Deep Learning Final Project

Introduction to Artificial Intelligence

Economics / Bigdata Science / Japanese Culture
Junghun Lee
December 18, 2022

Contents

Abstract

This project is to directly classify the image dataset with 52 classes using what we learned during the semester. Commonly used convolutional neural networks (CNN) were used for classification. I used Multiple layers to increase the Accuracy, and use **Swish (Sigmoid Linear Unit)** for the activation function, which is known to perform better in the presence of multiple layers. After **20 epochs** with random seeds designated as 42, the accuracy was approximately **96.9103%** on test(validation) data set. What a nice performance that I think.

Project Purpose

The goal of this project is how well we classify the dataset given by 52classes. To achieve the goal of the project, first I devised neural networks such as LSTM or GRU. However, I realized that it didn't have much power in image classification, and I changed the direction to match the nature of the dataset of black-and-white images. There were already several models in the image classification problem (for example, networks such as Wave-mix and Resnet). I have boldly abandoned the new model to achieve the time limit conditions given to the project. Instead, I used a familiar model, which is named 'CNN'. I used **SiLU** as a loss function to induce better performance even through multiple layers.

Background Theory

The theoretical background of **CNN** was first introduced in LeCun's 1989 paper "*Backpropagation applied to handwritten zip code recognition*". **LeCun** later proposed a network called ***LeNet*** in 1998, which is the **first CNN**. It was simplified in 2003 by Behnke's paper "*Hierarchical Neural Networks for Image Interpretation*" and Simard's paper "*Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*", and many studies have been conducted. Following figure is the simplified structure of the CNN.

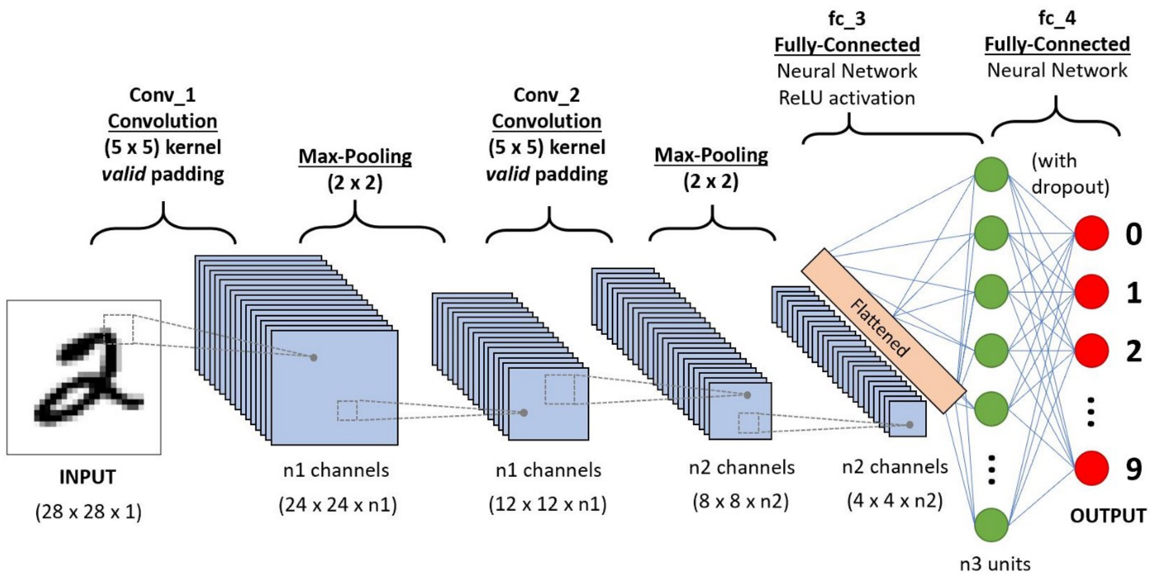


Figure 1. Convolutional Neural Network, by Sumit Saha

As followed figure, CNN learns the characteristics through multiple layers of the input image and finally predicts the class. In this process, ReLU (Rectified Linear Unit) is usually used as an activation function, but in this project I used **SiLU**(Sigmoid Linear Unit), which shows slightly better performance in multi-layer situations, to further boost performance.

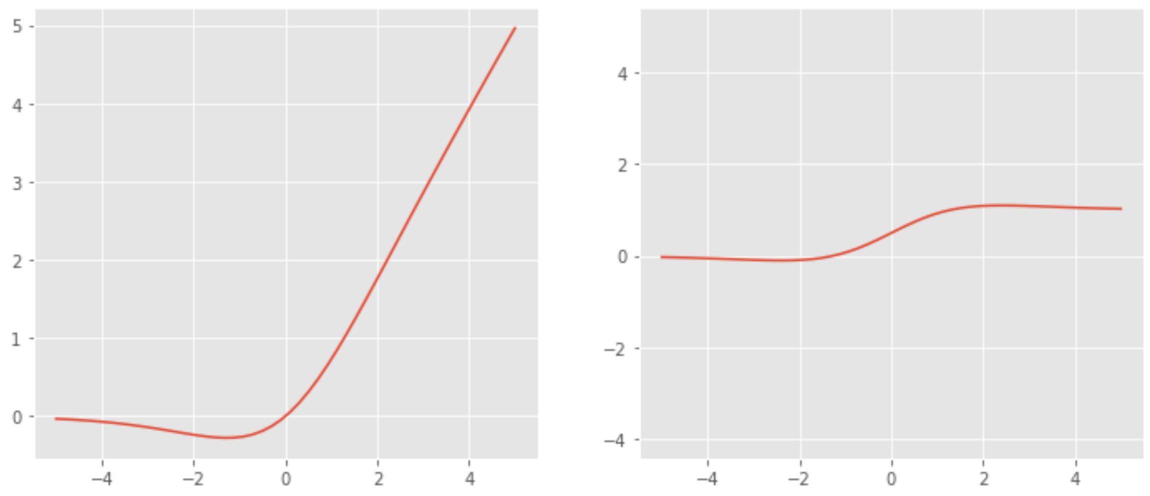


Figure 2. SiLU function, image by Subin Ahn

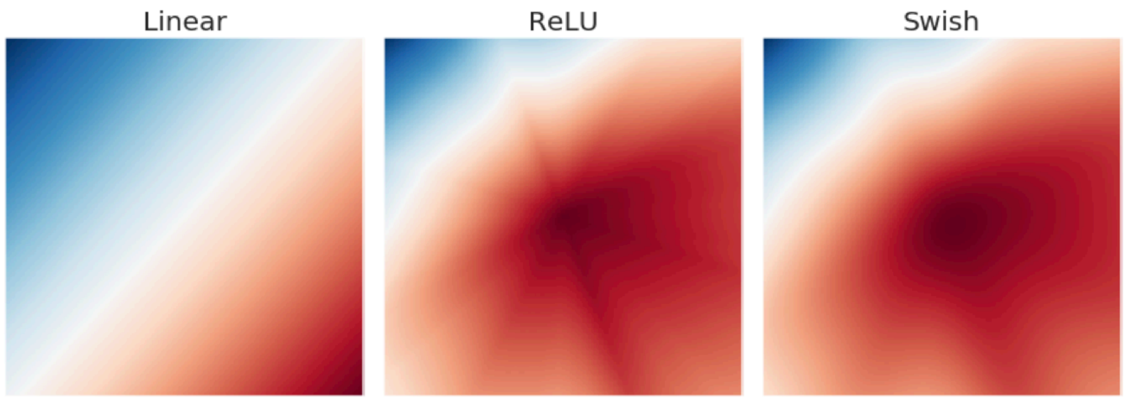


Figure 3. The comparison of Activation functions. Image by Prajit Ramachandran, Barrett Zoph, Quoc V. Le (2017)

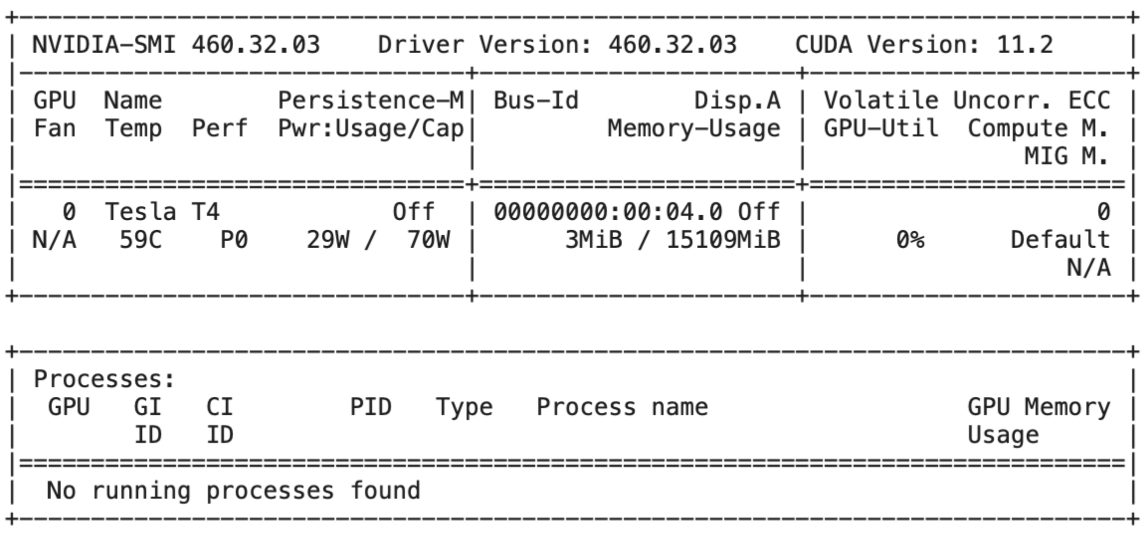
$$f(x) = \frac{x}{1 + e^{-x}} = x \cdot \sigma(x)$$

$$f'(x) = f(x) + \sigma(x)(1 - f(x))$$

The SiLU function is The published activation function made by Google Brain researcher (Prajit Ramachandran, Barrett Zoph, Quoc V. Le) at 2017. This activation function shows better performance in **deep multi-layer convolutional neural network** compared to ReLU. In situations where the region of x is less than or equal to 0, the SiLU function allows a slight negative region activation. According to the paper, it have shown better performance than ReLU in image classification problems such as CIFAR and MNIST.

Preset

There's something I have to do first before I make a model. In colab, GPU environment and Dataloader should be set according to project restrictions.

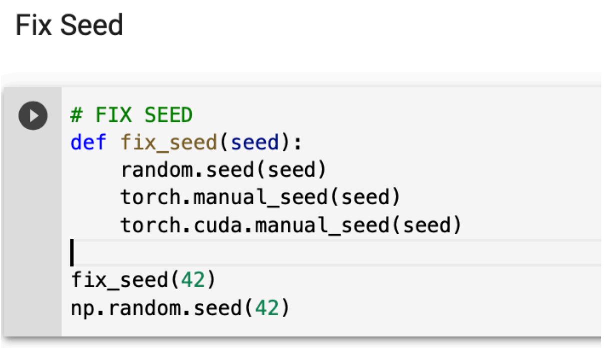


```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   59C    P0    29W /  70W |      3MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 4. Google Colab GPU Setting

And I designated random seed to 42, which is also our project's restrictions.



Fix Seed

```
# FIX SEED
def fix_seed(seed):
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)

fix_seed(42)
np.random.seed(42)
```

Figure 5. Fix Seed

We also need to check how the data looks like. Using pyplot, lets check a one of image.

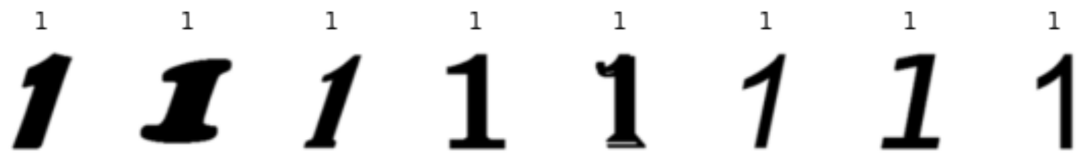
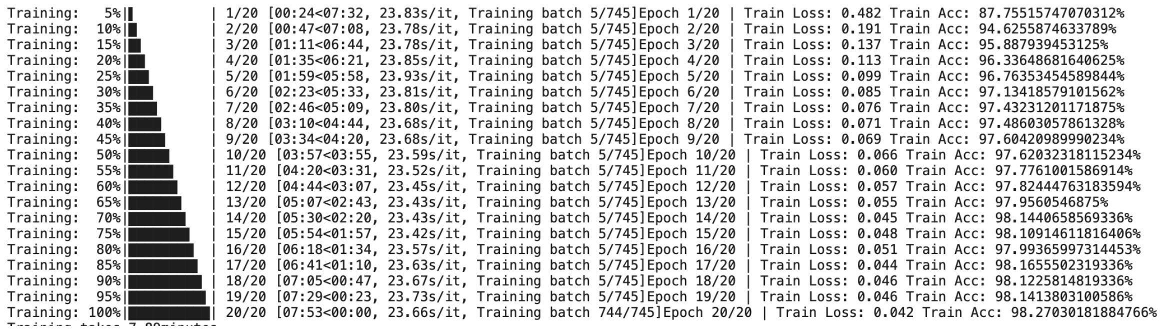


Figure 6. Check images

Model Construction

The size of the input image was 100 in height and 100 in width. The Data Set had 52 classes. Train data totaled 37232 and number of Vaildation Data is 7800. For learning, I initially constructed a convolutional neural network using two layers. However, the accuracy was less than 95 percent of what I was aiming for. In order to increase the accuracy, first I increased the number of layers. The number of layers has been adjusted appropriately because too many layers can cause gradient vanishing problems. Finally, seven layers of acuity resulted in a stable 95% record for test data. Here, I found out the SiLU activation function while looking for something special to differentiate my model. As a result of applying SiRU instead of ReLU function for all activation functions, performance is improved.



```
Training:   5%|█                   | 1/20 [00:24<07:32, 23.83s/it, Training batch 5/745]Epoch 1/20 | Train Loss: 0.482 Train Acc: 87.75515747070312%
Training:  10%|██                  | 2/20 [00:47<07:08, 23.78s/it, Training batch 5/745]Epoch 2/20 | Train Loss: 0.191 Train Acc: 94.6255874633789%
Training:  15%|███                 | 3/20 [01:11<06:44, 23.78s/it, Training batch 5/745]Epoch 3/20 | Train Loss: 0.137 Train Acc: 95.887939453125%
Training:  20%|████                | 4/20 [01:35<06:21, 23.85s/it, Training batch 5/745]Epoch 4/20 | Train Loss: 0.113 Train Acc: 96.33648681640625%
Training:  25%|█████               | 5/20 [01:59<05:58, 23.93s/it, Training batch 5/745]Epoch 5/20 | Train Loss: 0.099 Train Acc: 96.76353454589844%
Training:  30%|██████              | 6/20 [02:23<05:33, 23.81s/it, Training batch 5/745]Epoch 6/20 | Train Loss: 0.085 Train Acc: 97.13418579101562%
Training:  35%|███████             | 7/20 [02:46<05:09, 23.80s/it, Training batch 5/745]Epoch 7/20 | Train Loss: 0.076 Train Acc: 97.43231201171875%
Training:  40%|████████            | 8/20 [03:10<04:44, 23.68s/it, Training batch 5/745]Epoch 8/20 | Train Loss: 0.071 Train Acc: 97.48603057861328%
Training:  45%|█████████           | 9/20 [03:34<04:20, 23.68s/it, Training batch 5/745]Epoch 9/20 | Train Loss: 0.069 Train Acc: 97.60420989990234%
Training:  50%|██████████          | 10/20 [03:57<03:55, 23.59s/it, Training batch 5/745]Epoch 10/20 | Train Loss: 0.066 Train Acc: 97.62032318115234%
Training:  55%|███████████         | 11/20 [04:20<03:31, 23.52s/it, Training batch 5/745]Epoch 11/20 | Train Loss: 0.060 Train Acc: 97.7761001586914%
Training:  60%|████████████        | 12/20 [04:44<03:07, 23.45s/it, Training batch 5/745]Epoch 12/20 | Train Loss: 0.057 Train Acc: 97.82444763183594%
Training:  65%|█████████████       | 13/20 [05:07<02:43, 23.43s/it, Training batch 5/745]Epoch 13/20 | Train Loss: 0.055 Train Acc: 97.9560546875%
Training:  70%|██████████████      | 14/20 [05:30<02:20, 23.43s/it, Training batch 5/745]Epoch 14/20 | Train Loss: 0.045 Train Acc: 98.1440658569336%
Training:  75%|███████████████     | 15/20 [05:54<01:57, 23.42s/it, Training batch 5/745]Epoch 15/20 | Train Loss: 0.048 Train Acc: 98.10914611816406%
Training:  80%|████████████████    | 16/20 [06:18<01:34, 23.57s/it, Training batch 5/745]Epoch 16/20 | Train Loss: 0.051 Train Acc: 97.99365997314453%
Training:  85%|█████████████████   | 17/20 [06:41<01:10, 23.63s/it, Training batch 5/745]Epoch 17/20 | Train Loss: 0.044 Train Acc: 98.1655502319336%
Training:  90%|██████████████████  | 18/20 [07:05<00:47, 23.67s/it, Training batch 5/745]Epoch 18/20 | Train Loss: 0.046 Train Acc: 98.1225814819336%
Training:  95%|███████████████████ | 19/20 [07:29<00:23, 23.73s/it, Training batch 5/745]Epoch 19/20 | Train Loss: 0.046 Train Acc: 98.1413803100586%
Training: 100%|████████████████████| 20/20 [07:53<00:00, 23.66s/it, Training batch 744/745]Epoch 20/20 | Train Loss: 0.042 Train Acc: 98.27030181884766%
```

Figure 7. Model Training Progress

The learning rate is 0.001 for hyper parameter and the epoch is 20 times. Surprisingly, it showed 98.27% Accuracy for Train Data Set.

Model test and Result

Using pyplot, I expressed the results of learning Train Data Set.

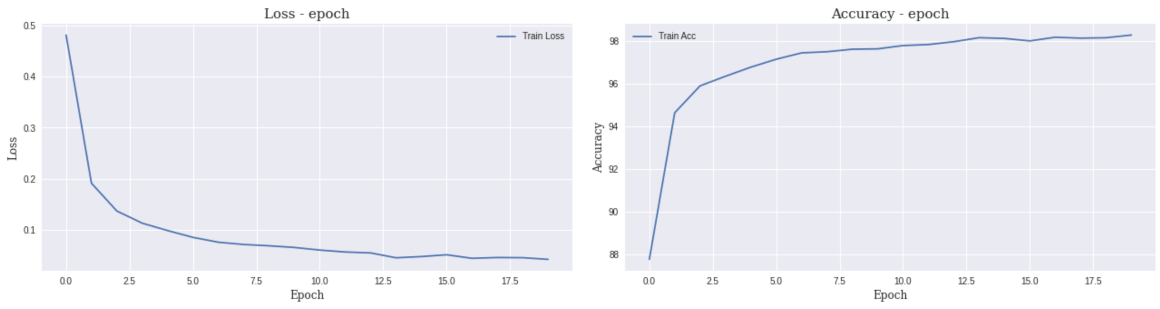
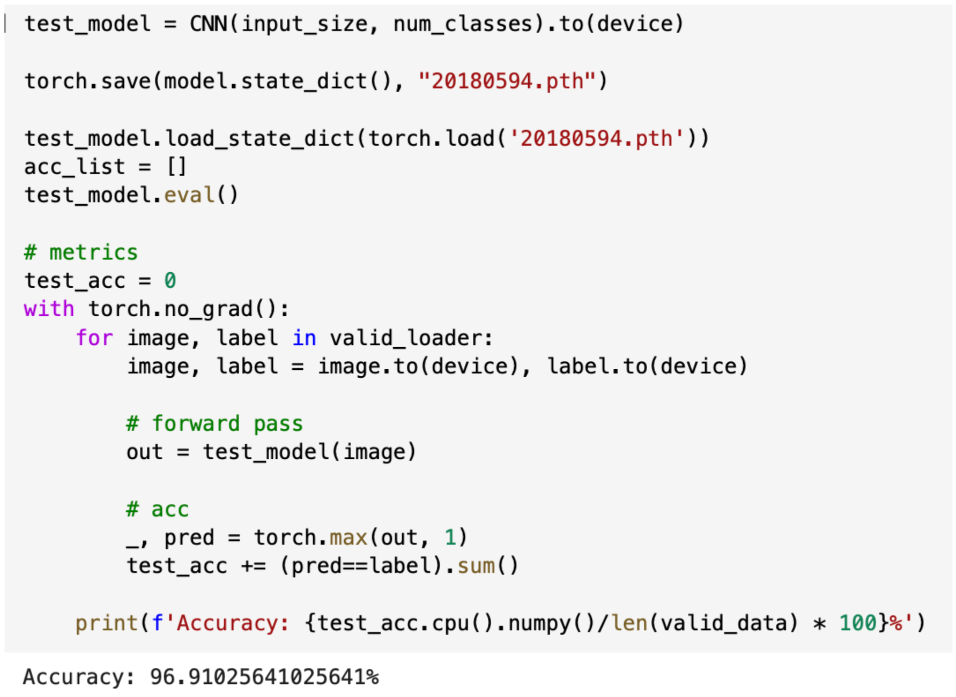


Figure 8. Model Training Result

The total learning time was measured to be about 8 minutes (7.89minutes) in Tesla T4 Environment. This is much faster than the project time restriction.



```
test_model = CNN(input_size, num_classes).to(device)

torch.save(model.state_dict(), "20180594.pth")

test_model.load_state_dict(torch.load('20180594.pth'))
acc_list = []
test_model.eval()

# metrics
test_acc = 0
with torch.no_grad():
    for image, label in valid_loader:
        image, label = image.to(device), label.to(device)

        # forward pass
        out = test_model(image)

        # acc
        _, pred = torch.max(out, 1)
        test_acc += (pred==label).sum()

    print(f'Accuracy: {test_acc.cpu().numpy()/len(valid_data) * 100}%')
```

```
Accuracy: 96.91025641025641%
```

Figure 9. The Accuracy onTest(Validation) Set

Finally, the Accuracy for the test (validation) dataset exceeded the existing target of 95%, reaching **96.91%**, which is close to 97%.

References

Figure 1. Convolutional Neural Network, by Sumit Saha (https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

Figure 2. SiLU function, image by Subin Ahn (https://subinium.github.io/introduction-to-activation/)

Figure 3. The comparison of Activation functions. Image by Prajit Ramachandran, Barrett Zoph, Quoc V. Le (2017) (https://arxiv.org/pdf/1710.05941v1.pdf)

Backpropagation applied to handwritten zip code recognition, LeCun's (1989)

Hierarchical Neural Networks for Image Interpretation. Behnke*(2003)*

Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. Simard(2003)

Deep Learning with Python. Francois Chollet(2017)

Swish : A Self-Gated Activation Function, Google Brain, Prajit Ramachandran, Barrett Zoph, Quoc V. Le (2017)