Handwritten Alphabet Recognition with MLP, CNN, and SVM

Gu T.
V00890534

July 30, 2021

Contents

1 Introduction **1**

2 Related Work **1**

3 Problem Formulation **1**

3.1 Dataset 1

3.2 Model Evaluation 2

4 Methodology **3**

4.1 Data Preprocessing 3

4.2 MLP 4

4.3 CNN 4

4.4 SVM 4

5 Results and Discussions **4**

5.1 MLP 4

5.2 CNN 6

5.3 MLP vs. CNN 7

5.4 SVM 8

5.5 CNN vs. SVM 10

6 Conclusion **10**

List of Tables

1 MLP results 5

2 CNN results 6

3 SVM results 9

List of Figures

1 25 samples in dataset 2

2 5-Fold Cross Validation 3

3 MLP graph 5

4 MLP graph 6

5 MLP vs. CNN (batch size = 128, epochs = 10) 7

6 MLP (batch size = 128, epochs = 10) vs. CNN (batch size = 128, epochs = 5) 8

7 SVM results 9

8 CNN vs. SVM 10

Abstract

The abstract goes here.

1 Introduction

With the bloom in machine learning, handwritten alphabet recognition becomes a mature technique with wide application including handwriting input and Optical Character Recognition(OCR). This problem has been tackled by various machine learning algorithms and the outcomes are mostly pleasant. Given enough data, any basic deep learning algorithm can reach a recognition accuracy over 95%.
Patel shares a dataset with 372,451 samples and there are many successful implementations[1]. However, These implementations concerns majorly on the accuracy, ignoring the training cost. It is well-known that machine learning algorithms often implies a prolonged training time and time is critical in real world, especially business application. This vacancy in speed evaluation is reasonable. Since each implementation only involves one algorithm and these algorithms are run under different environments, comparing their training time is meaningless.
In this project, three popular machine learning algorithms, as follows, will be implemented and evaluated under the same environment.

1. Multi-layer Perceptron (MLP)
2. Convolutional Neural Network (CNN)
3. Support Vector Machine (SVM)

The comparison of their performance will demonstrate their corresponding advantage and disadvantage.

2 Related Work

As currently popular machine learning algorithms, there are both MLP[2] and CNN[3] implementations based on this dataset with an outstanding validation accuracy over 99%. On the contrary, no implementation based SVM, an old school algorithm, is used on this dataset. However, there exists SVM solution[4] on a similar dataset, MNIST, suggesting that SVM has the capability to do image recognition.
Jiang et al. have done a similar project in 2018, comparing KNN, SVM, MLP, CNN and Transfer Learning[5]. Unfortunately, their article did not come up with much conclusion and their source code on GitHub is no longer accessible. Also, their SVM shows a performance even "worse than arbitrary guess". Considering the success in SVM classifiers on MNIST[4], their implementation is likely to be problematic.

3 Problem Formulation

3.1 Dataset

The dataset is a CSV file containing 372,451 rows and 785 columns where each row represents a column. For each sample, the first column is its label represented as an integer $l \in [0, 25]$.

Each number from 0 to 25 means a to z in alphabet. There are 26 possible classes. The rest 784 columns is a flattened 28×28 matrix I.

$$I = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,28} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,28} \\ \vdots & \vdots & & \vdots \\ g_{28,1} & g_{28,2} & \cdots & g_{28,28} \end{bmatrix}$$

Each g in the matrix is in the interval $[0, 255]$, representing a gray scale color. The whole matrix I is a gray scale image with a handwritten character in it. Figure 1 demonstrates 25 samples transferred from matrix I. This is a classical classification problem with 784 features

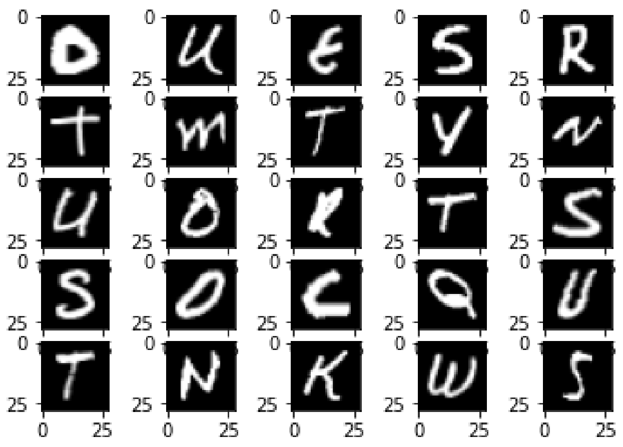

Figure 1: 25 samples in dataset

and 26 classes. The task of the algorithm is to find pattern in the given feature matrix and label vector, in order to classify other images.

3.2 Model Evaluation

Normally, the metrics used for neural networks is the loss relevant to epochs. However, this research involves SVM, which is a different system. Therefore, the comparison will be mainly based on accuracy and time.
Accuracy is not the best metrics since it is inconclusive when samples' distribution is severely unbalanced. For example, when 99% percent of the sample belongs to class A and only 1% belongs to class B, a model classifying all input into A will have a 99% accuracy. Thus, accuracy sometimes does not truly represent the quality of a model. However, observation on the dataset shows that in this case the 26 classes have a balanced distribution. It is assumed to be safe to use accuracy.
As mentioned before, the training time various in different environments, making it unfair for evaluation. In this project, the environment will be kept stable for every models.

For some models, various training data may cause a difference in performance. To ensure the result is representative, 5-fold cross validation is used. Figure 2 demonstrate the process of 5-fold cross validation. The whole dataset is split into 5 parts with nearly the same size D_1, D_2, D_3, D_4, D_5. Each configured model M will be trained and validated 5 times. For the

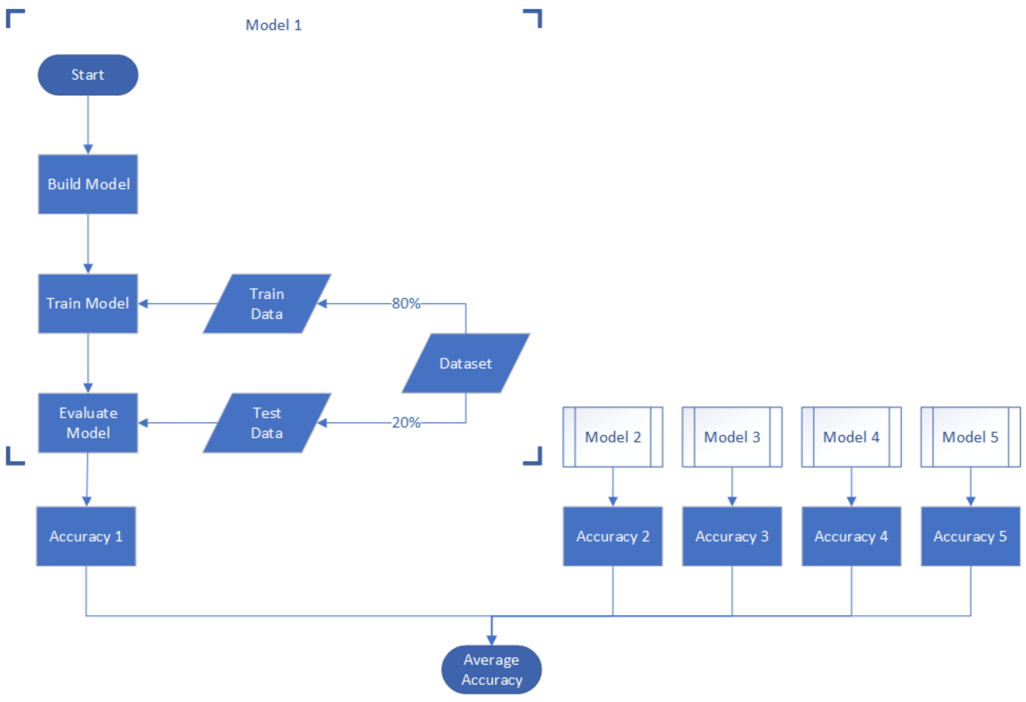

Figure 2: 5-Fold Cross Validation

i^{th} time, D_i will be used as training set while the rest as validation set; after training, the training time t_i will be recorded along with the validation accuracy a_i. The performance of a model is represented by the average.

$$t(M) = \frac{1}{5}\sum_{i=1}^{5} t_i(M)$$

$$a(M) = \frac{1}{5}\sum_{i=1}^{5} a_i(M)$$

4 Methodology

All experiments are coded in python, which has many packages useful in data processing and machine learning. The data pre-processing is done using NumPy. The implementation of MLP and CNN is based on TensorFlow framework and Keras. The SVM uses SVC in scikit-learn.

4.1 Data Preprocessing

For the first step, the whole dataset is loaded into a NumPy array. To generate proper input for MLP and CNN, the feature matrix shall first be normalized. Since the each value g is limited in the interval of $[0, 255]$, the normalization is simple. Divide each g by 255 is sufficient.

$$g \leftarrow \frac{g}{255}$$

As neural networks in Keras take categorical labels, it is necessary to transfer the original value to categorical vector. In this case, labels have 26 possible values (0-25), thus we do the following for each label value l to get a vector l':

$$l' \leftarrow (z_0, z_1, \cdots, z_{25})$$

$$z_i = \begin{cases} 1, & i = l \\ 0, & i \neq l \end{cases}$$

4.2 MLP

In this project, a MLP with 3 hidden layers, each with 128 nodes will be used. The batch size are chosen to 256 with 10 epochs. Based on structure above, different optimizers, activation functions, and dropout rates will be used. The optimizers chosen are Stochastic Gradient Descent (sgd), Adaptive Moment Estimation (adam), and Root Mean Square Prop (rmsprop). Sigmoid and ReLU activation function will be compared and as for dropout rate, it is a difference between no dropout (0) and small dropout (0.2).

4.3 CNN

CNN has a relatively similar structure as MLP. There are great differences but they are both neural network based algorithms. Thus, the optimal hyperparameters found for MLP will be used on CNN. With the optimal setting of MLP, a CNN with 1 convolution layer, 1 pooling layer, and 1 fully connected layer is built. The experiments on CNN will focus on the relationship between batch size (128, 256, 512) and epochs (5, 10, 20).

4.4 SVM

SVM has less option to modify. The kernel function is Radial Based Function (rbf). The comparison will be done between hard margin SVM and soft margin SVM along with 12 different γ values from 1×10^{-5} to 2×10^{-1}. It is worth mention that, if the hard margin SVM success to converge, implying that the dataset is linear separable, there is no reason to use soft margin SVM.

5 Results and Discussions

5.1 MLP

In total, 12 MLP configurations are trained and validated. Table 1 shows the result briefly, with only average test accuracy a and average training time t from 5-fold cross validation. Figure 3 gives a direct comparison on different configurations.

Obviously, tested configurations with sgd optimizer performs terribly ($a < 21\%$) on this problem while the other two optimizers do a decent job ($a > 98\%$). Although many former implementations shows that sgd is not the best solution, such performance is still abnormal. Observation on the training log tells that the loss has almost no decrease with epochs. It is more likely that the learning rate, which is set to default (0.01), is not suitable for this particular task. Anyway, these sgd based models have a unacceptable accuracy. Thus, they

Optimizer	Activation Function	Dropout Rate	a	t
SGD	sigmoid	0	16.34	52.85
		0.2	15.65	55.69
	ReLU	0	20.02	52.57
		0.2	16.26	56.86
Adam	sigmoid	0	98.35	55.9
		0.2	98.36	59.34
	ReLU	0	98.35	55.15
		0.2	98.31	59.62
RMSProp	sigmoid	0	98.39	79.41
		0.2	98.34	83.34
	ReLU	0	98.38	79.51
		0.2	98.32	83.75

Table 1: MLP results

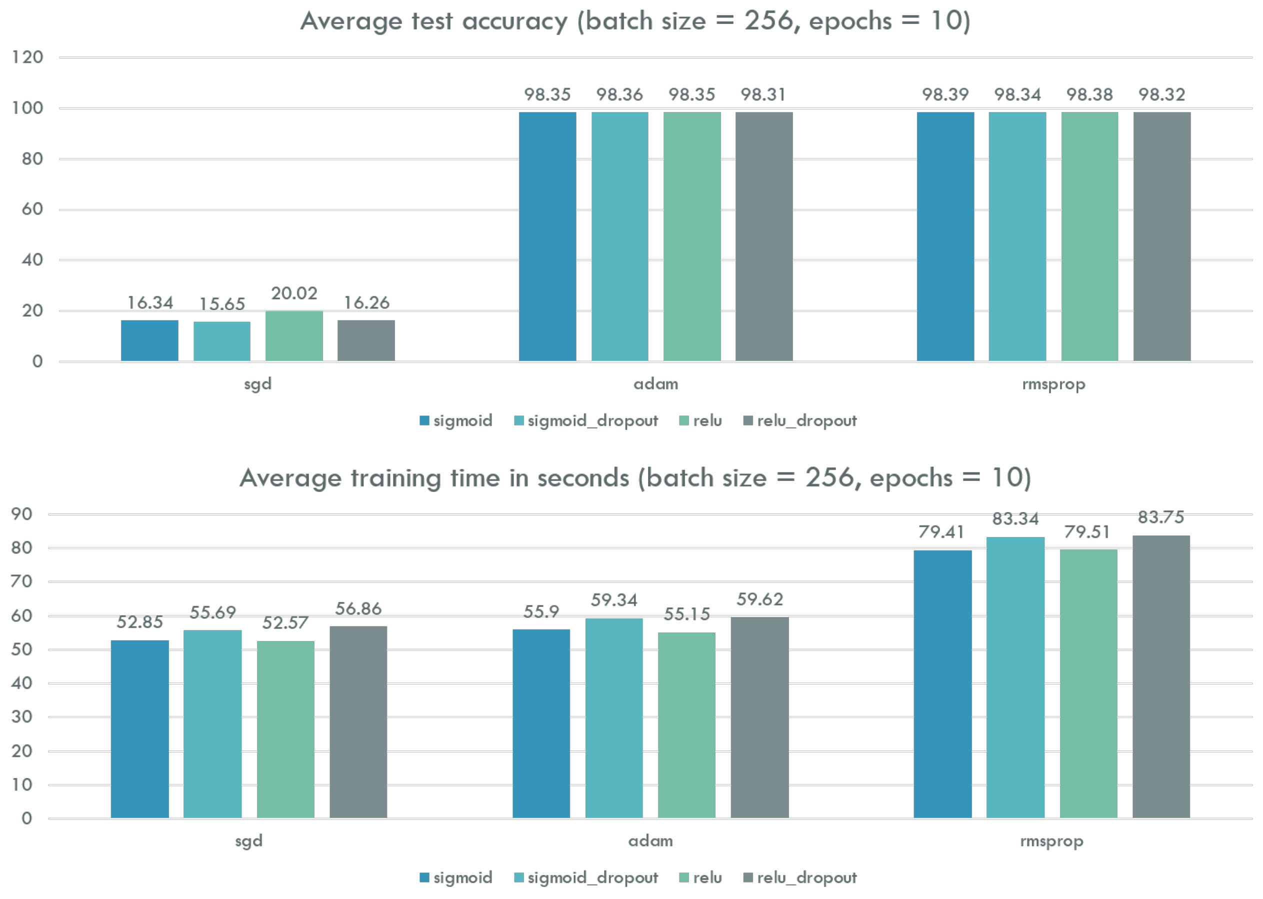

Figure 3: MLP graph

shall no longer be considered in the rest of this research. The result shows that adam has a better performance on training time, comparing to rmsprop. Therefore, adam is the best optimizer among the tested ones.
The average accuracy shows that dropout does not make any contribution to the accuracy. However, models with dropout takes longer time to train. Hence, dropout will not be used in further experiments. The training log tells that the training accuracy is close to the validation accuracy, which indicates that the model has no overfitting problem, leaving dropout unnecessary.
There is no significant difference on performance between two tested activation function. Thus, both of them are good to use. Here, we use relu. In conclusion, the configuration with adam optimizer, no dropout has the best performance. It will be used in further experiments with relu activation function.

5.2 CNN

With settings concluded from MLP, we use adam optimizer and relu activation function and no dropout in our CNN. There are 9 models tested in total with results shown in table 2. The change of a and t related to batch size and epochs are demonstrated in figure 4.

Apparently, with larger batch size, a will be smaller when the number of epochs is small.

Batch Size	Epochs	a	t
	5	98.78	61.56
128	10	99.00	115.28
	20	99.22	228.77
	5	98.53	41.31
256	10	98.98	77.59
	20	99.20	147.71
	5	98.27	31.74
512	10	98.85	57
	20	99.14	108.95

Table 2: CNN results

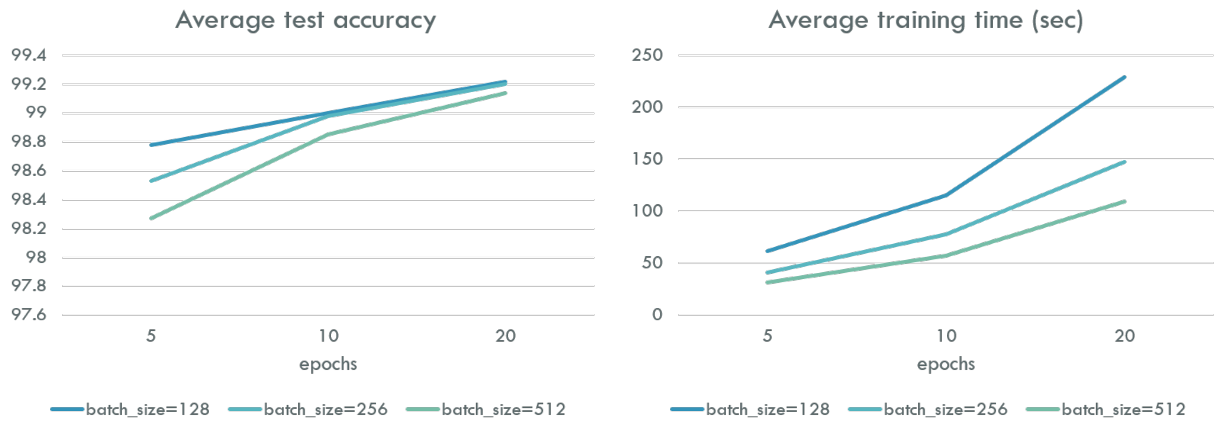

Figure 4: MLP graph

However, with the growth of epochs, the accuracy of different batch size converges to some upper limit. The shape of the curves are logarithmic. When the number of epochs is small, small batch size has an advantage on accuracy. However, once the number of epochs is sufficiently large, different batch size would yield to approximately the same accuracy.
The major advantage of large batch size is time taken for each epoch. The difference becomes even larger when the number of epochs is large. The shape of the curves fits $y = e^x$ or $y = x^2$. Based on the nature of curve of a and curve of t, we can conclude that, to reach a certain accuracy, the best strategy is to choose a batch size as large as possible and then select a sufficient number of epochs.
For example, when batch size is 128, at 10 epochs, the accuracy reaches 99.00% in 115.28 seconds; when batch is 512, at 20 epochs, it uses 108.95 seconds for 99.14% accuracy. With larger batch size, the model got a better accuracy with less training time.

5.3 MLP vs. CNN

With two algorithms evaluated, it is time to compare the performance of CNN against MLP. For a fair comparison, we chose the CNN with batch size of 128 and 10 epochs, which is the same as all MLP models.

This comparison in figure 5 shows that the accuracy of CNN is only 0.63% better than

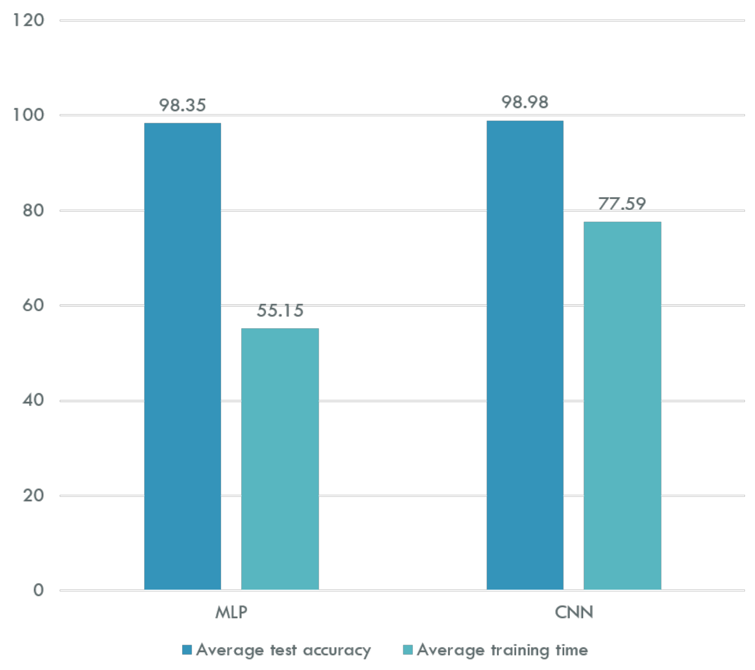

Figure 5: MLP vs. CNN (batch size = 128, epochs = 10)

MLP. However, it took extra 22 seconds. With such a small difference in accuracy, I almost concluded that MLP is faster. However, this comparison actually reveals the problem of using accuracy as metrics. To better show the real relation, check figure 6 for the performance of the same CNN at 5 epochs.
In fact, the accuracy of CNN at epoch 5 exceeds that of MLP at epoch 10. It only took the CNN model 42 seconds to reach this accuracy, meaning that CNN is still faster than MLP.

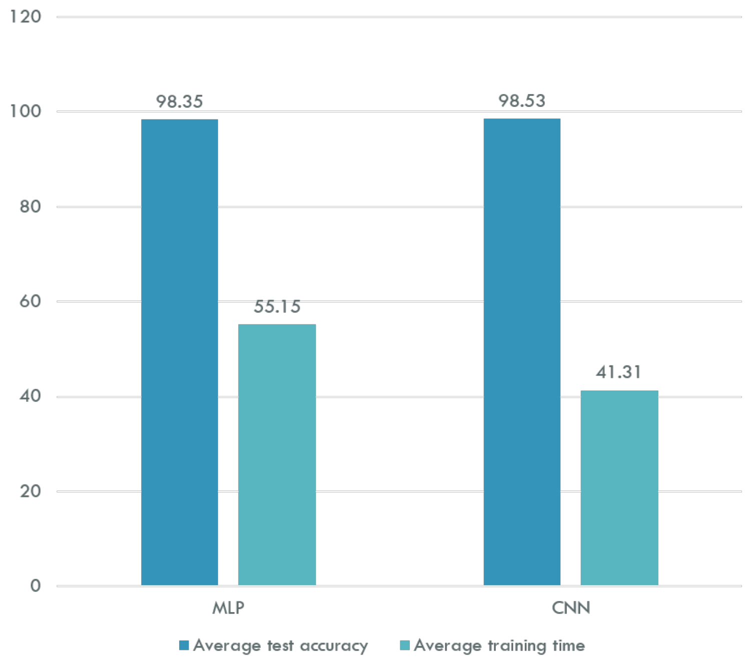

Figure 6: MLP (batch size = 128, epochs = 10) vs. CNN (batch size = 128, epochs = 5)

The lesson here is that, since the accuracy curve is logarithmic, neural networks may spend a long time to make a tiny progress when the accuracy is already high. Thus, any small difference in accuracy might actually be a big deal. Using loss for evaluation may make this difference more significant.

5.4 SVM

The first few attempts of training a SVM with the dataset failed. To be precise, SVM did not fail but they never finished. First, I assumed that the dataset is not linear separable. Thus, I adjusted the C value, making it a soft margin SVM. Unfortunately, nothing changed. Soon I realized, this problem is simply cause by the size of the dataset — it is way too large. There are successful implementations of SVM classifiers on MNIST dataset, which is a dataset with 1×10^4 samples. However, the dataset in this problem has over 3.7×10^5 samples, 37 times larger than MNIST.
While SVM fails on dealing with huge dataset, I still decide to train it with a small portion of the dataset. Namely, 0.25% of the dataset, which is 9311 samples, close to the size of MNIST, is used for training. The outcome is surprising: the SVM trained has a accuracy of 92.89%, validated using other 9311 samples.
SVM is proved to be valid on this problem. Further, since a hard margin SVM converges, it is unnecessary to use soft margin SVM. Thus, I tried 12 different γ values as shown in table 3. The change of accuracy and time along with γ is demonstrated in figure 7.
At $\gamma = 0.01$, a reaches the maximum while t is minimum. Undoubtedly, the optimal γ is close to 0.01.

γ	a	t
0.00001	54.34	46.95
0.00002	63.59	42.57
0.00005	76.33	30.37
0.00007	79.56	26.52
0.00009	81.60	23.63
0.0001	82.36	22.83
0.0003	86.87	15.7
0.0007	88.75	12.61
0.002	90.30	10.95
0.004	91.33	11.24
0.01	92.89	14.12
0.04	92.88	45.22
0.2	23.74	103.11

Table 3: SVM results

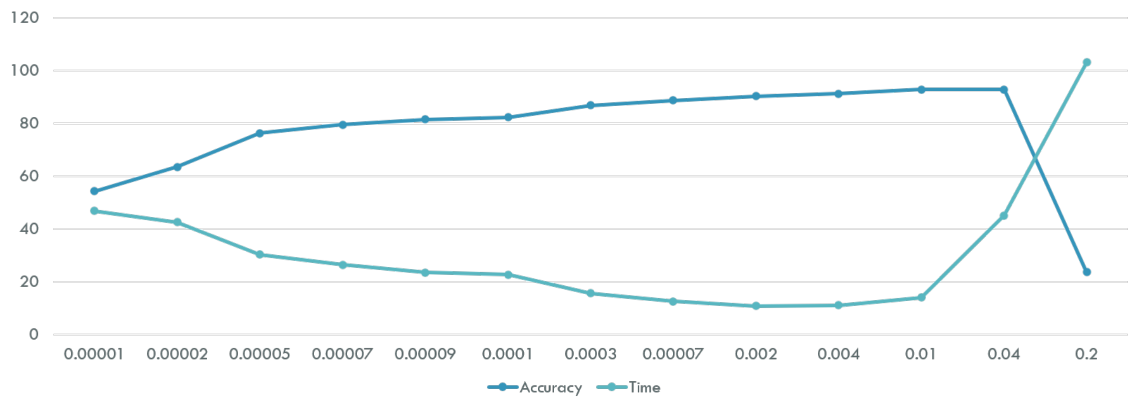

Figure 7: SVM results

5.5 CNN vs. SVM

We know that SVM is able to handle this task. However, comparing its performance from only 9,311 training sample with that of CNN with 372,451 training samples is unfair. Therefore, I decide to train the best configured CNN with 9,311 samples in order to see how CNN performs on small dataset.

In figure 8, both CNN and SVM are trained with 9,311 samples and validated against the

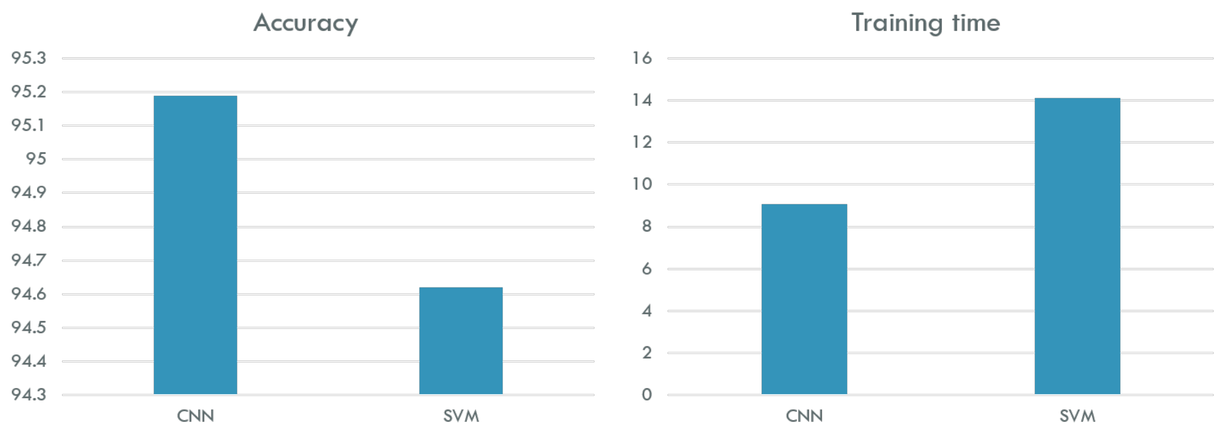

Figure 8: CNN vs. SVM

rest 99.75% of the dataset. As a result, the triumph belongs to CNN again. Even trained with a relatively small dataset, CNN still has better accuracy and shorter training time.

6 Conclusion

Now, we know that all three algorithms are capable to do handwritten alphabet recognition with a satisfying predicting accuracy. Also, it is fair to say CNN is the algorithm most suitable for this problem among all three algorithms tested. Another important conclusion is that, SVM is not a good choice for huge datasets.

Although there are some conclusions about how to choose hyperparameters drawn during the experiments, I argue that they are not valuable conclusion. First of all, they are not generalizable. These conclusion can only be apply on this particular problem. Actually, some settings tested proved to be terrible performs great on MNIST dataset. This implies that my experience in tuning these models will not help much when I try to solve another problem. Secondly, These optimal settings found are only approximated local optimal; they are not the precise global optimal. Without any theory backing up these choice, they are merely a posterior — we can only look for a good configuration by keep trying. There are so many hyperparameters and some of them are continuous, this is difficult.

The two reasons above are also drawbacks of data driven machine learning models, along with the difficulty of collecting such large amount of data. In addition, these models have nearly no explanatory value at all. Thus, the experience in solving one problem can hardly be transferred to a different situation.

Data driven models surely have great application value. However, they are more like a data mining tool instead of an intelligent agent. These drawbacks become its upper limit, preventing it from achieving strong AI. I look forward to future development solving these problems.

References

[1] S. Patel. (2018) A-z handwritten alphabets in .csv format. [Online]. Available: https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format/code

[2] (2021) Idss lab2 multilayer perceptron. [Online]. Available: https://www.kaggle.com/mrmorj/idss-lab2-multilayer-perceptron

[3] (2021) 99.9% accuracy on alphabet recognition by eda. [Online]. Available: https://www.kaggle.com/nohrud/99-9-accuracy-on-alphabet-recognition-by-eda

[4] (2016) Handwritten number recognition (2) — svm implementation on mnist handwritten number recognition. [Online]. Available: https://blog.csdn.net/ni_guang2010/article/details/53069579

[5] S. Jiang, X. Huang, and P. Wu. (2018) Comparing five image classification method: Knn, svm, bpnn, cnn and transfer learning. [Online]. Available: https://blog.csdn.net/btujack/article/details/80875652?utm_term=CNN%E5%92%8CSVM%E7%9A%84%E5%AF%B9%E6%AF%94&utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~all~sobaiduweb~default-0-80875652&spm=3001.4430