Name: Hanzhe Ye
Student ID: 1671744
Course: CMPUT 466
Instructor: Lili Mou

CMPUT 466 Final Project Report

Background

CIFAR, also known as Canadian Institute for Advanced Research, is a dataset consisting of thousands of tiny images collected and gathered from the web by a group of researchers. CIFAR-10 is a subset of the CIFAR dataset that has been selected and filtered for small machine learning tasks to improve training performances. This dataset is available at the website https://www.cs.toronto.edu/~kriz/cifar.html. For simplicity purposes, this dataset was directly downloaded and loaded to my program during execution by a python library called "TorchVision". To compare the performance of convolution models and traditional ML models on image classification, an experiment is designed where hyperparameters are controlled. Accuracy was used as the evaluation metric and test metric for all experimenting models. The objective is to minimize the losses, compare the prediction results, and select one of the best models with the highest test accuracy for this task.

Task Introduction

This project is designed to do image classification on the CIFAR10 dataset by using 3 different ML Models. In this dataset, there are 10 object classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each image inside the dataset has a size of 32 * 32 pixels with 3 color channels (RGB). There are around 60000 image data inside this dataset, 45000 of them were used as training data, 5000 of them were used as validation data, and 10000 of them were used as test data. A set of transformations has been applied to all 3 datasets so that these image datasets are transformed into tensors and properly normalized. Random horizontal flipping has been applied to the images in the training dataset to prevent overfitting issues.

In this task, three ML models have been used: Logistic Regression Model, Fully Connected Network, and Convolutional Neural Network. The structure of the Logistic Regression model is a Linear layer with a Softmax activation ($z = Xw + b, y = softmax(z)$). The loss function used is the cross-entropy loss, which is typical for logistic models.

The next model is a fully-connected neural network. There are 2 hidden linear layers with ReLU activation in this network. But, to differentiate it from the Logistic Regression model, this FCN structure is designed for linear classification. As a result, there are no non-linear activation functions in the last linear layer ($y = Relu(Relu(Xw1)\ w2)w3$). The loss function used is the Mean Square Error loss, which is suitable for linear classification with a linear boundary.

The last model is a convolutional neural network with a structure similar to VGG-16. Different from the previous two networks, two convolution layers with ReLU activation and max pooling have been added on top of the FCN layers to abstract and extract important features from the training images. In order to prevent overfitting, two small dropout layers have been added to the bottom of the two linear layers with a dropoff probability of 0.2. The last layer in FCN is a small softmax activation to calculate the estimation of class probabilities. The cross-entropy loss has been used to calculate the classification loss.

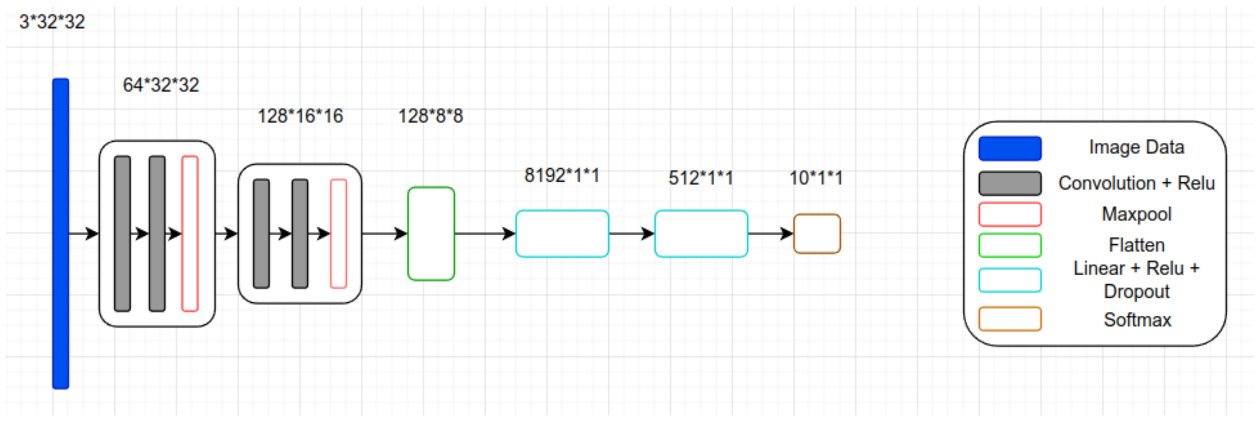


Figure 1.1: Structure of the ConvNet

To enhance the efficiency, the power of GPU has been harnessed to train these ML models. The entire training, evaluating, and testing process has taken place on a computer with a Nvidia 1660Ti GPU and 16GB RAM. The cuda module was loaded by pytorch functions.

The input tensors for all 3 models during training have a shape of (batch_size, 3, 32, 32). For the Logistic model and FCN model, these batches of images will be flattened to (batch_size, 3072) first before being fed to the first linear layer of the neural network. There is also a flatten layer in the CNN model between the last Maxpooling layer and the first linear layer. This is because the linear layers can only accept two dimensional matrices where one dimension must be the batch size of each dataset.

The output tensors for all 3 models are matrices with a size of (batch_size, 10), where 10 is the number of classes. The output returns 10 probability values for 10 categories of an image where each of them has a range of [0, 1]. The category with the maximum predicted value will be selected as the predicted category for a specific image. As a result, the predicted classes has a shape of (batch_size, 1), where each element is an integer in the range of [0, 9].

The validation and test accuracy have been set to be the real goals of the task. The accuracy was calculated by the proportion of predicted classes that are equal to the actual classes given by the validation or test dataset (groundtruth). The training process lasts for 50 epochs for all 3 models, for each epoch, the model will be validated on the validation set, both validation loss and accuracy will be calculated. The training function will return the best model trained on the best epoch where the validation accuracy reaches the highest point. The validation loss on that epoch will also be returned.

Hyperparameters

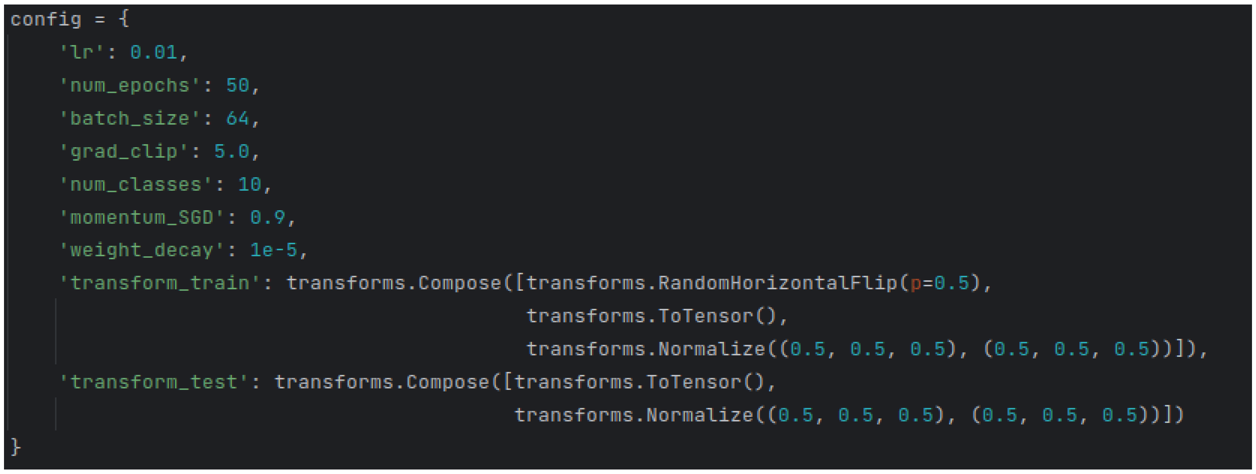
To improve the experiment and reflect the quality of the models, all hyperparameters are controlled – the learning parameters are exactly the same for all models. To find the best learning rate which makes the difference in the performance of these models maximized, the program has been run for several times. For each time, a learning rate α has been selected from the range [0.001, 0.1] and the test accuracy for all models has been recorded. After several experiments, $\alpha = 0.01$ has been observed to be the best learning rate. At this α, all the models can converge fast enough while none of them are overshooting.

The optimizer used by all 3 models is SGD, which is robust in minimizing the loss when the complexity of the model is high and weight capacity is large. Compared to Adam, SGD is preferred when the data is enormous and prediction accuracy is an essential factor. Although Adam uses adaptive learning rate for gradient descent which can make the convergence of the loss faster, the loss might be misled to a wrong direction and overshoot when the objective function is complex.

The gradient clipping was still used to prevent exploding gradients for the Logistic model and the CNN model. Although, it is hard for the gradient to explode since all data has been normalized to a small range. The weights are all initialized by Xavier Initialization for all 3 models to prevent vanishing gradient problems.

The batch size has been set to 64 instead of 256 since it requires a lot of GPU sources to compute gradients when the batch size is large. To prevent overfitting issues during training and penalizing dominated weights, L2 regularization with a decay parameter of 1e-5 has been introduced to the loss functions of all 3 models. The momentum for SGD is set to 0.9, which is the default setup for neural network and ML model training. The momentum of SGD is defined as a method to help accelerate the gradient in the right directions, thus leading to faster convergence for models.

Finally, a set of hyperparameters has been decided. After feeding these parameters to the models and training for several times, the differences in test accuracies were fully maximized. Here are the final hyperparameters listed, which is exactly the same in the submitted codes:

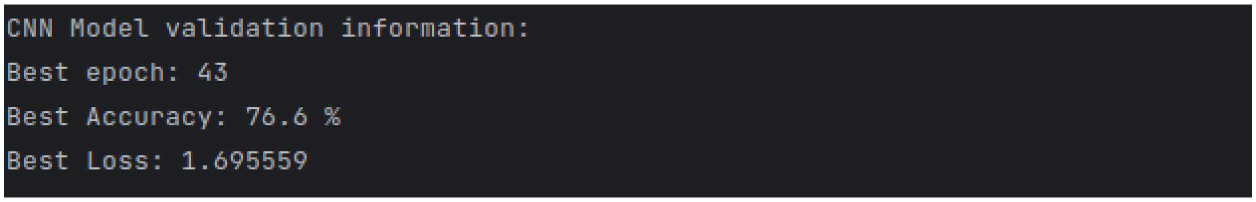


```
config = {
    'lr': 0.01,
    'num_epochs': 50,
    'batch_size': 64,
    'grad_clip': 5.0,
    'num_classes': 10,
    'momentum_SGD': 0.9,
    'weight_decay': 1e-5,
    'transform_train': transforms.Compose([transforms.RandomHorizontalFlip(p=0.5),
                                           transforms.ToTensor(),
                                           transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]),
    'transform_test': transforms.Compose([transforms.ToTensor(),
                                          transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
}
```

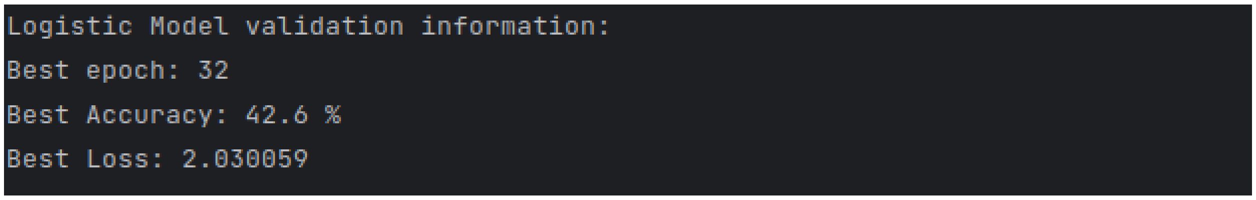
Figure 2.1: Final Hyperparameters list

Results

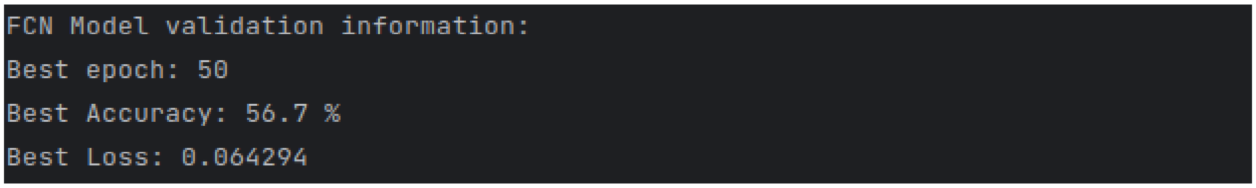
Here are the evaluation results after training each of the 3 models for 50 epochs by using preferred hyperparameters mentioned above. The model trained from the best epoch with the highest validation accuracy has been saved. The validation losses of the 3 models trained from the best epoch have also been recorded and displayed:



```
CNN Model validation information:
Best epoch: 43
Best Accuracy: 76.6 %
Best Loss: 1.695559
```



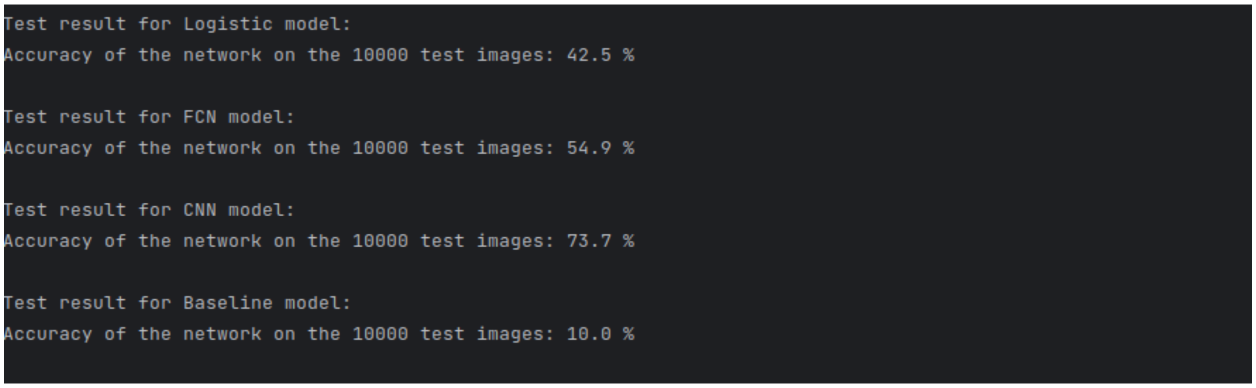
```
Logistic Model validation information:
Best epoch: 32
Best Accuracy: 42.6 %
Best Loss: 2.030059
```



```
FCN Model validation information:
Best epoch: 50
Best Accuracy: 56.7 %
Best Loss: 0.064294
```

Figure 3.1, 3.2, and 3.3: Evaluation Results for 3 models with default Hyperparameter

Here are the performances of all 3 best saved models on the testing dataset. The accuracy has been selected as the test metric. In addition to the previously trained models, the baseline model – which is simply returning the first category as the prediction for every input feature x, has also been tested along with 3 other models for comparison purposes:



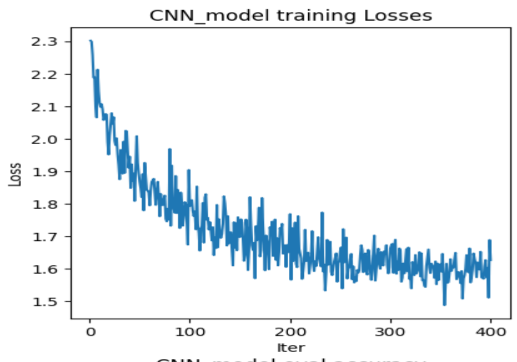
```
Test result for Logistic model:
Accuracy of the network on the 10000 test images: 42.5 %

Test result for FCN model:
Accuracy of the network on the 10000 test images: 54.9 %

Test result for CNN model:
Accuracy of the network on the 10000 test images: 73.7 %

Test result for Baseline model:
Accuracy of the network on the 10000 test images: 10.0 %
```

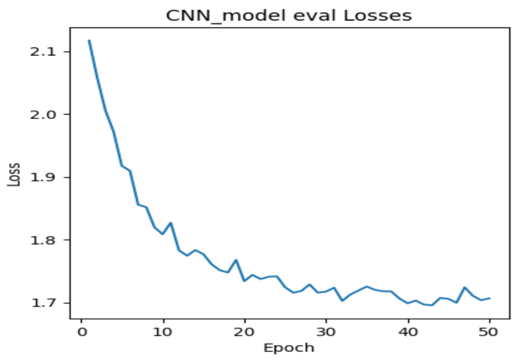
Figure 3.4: Test Result for 3 models along with the trivial baseline

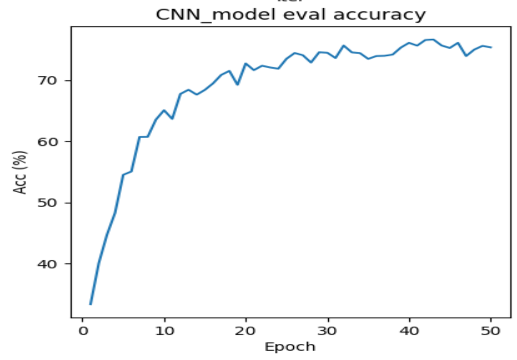
According to the information collected and displayed above, all 3 models performed better than the baseline model, which is simply making trivial predictions on the test dataset. The CNN model is the best model among all 4 models for the CIFAR-10 image classification task – it has both the best validation accuracy and test accuracy, which just met our criteria. Notice that the evaluation loss of the CNN model is not the minimum among all 3 models, this is because the cross-entropy loss exaggerates the small discrepancies between the prediction probabilities and the actual targets. For MSE loss, all these small discrepancies have been almost neglected when the data range is small, especially when the probabilistic data ranges from [0, 1]. This is why FCN performed much worse even when its loss seemed to be the smallest.

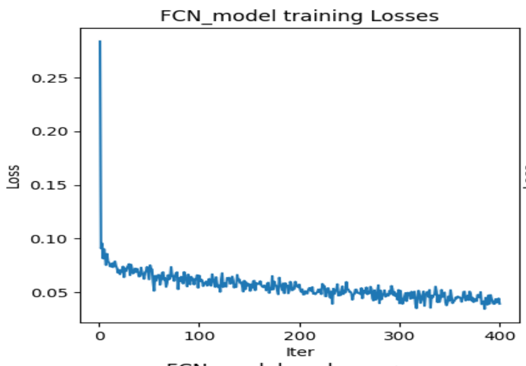
The CNN model has outcompeted the pure fully-connected networks because the convolution layers can actually discover spatial relationships and ordering from the input image pixels and abstract them during the forward pass. By extracting important features and filtering out unnecessary noises, the CNN model can make the image learning process more stable and focused. For example, in the training dataset, around 50% of input images have been horizontally flipped. The validation dataset also contained some images that are just offset by 1 or 2 pixels from the original training images. If we flatten these images into d dimensional vectors and feed them into a pure fully-connected neural network, the order information of each pixel from these images will be lost – this will make the predicted class probabilities on these images completely different from the predicted class probabilities on the original images. In other words, the traditional fully-connected networks will be fully tricked by the noises on these input images.

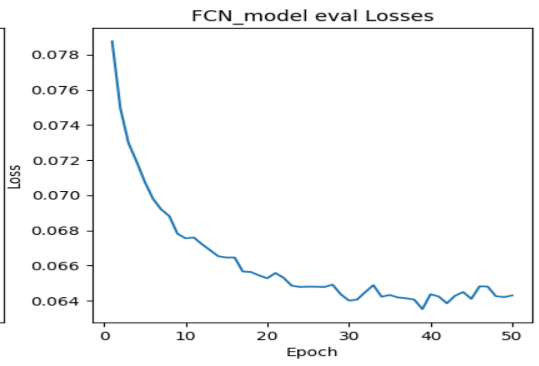
Another interesting result discovered from the evaluation result and test result from this task is that the FCN model with no activation in the last layer actually worked a little bit better than the logistic model. This is counterintuitive, because softmax activation is designated for k-categories classification tasks. Fortunately, this strange result has been explained by the following loss and accuracy graphs:

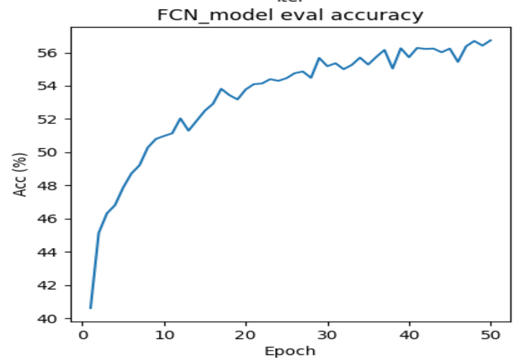

CNN_model training Losses
Loss
Iter
2.3
2.2
2.1
2.0
1.9
1.8
1.7
1.6
1.5
0
100
200
300
400


CNN_model eval Losses
Loss
Epoch
2.1
2.0
1.9
1.8
1.7
0
10
20
30
40
50


CNN_model eval accuracy
Acc (%)
Epoch
70
60
50
40
0
10
20
30
40
50


FCN_model training Losses
Loss
Iter
0.25
0.20
0.15
0.10
0.05
0
100
200
300
400


FCN_model eval Losses
Loss
Epoch
0.078
0.076
0.074
0.072
0.070
0.068
0.066
0.064
0
10
20
30
40
50


FCN_model eval accuracy
Acc (%)
Epoch
56
54
52
50
48
46
44
42
40
0
10
20
30
40
50

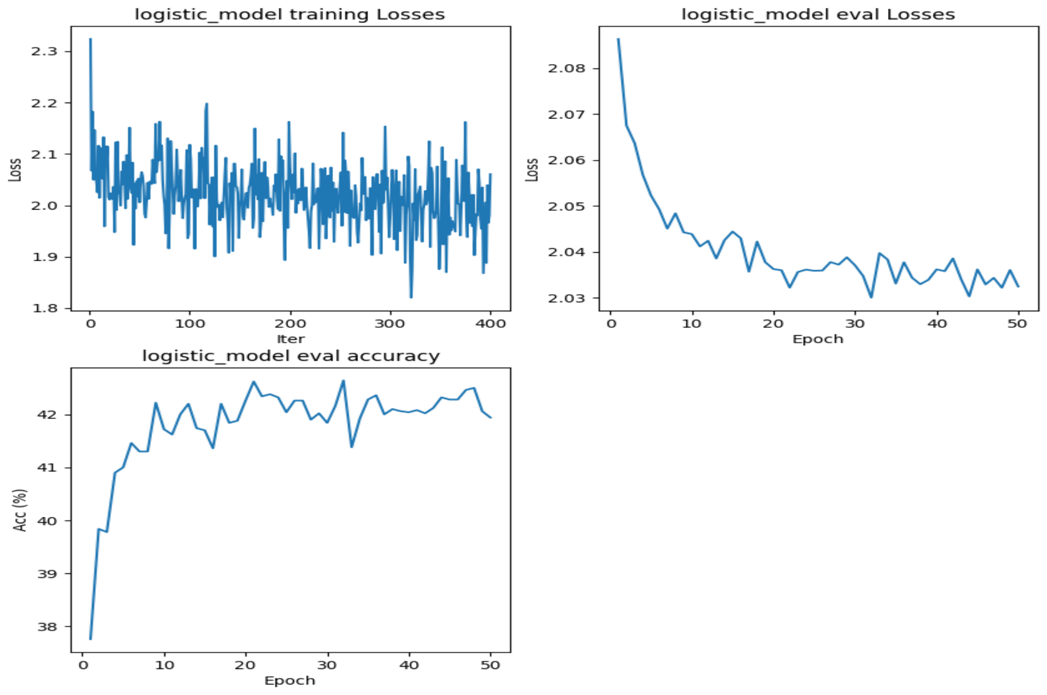


Figure 3.5, 3.6, and 3.7: Compare the FCN model and the logistic model

From the graphs above, we could see that the FCN model has a learning process that is quite similar to that of the CNN model, where the loss is gradually minimized with no big fluctuations. But for the logistic model, the variance of the loss during training is so big that it is so hard to tell if the training cross-entropy loss is actually minimizing or not. As you could also observe, there are several peaks in the evaluation loss graph and accuracy graph for the logistic model, which means that its entire training process is not stable. This is because the learning parameters for the logistic model are not enough and the hypothesis class is too simple that both the representative bias and variance of this model are higher than the FCN model. Also, the image features are too complex which requires a lot of parameters and non-linear activations to be fully learned. This resulted in the underfitting issue and a poor generalizability in the traditional logistic classifier.

Another possible issue is that the exponents inside the softmax function will cause the overflowing issue in the forward pass – the predicted class probabilities might be far much smaller for some categories which may cause float point errors and some weights in the hidden layer will become so large. Also, in the back propagation, the gradient is more likely to be exploded due to the exploding weights in the forward pass. In that case, the gradient descent process is more likely to overshoot and make the weight optimization even worse. Even though a smaller learning rate, gradient clipping, and regularization is useful to prevent this issue, to maximize the accuracy and maintain a fast convergence, we can't make the learning rate too small or lambda too large. In that case, the fluctuation still exists in loss during training.

Conclusion

The CNN model has clearly become the best model for image classification tasks, both in stability, high prediction accuracy, and better generalizability. By comparing different types of results collected from training, validation, and testing processes, we had a systematic picture of how well our CNN model performed among all three ML models. In fact, CNN models have gradually become the mainstream models that are widely used in various image classification tasks. The CNN model deployed in this task used a structure similar to VGG-16 structure. Beyond that, LeNet, Googlenet, and YOLO are more complicated CNN structures that have been designed and implemented for various types of computer vision projects. There is a bright future for both machine learning and neural networks. For CNN, it could be seen as a capstone for us to explore the full potential of ML algorithms in dealing with real-world tasks.