Convolutional Neural Networks and Support Vector Machines for Image Classification

Chloe Mills, Shania Wan-Bok-Nale, Khoi Nguyen
McGill University

COMP 551 Mini Project 4

1 Abstract

In this project, we were tasked to reproduce a scientific paper. We chose Abien Fred Agarap's paper [1] in which the main topics are Convolutional Neural Networks with Support Vector Machines in order to classify images in Fashion-MNIST and MNIST datasets. We compared the performance of this model with a CNN softmax model in order to see if an output SVM layer increases or decreases the test accuracy when classifying images in the datasets. We experimented with different hyper-parameters as well as parameters to see which model performed the best. We found that the CNN model with softmax output layer results in test accuracy of 99.32% and 91.88% for MNIST and Fashion-MNIST datasets respectively. On the other hand, the CNN model with a SVM output layer results in test accuracy of 99.17% and 91.75% for MNIST and Fashion-MNIST datasets respectively.

2 Introduction

We were tasked to reproduce a scientific work of our choice. Reproducibility is a critical aspect of scientific research. It provides the author's work with credibility and allows the aim of the research to be expanded. As a new and emerging field, Machine Learning does not have clear guidelines compared to the general sciences. Hence, the community has been encouraging the reproducibility checklist. Here we attempt to reproduce the results of the following paper: "An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification"[1] written by Abien Fred Agarap. The paper experiments with MNIST and Fashion-MNIST datasets using two models, namely Convolutional Neural Network with Support Vector Machines and Convolutional Neural Network with Softmax.

3 Scope of reproducibility

Abien Fred Agarap's paper[1], claims that using a Support Vector Machine (*SVM*) with Convolutional Neural Networks (*CNN*), does not improve the results of Image classification for the MNIST dataset nor for the Fashion-MNIST dataset compared to using Softmax activation function with CNN. In fact [1] claims that using Softmax activation function instead of SVM performs slightly better. Although, they point out that the model that uses SVM may achieve better results if preprocessing techniques were implemented.

The author also claims that using L2-SVM loss function rather than L1-SVM loss function produces better results on average since it is differentiable [1]. In our experiments we test out this claim.

4 Methodology

We used Abien Fred Agarap's code [2] with minor adjustments to account for older TensorFlow libraries used and adjustments to the SVM function. We used SVM optimizer as shown in equation (2). Abien Fred Agarap uses Euclidean norm squared which is defined as $\|w\|_2^2 = (\sqrt{\sum_{i=1} w_i^2})^2 = \sum_{i=1} w_i^2$ in place of $w^T w$ in equation (2).

4.1 Support Vector Machines

Support Vector Machines are used for binary classification but can be modified to support multi-class classification. We tested the following equations in order to determine the best one for the SVM layer in our CNN model. SVMs learning uses one of the following optimization equations.

$$\min_{w} \frac{1}{N} w^T w + C \sum_{n=1}^{N} \max(0, 1 - y_n(w^T x_n + b)) \tag{1}$$

$$\min_{w} \frac{1}{N} w^T w + C \sum_{n=1}^{N} \max(0, 1 - y_n(w^T x_n + b))^2 \tag{2}$$

SVMs learn the weight parameter w where C is a constant, y_n is the corresponding label and the predict function is $w^T x_n + b$ [1]. L1-SVM and L2-SVM are equations 1 & 2 respectively where L1-SVM minimizes hinge loss and L2-SVM minimizes the squared hinge loss.

4.2 Convolutional Neural Networks with Support Vector Machines

Convolutional Neural Networks is an artificial neural network that is most suited for computer vision. A popular use for CNNs is image classification. CNNs were inspired by Multi Layer Perceptrons but take a different approach when it comes to regularization. They may be built using multiple hidden layers which capture the spatial dependencies between the pixels of an image. A CNN model contains many layers with lower connectivity; in most cases they are used with a softmax layer as the final layer. The report "Deep learning using linear support vector machines" [3] challenges this norm by introducing Support Vector Machines to CNNs. Thus, we reproduced both a CNN with Support Vector Machine and a CNN with Softmax and compared.

4.3 Datasets

The MNIST dataset is one that is widely used by the Machine Learning community as the baseline dataset used in image processing and classification. It contains 60,000 training images and 10,000 testing images of handwritten digits. It has 10 classes and the dataset is evenly distributed among the classes. On the other-hand, Fashion-MNIST is a dataset developed by Zalando which consists of fashion and clothing items within 10 classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. It was developed as an successor to the MNIST handwriting dataset to further challenge classification models. The dataset itself contains 70,000 instances: 60,000 of them for training and 10,000 of them for testing. 1 instance of the dataset consists of a 28x28 grayscale image and it's classification target. When importing the dataset we normalized the values such that they are in the range of 0-1. We split the training dataset into 50,000 for training and 10,000 for validation. Figure 1 outlines the perfectly equal class distribution which ensures the model will not be biased towards a class.

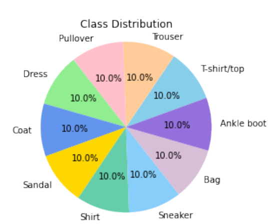

Figure 1: Equal class distribution in Fashion MNIST dataset

4.4 Hyper-parameters

We optimized the CNN-SVM model by testing different values of the penalty parameter C. The cost (penalty) parameter is a hyper-parameter of the CNN-SVM model that decides how much the linear seperator should be able to "bend" for classification. For a small cost, a decision boundary with a large margin is chosen at the expense of more misclassification. For a higher cost, we aim to classify more points correctly. We decided to use the Fashion-MNIST dataset as it is more challenging in order to find the best penalty parameter. Figure 4 in the appendix highlights our training accuracies over different values of C and Figure 5 in the appendix highlights the training loss. We then ran our model on the test set and table 1 highlights our results. We also experimented with different batch sizes further down.

C penalty Hyper-Parameters value	Test Accuracy(%)
1	91.65
2	91.75
5	92.01
10	91.79

Table 1: Test accuracy of CNN-SVM over different Hyper-parameters on Fashion-MNIST

We observe that $C = 5$ gives us the best test accuracy of 92.01%.

4.5 Experimental setup and code

We used Abien Fred Agarap's code [2] and modified the parameters (e.g. activation functions) and hyper-parameters (e.g. batch_size, penalty) of the functions and classes according to our experiments. However, for ablation studies, it involved further modifications to the code itself which we detail below.

The code provided by the author was written using the version 1 of tensorflow. In order to run the provided code, we imported version 1 of tensorflow, disabled the version 2 of tensorflow, added the import module input_data.py and imported Fashion-MNIST dataset from outside sources.

The layers of the CNN model are outlined in Appendix and an adaptation from [1].

In our experiments, we performed ablation studies to understand the impact of various components of the model regarding the final test accuracy. We individually removed dropout, one convolutional layer, both convolutional layers, activation functions, pooling layers, and the fully-connected layer and tested it against both the MNIST and FashionMNIST datasets.

Dropout: For dropout, we simply commented out the code involved. We plugged the result of the fully-connected layer directly into the readout layer, skipping the dropout layer. This is then sent into a Softmax layer or SVM layer, in order to obtain the results.

1 Convolutional layer: The CNN consists of 2 convolutional layers, which we decided to remove the second layer. We commented out the code pertaining to the second layer. We fed the first layer after ReLU and pooling into the fully-connected layer, skipping the second layer and its activation function and pooling. However, the shape of the first layer's matrix differs from the result of the shape of the second layer's matrix, which the fully-connected layer is expecting. Hence, we also modified the shape of the fully-connected layer's matrix in order to allow for proper matrix multiplication. We modified the shape from $(7 * 7 * 64, 1024)$ to $(14 * 14 * 32, 1024)$.

2 Convolutional Layers: Next, we removed both convolutional layers. We passed the input directly into the ReLU and pooling layers, skipping the first convolutional layer, and then fed the result into the second ReLU and pooling layers. This is then passed into the fully-connected layer, which we modified the shape from $(7 * 7 * 64, 1024)$ to $(7 * 7 * 1, 1024)$. Since the result from the second pooling layer is now different, we also had to modify the reshape parameters from $(-1, 7 * 7 * 64)$ to $(-1, 7 * 7 * 1)$.

Activation Functions: Each of the 2 convolutional layers and the fully-connected layer has a ReLU activation function attached to it. It was a simple process to remove the activation functions and plugging the convolutional layers straight into the pooling layers.

Pooling Layer: There is a pooling layer that follows each of the convolutional layers. We sent the result after each activation function of the convolutional layers to the next layer, skipping the pooling layer. We did not have to modify the shape of the second convolutional layer. However, we did have to modify the shape of the fully-connected layer, from $(7 * 7 * 64, 1024)$ to $(28 * 28 * 64, 1024)$, which the result of the second convolutional layer is passed into. We noticed that each of the 2 pooling layers shrunk the size of the matrix by a factor of 4, therefore without them, our second convolutional layer's matrix's size increased by a factor of 16. Therefore, we also had to change the parameters of the reshape/flatten function accordingly from $(-1, 7*7*64)$ to $(-1, 28*28*64)$. Because of the noticeable increase in size of the matrices, the training process took up a significant amount of memory and time.

Fully-connected Layer: There exists a fully-connected layer in between the second convolutional layer and the dropout layer. To remove this, we simply sent the result of the second convolutional layer after ReLU and pooling straight to dropout. However, the result of dropout is multiplied with the readout layer, therefore we have to modify the shape of the readout layer's matrix. We changed it from $(1024, 10)$ to $(3136, 10)$ to match the flattened pooling of the second convolutional layer of $(batch\_size, 3136)$

4.6 Computational requirements

We ran the models on our local computers. The hardware used were 2 MacBooks with Intel's 8th gen Core i5 with 8GB of RAM, a laptop with AMD Ryzen 9 5900HS with 8 cores and with 16GB of RAM, and a desktop with AMD Ryzen 5 2600 with 6 cores and 16GB of RAM. No GPUs were used in our experiments.

Most of our experiments were done with batch size of 128; in which our CNN-SVM model had an average runtime of 860 seconds, and our CNN-Softmax model had had an average runtime of 840 seconds. We also experimented with different batch sizes of 512, 256, 64, 32, 16, 8 and their runtimes are 2809, 1641, 514, 363, 226, 168 seconds accordingly.

For both models, the experiments on average, used up 1.2GB of memory. However, the pooling layer ablation experiment used up 2.2GB of memory due to the expanded matrix size. Therefore, in our case, we had enough memory to run multiple instances of experiments simultaneously to maximize our CPU usage. A faster runtime would require the usage of GPUs.

5 Results

Our goal was to reproduce the results in [1] using CNN-softmax and CNN-SVM models. We performed ablation studies and modified the models in order to understand their robustness and to evaluate their performance. We also experimented with L1-SVM and L2-SVM loss to see if the author's claims such that L2-SVM which minimizes the squared-hinge loss, provides higher accuracy than using L1-SVM which minimizes the standard hinge loss.

5.1 Results reproducing the paper

We ran both the CNN-Softmax and CNN-SVM over both MNIST and Fashion-MNIST datasets. Figures 6,7,8,9 in the appendix highlight our corresponding training loss and training accuracy results.

We observe that the four figures have similar trends as the plots in the paper [1]. On the MNIST dataset, we see a sharp increase in accuracy to above 90% in only 100 epochs for both models. Similarly, the loss over both models in the MNIST dataset have high correlation with each other. However, we observe CNN-softmax to have a general trend of higher training accuracy and training loss on fashion-MNIST dataset. Table 2 features our test accuracy results over the two models and two datasets.

We observe CNN-Softmax to have higher test accuracies than CNN-SVM in both datasets. Thus, we arrive at the same conclusion as [1] which states that CNN with Support Vector Machines does not necessarily perform better than CNN with Softmax. In fact, CNN-SVM performs slightly worse. We conclude that the paper [1] is reproducible.

	MNIST	Fashion-MNIST
CNN-SVM	99.17	91.75
CNN-softmax	99.32	91.88

Table 2: Test accuracy of CNN-SVM & CNN-softmax over MNIST and Fashion-MNIST

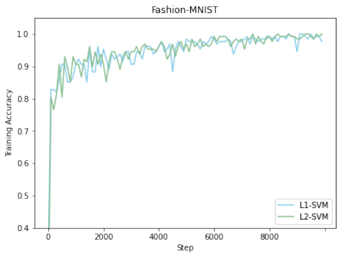

Figure 2: L1-SVM and L2-SVM training accuracy on Fashion-MNIST dataset

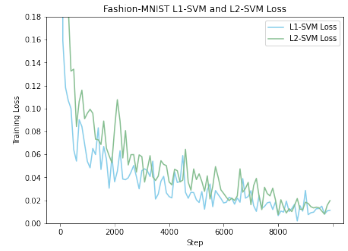

Figure 3: L1-SVM Loss and L2-SVM Loss on Fashion-MNIST dataset

5.2 Results improving SVM

When experimenting with L1-SVM compared to L2-SVM we found that L1-SVM actually performed slightly better than L2-SVM with test accuracy of 92.32% and 91.75% respectively on the Fashion-MNIST dataset. Since this difference is so small, we continued to use L2-SVM in our experiments since we wanted to reproduce the same results as [1]. Figures 2 & 3 display our results. They seem to both follow similar trends as the accuracy increases but L1-SVM seems to have sharper rises and falls in accuracy compared to L2-SVM.

For the MNIST dataset, using L2-SVM results in 99.17% and L1-SVM results in test accuracy of 99.21%. Figure 10 & 11 in Appendix display our results.

5.3 Ablation Studies

After removing each layer, we ran both CNN-SVM and CNN-Softmax and recorded the test accuracies on the MNIST and FashionMNIST datasets. After removing the layers, the accuracy decreased as expected, but by very little. On average, the difference compared to the original model was about 1%. Even after removing both convolutional layers, the difference was around 10%. What is astonishing, however, is that after individually removing the dropout layer and fully-connected layer, the accuracies increased instead. The accuracies from the dropout layer ablation using CNN-SVM surpassed the original model, while slightly falling behind using CNN-Softmax. Meanwhile, the fully-connected layer ablation achieved higher accuracies on average.

We speculate that the dropout layer ablation increases the accuracy because our model is not prone to overfitting, therefore dropping nodes would simply decrease the amount of data that is used to train the model. As for the fully-connected layer ablation, we speculate that it holds less importance to the model and might be causing our model to slightly overfit. According to our results, we find that the components with greatest importance would be the convolutional layers, followed by the pooling layers and finally the ReLU activation layers. This experiment displays how powerful and robust our CNN model is; even with important layers removed, it still produced high accuracy.

Layer(s) Ablated	MNIST SVM(%)	MNIST Softmax(%)	FashionMNIST SVM(%)	FashionMNIST Softmax(%)
No Layers Ablated	99.17	99.32	91.75	91.88
Dropout Layer	99.22	99.07	92.26	91.58
1 Convolutional Layer	99.06	98.76	92.22	91.73
2 Convolutional Layers	90.64	91.23	80.30	80.37
ReLU Layers	98.67	98.80	90.88	90.81
Pooling Layers	98.53	98.79	90.56	90.66
Fully-connected Layer	99.35	99.26	91.87	91.94

Table 3: Test accuracies of CNN-SVM and CNN-Softmax on MNIST and FashionMNIST datasets with layers ablated

5.4 Batch sizes and activation functions

We experimented with different batch sizes of 8, 16, 32, 64, 128, 256, 512. We found that the smaller the batch size, the faster the training process, but also the lower the test accuracy. This was inline with our expectations, since we are training with more samples per epoch. However, we found that slight increase in accuracy for batch sizes larger than 128 is no longer worth the long runtime.

Size: 8(%)	Size: 16(%)	Size: 32(%)	Size: 64(%)	Size: 128(%)	Size: 256(%)	Size: 512(%)
88.24	88.7	89.78	91.30	91.75	91.97	91.91

Table 4: Test accuracies of different batch sizes using CNN-SVM on FashionMNIST with best hyper-parameters

We also experimented with different activation functions on CNN-SVM such as ReLu, tanh, sigmoid, and Leaky-ReLu. The test accuracy results on the Fashion-MNIST dataset are 91.75%, 90.12%, 92.07% and 90.75% respectively. Surprisingly, sigmoid activation function achieved a higher accuracy than ReLu, whereas the other activation functions yielded a slightly lower accuracy.

6 Discussion

We were able to reproduce experimental results that support the claims of the paper. We experimented with L1-SVM in place of L2-SVM as well as using $w^T w$ as seen in equation (2) rather than Euclidean norm squared as seen in [1]. We did numerous additional experiments outside of the original paper, including ablation studies, effects of different batch sizes and activation functions and more. However, we did not have enough time and computational power to use k-fold with our experiments or run multiple runs of the same experiment, therefore we were not able to account for potential variance between runs.

6.1 What was easy

The code that we used to experiment on was very clear. The author's github explained well how to run the code. He commented his code thoroughly which made it easier for us to understand the logic when looking at his Support Vector Machines and Softmax implementations.

6.2 What was difficult

It was difficult to import the datasets. As the author created his code during the time that tensorflow was still in version 1, we encountered numerous Module Errors. The method to import the datasets were also different from the one that we usually used and instead we needed additional resources to add the import module and Fashion-MNIST dataset as GZ files. The author implemented SVM using Euclidean norm squared in place of $w^T w$ in equation (2). So understanding the reasoning behind this decision was difficult. Since the squared Euclidean Norm is simply squaring the weights and taking the sum; understanding why this produces accuracy on par with equation (2), involves a vast understanding of mathematics and algorithms.

Although the code was commented well, it was still sometimes difficult to modify the code during our ablation studies. Removing a layer usually changes the size of the matrices that are being multiplied together. This meant we had to determine the correct dimensions of the different layers' matrices and adjust it accordingly.

6.3 Statement of Contributions

All three members of the group contributed equally to this project. Nguyen worked on the ablation, Mills worked on the SVM implementation and Wan-Bok-Nale worked on the reproduction of the paper.

References

[1] Abien Fred Agarap. 2017.*An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification*, agarap2017architecture, arXiv preprint arXiv:1712.03541, (2017).

[2] abien_fred_agarap_2017_1098369, Abien Fred Agarap, AFAgarap/cnn-svm v0.1.0-alpha, 2017, 10.5281/zenodo.1098369, https://doi.org/10.5281/zenodo.1098369,

[3] Yichuan Tang. 2013. Deep learning using linear support vector machines. arXiv preprint arXiv:1306.0239 (2013)

7 Appendix

CNN architecture:

1. Input: 32 x 32 x 1
2. Convolutional Layer 1: 5 x 5, 32 filters
3. ReLu
4. Pooling: 2 x 2, 1 stride
5. Convolutional Layer 2: 5 x 5, 64 filters
6. ReLU
7. Pooling: 2 x 2, 1 stride
8. Fully-connected Layer: 1024 hidden units
9. Dropout: p = 0.5
10. Softmax Readout Layer: 10 output classes

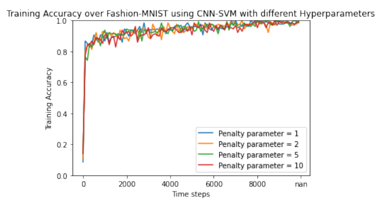

Figure 4: CNN-SVM Training Accuracy over Fashion-MNIST with different Hyper-parameters

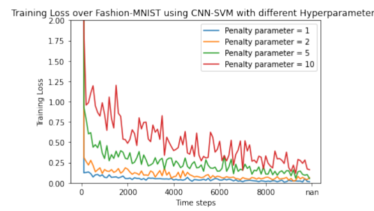

Figure 5: CNN-SVM Training Loss over Fashion-MNIST with different Hyper-parameters

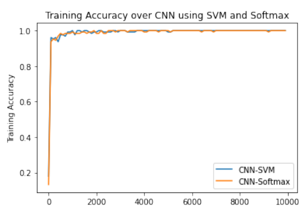

Figure 6: Training Accuracy on MNIST dataset

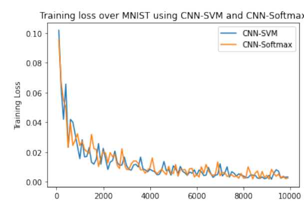

Figure 7: Training Loss on MNIST dataset

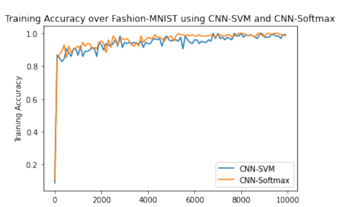

Figure 8: Training Accuracy on fashion-MNIST dataset

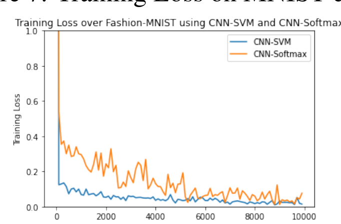

Figure 9: Training Loss on fashion-MNIST dataset

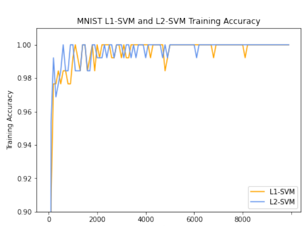

Figure 10: L1-SVM and L2-SVM training accuracy on MNIST dataset

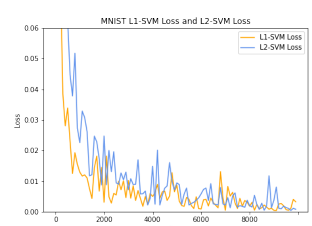

Figure 11: L1-SVM Loss and L2-SVM Loss on MNIST dataset