Final Project

Yinqi (Bill) Sun
New York University
ys3540@nyu.edu

Honor Pledge:

"I affirm that I will not give or receive any unauthorized help on this academic activity and that all the work I submitted is my own."
Signature: Yinqi Sun
Code Availible: `https://github.com/sunyinqi0508/ml_proj`

1. INTRODUCTION

The problem I'm trying to solve in the project is:

Learning the 'Handwriting Accent'

As we may know, people from different regions may have different writing styles ('accent') [1, 2] on the same characters. This is largely influenced by the culture and his native language. For example, some eastern asian people tends to write the letter 't' from the horizontal bar '-' and then the vertical part '*l*'. And the Japanese people tends to write number '0' clockwise while the rest of the world tends to write them in counterclockwise direction [3]. There could potentially be much more factors that affects the handwriting style, I'm trying to use machine learning techniques to build a model to classify writing accents from people. As far as I know, there's probably no research on similar topics.

2. APPLICATION

The first and most obvious application of this project is in identification of anonymous writings useful in criminal investigation. It can also be used to extract more information from anonymous data (e.g. feedback, exams) for research purposes, for example, analyzing scores of test-takers from different countries.
The second but more important application is in OCR. Because different people write the same characters differently. Ideally, the OCR programs should learn each person's writing style in order to improve accuracy. But this may be too expensive and requires too much data before it's accurate enough to be usable. Especially in mobile devices like phones and smartwatches where writing recognition is mostly demanded. Because people's writing styles can be grouped into several accents, we can first pre-train models for popular accents. And the accent of the user can be chosen according to the current region or based on his previous handwritings or it can be manually chosen by the user himself.

3. IMPLEMENTATION

In this project, I'll try to use classification methods such as KNN and deep neural networks to achieve the goal. The model is elaborated in Chapter 4

3.1 Goals and Validation

The ultimate goal is to build a model that categorize the writing style of people around the world into major accents. However, this would require much larger datasets. So, just to prove the concept, I simplified the goal to be:

'Distinguishing hand-written numbers and English characters between Chinese people and westerners.'

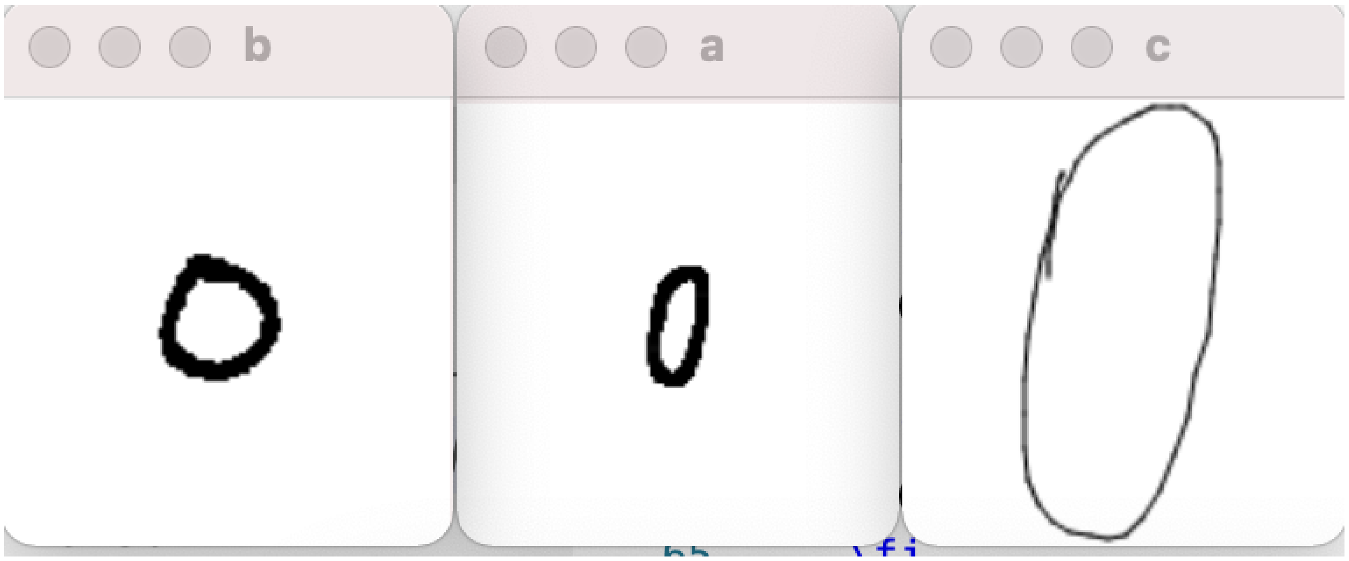
3.2 Datasets

For handwriting data from Chinese people, I used HIT-OR3C dataset [4] and CASIA [5] dataset.
The HIT-OR3C offline dataset provides Chinese handwritings of 10 numbers, 26 lowercase English characters, 26 uppercase

English characters and several thousands of Chinese characters. The data are collected from 122 users, each user wrote all these characters in an order described in labels.txt with a stylus. The data are directly collected by the writing pad.
The other Chinese handwriting dataset I used is the CASIA dataset [5], like the HIT-OR3C, CASIA contains all the 62 characters we are interested in and several thousands of Chinese characters but are collected from over 1000 users. But Instead of using stylus, this dataset is collected from handwritings on paper using OCR.
For handwriting data from west people. I used the NIST dataset [6]. It is collected from employees of Census Bureau in Maryland. It contains 62 different characters written on paper and each character has more than 5000 different samples.

3.2.1 Data Preprocessing

Reading the data: Each dataset uses different format to efficiently store it. The HIT-OR3C dataset uses their own data format described in one of the documentations. `http://www.iapr-tc11.org/dataset/OR3C_DAS2010/v1.0/OR3C/online/File%20style(English).doc`
And the CASIA and NIST dataset both uses png files to store individual sample. I simply used the OpenCV library to read png data.
Data Unification: Because it's difficult to find a single dataset that has the region of the writer labelled, I have to resort to use different datasets. However, each dataset will also differ on the method it used to collect the data (digital input or OCR from paper and the OCR algorithm being used) and the postprocessing techniques. In order to prevent our models from learning those irrelevant features, we need to eliminate those features as much as possible. For example, in the graph shown below, the left data(b) is from NIST dataset, and the right most data(c) is from HIT-OR3C. We can see that the stroke from HIT-OR3C is much thinner and it's stretched to fit the whole canvas. To eliminate these features, I thickened the stroke using a dilate filter with 14*14 kernel. Then I randomly zoomed it out while padding it with white pixels to maintain the size. Finally, I applied a clamped cubic color filter to reduce the number of gray level. We can see the resulting image in the middle (a) seems to have successfully eliminated most of noticeable irrelevant features while still maintaining the its original shape (c). The preprocessing code for HIT-OR3C dataset can be found in chinese_proc.py on my github repo.



Randomization Although we can eliminate noticeable features from preprocessing, there could still be potential irrelevant features that are picked up by the neural network due to its high flexibility. I may need to introduce more noises to obfuscate them. I noticed tensorflow offered some out-of-box randomization functions for images such as rotating and scaling. Because the feature we'd like to learn lies in the shape of the characters instead of its size and direction which is largely dependant on the data extraction and postprocessing methods.

4. MODELS

Because this is primarily a classification problem. In this project I'll first use simple classification methods such as KNN which are cheaper to train and much easier to interpret. I'll also use deep learning methods because it's generally better on complicated tasks like image recognition.

4.1 Neural Network Classifier

In the DNN classifier, I'll use a 2-network architecture inspired from the representation network from GQN (Featured in my blog post assignment), the first network is a three-layer convolutional network to extract features from the 128*128*1 image. I added max pooling layers after each convolutional layers to reduce its sensitivity on the location of features. the convolutional network output a 2*2*256 feature vector extracted from each image sample. The second network is has 3 layers, it further classifies the feature vector into 2 categories. The two networks are trained together with Adam optimizer (an SGD optimizer that automatically tunes learning rate).

The data set was split into training set and validation set using skleran.train_test_split. I used tensorflow (keras) to build the model (see train.py). The batch size can be controlled manually after each epoch, in each training, I adjusted it manually based on the validation accuracy. It almost automatically trains the model using my GPU with CUDA. The shape of the 2 neural networks are shown below:

Model: "sequential":

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 123, 123, 256)	9472
max_pooling2d (MaxPooling2D)	(None, 20, 20, 256)	0
conv2d_1 (Conv2D)	(None, 15, 15, 256)	2359552
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_2 (Conv2D)	(None, 2, 2, 256)	2359552
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 256)	262400
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 1)	65

4.2 KNN Classifier

For comparison, I also implemented a naive KNN classifier using Scikit-Learn. Because KNN is known to scale poorly to higher dimensions, I first preprocessed the 128x128 (=16384 dimensions) images to reduce its size to 24x24 (= 576 dimensions), to prevent loss of too much information during scaling, I first cropped the image the bounding box of its content and then pad it to equal width and height so that scaling it to 24x24 won't distort the image. The code for KNN preprocessing can be found in **train.py** knn_preproc() bounding_box(). The code for KNN classifier is in knn()

4.3 DNN based recognizer

Since we want to simulate the application in OCR, I also built a DNN based classifier that recognizes the characters. I noticed that some of the state-of-the-art Classifier for MNIST dataset [7] uses similar architectures as my accent recognizer. So I simply changed the loss function of my accent recognizer to SparseCatagorialCrossentropy and it can achieve over 90% accuracy on NIST dataset after 10 epochs of training but it trains much faster due to less neurons and hidden layers. Because this OCR network isn't the main focus of this work, I'll just use this implementation for simplicity.

5. EXPERIMENTS AND RESULTS

5.1 CNN based Accent Classifier Performance

The training set has about 180000 data point, about half labelled Chinese and half western. The batch size is 64 and the initial learning rate for Adam optimizer is 0.0001 The initial results shows that the model converges after 2 epoch with validation accuracy of 0.9967.

```
Epoch 1/1
2747/2747 - 1209s 438ms/step - loss: 0.1594 - accuracy: 0.9236 - val_loss: 0.0087 - val_accuracy: 0.9976
Epoch 2/2
2747/2747 - 1203s 438ms/step - loss: 0.0059 - accuracy: 0.9981 - val_loss: 0.0132 - val_accuracy: 0.9967
```

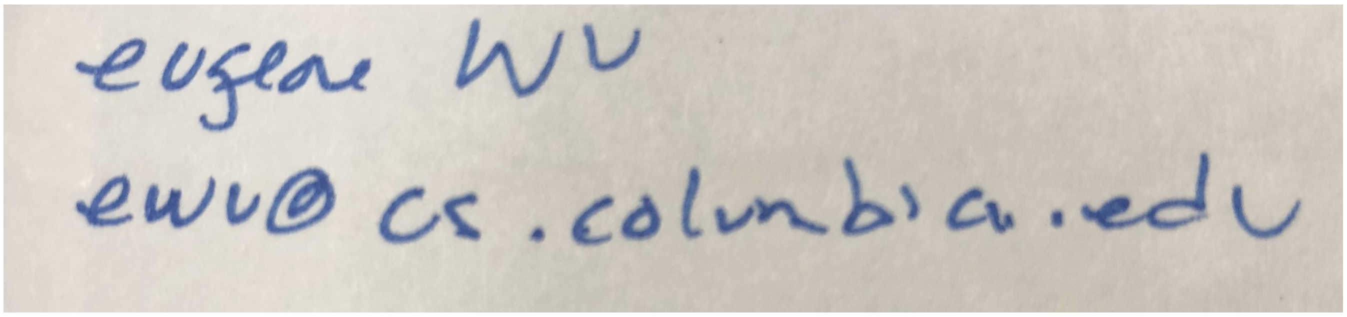
5.2 KNN based Accent Classifier Performance

The KNN Classifier is able to have about only 63% accuracy despite adding some preprocessing. I guess it's because the differences between writing accents are too subtle, at least at pixel level. Also, KNN isn't known to perform poorly in high dimensions.

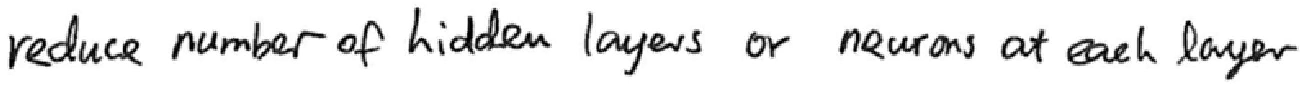
5.3 OCR Performance

I trained the same character recognition networks seperately with training data from Chinese datasets(HIT-OR3C+CASIA), NIST dataset, and the mix of Chinese and NIST datasets. After 10 epochs, they achieved accuracy of 89%, 91% and 79% respectively. If the data is first classified by its accent with accuracy of over 99.9%, the overall accuracy of character recognition will be about 90% which is significantly higher than doing OCR without accent recognition.

5.4 Real world data



eugene wu
ewu@ cs . columbia . edu



reduce number of hidden layers or neurons at each layer

From real-world data I used my handwriting from the scan of my final exam answer sheet, the handwriting from my grandma and the signiture from my advisor in the US. Because I need to process these samples manually, I only took about 10 samples each. And the accent recognizer can identify my grandma's writing with 100% accuracy and my handwriting with 60% accuracy but it recognizes all 11 samples from my US advisor's handwritings as Chinese. I think one possible reason is that as a Chinese American his writing accent is mostly influenced by his early education in China.

6. CODE USAGE

After downloading the three datasets, we need to first preprocess them. The code data_read.py contains functions for reading the HIT-OR3C and NIST dataset and output them as numpy array binary file. The HIT-OR3C dataset is then processed in chinese_proc.py. The CASIA dataset is read and preprocessed in cn2.py.
We can then train the models with functions in train.py.
After first loading the data, we generate labels for them. For accent identification, there're only two classes Chinese and western. For character classification we need to generate labels accroding to the data, I provided functions that output labels for each dataset in data_read.py and cn2.py. Note that the labels I output are case sensitive, (e.g. A and a are in 2 different classes) however, because many upper and lower case characters are not quite distinguishable and the CASIA dataset doesn't seperate upper and lower case characters, I decided to not distinguish between upper and lower case characters which gives us 36 classes in total. For this reason, we also need to preprocess the labels generated by the reader using the following code.

```
labels = np.where(labels>=36, labels - 36, labels)
```

dnn_binary_model() will return the accent recognition model, and dnn_categorial_model() will return the character recognition model. dnn_train() will train the dnn model with (data, label) from (train_x, train_y). dnn_load_weights() will load saved model from file. You may import this python file to use these functions.

```
from train import *
```

7. FUTURE WORK

From the real-world test of the experiment, I learned that the real-world data can be much more complicated and classifing accent by the region is not a good idea. So, in the future works, I may explore using unsupervised learning techniques such as PCA and clustering to mine the primary accent categories before the classification and hopefully this will make the classification results more sensible.

8. REFERENCES

[1] "Handwriting accents penmanship can betray language identity," 2016. https://www.translatemedia.com/translation-blog/handwriting-accents-penmanship-can-betray-language-identity/.

[2] "Regional handwriting variation," 2016. https://en.wikipedia.org/wiki/Regional_handwriting_variation.

[3] "How do you draw a circle? we analyzed 100,000 drawings to show how culture shapes our instincts," 2017. https://qz.com/994486/the-way-you-draw-circles-says-a-lot-about-you.

[4] "HIT-OR3C dataset," 2010. http://www.iapr-tc11.org/mediawiki/index.php?title=Harbin_Institute_of_Technology_Opening_Recognition_Corpus_for_Chinese_Characters_(HIT-OR3C).

[5] "CASIA dataset," 2011. http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html.

[6] "The NIST dataset," 2016. https://www.nist.gov/srd/nist-special-database-19.

[7] "Mnist classifier with average 0.17https://github.com/Matuzas77/MNIST-0.17.