Handwritten Character Recognition using Convolutional Neural Networks with TensorFlow

Calvin Ferraro, Ruslan Polichshuk, ZhiZhou He, Saichidvilas Renukunta
csferrar@asu.edu, rpolichs@asu.edu, zhizhouh@asu.edu, srenuku1@asu.edu

Abstract* - the recent achievement in Deep Learning (DL) has opened new doors for Artificial Intelligence development and Automatization. However, there are numerous different Neural Network (NN) architectures which are not studied. Moreover, it is important to know how to optimize hyperparameters of NN and adjust them to get the highest efficiency. In this paper we analyze six different Convolutional NN (CNN) architectures by building a Handwriting Character Recognition tool as a metric. This article mainly focuses on offline recognition of handwritten digits and Latin letters by detecting individual characters first. CNN model was built in TensorFlow and trained by using MNIST and CoMNIST datasets. Finally, as an additional challenge in scope of the project for EEE511 class our team created a GUI for manual testing of the developed models, which includes text segmentation. The final model can be considered as a raw but complete Handwritten Text Recognition tool.

***Keywords* - Handwritten Text Recognition, Convolutional Neural Network, TensorFlow**

I. INTRODUCTION

Recent achievements in deep learning (DL) have led to a renewed interest in the development of optical character recognition (OCR) techniques. The OCR, defined as a system that translates characters into machine-encoded format [1], began in the first half of the last century. For this purpose, mechanical machines have been utilized; whereas, the text had to be in a specific format. Today, DL algorithms provide an opportunity to decode very complex text structures, such as handwritten text, into digital format. The complete comprehensive survey on OCR and related techniques is presented in reference [2]. One of the most complex problems in this area is handwritten text recognition (HTR).

There are two distinguished types of HTR: online and offline algorithms [3]. *Online HTR* transforms signals from pen tips (also called digital ink) into text. An example of the application of online HTR is the interpretation of stylus written text by note-taking software. The idea of *offline HTR* is to digitize already written text from pictures or surfaces. Off-line handwritten text recognition has many possible applications. This can include some examples such as signature recognition, document digitization, and automated mail sorting. There are also many applications for handwritten text recognition in mobile applications, such as check deposits in banking applications and language translations done by software such as Google Translate.

There are two main difficulties associated with handwritten HTR, feature extraction and character extraction. Due to the flexibility of handwritten characters, there are a large variety of written character shapes. That raises issues on extracting features from different handwritten characters. Character extraction is the process to separate images into constituent characters. Handwritten character recognition has a similar interest. On the other hand, less variability and independence in language makes this problem easier. In the current project, we consider offline handwritten digits recognition only. The main challenges related to offline handwritten digits recognition are [4]: (i) numerous variations on the writing of the same digit, (ii) limited sources of the high-quality data, and (iii) digit segmentation. Therefore, despite advances in developing deep learning-based techniques, handwritten digits recognition remains a challenging problem. CNN can be the key to solve those problems.

CNN architecture consists of two types of hidden layers. Convolutional layer, the layer employed to extract the high-level features from images. Pooling layer which is responsible for reducing spatial size and extract dominant features that characterize the handwritten text. These two types of layers enable CNN to reduce the size of sampled data while keeping critical features of the handwritten text. Due to the advantage of CNN, it has been widely applied in image processing, classification, segmentation and so on. Despite its wide application, CNN (as any other algorithms) require optimization of hyperparameters, layers setup, and architectures for each separate problem or class of problems. Therefore, the scope of the current paper covers this problem.

Problem Formulation & Objective

In their paper Siddique et al [5] implement CNN with different layups for digit recognition to make a comparison between them. The main objective is to perform parametric study by changing layer layup in CNN architecture. This allows to trace the variability of the accuracy versus number and order of layers in the model. In addition to that, the most accurate solution can be found. Authors considered six configurations, and reached 99.21% validation accuracy in 15 epochs (case 2). Our paper considers all these cases, and shows comparison with the paper results. The key objective of our paper is to achieve comparable results and perform parametric study on hyperparameters. The novelty of the current study is not in the development of HTR itself, but in providing of the comparison between different CNN architectures in order to determine the most efficient one, and see how different layers affect the final performance.

As an additional objective for this project, we utilized the built CNN for Latin letter recognition, developed GUI for manual tests and demonstration with implemented character segmentation. The final algorithm can be considered as a complete HTR tool; however, with some limitations, which will be discussed later. Since this project is done in scope of the EEE511 class, our target was to learn how to construct basic NN, implement them in TensorFlow, adjust hyperparameters, and apply in real software. The basic, but important case of text recognition is ideal for these purposes.

II. METHODS

Resources

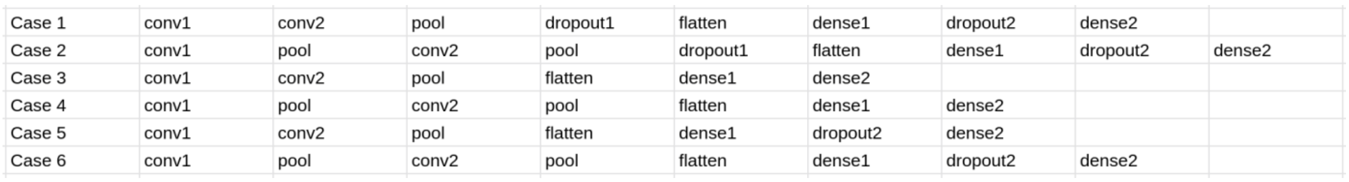
We use Google Colaboratory for the development of our code and training process. This allows us to share the code quickly and run on a shared resource if sufficient computing resources are unavailable. Colab also allows for easy management of dependencies as, most often, they are supplied as a default option. The code is written using 3.7.12 Python with Tensorflow 2.7.0. Tensorflow is employed to implement neural network architectures with sequential models. Adam optimizer which means learning rate is 0.005 is applied.

As for hardware, a Nvidia GPU preinstalled in Google Colab is used to accelerate the computation process. NVIDIA-SMI 495.44 with driver version 460.32.03 and CUDA version 11.2 help debug and run our code. Since we utilize Tensorflow, the Tensor Processing Unit (TPU) is more suitable here; however, it was not available during the training process.

Datasets

For this paper we utilize the Modified National Institute of Standards and Technology (MNIST) dataset. This dataset contains 28 by 28 pixel images of handwritten digits. There are 60000 training data scanned images, and 10000 images are for validation. These images are represented as a 28 by 28 matrix, containing information of the intensity (from 0 to 255) of pixels in grayscale. The output is a scalar which represents the number of the class. Input data has been preprocessed in two steps: matrix was flattened to 784-dimensional vector and then normalized. Output has been categorized and represented as a vector with values between 0 and 1. For example, digit 4 corresponds to [0 0 0 0 1 0 0 0 0 0].

Similarly, as for handwritten Latin and cyrillic letters, CoMNIST dataset is utilized. This dataset provides 12828 278 by 278 pixel images in .png format. Despite the larger size of the input vector,



Case 1	conv1	conv2	pool	dropout1	flatten	dense1	dropout2	dense2	
Case 2	conv1	pool	conv2	pool	dropout1	flatten	dense1	dropout2	dense2
Case 3	conv1	conv2	pool	flatten	dense1	dense2			
Case 4	conv1	pool	conv2	pool	flatten	dense1	dense2		
Case 5	conv1	conv2	pool	flatten	dense1	dropout2	dense2		
Case 6	conv1	pool	conv2	pool	flatten	dense1	dropout2	dense2	

Figure 1: Definitions of each case defined in Siddique et al. [5]

the data was compressed to 28 by 28 matrix size in order to reduce computational time and have comparable results. The output data is also a scalar containing 26 classes, from 0 to 25, which correspond to A through Z. Summarizing, CoMNIST has much less data points and more classes, which provides a smaller amount of data for one separate class, compared to the digits case. In addition, letters have higher variability, such as lowercase and uppercase, Italic and non-Italic, cursive and block letters. Therefore, it is predicted to have lower accuracy, compared to digit recognition.

III. IMPLEMENTATION AND SIMULATION

Deep learning layers setup

To construct a desired layers layup, several types of layers have been utilized. The organization of these layers into each of the six cases defined in Siddique et al. are seen in Figure 1 [5]. Following are the layers as are used in our project:

- **conv1**:

Convolutional layer consisting of 32 filters, 3x3 kernel size, and a ReLU activation. This serves as the input layer in all cases.

- **conv2:**

Convolutional layer consisting of 62 filters, 3x3 kernel size, and a ReLU activation.

- **pool:**

A max pooling layer with a pool size of 2x2 pixels

- **dropout1:**

A dropout layer with a 25% dropout rate.

- **dropout2:**

A dropout layer with a 50% dropout rate.

- **flatten:**

A standard flattening operation to convert the input from the 2D representation output by convolutional layers to the 1D input accepted by the fully connected layers

- **dense1:**

Fully connected layer containing 128 neurons and a ReLU activation function.

- **dense2:**

Fully connected layer containing 26 output neurons for Latin character recognition (or 10 for digits) and a softmax activation.

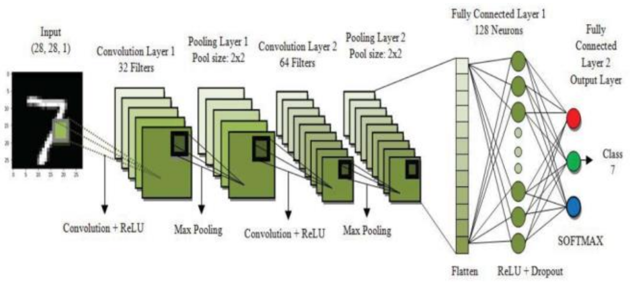
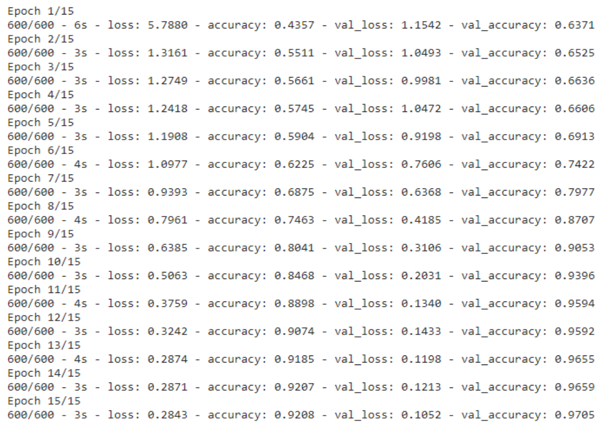


Figure 2. Seven-layered CNN architecture

Adam optimizer with learning rate of 0.005 and was utilized. The training was performed using 15 epochs with batch size of 100 points. In the in figure 3 below the compilation log file is presented. It shows training and validation accuracy at each epoch.



```
Epoch 1/15
600/600 - 6s - loss: 5.7880 - accuracy: 0.4357 - val_loss: 1.1542 - val_accuracy: 0.6371
Epoch 2/15
600/600 - 3s - loss: 1.3161 - accuracy: 0.5511 - val_loss: 1.0493 - val_accuracy: 0.6525
Epoch 3/15
600/600 - 3s - loss: 1.2749 - accuracy: 0.5661 - val_loss: 0.9981 - val_accuracy: 0.6636
Epoch 4/15
600/600 - 3s - loss: 1.2418 - accuracy: 0.5745 - val_loss: 1.0472 - val_accuracy: 0.6606
Epoch 5/15
600/600 - 3s - loss: 1.1908 - accuracy: 0.5904 - val_loss: 0.9198 - val_accuracy: 0.6913
Epoch 6/15
600/600 - 4s - loss: 1.0977 - accuracy: 0.6225 - val_loss: 0.7606 - val_accuracy: 0.7422
Epoch 7/15
600/600 - 3s - loss: 0.9393 - accuracy: 0.6875 - val_loss: 0.6368 - val_accuracy: 0.7977
Epoch 8/15
600/600 - 4s - loss: 0.7961 - accuracy: 0.7463 - val_loss: 0.4185 - val_accuracy: 0.8707
Epoch 9/15
600/600 - 3s - loss: 0.6385 - accuracy: 0.8041 - val_loss: 0.3106 - val_accuracy: 0.9053
Epoch 10/15
600/600 - 3s - loss: 0.5063 - accuracy: 0.8468 - val_loss: 0.2031 - val_accuracy: 0.9396
Epoch 11/15
600/600 - 4s - loss: 0.3759 - accuracy: 0.8898 - val_loss: 0.1340 - val_accuracy: 0.9594
Epoch 12/15
600/600 - 3s - loss: 0.3242 - accuracy: 0.9074 - val_loss: 0.1433 - val_accuracy: 0.9592
Epoch 13/15
600/600 - 4s - loss: 0.2874 - accuracy: 0.9185 - val_loss: 0.1198 - val_accuracy: 0.9655
Epoch 14/15
600/600 - 3s - loss: 0.2871 - accuracy: 0.9207 - val_loss: 0.1213 - val_accuracy: 0.9659
Epoch 15/15
600/600 - 3s - loss: 0.2843 - accuracy: 0.9208 - val_loss: 0.1052 - val_accuracy: 0.9705
```

Figure 3. Code running log file

IV. RESULTS AND DISCUSSIONS

Digit Recognition Results

Training for each of six cases has been performed, and summary of all results can be seen in the figure 8. The training and validation accuracies for literature (figure 7) and the developed models shows similar trends and values. The maximum deviation in overall performance is 0.6% for the case 4. For the rest of the cases, deviation lies in range of 0.1-0.5%. However, the developed model show slightly less performance. There can be several reasons. One of them can be related to the optimizer. In the original paper there are no specifications on the optimizer settings. In addition, stochastic gradient method may provide different results at different stages. The maximum performance accuracy achieved is 99.03%.

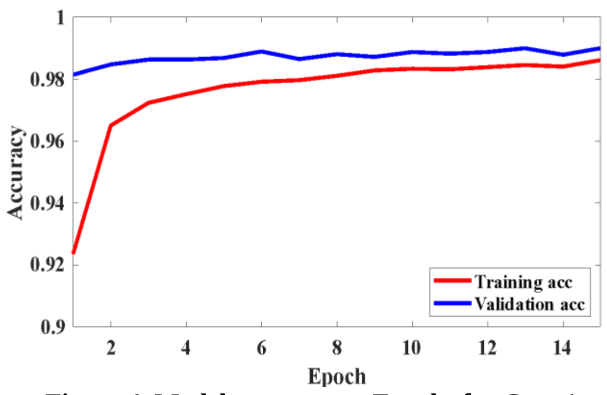


Figure 4. Model accuracy vs Epochs for Case 1

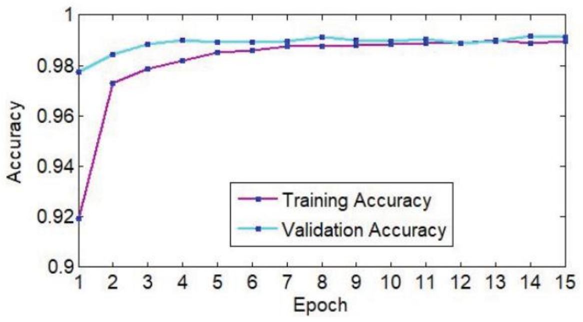


Figure 5. Model accuracy vs Epochs for Case 1 from source paper [5]

Figure 4 and 5 show accuracies of both literature and our algorithms for case 1 against number of epochs. It observed, that training accuracy rapidly increase from epoch 1 to 2, and then remains above 97%. It shows, that for the epoch 1 the initial guess was far from local minima. After epoch 5, results are fluctuating around the extremum point. All results show accuracies about 99% for the last epoch. It is high enough to get the right prediction for a random picture. In Figure 6 below, there are 5 random handwritten digits, and all of them predicted precisely.

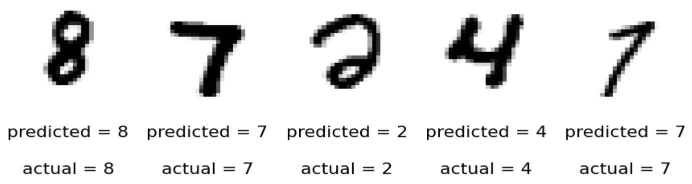
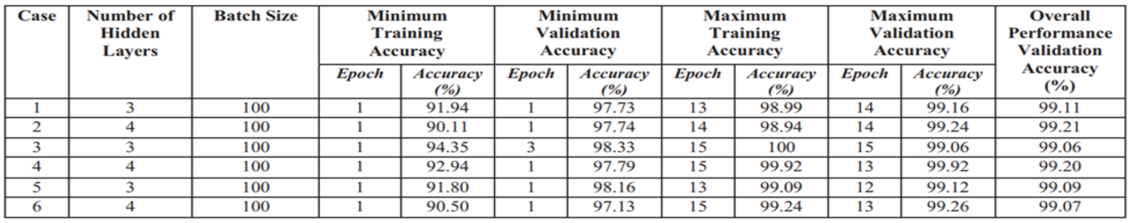
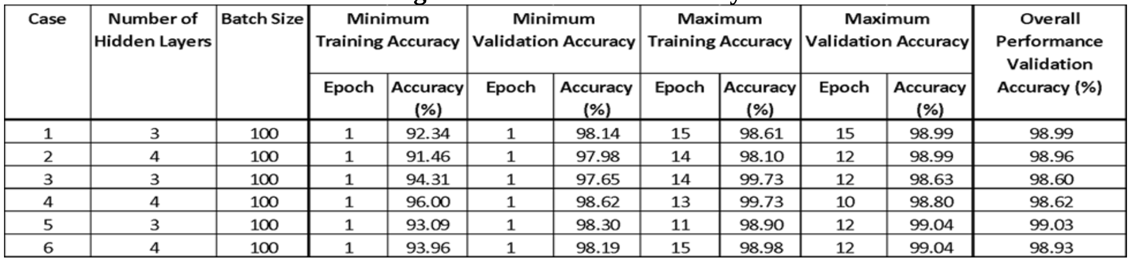


Figure 6. Model validation



Case	Number of Hidden Layers	Batch Size	Minimum Training Accuracy		Minimum Validation Accuracy		Maximum Training Accuracy		Maximum Validation Accuracy		Overall Performance Validation Accuracy (%)
			Epoch	*Accuracy (%)*	*Epoch*	*Accuracy (%)*	*Epoch*	*Accuracy (%)*	*Epoch*	*Accuracy (%)*	
1	3	100	1	91.94	1	97.73	13	98.99	14	99.16	99.11
2	4	100	1	90.11	1	97.74	14	98.94	14	99.24	99.21
3	3	100	1	94.35	3	98.33	15	100	15	99.06	99.06
4	4	100	1	92.94	1	97.79	15	99.92	13	99.92	99.20
5	3	100	1	91.80	1	98.16	13	99.09	12	99.12	99.09
6	4	100	1	90.50	1	97.13	15	99.24	13	99.26	99.07

Figure 7. Literature Summary



Case	Number of Hidden Layers	Batch Size	Minimum Training Accuracy		Minimum Validation Accuracy		Maximum Training Accuracy		Maximum Validation Accuracy		Overall Performance Validation Accuracy (%)
			Epoch	Accuracy (%)	Epoch	Accuracy (%)	Epoch	Accuracy (%)	Epoch	Accuracy (%)	
1	3	100	1	92.34	1	98.14	15	98.61	15	98.99	98.99
2	4	100	1	91.46	1	97.98	14	98.10	12	98.99	98.96
3	3	100	1	94.31	1	97.65	14	99.73	12	98.63	98.60
4	4	100	1	96.00	1	98.62	13	99.73	10	98.80	98.62
5	3	100	1	93.09	1	98.30	11	98.90	12	99.04	99.03
6	4	100	1	93.96	1	98.19	15	98.98	12	99.04	98.93

Figure 8. Summary for the developed model

One case that demands some attention is Case 3. It can be seen in Figure 7 and Figure 8 that this case is the worst performing out of both implementations of the same network. The reason behind this is that no dropout layers and only one pooling layer are used in Case 3 as can be seen in Figure 1. The absence of these layers causes the model to overfit to the training samples and thus perform poorly.

Letter Recognition Results

Figure 9 illustrate the results for handwritten letters recognition. To implement letter recognition, a modified CNN is utilized. The modified CNN consists of 2 convolutional layers as the first two layers, followed by one pooling layer. Sequentially, a flatten layer, first density layer, dropout layer and second dense layer after the pooling layer, as can be seen in Case 5 in Figure 1. After 15 training epochs, the maximum training accuracy can be 98.79%. Meanwhile, the maximum validation accuracy is only 88.03%, lower than that of handwritten digit recognition. That is reasonable as the complexity of handwritten letters significantly increases. Apart from that, CoMINST offers fewer training points while having more classes when compared to MNIST dataset.

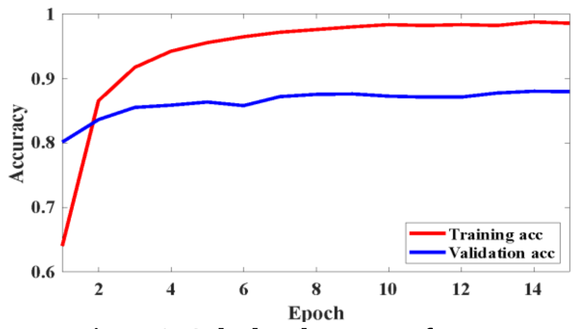


Figure 9. Calculated accuracy for case 5

However, the accuracy is still sufficient to precisely recognize handwritten letters. As shown in Figure 10, all handwritten letters are predicted precisely, none of five letters was recognized wrongly.

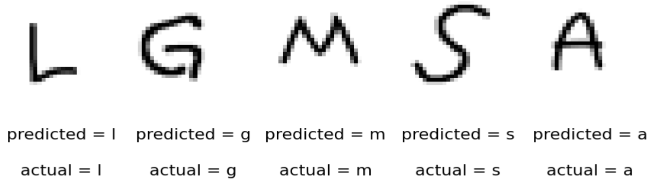
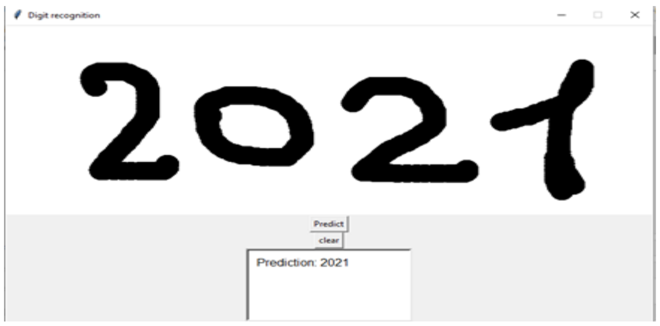


Figure 10. Predicted values for random images

Segmentation & Demonstration

Figures 11 and 13 show the developed GUI for manual testing of the developed algorithms. It allows to manually draw a text or set of digits, and it will show the predicted values. Later, it can be used for data collection and improving of the dataset.

In practice, handwritten texts are often multiple characters instead of the single character representation shown previously. A number consisting of four digits (2021 in Figure 11) and a word containing three letters (ASU in Figure 13) are found to be predicted successfully. As shown in figure 12, words containing multiple digits or letters are firstly segmented into different files before recognition. After being recognized individually, the predicted results were put together. The success shows the potential of CNN used in handwritten text recognition.



Prediction: 2021

Figure 11. prediction of Digit recognition

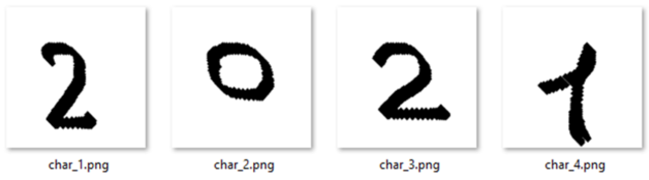
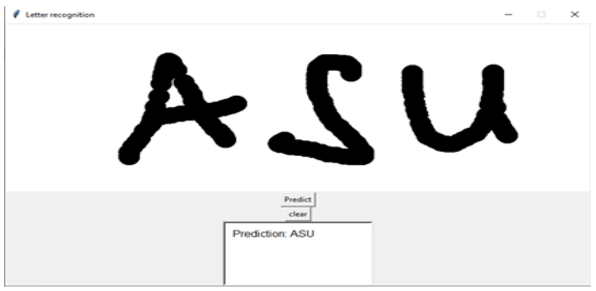


Figure 12. Segmented digits in different files



Prediction: ASU

Figure 13. Picture depicting Character recognition

V. CONCLUSIONS

While our developed model does not produce the exact results reported by Siddique et al., the results that we found were very close. For each of the six defined cases, our developed model was within 0.1-0.5% loss. The maximum performance achieved is 99.03%, and the maximum training accuracy is 99.73%. We detected, that elimination of dropout layer leads to overfitting. Using this algorithm, we could build letter recognition, and by application of text segmentation get HTR.

Furthermore, our initial goal of recognizing handwritten text was achieved, albeit not in the way that we had initially intended. While we had hoped to implement a more complex algorithm, we instead took a bottom-up approach to handwritten text recognition. We did this by first implementing an algorithm to recognize single characters and digits, and then implementing a word segmentation algorithm that would break down each word (or number) into single characters such that they might be passed into the same code used previously. Overall, this was a more educational approach to the project as it taught all of us about how handwritten text is recognized in state-of-the-art algorithms. In addition, we have learned how to implement different CNN architectures, use TensorFlow for NN construction, receive experience with text recognition.

REFERENCES

[1] C. C. Tappert, C. Y. Suen, and T. Wakahara, ‘‘The state of the art in online handwriting recognition,’’ IEEE Trans. Pattern Anal. Mach. Intell., vol. 12, no. 8, pp. 787–808, Aug. 1990, doi: 10.1109/34.57669.

[2] Memon, Jamshed, et al. "Handwritten optical character recognition (OCR): A comprehensive systematic literature review (SLR)." *IEEE Access* 8 (2020): 142642-142668.

[3] Vashist, Prem Chand, Anmol Pandey, and Ashish Tripathi. "A comparative study of handwriting recognition techniques." *2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM)*. IEEE, 2020.

[4] Balaha, H. M., Ali, H. A., Youssef, E. K., Elsayed, A. E., Samak, R. A., Abdelhaleem, M. S., ... & Mohammed, M. M. (2021). Recognizing Arabic handwritten characters using deep learning and genetic algorithms. *Multimedia Tools and Applications*, 1-37.

[5] F. Siddique, S. Sakib and M. A. B. Siddique, "Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers," 2019 5th International Conference on Advances in Electrical Engineering (ICAEE), 2019, pp. 541-546, doi: 10.1109/ICAEE48663.2019.8975496.

[6] Brownlee, J. “Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras”. Machine Learning Mastery (2016)

[7] Brownlee, J. “How to Develop a CNN for MNIST Handwritten Digit Classification”. Machine Learning Mastery (2019)

[8] GitHub. MNIST digits classification with TensorFlow 2 (2020) https://github.com/antonio-f/TensorFlow2_digits_classification-Linear_Classifier-MLP/blob/master/TensorFlow2_digits_classification-Linear_Classifier-MLP/digits_classification.ipynb

APPENDIX

A. Links

Presentation:https://docs.google.com/presentation/d/10WdTyWiSRpdtGs8oyzDdoLcIJkPPe-r1O5kf8ETLbBo/edit#slide=id.gfba47225db_2_201
GitHub:https://github.com/calferraro/HandwrittenCharacterDemo

B. Code

Following is the code that both trains and runs the GUI. There are three command line options that can be used to modify how the code runs: --case, --train, --isDigit. The case argument allows the user to specify which of the six cases to use. This affects both the GUI and the training, so if a specific model is to be trained, this argument must be utilized. If this is not set, the default case is Case 1. Next, the --train argument is used to specify that a model is to be trained. If it is not set, the default option is to run the GUI. The last command line argument is the --isDigit option. This allows the user to specify whether they are inputting a digit or a character. The default option is to use the character recognizer.

```
# Initial setup
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt

import numpy as np
from tensorflow.keras.utils import to_categorical  # Will be used to categorize an output layer neurons

# Import all required layers types
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dense
from keras.layers import BatchNormalization
from keras.layers import Flatten
from keras.layers import Dropout
from keras.datasets import mnist

# Load optimizers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD

# Imports for gui
import tensorflow as tf
from keras.models import load_model
import cv2
import numpy as np
import os
```

```
from PIL import ImageTk, Image, ImageDraw
import PIL.Image as Img
import PIL
import tkinter as tk
from tkinter import *
import pytesseract
pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-
OCR\\tesseract.exe'

import argparse

case = [
   ['conv1', 'conv2', 'pool', 'dropout1', 'flatten', 'dense1', 'dropout2',
'dense2'],
   ['conv1', 'pool', 'conv2', 'pool', 'dropout1', 'flatten', 'dense1',
'dropout2', 'dense2'],
   ['conv1', 'conv2', 'pool', 'flatten', 'dense1', 'dense2'],
   ['conv1', 'pool', 'conv2', 'pool', 'flatten', 'dense1', 'dense2'],
   ['conv1', 'conv2', 'pool', 'flatten', 'dense1', 'dropout2', 'dense2'],
   ['conv1', 'pool', 'conv2', 'pool', 'flatten', 'dense1', 'dropout2',
'dense2']
]

def CNN_modelSetup(args, layers):
 CNN_model = Sequential()

 for l in layers:
   if(l == 'conv1'):
     CNN_model.add(Conv2D(32, (3, 3), activation='relu',
kernel_initializer='he_uniform', input_shape=(28, 28, 1))) #conv1
   if(l == 'conv2'):
     CNN_model.add(Conv2D(64, (3, 3), activation='relu',
kernel_initializer='he_uniform')) #conv2
   if(l == 'pool'):
     CNN_model.add(MaxPooling2D((2, 2))) #pool1
   if(l == 'dropout1'):
     CNN_model.add(Dropout(0.25))
   if(l == 'flatten'):
     CNN_model.add(Flatten())
   if(l == 'dense1'):
     CNN_model.add(Dense(512, activation='relu',
kernel_initializer='he_uniform'))
   if(l == 'dropout2'):
     CNN_model.add(Dropout(0.5))
   if(l == 'dense2'):
     CNN_model.add(Dense(10 if args.isDigit else 26, activation='softmax'))
 # compile model
 opt = Adam(learning_rate=0.001)
 # opt = SGD(learning_rate=0.01, momentum=0.9)
 CNN_model.compile(optimizer=opt, loss='categorical_crossentropy',
metrics=['accuracy'])
 return CNN_model

def character_gui(case):
   classes_letter=['A','B','C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']
   width = 800
   height = 300
   center = height//2
   white = (255, 255, 255)
   green = (0,128,0)

   modelCNN=tf.keras.models.load_model('LetterRecognition_case4.h5') # Load our
model

   def paint(event):
       x1, y1 = (event.x - 10), (event.y - 10)
       x2, y2 = (event.x + 10), (event.y + 10)
       cv.create_oval(x1, y1, x2, y2, fill="black",width=10)
       draw.line([x1, y1, x2, y2],fill="black",width=10)

   def model():
       filename = "./image.png"
       image1.save(filename)
       nchar=segmentation(filename)
       txt.insert(tk.INSERT, "Prediction: ")
       for i in range(nchar):
           pred=testing(i+1)
           txt.insert(tk.INSERT,"{}".format(classes_letter[pred[0]]))

   def clear():
       cv.delete('all')
       draw.rectangle((0, 0, 5000, 5000), fill=(255, 255, 255, 0))
       txt.delete('1.0', END)
       import os
       filename = "./image.png"
       nchar=segmentation(filename)
       for i in range(nchar):
           os.remove('./char_' +str(i+1)+'.png')

   def testing(charnum):
       img=cv2.imread('char_' +str(charnum)+'.png',0)
       img=cv2.bitwise_not(img)
       #cv2.imshow('img',img)
       img=cv2.resize(img,(28,28))
       img=img.reshape(1,28,28,1)
       img=img.astype('float32')
       img=img/255.0
       pred=modelCNN.predict(img)
       classes_x=np.argmax(pred,axis=1)

       return classes_x

   def segmentation(file):
       #file = "./image.png"
       img = cv2.imread(file)
       h, w, _ = img.shape
       boxes = pytesseract.image_to_boxes(img)
       charcoordintates=boxes.splitlines()
       nchar=len(charcoordintates)
       k=1;
       for b in boxes.splitlines():
           b = b.split(' ')
           img = cv2.rectangle(img, (int(b[1]), h - int(b[2])), (int(b[3]), h -
int(b[4])), (0, 255, 0), 2)
           fp = open(file,"rb")
           imageObject = PIL.Image.open(fp)
           #imageObject = Image.open('image.png')
           cropped = imageObject.crop((int(b[1]),
                                       h-int(b[4]),
                                       int(b[3]),
                                       h-int(b[2]),
                                     ))
           card = Img.new("RGBA", (300, 300), (255, 255, 255))
           img2 = cropped.convert("RGBA")
           x, y = img2.size
           ov=75
           card.paste(img2, (ov, ov, x+ov, y+ov), img2)
           card.save('char_' +str(k)+'.png', 'PNG')
           k=k+1
       return nchar

   root = Tk()
   ##root.geometry('1000x500')

   root.resizable(0,0)
   cv = Canvas(root, width=width, height=height, bg='white')
   cv.pack()

   # PIL create an empty image and draw object to draw on
   # memory only, not visible
   image1 = PIL.Image.new("RGB", (width, height), white)
   draw = ImageDraw.Draw(image1)

   txt=tk.Text(root,bd=3,exportselection=0,bg='WHITE',font='Helvetica',
               padx=10,pady=10,height=5,width=20)

   cv.pack(expand=YES, fill=BOTH)
   cv.bind("<B1-Motion>", paint)

   ##button=Button(text="save",command=save)
   btnModel=Button(text="Predict",command=model)
   btnClear=Button(text="clear",command=clear)
   ##button.pack()
   btnModel.pack()
   btnClear.pack()
   txt.pack()
   root.title('Letter recognition')
   root.mainloop()

def digit_gui(case):
   classes=[0,1,2,3,4,5,6,7,8,9]
   width = 800
```

```python
height = 300
center = height//2
white = (255, 255, 255)
green = (0,128,0)

modelCNN=tf.keras.models.load_model('CNN_HDR_case6.h5') # Load our model

def paint(event):
    x1, y1 = (event.x - 10), (event.y - 10)
    x2, y2 = (event.x + 10), (event.y + 10)
    cv.create_oval(x1, y1, x2, y2, fill="black",width=10)
    draw.line([x1, y1, x2, y2],fill="black",width=10)

def model():
    filename = "./image.png"
    image1.save(filename)
    nchar=segmentation(filename)
    txt.insert(tk.INSERT, "Prediction: ")
    for i in range(nchar):
        pred=testing(i+1)
        txt.insert(tk.INSERT,"{}".format(classes[pred[0]]))

def clear():
    cv.delete('all')
    draw.rectangle((0, 0, 5000, 5000), fill=(255, 255, 255, 0))
    txt.delete('1.0', END)
    import os
    filename = "./image.png"
    nchar=segmentation(filename)
    for i in range(nchar):
        os.remove('./char_' +str(i+1)+'.png')

def testing(charnum):
    img=cv2.imread('char_' +str(charnum)+'.png',0)
    img=cv2.bitwise_not(img)
    #cv2.imshow('img',img)
    img=cv2.resize(img,(28,28))
    img=img.reshape(1,28,28,1)
    img=img.astype('float32')
    img=img/255.0
    pred=modelCNN.predict(img)
```

```
        classes_x=np.argmax(pred,axis=1)

        return classes_x

    def segmentation(file):
        #file = "./image.png"
        img = cv2.imread(file)
        h, w, _ = img.shape
        boxes = pytesseract.image_to_boxes(img)
        charcoordintates=boxes.splitlines()
        nchar=len(charcoordintates)
        k=1;
        for b in boxes.splitlines():
            b = b.split(' ')
            img = cv2.rectangle(img, (int(b[1]), h - int(b[2])), (int(b[3]), h -
int(b[4])), (0, 255, 0), 2)
            fp = open(file,"rb")
            imageObject = PIL.Image.open(fp)
            #imageObject = Image.open('image.png')
            cropped = imageObject.crop((int(b[1]),
                                        h-int(b[4]),
                                        int(b[3]),
                                        h-int(b[2]),
                                      ))
            card = Img.new("RGBA", (300, 300), (255, 255, 255))

            img2 = cropped.convert("RGBA")
            x, y = img2.size
            ov=75
            card.paste(img2, (ov, ov, x+ov, y+ov), img2)
            card.save('char_' +str(k)+'.png', 'PNG')
            k=k+1
        return nchar

    root = Tk()
    ##root.geometry('1000x500')

    root.resizable(0,0)
    cv = Canvas(root, width=width, height=height, bg='white')
    cv.pack()
    # PIL create an empty image and draw object to draw on
    # memory only, not visible
    image1 = PIL.Image.new("RGB", (width, height), white)
    draw = ImageDraw.Draw(image1)

    txt=tk.Text(root,bd=3,exportselection=0,bg='WHITE',font='Helvetica',
                padx=10,pady=10,height=5,width=20)

    cv.pack(expand=YES, fill=BOTH)
    cv.bind("<B1-Motion>", paint)

    ##button=Button(text="save",command=save)
    btnModel=Button(text="Predict",command=model)
    btnClear=Button(text="clear",command=clear)
    ##button.pack()
    btnModel.pack()
    btnClear.pack()
    txt.pack()
    root.title('Digit recognition')
    root.mainloop()

def main(args):
 if(args.isDigit):
    # do digit recognition code here
    # Data load and preprocess
    (Xtr_0, Ytr_0), (Xts_0, Yts_0) = mnist.load_data()

    Xtr = Xtr_0.reshape((Xtr_0.shape[0], 28, 28, 1))  # Training input reshape
matrix -> vector
    Xts = Xts_0.reshape((Xts_0.shape[0], 28, 28, 1))  # Testing input reshape
matrix -> vector

    Ytr = to_categorical(Ytr_0) # Represent an output as a vecotr 1x10; exp: 4 -
> [0 0 0 0 1 0 0 0 0 0]
    Yts = to_categorical(Yts_0) # Same for test data

    # Data normalization
    Xtr_norm = Xtr.astype('float32')/255.0
    Xts_norm = Xts.astype('float32')/255.0

    if(args.train):
      # call training function here
      # Model training
      CNN_mod = CNN_modelSetup(args, case[args.case])
      CNN_mod.fit(Xtr_norm, Ytr, validation_data=(Xts_norm, Yts), epochs=15,
batch_size=100, verbose=2)
      scores = CNN_mod.evaluate(Xts_norm, Yts, verbose=0)

      CNN_mod.save('./models/DigitRecognition_case{}.h5'.format(args.case))
    else:
      # call gui function here
      digit_gui(args.case)
  else:
      # do character recognition code here
      with open("./data/latin_data.csv") as file_name:
          X_all_0 = np.loadtxt(file_name, delimiter=",")

      X_all=X_all_0.reshape(X_all_0.shape[0], 28, 28, 1)
      X_all=X_all.astype('float32')

      with open("./data/latin_label.csv") as file_name:
          Y_all_0 = np.loadtxt(file_name, delimiter=",")
      Y_all=to_categorical(Y_all_0)

      Y_all_0.shape

      numbyclass=[493, 496, 508, 461, 500, 482, 509, 509, 476, 458, 460, 523,
547, 521, 465, 526, 484, 470, 519, 477, 475, 480, 465, 490, 545, 483]
      Trnumbyclass=[394, 397, 406, 369, 400, 386, 407, 407, 381, 366, 368, 418,
438, 417, 372, 421, 387, 376, 415, 382, 380, 384, 372, 392, 436, 386]

      currentidx=0
      tridx=[]
      tsidx=[]
      for i in range(26):
          tridx=np.append(tridx,range(currentidx, currentidx+Trnumbyclass[i]))
          tsidx=np.append(tsidx,range(currentidx+Trnumbyclass[i],
currentidx+numbyclass[i]))
          currentidx=currentidx+numbyclass[i]

      tridx=tridx.astype(int)
```

```
     tsidx=tsidx.astype(int)



     Xtr=np.array([X_all[x] for x in tridx])
     Xts=np.array([X_all[x] for x in tsidx])
     Xtr_0=np.array([X_all_0[x] for x in tridx])
     Xts_0=np.array([X_all_0[x] for x in tsidx])
     Ytr=np.array([Y_all[x] for x in tridx])
     Yts=np.array([Y_all[x] for x in tsidx])
     Ytr_0=np.array([Y_all_0[x] for x in tridx])
     Yts_0=np.array([Y_all_0[x] for x in tsidx])

     if(args.train):
         print('start training')
         # call training function here
         CNN_mod = CNN_modelSetup(args, case[args.case])
         CNN_mod.fit(Xtr, Ytr, validation_data=(Xts, Yts), epochs=15,
batch_size=50, verbose=2)
         scores = CNN_mod.evaluate(Xts, Yts, verbose=0)

         CNN_mod.save('./models/LetterRecognition_case{}.h5'.format(args.case))
     else:
         # call gui function here
         character_gui(args.case)

if __name__ == '__main__':
 parser = argparse.ArgumentParser()

 parser.add_argument('--train', action="store_true")
 parser.add_argument('--case', type=int, default=0)
 parser.add_argument('--isDigit', action="store_true")

 args = parser.parse_args()

 main(args)
```

C. Lessons Learned

end of semester reflection - lessons learned from working on the final project						
Team # and names of team members						
Team #2	Calvin Ferraro	Ruslan Polichshuk	Zhizhou He	Saichidvilas Renukunta		
Project title	Handwritten character recognition					
	literature (not well written or self-contained, not specific on implementation, no data source indicated, no source code indicated…)	setting up the environment and obtaining data	to have the first successful test run (issues during debugging, compatibility problems…)	obtaining results (algorithm/method is dificult to implement, hyper parameters difficult to tune...)	obtaining results (cannot duplicate what was reported in paper, if so, why?)	reporting (Intro, method, result, discussions, …)
specific & detailed evidence is required to support claims (e.g., links, repository sites, equation #, figure #, paragraphs, sections, etc…)	We could have found a better source	We had no issues setting up our environment our collecting data. We used MNIST and CoMNIST as our data sets which are both easily obtainable. For our development environment we used Google Colaboratry, which requires no setup before use.	We did have one debugging issue in implementing the algorithm. Our initial code did not run because we had forgotton to normalize the input. This resulted in the loss not converging. Our lesson learned in debugging is how important it is to normalize input features	Hyperparameter tuning was the main topic of our paper, so it was not a difficult task to implement the networks defined. All was very well defined in the source paper. We have learned how to utilize Tensorflow to build CNN model using Sequential().	Our results matched those of the paper very closely. The only difference is that we had to use a different optimizer than was used in the source because we had some issues with convergence. Otherwise, no issues in optaining results.	We feel that there was a gap in understanding throghout the semester in what we thought was asked in reports and what was expected. As can be seen in the comments from the TA on our updates, often times we did not go deep enough in descriptions of certain areas. This was often caused in either a lack of space in the report or a lack of thorough

D. Contribution form

Assignment name	Final report package							
Team # 2								
Student name: Zhizhou He	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to equation #, figure #, paragraphs, sections, etc…)			Presenting result in the final report and give explanation	Introduction and method,editing and update on final report		SR	RP	CF
Student name: Saichidvilas Renukunta	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to equation #, figure #, paragraphs, sections, etc…)	Did survey of similar papers online, learnt new techniques			Appendix and Results, Formatting Proof reading		ZH	RP	CF
Student name: Calvin Ferraro	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to equation #, figure #, paragraphs, sections, etc…)	Found source pape	Generalized code to build any of the specified cases. Added segmentation to the demo gui so that multi character strings can be recognized.	Generated loss/accuracy graphs	Added code and how to run it in Appendix. Wrote conclusion section and contributed to the results section		ZH	SR	RP
Student name: Ruslan Polichshuk	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to equation #, figure #, paragraphs, sections, etc…)	Literature review in the introduction section; code sousres	Wrote a training code for digits and letters, impelemented GUI, helped with text segmentation (to store searate letters as pictures)	Training for all cases for digits, and one case for letters. Generated examles for the paper	Introduction, part of methodology and results, finilizing the whole paper		ZH	SR	CF