DeepMotion: Handwriting English Character Classification Based on the Motion Sensing on Pen

COGS181 Final Project

Neural Networks/Deep Learning by Professor Zhuowen Tu

Chen Yang
Computer Science and Engineering
UC San Diego
chy099@ucsd.edu

Wanze Xie
Computer Science and Engineering
UC San Diego
waxie@ucsd.edu

ABSTRACT

This paper introduces DeepMotion, and end-to-end solution that adopts deep learning to recognize the English character that users are writing in real time by reading in the motion data of the pen collected using a sensor attached on its tail. This is aimed to be a cheaper alternative to the smart pen that is prevalently used for touch screens. At a high level, we deployed and tested three different supervised deep learning models including convolutional neural network (CNN), recurrent neural network (RNN), and dynamic recurrent neural network (dynamic RNN). An important aspect of our study is to explore the feasibility of recognizing handwriting character based on people's handwriting stroke and to evaluate the performance of different deep learning strategies and tuning hyperparameters on each deep learning strategy on our pen motion datasets.

KEYWORDS

CNN, RNN, neural network, character recognition, stroke recognition, handwriting motion capture.

1 INTRODUCTION

Writing on the touch screen using smart pen has been prevalent in today's tech world, mostly because of its convenience where people can store and retrieve notes from the cloud on different devices. However, this approach presents couple limitations: 1) requiring the collaboration between the digital pen and the computing device that has either a touch screen or digital writing pad. 2) prices are high because people need to purchase the device and the pen accessories. 3) potentially failed to favor the group of people who enjoy the feeling of writing on the paper using a traditional pen or pencil. On the other hand, there are also study shows that hypertext reading (Rakefet, 2016) and writing (Subrahmanyam, 2013) can be distracting and often negatively affecting people's attention on their work.

With the advent of the state-of-the-art machine learning strategies and the good performance of deep learning on data classification and prediction, many studies have come up with results that can be used to solve above problems. Image processing tools such as Optical Character Recognition (OCR) (smith, 2007) that can transfer handwritten scripts to digital text files is a promising solution to these problems, because people can take a take a picture of the notes and convert it to be an editable text file on digital devices. Such approach can be more efficient if we could get rid of the procedure of letting user to use another device to take the picture and upload. Inspired by this idea, we hope to implement a simpler and more intuitive way to combine both the advantage of writing on the paper and the convenience of keeping an editable record on the cloud.

Based on the study of character recognition using motion data instead of image data, we can simplify the pipeline into following: 1) user writes notes using a pen with sensor attached. 2) sensor sends collected data to the cloud server through the Bluetooth connected smartphone. 3) the server side does the character recognition and post processing to generate accurate paragraphs in a file stored in the cloud. This study will analyze on the training of the model used to complete the character recognition based on the motion data, as well as present our self-made device used to do the motion capture.

In this paper, we specifically focused on building a neural network model that can classify elementary English characters including a, b, c, d and e, but the idea is applicable to most characters in today's language systems. With our best efforts, we respectively collected over 1,000 handwriting data for each character from different people. We also 3D printed a model for mounting a 9-DOF motion sensor onto a normal sharpie for handwriting data collecting and uses Arduino as transferring interface between the sensor end and the computer end. The goal for this prototype is that when each new character is written, we will be able to recognize the pattern of the stroke and convert it to the corresponding character onto computer screen in real time.

2 METHOD

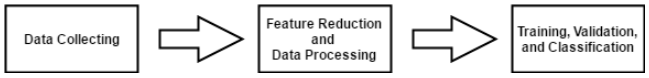

Figure 1: The Deep Learning Pipeline Built for the Study.

Figure 1 shows the pipeline we built for the study, which includes four different stages. The rest of this section will describe how we designed each of the step in the pipeline.

2.1 Data Collecting

In terms of the sensor we used to collect motion data, we are using MPU9250 9-axis motion sensor, Table 1 shows the spec of this sensor. We also experimented with the MPU 6050 6-axis motion sensor but decided not to use it due to its low accuracy and lack of magnetoscope field data (which cause the inaccurate conversion on yaw, pitch, raw motion data).

	MPU9250
Degree of Freedom	9
Built-in Sensors Type	Accelerometer Gyroscope Compass
Clock Interval	10-20ms

Table 1: MPU9250 Motion Sensor Tech Spec.

We designed a 3D model and printed it out using a 3D printer, so that we can attach the motion sensor to the top (tail) of a pen. Figure 2 shows the 3D model and Figure 3 shows the printed version. We removed the supporter and mounted it onto our sharpie. We used Arduino Uno R3 as the hardware motherboard. We manually hooked up the motion sensor with the motherboard, and programmed the hardware source code for reading in sensor data and streaming processed data to computer. We also mounted a 2x16 pixel LED display so that we can have real-time feedback when collecting the data. Figure 4 shows a real image of the entire hardware system.

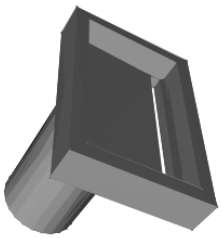

Figure 2: The rendered 3D model result of pen sensor mounting device. This model is the real model we used for 3D-printing.

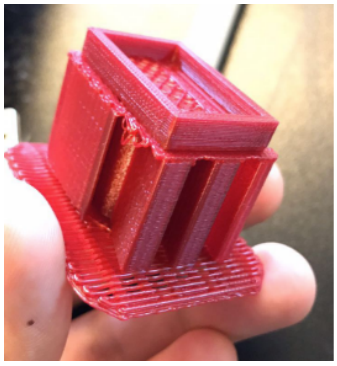

Figure 3: The printed result of our 3D model of the mounting device.

A big challenge that we faced during collecting the data is that we only want to avoid the noise data when the pen holder is not writing characters. Therefore, the timing of the start and the end of writing a character need to be detected. So, we implemented an extra push down button for the sake of making collecting data easier. The system can be simplified if we can have a pressure sensor on the tip of the pen. Figure 5 shows a volunteer working on creating data sample for this study.

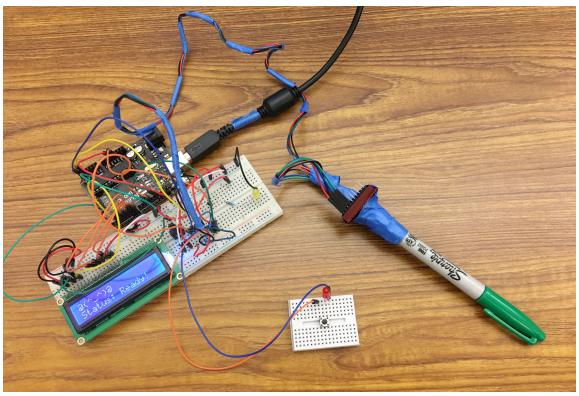

Figure 4: The hardware-sensing prototype we build for collecting and preprocessing pen's motion data.

One challenge during collecting data is that we only want to collect the motion of the pen when the pen holder is actually writing characters, and therefore the timing of the start and the end of writing a character can be hard to accurately detect based on the data variation itself. Therefore, we implemented an extra push down button for the sake of making collecting data easier. To collect the temporal data of the motion of the pen when writing each single character, the writer will press the push down button right before he or she is about to write a character, and release the push down button by the time he/she finished writing a character. Figure 5 shows a volunteer is working on creating data sample for this study.

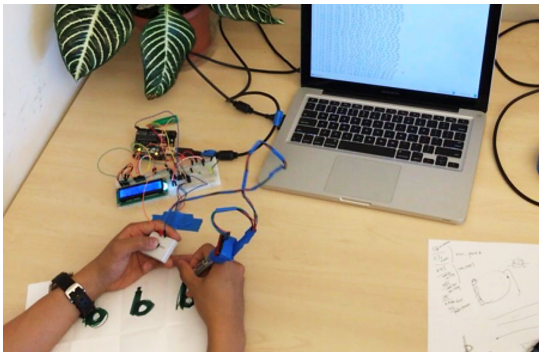

Figure 5: A volunteer is creating data sample for this study.

Nevertheless, noises still exist because we find it hard for the writer to accurately press the push down button at exactly the time starting to write. And the duration time of writing each character is indeterministic as well. As a result, we designed an interpolation method to normalize the temporal data, which we will discuss in the following Data Processing section (2.3).

2.2 Feature Reduction

Our motion sensor provides 9 degrees of freedom data sensing, which means each data sample contains 9 data: accelerometer data on xyz axis, gyroscope data on xyz axis, and the magnetometer data on xyz axis. During our data collection study, we discovered that people usually finish writing a single character within 600-1500ms. Since the MPU9250 motion sensor is able to return sampling data in up to 15ms, each character will contain varying 40 - 100 data points. By multiplying the number of data (which is 9), we can estimate the number of features for each character to be around 500 - 1000.

Therefore, to reduce training time, we decide to perform feature reduction by converting the collected raw data to standardized aircraft principal axes data: Yaw, Pitch and Roll. The standard way of calculating yaw, pitch and roll only require the data from accelerometer and gyroscope, but we add magnetometer data to the calculation in an effort to make the data more stable and accurate, and it turns out that the variation of data is smoother and more stable.

2.3 Data Processing

While we were recording the raw data, we perform yaw, pitch, and roll (ypr) calculation in real time on the Arduino motherboard. In the end, we save the ypr value combining with the raw 9-DoF data into the serial monitor. The Figure 7 shows an overview of this processing stream.

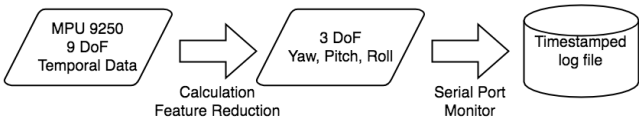

Figure 7: The flow chart showing the process of how data is read from MPU9250 and transfer to a log file for next step processing.

As mentioned above, the biggest challenge in this preprepressing pipeline is that the time it takes to write a single character varies quite a lot from characters to character as well as from person to person. Therefore, a sensible approach is needed to normalize the data so that the number of features for each data stays the same. Based on our implementation of MPU9250 sensor, it will return the ypr data for every 15 milliseconds, and each character takes 600 - 1500 milliseconds.

As a result, we used following procedure to perform data normalization: 1) extract out the pattern of the variance of the temporal data during the single character writing. 2) interpolate between adjacent temporal data sample to approximately predict the data during any time in this specific time period. In order to preserve as many meaningful features as possible without too much tradeoff for the computation time, we decided to up sampling and extract 300 features for each data by retrieving 300 ypr data from the interpolation model for each data sample. Table 2 shows the data size after the data processing.

Data Set	Sample Number	Feature Dimension
Character “a”	1,022	300 (3 * 100 ypr)
Character “b”	1,049	300 (3 * 100 ypr)
Character “c”	1,155	300 (3 * 100 ypr)
Character “d”	1,015	300 (3 * 100 ypr)
Character “e”	1,024	300 (3 * 100 ypr)

Table 2: Sample number and feature number of the raw input data set for each character.

In order to perform the classification, we combined all five data sets into one data set with label, showed in Table 3.

Combination	Sample Number	Feature Dimension
“a-b-c-d-e”	5,265	300 (3 * 100 ypr)

Table 3: Combined data set and feature information.

2.4 Classification and Model Training

By adopting different deep learning approach to test out the performance, we built a data set where the pre-processed motion data is the input data, and labels (targets) are 5 different class that is labelled as 0, 1, 2, 3, 4, corresponding to the character a, b, c, d, e. We adopted one-hot encoding on the labels because we do not want to compare the labels based on their alphanumeric sequence. As a result, in our training set, the dimension of the motion data is (4212,300), and the dimension of the label data is (4212,5), since one hot encoding expand the dimension from scalar to 5 length arrays. And for the testing set, the dimensions are (1053,300) and (1053,5).

Our training platform is the cloud computing platform provided by our school, which composed of powerful GPU

clusters. With this support, we are able to test large number of different cases with different hyper parameters in a reasonable time. Due the lengthy training time of recurrent neural network, we also implemented multithreading to speed up the training progress and tuning hyperparameters. Our result on different training models can be visualized in the following section.

3 EXPERIMENT SETUP AND RESULTS

3.1 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) is a class of deep, feed-forward artificial neural networks. In image processing cases, CNN can perform classification relatively well compare to some linear models. We decided to implement a CNN solution to perform classification on the motion handwriting data. To design and find out the best model for our dataset, we need to tune these six parameters: number of convolutional layers, batch size, filter width, size of window and stride, pooling strategy, and padding strategy. Based on these parameters, we performed couple sets of experiments to explore the best solution with CNN.

3.1.1 CNN Architecture

Considering that our datasets are primarily composed of 1D data, which is data along a time sequence. So instead of using existing architectures like VGG or ResNet that are primarily built for Image Recognition, we decided to implement our own 1D convolution layers based on the TensorFlow example for MNIST datasets [4]. Figure 8 shows the basic architecture of our CNN model in general.

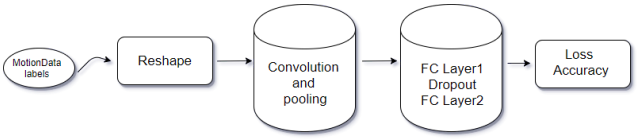

Figure 8: A Basic Architecture of our CNN Model.

During experimentation, we designed eight different models based on giving different number of convolutional layers, pooling functions, batch size during each training iteration, and activation functions. In the following sections, we will discuss how each model performs differently with different hyper parameter settings.

3.1.2 Convolution Layers

Our first goal is to test how the number of convolutional layers might affect the performance of the CNN model on our datasets. We designed 3 type of CNN models, including 2-layer, 3-layer and 5-layer. But to make it easier to identify the difference, we tried our best to keep the rest configurations of the model identical. For the pooling function, we choose the max pooling as default, and we use RuLU as the activation function for all 3 models. The Dropout layer is in between 2 FC layers. The rest configurations of the models are shown in the Table 4.

Layer Number	Feature Extract on Each Layer	FC Layer (Neurons)	Max Pooling (width, stride)
2	32, 64	32*32	(2,2), (2,2)
3	32, 64, 128	64*64	(2,2), (2,2), (3,3)
5	32, 64, 64, 64, 128	64*64	(2,2), (2,2), (15,1), (15,1), (5,5)

Table 4: Experiment Condition for CNN Model Selection.

During the training, we set the default batch size to be 50 for all three models, and set iteration number to be 20,000 which has been proved to be an adequate number of iterations that can bring training loss to convergence. One thing note is that, as expected, the more layer we introduce in our model, the slower the training progress would be. The training result can be visualized in the following Table 5.

Cond. ID	No. of Convolution Layer	Training Accuracy	Training Loss	Testing Accuracy
1	2 layers	1.0000	0.0596	0.9858
2	3 layers	1.0000	0.0785	0.9829
3	5 layers	1.0000	0.1624	0.9791

Table 5: Experiment Results for CNN Model Convolution Layer Selection.

From the table, it has shown trend that 2-layer has shown good enough performance. When we increment the number of layers to be 3 or 5, the training loss and test accuracy both goes worse and we think it is causing overfitting on the training datasets, so that it has less generalization ability. In the Figure 9, we also plotted out the trend of the training loss and testing accuracy over each iteration, we extend the x axis to be logarithmic so it is easier to identify the trend across the iterations.

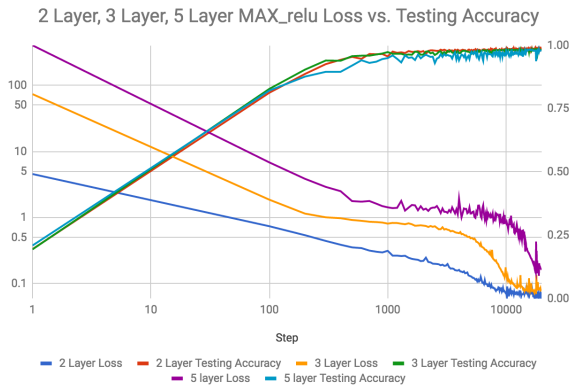

Figure 9: Experiment Results Comparison for CNN Model Convolution Layer Selection.

In the Figure 9, the vertical axis on the left side shows the value of the training loss, and the vertical axis on the right side shows the value of the test accuracy. The result is aligned with our previous expectation. The trend has been obvious that the 2-layer model performs good training result over the iteration. Through the test accuracy, we can also identify that the curve of the 2-layer model is more stable than the rest two.

3.1.3 Pooling Functions

We believe pooling function is also a major part that might affect our performance. Since we are using the pooling function provided by TensorFlow (tf.nn.pool), and it only support max pooling and average pooling, we will thus be testing on the those two type of pooling function and see the difference. Since from our previous experiment. 2-layer model shows the best training result, we will thus be testing using the 2-Layer model. The configuration of the 2-layer model will be the same as before, except that we will use "MAX" and "AVG" in the pooling function instead of just max pooling. The following graph shows the difference between the two. Note that for the sake of convenience, we combined the curve that we will use for later analysis are also incorporate in this graph but they should be ignored for now.

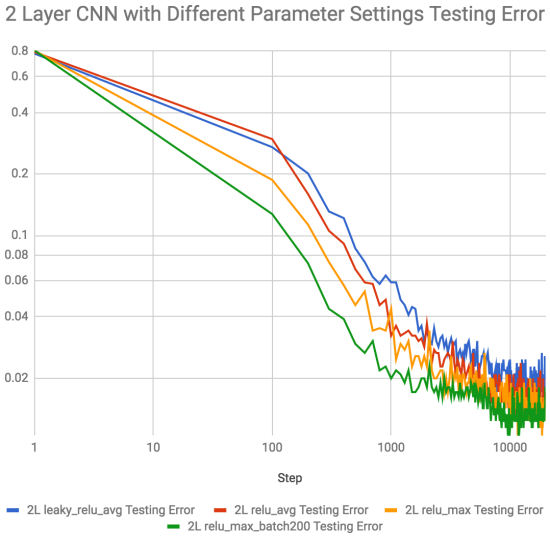

Figure 10: Experiment Results Comparison on Testing Error for CNN Model Pooling Functions Selection.

From the Figure 10 above, if we focus on red (avg) and orange (max) curve, we can identify that the max pooling gives a better training result. The blue curve (avg) that adopts Leaky RuLU as activation function performs also not as good as the model that adopts max pooling. The final testing error for max pooling, average pooling and average pooling with Leaky RuLU as activation function are respectively 0.9858, 0.9839, and 0.9801, and we can say that the max pooling function definitely fits our datasets better. The similar result can also be deduced from the following Figure 11 as well if we examine the training loss, where the purple curve (max) gives lower training loss across the training compared to the yellow and blue one.

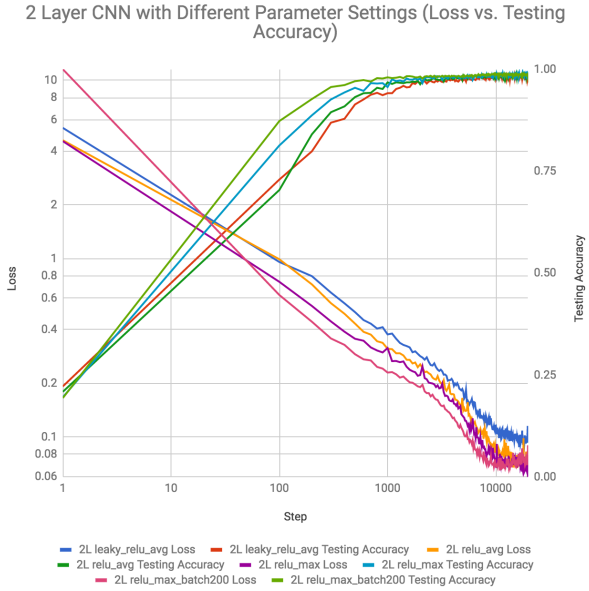

Figure 11: Experiment Results Comparison on Training Loss and Testing Accuracy for CNN Model Pooling Functions Selection.

3.1.4 Batch Size

From our previous experiment, the variance of the convolution layers and the choices of pooling functions actually does not produce any model that performs better than our original default model, which is the 2-layer model with max pooling and RuLU activation function. However, this experiment has proved that choosing larger batch size over each iteration can significantly yield better training result, both better training accuracy and lower training loss. Instead of 2-layer model, we performed the experiment on the 3-layer model, where we choose batch size range from 50 to 200. The configuration of the model can be well represented with the following Table 6.

Cond. ID	No. of Convolution Layer	Batch Size	Pooling Function	Act. Function
1	2 layers	50	MAX	RuLU
2	3 layers	100	MAX	RuLU
3	5 layers	200	MAX	RuLU

Table 6: Experiment Conditions for CNN Model Batch Size Selection.

During the training, we discovered that as we increase the batch size, the training takes longer time proportionally. The training result is shown in the Figure 12 and Figure 13 below.

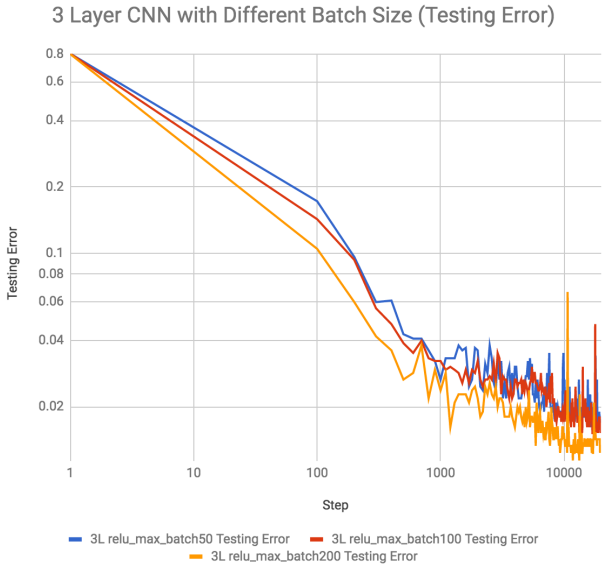

Figure 12: Experiment Results Comparison on Testing Error for CNN Model Batch Size Selection.

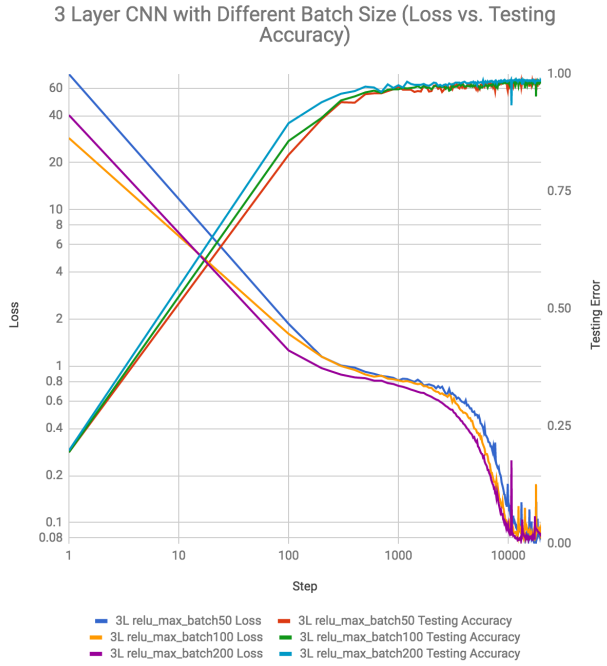

Figure 13: Experiment Results Comparison on Loss and Testing Accuracy for CNN Model Batch Size Selection.

The trend of the curve can be easily identified through the graph as well. In the Figure 12 which demonstrate the test error across the training progress, the curve with batch size of 200 performs better than the other 2 curves all along, and the curve of batch size 100 outperforms the batch size of 50 as well. The same trend can be identified in the training loss in the figure 13, but interestingly the batch size of 200 yield higher training loss in the beginning, but performs much better later on.

One of the reasons we choose to perform the batch size experiment on the 3-layer model is that we want to see if we could make 3-layer model to achieve as almost as good result as our default 2-layer model. And to our surprise, the batch size of the 200 on the 3-layer model even outperforms the 2-layer model. We compared the final test error and training loss with the 2-layer model, which is reflected in the Table 7 below.

Model	Training Accuracy	Training Loss	Testing Accuracy
2_layer_batch_50	1.0000	0.0836	0.9858
3_layer_batch_50	1.0000	0.0785	0.9829
3_layer_batch_100	1.0000	0.0868	0.9848
3_layer_batch_200	1.0000	0.0836	0.9867

Table 7: Experiment Results for CNN Model Batch Size Selection.

3.1.5 Activation Functions

Lastly, we did experiment on the activation function, in an effort to see if using activation functions other than ReLU can yield a better result. In this experiment, we avoided experimenting on sigmoid function and Tanh function due to time constraints, also because study has shown [5] that drawbacks like killing gradients when saturated and the expensiveness of exponential function makes them unfavorable in most CNN models.

The activation functions that are introduced in our experiment our ReLU, ELU and Leaky ReLU with alpha=0.2, which are all well-known for their merits like do not saturate and computationally efficient. We performed the comparison of the ReLU function and ELU function on the 5-layer model, and performed the comparison of the Leaky ReLU activation function and ReLU activation function on the 2-layer model. We can first examine the difference between the ELU and ReLU function with the Figure 14 show below.

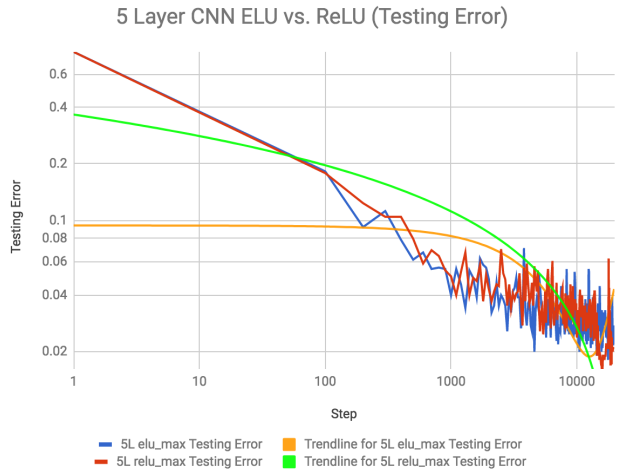

Figure 14: Experiment Results Comparison on Testing Error for CNN Model Activation Functions Selection.

When training, while it is not very obvious, but the stats shows that the ELU activation function take longer time, which is expected because ELU function involves exponential calculation. In the graph shown above, the blue and red curve shows the testing error of the training model across the 20k iterations, but the pattern is not clearly distinguishable. However, if we introduce the trendline for each curve, we can clearly see that although ELU activation function yields better result in almost first half of the training, but is surpassed by the ReLU function when the iterations reaches above 15k. A similar result can be more obviously retrieved using if we look at the training loss graph (Figure 15), where the ReLU activation function yields better result along the overall training iteration.

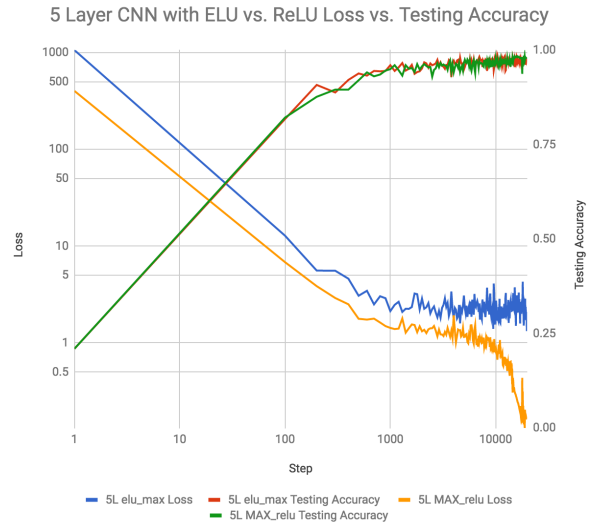

Figure 15: Experiment Results Comparison on Testing Error for CNN Model Activation Functions Selection.

The comparison of the performance of Leaky ReLU with alpha=0.2 and the ReLU has already been introduced in the previous section, where the ReLU function also shows a better performance than the leaky ReLU. And we can draw the result as shown below.

Cond. ID	Model	Training Loss	Testing Accuracy
1	2_layer_leaky_relu	0.0929	0.9801
2	2_layer_relu	0.0689	0.9839

Table 8: Experiment Results for CNN Model Activation Functions Selection.

3.1.6 Outcome

As a conclusion of our experiment, we found that the 2-layer model with ReLU activation function and max pooling function in general yields result that is more toward idea. And we also find that increase the batch size despite increasing the training time period yet in general can boost the performance to a higher level. Therefore, we increased the batch size of our 2-layer model to 200 and get the result shown in the Figure 16 below.

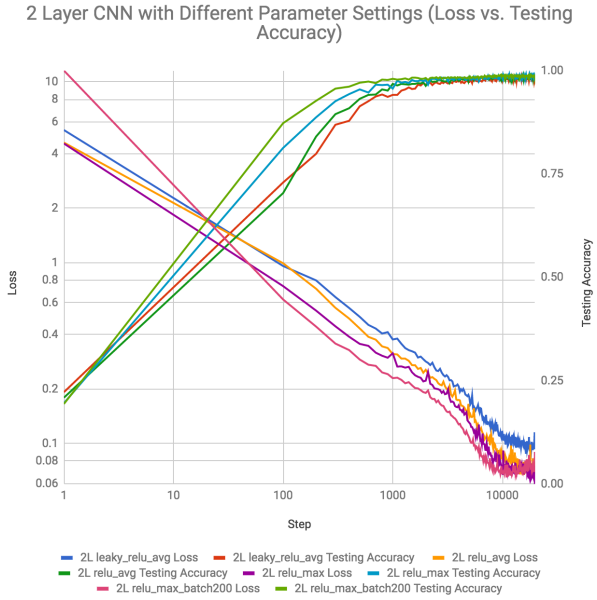

Figure 16: Experiment Results Comparison on Testing Error for CNN Model Outcome.

The pink curve and the light green curve shows the performance of the 2_layer_max_relu with batch_200 model, which surpasses all the other models shown in the graph. In the end, we obtained the training result of 0.9867 as the final testing accuracy and 0.0765 as the corresponding training loss.

During our training, we also discovered that the number of neurons in the fc layer, and the number of features we decide to

extract out in each convolutional layer will significantly affect the training time. We do believe that a better model might exist if we continue tune with the neurons and hidden layers, but considering the significant amount of training time, we decided to put it as topic in our further studies. But as far as our experiment has shown, two 1D convolution layer with ReLU activation function and max pooling with high batch size can produce the optimal result.

3.2 Recurrent neural network (RNN)

Recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behavior. After exploring the possibility of utilizing CNN to perform classification, we decided to implement the RNN solution to perform classification, since the motion data of the handwriting can be considered as temporal data. In the first attempt, we decided to implement a basic RNN network with LSTM cell. In this solution, there are three main parameters need to be tuned: learning rate, number of hidden unit in LSTM cell, and mini-batch size (or batch size). Since the relationship between these three parameters are not independent, for example, by isolating two parameters and find the best third parameter might not lead to the best third parameter if we change the other two parameters, we implemented a multi-threaded permutation engine to find the best parameter.

In this permutation engine, we iterated all the possible combination of these parameter, and trained them parallelly to save training and test time. Here is the table of all the combination we used in this set of RNN experiment. The last two columns reported the final training accuracy and testing accuracy on the model.

Exp. ID	Hidden	Learning	Batch	Training	Testing
0	64	0.001	128	0.916927	0.874568
1	64	0.001	256	0.949532	0.90046
2	64	0.001	512	0.921747	0.873993
3	64	0.005	128	0.97108	0.913694
4	64	0.005	256	0.973348	0.91542
5	64	0.005	512	0.989793	0.93786
6	64	0.01	128	0.976751	0.916571
7	64	0.01	256	0.993762	0.943613
8	64	0.01	512	0.993195	0.934407
9	128	0.001	128	0.988942	0.9229
10	128	0.001	256	0.98951	0.933832
11	128	0.001	512	0.99518	0.940736
12	128	0.005	128	0.999433	0.941312
13	128	0.005	256	0.998582	0.943613
14	128	0.005	512	0.999149	0.947066
15	128	0.01	128	0.998866	0.93786
16	128	0.01	256	0.999433	0.952819
17	128	0.01	512	1	0.948792
18	256	0.001	128	0.997448	0.939586
19	256	0.001	256	0.998015	0.933832
20	256	0.001	512	0.999716	0.935558
21	256	0.005	128	0.999716	0.944189
22	256	0.005	256	1	0.947066
23	256	0.005	512	0.999149	0.93901
24	256	0.01	128	1	0.947066
25	256	0.01	256	1	0.952244
26	256	0.01	512	1	0.945339
27	512	0.001	128	1	0.94649
28	512	0.001	256	1	0.944189
29	512	0.001	512	1	0.945915
30	512	0.005	128	1	0.939586
31	512	0.005	256	1	0.937284
32	512	0.005	512	1	0.934983
33	512	0.01	128	1	0.944764
34	512	0.01	256	1	0.935558
35	512	0.01	512	1	0.938435

Table 9: Experiment Conditions for RNN Model Selection.

On a side note, we also understand that the number of learning step also strongly influenced on the final training and testing

accuracy. However, due to the time constraint, and the purpose of this experiment (understanding how different parameters and model design lead to different result), we decided to use 15,000 number of learning steps across all the 36 experiment cases. At the end, it turns out this decision leads to a relative good result, which most of the experiment cases have converged down to about 90%-95% training and testing accuracy. Also, for control the effect on random training and testing set, we use train_test_split library in the sklearn to generate training set (66.6%) and testing set (33.3%) with a given pseudo random seed.

3.2.1 Number of Features in LSTM Cell

One of the most important parameter in the RNN network is the number of hidden unit that used in the LSTM cell. In the experiment we designed, we experimented four different number of hidden unit in the LSTM cell: 64 units, 128 units, 256 units, and 512 units. In order to get a better sense of how the changing of number of units changes the classification performance, we compute the average training and testing accuracy across the experience case: 0-8, 9-17, 18-26, and 27-35. The table below shows the result of the average computation.

No. of Units	Exp. ID	Training Accuracy	Testing Accuracy
64	0-8	0.965126	0.912287
128	9-17	0.996566	0.940992
256	18-26	0.999338	0.942654
512	27-35	1	0.940800

Table 10: Experiment Conditions for RNN Model Number of Features in LSTM Cell Selection.

Regrading to the training accuracy performance, we can discover that when we increase the number of feature unit in the LSTM cell, the average training accuracy keep improving. And when number of hidden units in LSTM cell equals 256, we can get best accuracy on testing set. However, when we increase the number of to 512, we can notice a drop in the testing accuracy. A reasonable theory behind this can be too many features in LSTM cell leads to the overfitting of the model. To understand this finding, we plot a comparison line chart on experiment 4, 13, 22, 3, which all have same parameter settings, but number of hidden units in LSTM cells differs.

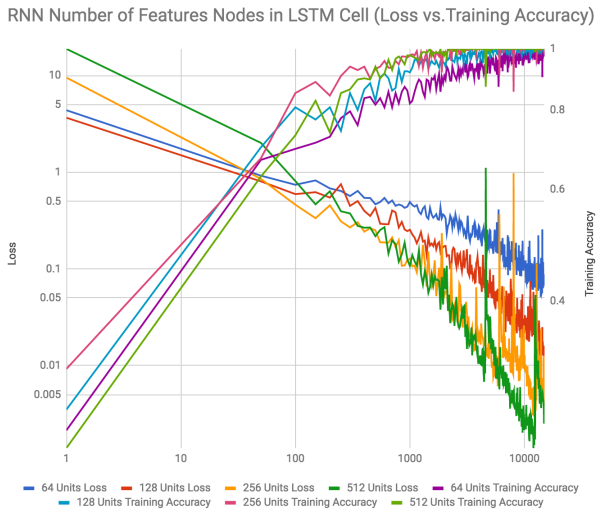

Figure 16: Experiment Results Comparison on Loss and Training Accuracy for RNN Model Number of LSTM Features Selection.

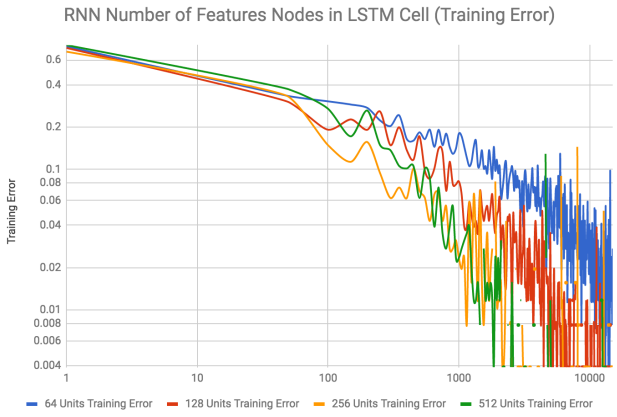

Figure 17: Experiment Results Comparison on Training Error for RNN Model Number of LSTM Features Selection.

From the above Figure 16 and 17, we can see that when we increase the number of hidden units in LSTM cell from 64 to 512, both training mini-batch loss and training error will decrease faster. Please note that the vertical and horizontal axis are all in log scale in order to show the trend better.

3.2.2 Learning Rate

In the experiment we designed, we also explored three possible learning rate in our solution: 0.001, 0.005, 0.01. The reason we chose these three set of value is we want to understand by increasing or decreasing the learning rate, the positive or negative impact on the speed of converging, and the change on batch loss and final accuracy. After running the experiment, we grouped

difference cases with their learning rate. The summary of the case comparison is in the table below.

In order to remove the effects of changing the other two parameters (hidden feature number and batch size), similar to the average computation we performed in 3.2.1, we compute the average within these three group of experiment. Here is the result of this average computation:

Learning Rate	Exp. ID	Training Accuracy	Testing Accuracy
0.001	0, 1, 2, 9, 10, 11, 18, 19, 20, 27, 28, 29	0.97975	0.92433
0.005	3, 4, 5, 12, 13, 14, 21, 22, 23, 30, 31, 32	0.99418	0.93675
0.01	6, 7, 8, 15, 16, 17, 24, 25, 26, 33, 34, 35	0.99683	0.94145

Table 11: Experiment Conditions for RNN Model Learning Rate Selection.

Base on the result of the average computation, we can see that the larger the learning step is, the better the result of training and testing accuracy. To verify this finding, we isolated the other two parameters and focused on the change of the learning rate. The following plot compares the change in the training loss in experiment case 19, 22, and 25.

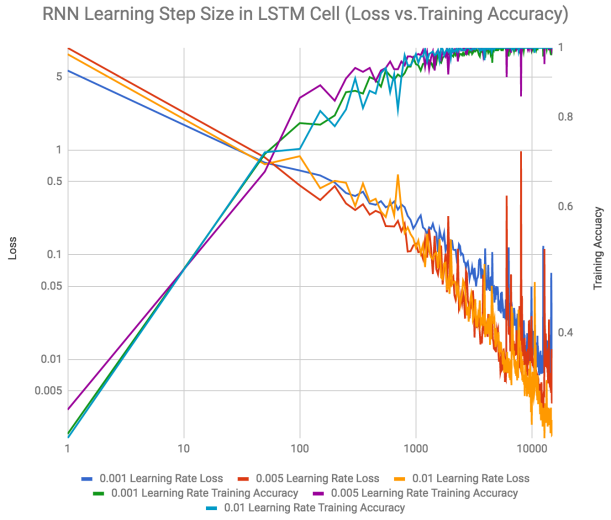

Figure 18: Experiment Results Comparison on Training Error for RNN Model Learning Rate Selection.

Based on the above Figure 18, we can clearly see that when we increase the learning rate, we can see the loss decrease faster (compare 0.001 learning rate loss vs. 0.01 learning rate loss). And the faster decrease on the loss leads to faster converge on the model, which provides a better training and testing accuracy with same number of learning iterations.

3.2.3 Batch Size

Another parameter that will influence the final outcome of the model is the batch size that used in training. We want to explore how does the change in batch size impact the final outcome of the classification result. We use the sample approach as we have in the previous two sections, which is taking the average of across cases to peek into the effect of tuning batch size. We group all the experiment result into three group.

Batch Size	Exp. ID	Training Accuracy	Testing Accuracy
128	0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33	0.98743	0.93071
256	1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34	0.99184	0.93666
512	2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35	0.99149	0.93517

Table 12: Experiment Conditions for RNN Model Batch Size Selection.

Base on the result from Table 12, when we increase the batch size from 128 to 256, there is an increase in both training and testing accuracy. However, when we keep growing the batch size to 512, both training and testing accuracy dropped. A reasonable guess to this phenomenon is that when we increase the batch size for each iteration, it will take longer for model to converge. Since we only use 15,000 iteration steps for all the experiments, the final accuracy of the model drops. To get a better sense on the trend, we selected three experiment cases with different batch size but isolated two other parameters: 21, 22, and 23. We plotted the training batch loss and training accuracy base on the iteration steps in Figure 19.

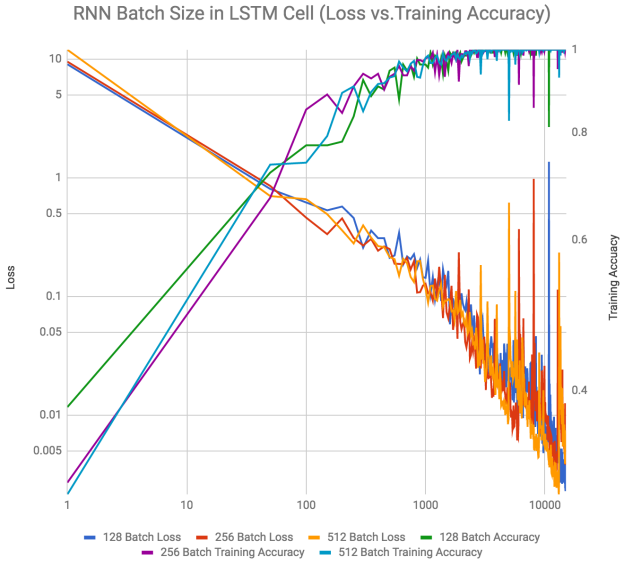

Figure 19: Experiment Results Comparison on Training Error for RNN Model Batch Size Selection.

Based on the above chart, we have noticed that there is no significant difference on the trend of the change on the loss. Although, it seems like increase the batch size into a larger value will make the change in loss and change in training accuracy become less stable. One guess to this finding is that since the batch becomes larger, changes to the model after each iteration will become larger. Since the logic for fetching the batch is random, so there is a chance in each learning iteration to select some data with bias (such as all the data mainly belong to one or two classes). In general, using 256 as batch size seems a reasonable choice.

3.2.4 Outcome

Based on above analysis on three main parameters we used in basic RNN model: number of hidden feature in LSTM cell, learning rate, and batch size; we discovered that the effect on tuning each parameter on the outcome of the final classification model. Here is the table of best five RNN models we discovered in this experiment process:

Exp ID.	Hidden Element	Learning Rate	Batch Size	Training Acc.	Testing Acc.
16	128	0.01	256	0.99943	0.95281
25	256	0.01	256	1	0.95224
17	128	0.01	512	1	0.94879
14	128	0.005	512	0.99149	0.94706
22	256	0.005	256	1	0.94706

Table 13: Top 5 Models based on Testing Accuracy in RNN Permutation Experiment.

Based on our research on the feasibility of deploying RNN model to our classification problem, we discovered the best hyperparameter to the classification, and be able to save these parameter for next stage real-time classification.

3.3 Dynamic RNN

Some study has shown that more sophisticated RNN architecture can yield better training result. One of the model to be taken into consideration is the dynamic recurrent neural network. Markus Mayer [6] mentioned in his experimental study that dynamic RNN model can produce the optimal result LSTM networks which is similar to the network that we are building. The biggest difference between RNN and dynamic RNN is that dynamic RNN allows different sequence length in different batches. By feeding batches of variable sizes, we will be able to perform fully dynamic unrolling of inputs and thus can expect better result.

Due to our limited datasets, relatively trivial classification task and time constraints, we didn't implement the training model with the dynamic RNN architecture. However, it is a promising approach to try out when we enrich our character motion data sets to the full set of alphabets.

4 CONCLUSIONS

In this paper, we presented DeepMotion, a brand-new approach to recognize handwriting characters in real time with deep learning models. We build up a complete pipeline from equipment setup, data collecting and raw data preprocessing to experiment design, algorithm implementation and result analysis.

To explore the best model for this special kind of datasets, we experimented through different hyper parameters in both convolutional neural network model and recurrent neural network model. In terms of the final testing error, the CNN model yields better result than the RNN model, which is to our surprise, because RNN is well-known for processing temporal data that is of our kind, but CNN outperforms in general across our experimentations. On the other hand, this also suggests that further study is needed to examine our way of processing raw data and the architecture of our self-designed deep learning model.

We believe DeepMotion can be a feasible way for character recognition in real applications, if our already obtained classification accuracy of 98% can be kept when our datasets get enriched with more character labels, as in the most recent speech recognition study [7], the accuracy of 94.4% is already ideally practical in the real world. In the future, we would like to implement a real-time recognition application based on our trained model so that our researcher can see timely feedback on the screen when a character is written. In addition, we would also enlarge our datasets to accommodate more characters into our model and make it useful in the actual product.

5 BOUNUS POINTS

5.1 Novel Ideas and Applications

a) We conducted researches on people's behavior of handwriting and did competitive analysis of many existing solutions, one of which is iSkn Slate [8], yet which still requires a writing pad with sensor. As a result, we finally came up with this innovative way by just using handwriting strokes as an input to the computing device without the need for writing pad or computer screen.

b) Our study in this paper has shown that deep learning models have made this idea feasible for set of characters, and very promising for larger set of characters as well if a more sophisticated model is designed.

c) We build the whole pipeline from equipment setup to data collection to training model design from scratch.

5.2 Large Efforts on Our Own Data Collection, Preparation, and Preprocessing

a) We purchased both MPU6050 and MPU9250 in an effort to find out the suitable sensor for our data collection.

b) An actual physical prototype is built using Arduino UNO R3 kit.

c) We designed algorithm for feature deduction and interpolation strategy to convert temporal raw data to trainable/learnable features.

d) Optimizations on our algorithms using multi-threading are used to boost our training progress on RNN model.

5.3 Comprehensive and State-of-the-art Deep Learning methodology and Result.

a) For each deep learning model, we tested on wide range of combinations of hyper parameters in order to understand how each parameter might affect performance of the training model on our datasets.

b) Instead of directly using existing models such as GoogleNet or ResNet, we spent effort to build neural network models specifically for our own data sets and tuned out the optimal hyper parameters across all experiments.

6 DOWNLOADS

1. Download pre-processing source code:
http://azureric.org/static/cogs181/final/pre_processing.py
2. Download experiment source code:
    a. CNN:
    http://azureric.org/static/cogs181/final/train_cnn.zip
    b. RNN:
    http://azureric.org/static/cogs181/final/train_rnn.zip
3. Download raw data set:
    a. Character "a":
http://azureric.org/static/cogs181/final/run_letter_a.csv
    b. Character "b":
http://azureric.org/static/cogs181/final/run_letter_b.csv
    c. Character "c":
http://azureric.org/static/cogs181/final/run_letter_c.csv
    d. Character "d":
http://azureric.org/static/cogs181/final/run_letter_d.csv
    e. Character "e":
http://azureric.org/static/cogs181/final/run_letter_e.csv
4. Video for EC:
http://azureric.org/static/cogs181/final/video.zip

ACKNOWLEDGMENTS

During this study, we have been fortunate to have the support and friendship of many individuals. First and foremost, we thank our professor, Zhuowen Tu. Over the weeks, Zhuowen has provided lots of great advises on our study, and he's suggestions have great inspired us to explore the new application of state-of-the-art deep learning field. We would like to next say thank you to all the TAs of this class: Long Jin, Zeyu Chen, and Adilijiang Ainihaer . They have provided great support to our study and even helped us to create more data sample as our training and testing set.

REFERENCES

[1] Ackerman, Rakefet and Goldsmith, Morris. 2011. "Metacognitive regulation of text learning: On screen versus on paper." Journal of Experimental Psychology (Jan. 2007), 18-32. DOI: http://ucelinks.cdlib.org/10.1037/a0022086

[2] Subrahmanyam, Kaveri and Michikyan, Minas. 2013. "Learning from Paper, Learning from Screens: Impact of Screen Reading and Multitasking Conditions on Reading and Writing among College Students." IJCBPL (Apr. 2013). DOI: https://www.igi-global.com/10.4018/ijcbpl.2013100101

[3] R. Smith. 2011."An Overview of the Tesseract OCR Engine," Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), Parana, 2007, pp. 629-633. DOI: http://ieeexplore.ieee.org/ 10.1109/ICDAR.2007.4376991

[4] TensorFlow, "Deep MNIST for Experts" (2017), https://www.tensorflow.org/get_started/mnist/pros

[5] Avinash Sharma V. "Understanding Activation Functions in Neural Networks"(2017), Medium Blog, https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[6] M. Mayer. "TensorFlow LSTM sin(t) Example" (2017), GitHub repository, https://github.com/sunsided/tensorflow-lstm-sin/blob/master/README.md

[7] C.C. Chiu, T.N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R.J. Weiss, K. Rao, K. Gonina, N. Jaitly, B. Li, J. Chorowski and M. Bacchiani, "State-of-the-art Speech Recognition With Sequence-to-Sequence Models" (2017), submitted to ICASSP 2018

[8] Lee, Dami. "iSKN Slate digitizes your paper doodles in real time using magnets" (2016), theverge, https://www.theverge.com/circuitbreaker/2016/10/27/13434636/iskn-slate-review-digitize-paper-drawings-magnetic-ring