

**ARTIFICIAL INTELLIGENCE PROGRAMMING PROJECT**

**Final Report - Final Project Presentation**

**Group 9**

– Hanoi, May 2021 –

Table of Contents

[I. Project Introduction](#_Toc68809412)

II. Project Management Plan

III. Data Collection

IV. Fundamental Algorithms

# I. Project Introduction

## 1. Overview

### 1.1 Project Information

* **Project name:** Offline English Handwriting Recognition
* **Group name**: Group 9

### 1.2 Project Team

|  |  |  |
| --- | --- | --- |
| **Full Name** | **Email** | **Role** |
| Đặng Hải Dương | duongdhhe150140@fpt.edu.vn | Leader |
| Đặng Thế Anh | anhdthe153776@fpt.edu.vn | Member |

### 1.3 Project Abstract

Offline English handwriting recognition is an important task due to the popularity of the English all over the world. This project presents an English handwritten word recognizer based on a sequence labeling method with deep convolutional neural networks (CNN) and recurrent neural networks (RNN). In addition, a connectionist temporal classification (CTC) loss function is utilized in order to eliminate the segmentation step required in conventional methods. The CNN layers are employed to extract the sequence of features from the word image. Altogether, the RNN layer with CTC function is used for labeling the input sequence.

## 2. Problem & Motivation

Despite the abundance of technological writing tools ,many people still choose to take their notes traditionally: with pen and paper. However, there are drawbacks to hand-writing text. It’s difficult to store and access physical documents in an efficient manner, search through them efficiently and to share them with others .Thus, a lot of important knowledge gets lost or does not get reviewed because of the fact that documents never get transferred to digital format. We have thus decided to tackle this problem in our project because we believe the significantly greater ease of management of digital text compared to written text will help people more effectively access, search, share, and analyze their records, while still allowing them to use their preferred writing method. The aim of this project is to further explore the task of classifying handwritten text and to convert handwritten text into the digital format. Handwritten text is a very general term, and we wanted to narrow down the scope of the  
project by specifying the meaning of handwritten text for our purposes. In this project, we took on the challenge of classifying the image of any handwritten word, which might be of the form of cursive or block writing. we believe that the most interesting and challenging  
part of this problem is the classification part, which is why we decided to tackle that instead of segmentation of lines into words, documents into lines, etc.

## 3. Solutions

We approach this problem with complete word images because CNNs tend to work better on raw input pixels rather than features or parts of an image. Given our findings using entire word images, we sought improvement by extracting characters from each word image and then classifying each character independently to reconstruct a whole word. In summary, in both of our techniques, our models take in an image of a word and output the name of the word.

## 4. Related Work The first prominent piece of OCR software was invented by Ray Kurzweil in 1974 as the software allowed for recognition for any font .This software used a more developed use of the matrix method (pattern matching). Essentially, this would compare bitmaps of the template character with the bitmaps of the read character and would compare them to determine which character it most closely matched with. The downside was this software was sensitive to variations in sizing and the distinctions between each individuals way of writing.

# II. Project Management Plan

## 1. Overview

### 1.1 WBS & Estimation

*[Create/Provide the project WBS & Estimation following the table template as below. In which, we categorize the WBS items into three levels of complexity (Simple, Medium, Complex) and estimate the total effort to complete each item in man-day]*

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **WBS Item** | **Complexity** | **Est. Effort** |
| ***1*** | ***Planning*** |  | **3** |
| 1.1 | - Understand Project | Simple | 1 |
| 1.2 | - Develop Schedule/ Time Budget | Simple | 2 |
| ***2*** | ***Data Collection*** |  | **15** |
| 2.1 | - Collect data from various trusted information sources | Simple | 5 |
| 2.2 | - Preprocessing & Data Augmentation | Complex | 10 |
| **3** | ***Training Model*** |  | **17** |
| 3.1 | Apply CNN model | Complex | 15 |
| 3.2 | -Test and complete the project | Simple | 2 |
| ***3*** | ***Implementation and design app*** |  | **10** |
| 3.2 | Develop a demo app | Complex | 9 |
| 3.3 | Perform deployment app | Easy | 1 |
| ***Total Estimated Effort (man-days)*** | | | ***45*** |

### 1.2 Project Risks

*[List out the details on project risks in the table below]*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Risk Description** | **Impact** | **Possibility** | **Response Plans** |
| 1 | Don’t have enough time | Project fail | Medium | Reduce some features |
| 2 | Can’t implement project | Project fail | High | Try another project which is easier to implement |
| 4 | Lack of data | The word in the image that user want to search don’t have in the database so the output will be not thing | Medium | +Update and add new image to the database. |
| 5 | Data lost | Waste a lot of time to process the data | Medium | + We need to have another backup of the database so that we can easily restore it after updating. |

## 2. Document Management And Source Code Management

*[Describe how you would manage project documents & their changes/versions] And*

*[Describe how you would manage project source codes & their changes/versions]*

*Github :* [*https://github.com/AutumnAdd3r/github.git*](https://github.com/AutumnAdd3r/github.git)

# III. Data Collection

1. **Data**

Our main resource for training our handwriting recognizer was the IAM Handwriting Dataset .This dataset contains handwritten text of over 1500 forms, where a form  
is a paper with lines of texts, from over 600 writers, contributing to 5500+ sentences and 11500+ words. The words were then segmented and manually verified; all associated  
form label metadata is provided in associated XML files. The source text was based on the Lancaster-Oslo/Bergen (LOB) corpus, which contains texts of full English sentences with a total of over 1 million words. The database also includes 1,066 forms produced by approximately 400 different writers. This database given its breadth, depth, and quality tends to serve as the basis for many handwriting recognition tasks and for those reasons motivated our choice of the IAM Handwriting Dataset as the source of our training, validation, and test data for our models. Last but not least, in deep learning large datasets–even with many pre-trained models–are very important and this dataset containing over 100K+ word instances met those requirements (deep learning model need at least 105 −106 training examples in order to be in position to perform well, notwithstanding transfer learning).

**1.1. Preprocessing & Data Augmentation**  
Before training our models with the dataset, we have applied various preprocessing and data augmentation techniques on our dataset in order to make our data more compatible with the models and to make our dataset more robust to real life situations.  
**1.2. Padding images**  
As mentioned above, the dataset consists of images of single words only. Moreover, the images are of different sizes because different words are of different lengths and  
heights. For instance the image of the word ‘error’ has a lower width than the image of the word ‘congratulations’ because of the length of the words. Similarly, the image  
heights differed among images due to the heights of their characters. For instance, the image of the word car has a lower height than the image of the buy since the characters of the word ‘buy’ extend above and below with ‘b’ and ‘y’. Our architectures, however, assumed the input images to be of the same size just like any other convolutional neural network architecture. This is essential as the weights of the layers are adjusted according to the first input image, and the model would not work as well if weights were not consistent, or changed shapes, for different inputs. Thus, we decided to make all the images of the same shape. It did not make sense to crop large images to an average size  
because cropping removes some characters from the image, which in turn can cause the image of a word to be interpreted differently. For instance cropping the image of the  
word ‘scholarly’ can make the image look like it is the image of the word scholar, and having two different labels for two similar images would definitely negatively affect the  
accuracy of our models. Moreover, we also thought that rescaling the image size would not work because of a few reasons. First of all, the aspect ratio of every image is different. This means that if we wanted to set the height or the width of an image to a specific value, the other dimension would have been different for all the other images with different aspect ratios. Secondly, if we resized the image on the height and width dimensions independently, then the image would get distorted, and we thought that this again would negatively affect our model since the inherent characteristics of the picture are being lost. Therefore, we decided to pad our images with whitespace to the maximum width and height present in our dataset. While doing so, the white space was added evenly on both sides of the height and the width dimensions. This approach does not necessarily change the inherent characteristic of the word image, and since the images of the same words would pretty much be padded with similar sizes, the relative relationship between the images do not change.

**1.3. Rotating images**  
Even though our dataset consists of the images of every word separately, some words within these images were slightly tilted. This was because the participants of the  
dataset were asked to write on blank paper with no lines, and some of the words were written in a more tilted fashion. This occasion happens very frequently in real life whether or not the page has lines, thus we decided to make our training data more robust to this issue by rotating an image towards the right by a very small angle with random probability and adding that image to our training set. This data  
augmentation technique helped us make our model more robust to some minor yet so frequent details that might come up in our test set.

# IV. Fundamental Algorithms

## Overview

**A. Connectionist Temporal Classification(CTC algorithm)**:

- An algorithm used to train deep neural networks in speech recognition, handwriting recognition and other sequence problems.

## What does CTC do?

In the case of creating an OCR (Optical Character Reader), CRNN (Convolutional Recurrent Neural Networks) are a preferred choice. They output a character-score for each time-step, which is represented by a matrix. We now need to use this matrix for:

* Training the Neural Network, i.e., calculating the loss
* Decoding the output of the Neural Network

CTC operation helps in achieving both tasks.

## Problems solved by CTC:

Imagine creating a dataset full of images of text and specifying each time-step of the image’s corresponding character, as shown in Fig2(a) There are a couple of issues with this approach:

* Annotating a dataset at the character level is a tedious task
* What if the character takes up more than one time-step? (as shown in Fig2(b)) It will result in duplication of the characters.

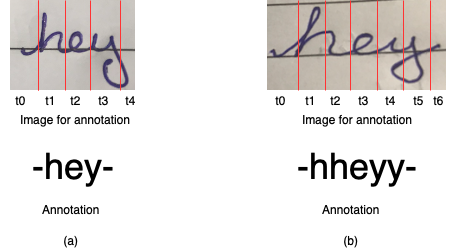


Fig2 (a) Shows the annotation for a case when each character takes up one time-step (b) Shows the annotation for a case when a few characters take up more than one time-step

Here, CTC comes to the rescue:

* CTC is formulated in such a way, that it only requires the text that occurs in the image. We can ignore both the width and position of the characters in an image.
* There is no need for post-processing the output of the CTC operation! Using decoding techniques, we can directly get the result of the network.

## CTC Working:

CTC works on the following three major concepts:

* Encoding the text
* Loss calculation
* Decoding

## Conclusion:

Initially, we looked at the problems faced with a naive Neural Network for the handwriting recognition task. Then, we looked into how CTC can solve those issues. Then, we saw how CTC functions by encoding the text, method of calculating the loss, and decoding the output from a Neural Network trained using CTC.

**B. Best part decoding (Greedy decoding):**

The idea of ​​this algorithm is:

At each time step, select the character with the specified maximum and combine them (the path with the specified maximum).

Then remove duplicates (duplicates) and blank characters (space) to give the final result a string of predicted word characters.

**C. CNN :**

Convolution neural network algorithm is a multilayer perceptron that is the special design for identification of two-dimensional image information . Always has more layers: input layer, convolution layer, sample layer and output layer. In addition, in a deep network architecture the convolution layer and sample layer can have multiple. CNN is not as restricted Boltzmann machine, need to be before and after the layer of neurons in the adjacent layer for all connections, convolution neural network algorithms, each neuron don't need to do feel global image, just feel the local area of the image. In addition, each neuron parameter is set to the same, namely, the sharing of weights , namely each neuron with the same convolution kernels to deconvolution image.

## Pseudo code

**1.Greedy search decoder:**

|  |
| --- |
| def decode\_best\_path(probs, ref=None, blank=0): |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Compute best path |
|  |

|  |
| --- |
| best\_path = np.argmax(probs,axis=0).tolist() |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Collapse phone string |
|  |

|  |
| --- |
| hyp = [] |
|  |

|  |
| --- |
| for i,b in enumerate(best\_path): |
|  |

|  |
| --- |
| # ignore blanks |
|  |

|  |
| --- |
| if b == blank: |
|  |

|  |
| --- |
| continue |
|  |

|  |
| --- |
| # ignore repeats |
|  |

|  |
| --- |
| elif i != 0 and b == best\_path[i-1]: |
|  |

|  |
| --- |
| continue |
|  |

|  |
| --- |
| else: |
|  |

|  |
| --- |
| hyp.append(b) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| # Optionally compute phone error rate to ground truth |
|  |

|  |
| --- |
| dist = 0 |
|  |

|  |
| --- |
| if ref is not None: |
|  |

|  |
| --- |
| ref = ref.tolist() |
|  |

|  |
| --- |
| dist,\_,\_,\_,\_ = ed.edit\_distance(ref,hyp) |
|  |

|  |
| --- |
|  |
|  |

return hyp,dist

## “Real” code

**CTC loss function:**

class CTCLayer(keras.layers.Layer):

def \_\_init\_\_(self, name=None):

super().\_\_init\_\_(name=name)

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

def call(self, y\_true, y\_pred):

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

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

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

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

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

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

self.add\_loss(loss)

# At test time, just return the computed predictions.

return y\_pred

**Greedy search decoder:**

def decode\_predictions(pred):

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

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

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

:, :max\_len

]

# Iterate over the results and get back the text.

output\_text = []

for res in results:

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

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

output\_text.append(res)

return output\_text

CNN :

def build\_model():

# Inputs to the model

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

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

# First conv block.

x = keras.layers.Conv2D(

32,

(3, 3),

activation="relu",

kernel\_initializer="he\_normal",

padding="same",

name="Conv1",

)(input\_img)

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

# Second conv block.

x = keras.layers.Conv2D(

64,

(3, 3),

activation="relu",

kernel\_initializer="he\_normal",

padding="same",

name="Conv2",

)(x)

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

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

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

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

x = keras.layers.Dropout(0.2)(x)

# RNNs.

x = keras.layers.Bidirectional(

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

)(x)

x = keras.layers.Bidirectional(

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

)(x)

x = keras.layers.Dense(

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

)(x)

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

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

# Define the model.

model = keras.models.Model(

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

)

# Optimizer.

opt = keras.optimizers.Adam()

# Compile the model and return.

model.compile(optimizer=opt)

return model

# Get the model.

model = build\_model()

model.summary()