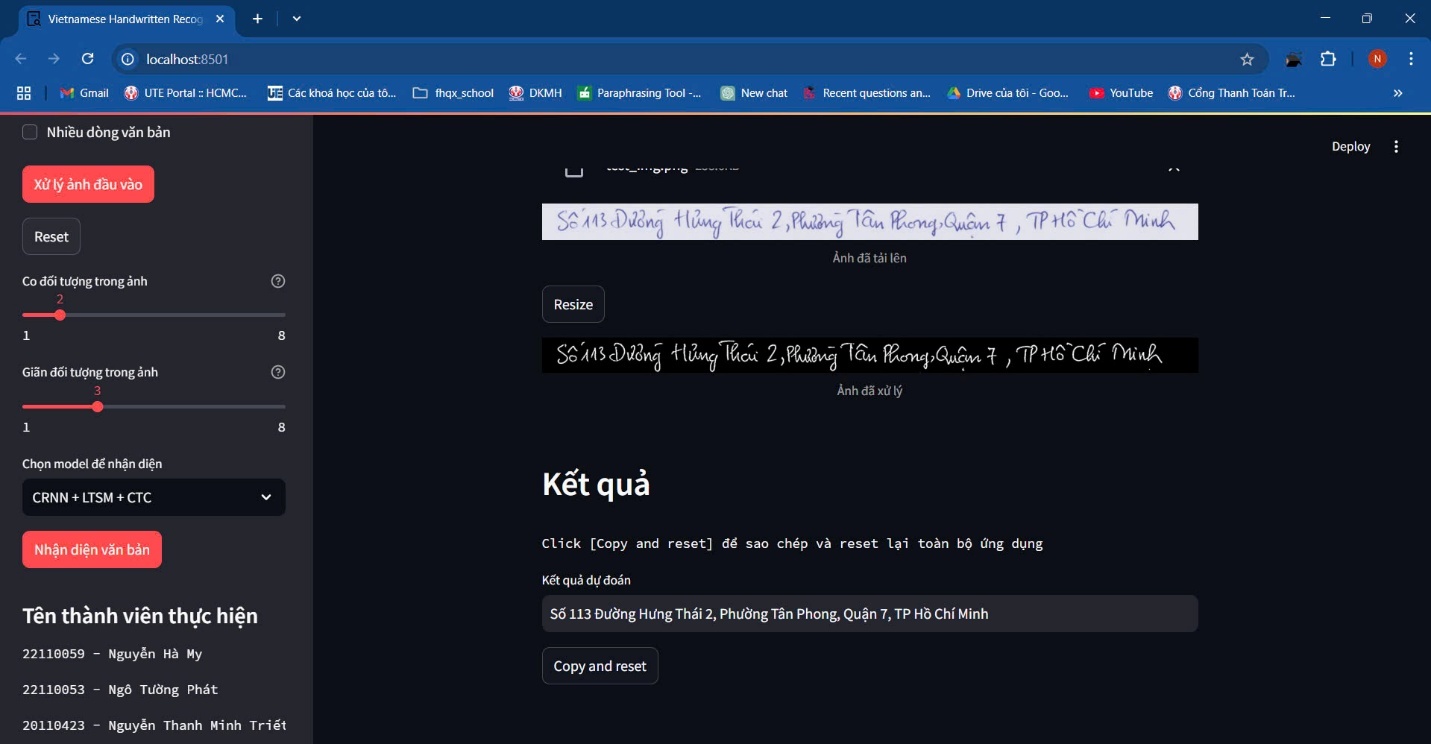


*Figure 20. User interface (4)*

* **Choose model to detect:**

*Figure 21. User interface (5)*

* **Detection:** Click test detection to detect and show the result



*Figure 22. User interface (6)*

**Application Features:**

* **Image Upload:**Users can easily upload images with handwritten text using a drag-and-drop interface or by selecting the “Browse Files” option.
* **Text Detection and Recognition:  
  The app employs a CRNN + LSTM + CTC model to accurately identify and extract text from uploaded images.**
* **Result Display:**Extracted text is shown directly beneath the uploaded image for user review.
* **Editing Tools:**Users can adjust the image by “stretching” it to enhance text recognition accuracy and use the “Reset” button to clear results or begin anew.

**How to Use the Application:**

1. Launch the app and select an image with handwritten text for upload.
2. Upload the image by dragging and dropping it into the interface or clicking “Browse Files” to select from your device.
3. Allow the app to process the image and extract the text.
4. Check the recognized text, which will appear below the uploaded image.
5. Use the editing tools to adjust the image if better recognition is needed.
6. Click “Reset” to clear the results and upload a new image.

**Note:** The accuracy of text recognition depends on the image quality and handwriting clarity. Ensure the image is sharp, and the text is legible and not too small or obscured.

# CHAPTER 5. CONCLUSION

## 5.1 Achievements

The development of the handwritten text recognition application has provided the team with valuable insights and skills, significantly enhancing our technical and collaborative capabilities:

1. Gained in-depth understanding of the software development lifecycle, from conceptualization to implementation and testing.
2. Strengthened teamwork and communication skills through collaborative problem-solving and task delegation.
3. Mastered advanced programming techniques using cutting-edge technologies such as CRNN, CTC Loss, and image processing libraries.
4. Developed practical experience in training deep learning models and optimizing them for real-world applications.
5. Improved research skills by exploring and adapting state-of-the-art OCR methodologies for Vietnamese handwritten text.

## 5.2 Advantages

The application offers several strengths that make it a valuable tool for users:

* Features an intuitive and user-friendly interface, designed to be accessible even for non-technical users.
* Provides core functionalities, such as image upload, text recognition, and editing tools, that effectively meet user needs.
* Utilizes a robust CRNN + LSTM + CTC model, achieving high accuracy (95%–100%) for Vietnamese handwritten text recognition.
* Supports flexible image preprocessing, enhancing recognition performance for varied image qualities.

## 5.3 Disadvantages

Despite its strengths, the application has areas that require further improvement:

* The user interface, while functional, lacks polished aesthetics and could benefit from a more modern design.
* Image prediction occasionally produces errors, particularly with low-quality or complex handwritten inputs.
* The TransformerOCR model underperforms compared to CRNN, with accuracy ranging between 50%–70%, limiting its reliability.
* Processing speed can be slow for large or high-resolution images, impacting user experience on lower-end devices.
* Limited support for diverse handwriting styles and non-Vietnamese text reduces its versatility.

# 

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy** | **Result** |
| CRNN + LTSM + CTC | 95% - 100% |  |
| TransformerOCR | 50% - 70% |  |

Table 1. Accuracy Table

## 5.4 Future Development

To enhance the application and address its current limitations, the team plans to implement the following improvements and expansions in future iterations:

1. **Mobile Application Support:**  
   Develop a mobile version of the application for iOS and Android platforms, ensuring accessibility and convenience for users on smartphones and tablets. This will include optimizing the interface for smaller screens and touch-based interactions.
2. **Improved Image Prediction Accuracy:**  
   Refine the image processing and text recognition algorithms to minimize errors in text prediction. This may involve incorporating advanced preprocessing techniques, such as enhanced noise reduction and adaptive thresholding, to handle challenging inputs like blurry or low-quality images.
3. **Higher Accuracy with TransformerOCR:**  
   Investigate and integrate improvements to the TransformerOCR model to boost its accuracy, aiming to match or surpass the performance of the CRNN + LSTM + CTC model (currently at 95%–100% accuracy). This could include fine-tuning the model with a larger and more diverse dataset of Vietnamese handwritten text.
4. **Enhanced User Interface Design:**  
   Redesign the user interface to improve aesthetics and usability, making it more visually appealing and intuitive. This includes streamlining navigation, adding interactive tooltips, and ensuring consistency across different devices.
5. **Support for Multilingual Recognition:**  
   Expand the application’s capabilities to recognize handwritten text in multiple languages beyond Vietnamese, catering to a broader user base and increasing its applicability in diverse contexts.
6. **Real-Time Recognition:**  
   Introduce real-time text recognition functionality, allowing users to capture images directly via a device camera and receive instant text extraction results. This would enhance the application’s practicality for on-the-go use cases.
7. **Cloud Integration and Scalability:**  
   Integrate cloud-based storage and processing to handle larger datasets and improve processing speed. This would also enable collaborative features, such as sharing recognized text or storing results securely online.
8. **Robust Error Handling:**  
   Implement comprehensive error-handling mechanisms to gracefully manage invalid inputs, such as non-image files or images with no recognizable text, providing clear feedback to users to enhance their experience.

By addressing these areas, the team aims to create a more robust, versatile, and user-friendly application that meets the evolving needs of users while maintaining high accuracy and performance standards.

# ASSIGNMENT TABLE

# 

|  |  |  |
| --- | --- | --- |
| **No.** | **Name** | **Assignment** |
| 1 | Ngô Tường Phát | Train and Build model CRNN to detect |
| Research and write code to use vietocr library |
| Building code to process the image |
| 2 | Nguyễn Hà My | Research the CRNN |
| Write code to crop the input image |
| 3 | Nguyễn Thanh Minh Triết | Building streamlit to build app interface |
| Building system to detect Vietnamese word |
| Research the CTC Loss |

*Table 2. Assignment table*

# 

# REFERENCES

1. **TensorFlow Documentation and Tutorials**
   * **Description**: Official TensorFlow documentation and tutorials covering the implementation of deep learning models, including CRNN, LSTM, and CTC Loss, which are essential for the project’s model training and implementation.
   * **Link**: <https://www.tensorflow.org/tutorials>
2. **Streamlit Documentation**
   * **Description**: Official documentation for Streamlit, providing guidance on building interactive web interfaces for Python applications, relevant to the project’s user interface development.
   * **Link**: <https://docs.streamlit.io/>
3. **Practical Guide to Image Processing for OCR**
   * **Description**: A PyImageSearch tutorial covering image preprocessing techniques (e.g., grayscale conversion, thresholding, and noise removal) using OpenCV, critical for the project’s image processing pipeline.
   * **Link**: <https://www.pyimagesearch.com/category/ocr/>
4. **VietOCR**
   * **Description**: An open-source repository providing a framework for Vietnamese OCR using deep learning models, including CRNN and Transformer-based approaches. It serves as a practical reference for the project’s Vietnamese OCR implementation.
   * **Link**: <https://github.com/pbcquoc/vietocr>
5. **Vietnamese Handwriting Recognition OCR**
   * **Description**: A GitHub repository focused on Vietnamese handwritten text recognition, offering code and insights for building OCR systems tailored to Vietnamese script.
   * **Link**: <https://github.com/TomHuynhSG/vietnamese-Handwriting-Recognition-OCR>
6. **Vietnamese Handwritten Text Dataset**
   * **Description**: A dataset of Vietnamese handwritten text images, useful for training and evaluating OCR models to improve the project’s model robustness.
   * **Link**: <https://drive.google.com/file/d/1-hAGX91o45NA4nv1XUYw5pMw4jMmhsh5/view?usp=sharing>