CIS680 Vision & Learning Assignment 2

Haoyuan Zhang (ID: 85174335)

October 21, 2017

Abstract

The homework is aimed to get our hands dirty with data pre-processing and augmentation. Total three parts and the corresponding reports show below, including figures, curves and analysis. All codes are compressed in another file (Use PyTorch).

1 Data Pre-processing and Augmentation

This section uses the required network structure and cifar10 data to test the influence of different image pre-processing operations. The below show the training loss and accuracy curve with respect to the epoch iteration times, as well as test accuracy.

1.1 Raw Images Data

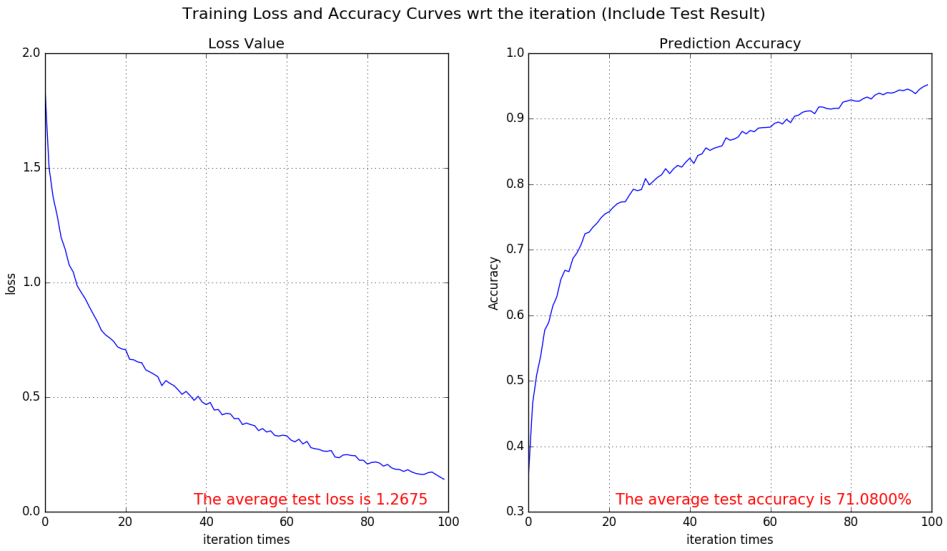

Figure 1: Training loss and accuracy curves (raw images)

1.2 Normalized Images Data

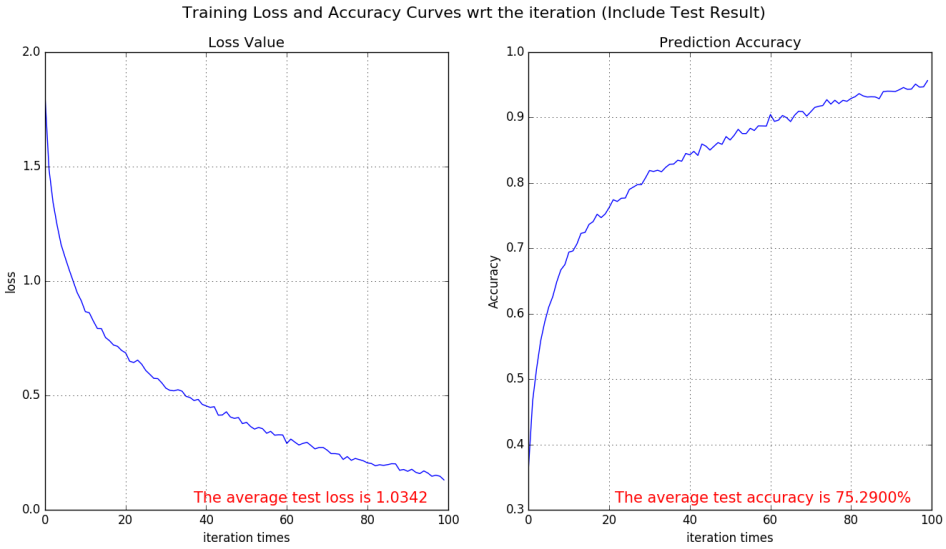

Figure 2: Training loss and accuracy curves (normalized images)

According to the result curve, the training process consists of more fluctuations so that needs more epoch iterations to converge. The test accuracy is higher than the model that trained by raw images data.

1.3 Normalized and Flipped Images Data

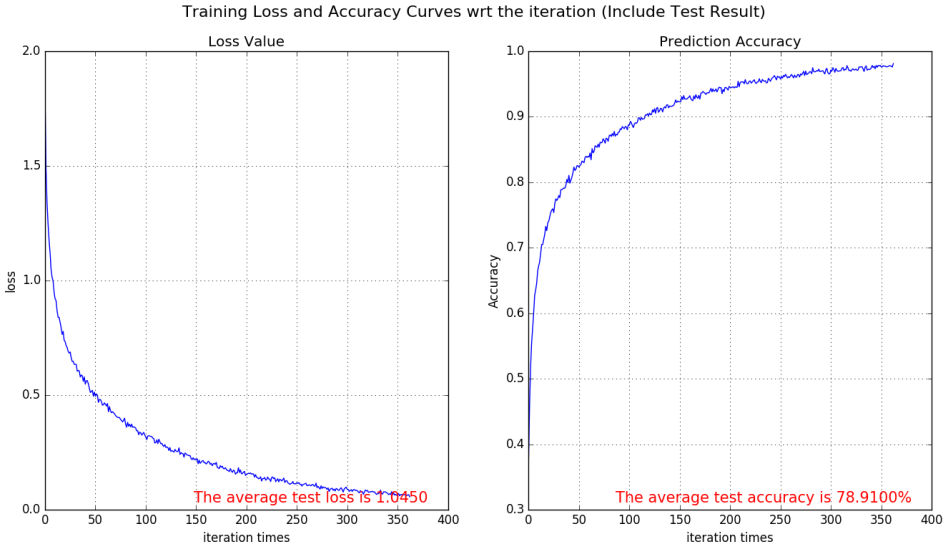

Figure 3: Training loss and accuracy curves (normalized and flipped images)

According to the result curve, the training process consists of even more fluctuations so that needs more epoch iterations to converge. The test accuracy is higher than the model that trained by normalized images data. The reason of those fluctuations is because the pre-processed data will somehow prevent the overfitting problems of the network.

1.4 Norm & Flipped & Shifted Images Data

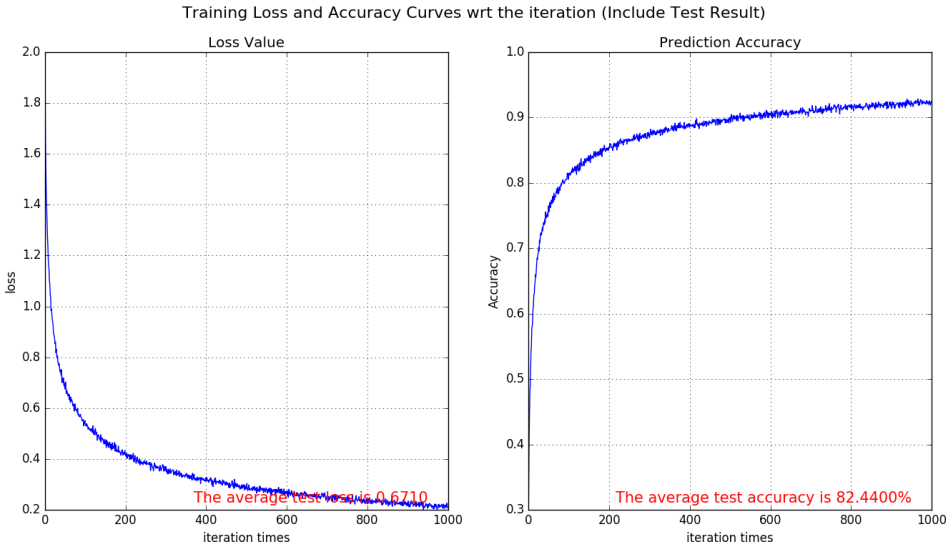

Figure 4: Training loss and accuracy curves (norm, flipped and shifted images)

With more complicated pre-processing data operations, the network is more difficult to train, that's why it has the most fluctuations compared with other three data types, such that needs more epoch times for converging. However, it performs the best to prevent overfitting problem, with the highest test accuracy.

2 CNNS and Vanishing Gradients on CIFAR-10

2.1 Own designed model

In this part, we design the network by ourself, the network should only be composed of convolutional layers with ReLU as activation function, pooling and softmax layers. Below is my designed network architecture.

Layers	Hyper-parameters
Convolution 1	Kernel size = (5, 5, 32), stride = (1, 1), padding = 2, followed by ReLU.
Pooling 1	Average operation. Kernel size = (2, 2), stride = (2, 2), padding = 0.
Convolution 2	Kernel size = (5, 5, 32), stride = (1, 1), padding = 2, followed by ReLU.
Pooling 2	Average operation. Kernel size = (2, 2), stride = (2, 2), padding = 0.
Convolution 3	Kernel size = (5, 5, 64), stride = (1, 1), padding = 2, followed by ReLU.
Pooling 3	Average operation. Kernel size = (2, 2), stride = (2, 2), padding = 0.
Convolution 4	Kernel size = (2, 2, 32), stride = (1, 1), padding = 0, followed by ReLU.
Convolution 5	Kernel size = (2, 2, 10), stride = (1, 1), padding = 0, followed by ReLU.
Pooling 4	Average operation. Kernel size = (2, 2), stride = (2, 2), padding = 0.
Softmax	Output channel = 10.

For the network training, the raw data is just normalized and all other training parameters are same as problem 1. The final training accuracy and test accuracy show below.

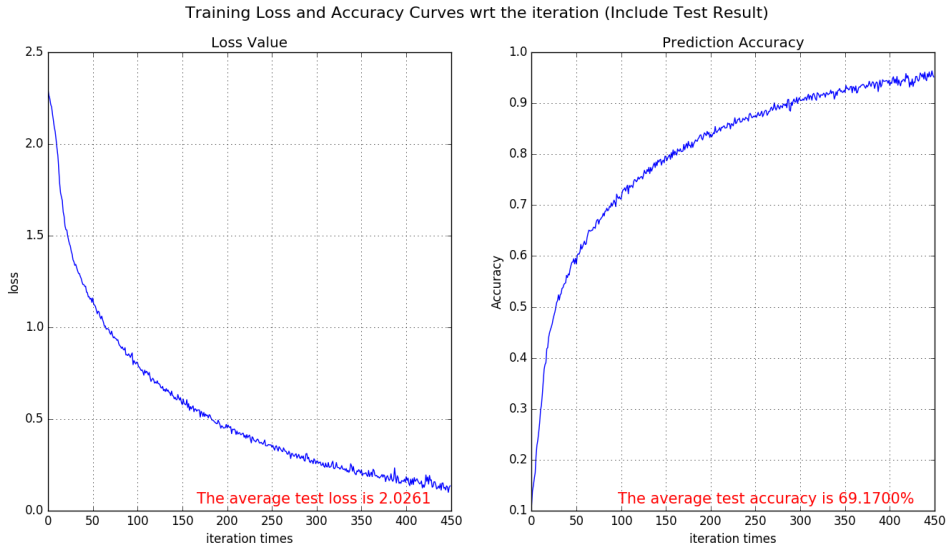

Figure 5: Training loss and accuracy curves (normalized images)

Compared to the network in problem 1 with the same data pre-processing operation (only normalization), after converting the fully connected layers into convolutional ones, there happen more fluctuations, and the final test accuracy decreases little bit to 69.17%, but still reasonable.

2.2 Gradient Magnitude in the First and Last Conv Layer

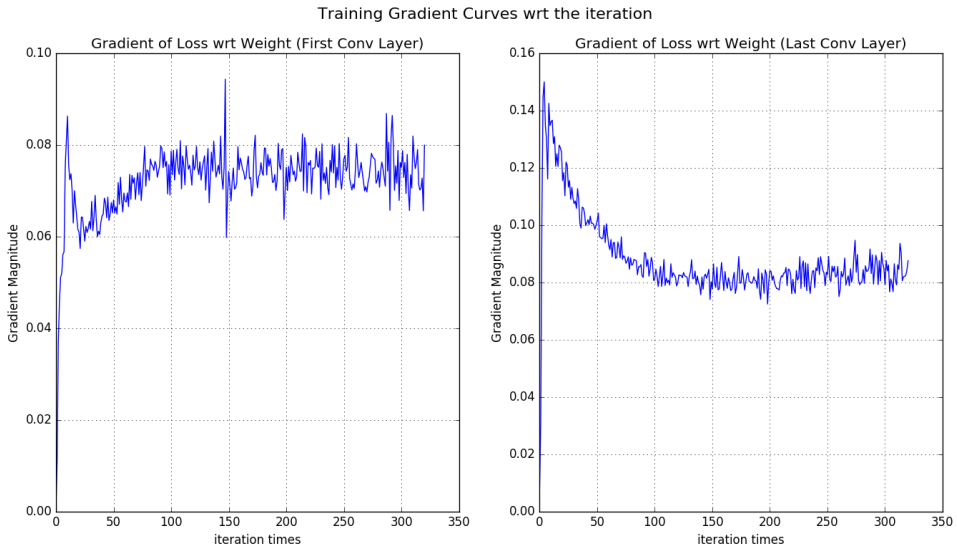

Figure 6: Loss Gradient wrt the Weight Parameter

At the start stage of training, the gradient in last layer is larger than the first one, since the gradient back-propagates from the end to start and needs to pass through multiple layers. With the training process going, the magnitude of gradients in both layers seem to converge to certain value which might represent the converging state of training.

2.3 Vanishing Gradient Problem

For this problem, we are supposed to add 20 more convolutional layers between the original first and second layers. Use the same data type, train the network again. Below show the training accuracy curve and gradient magnitude respectively.

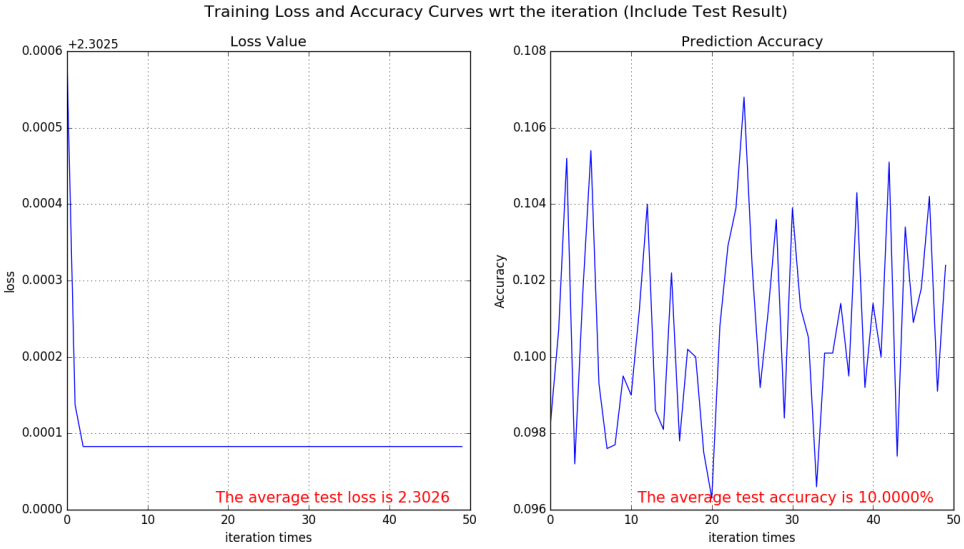

Figure 7: Training loss and accuracy curves (normalized images)

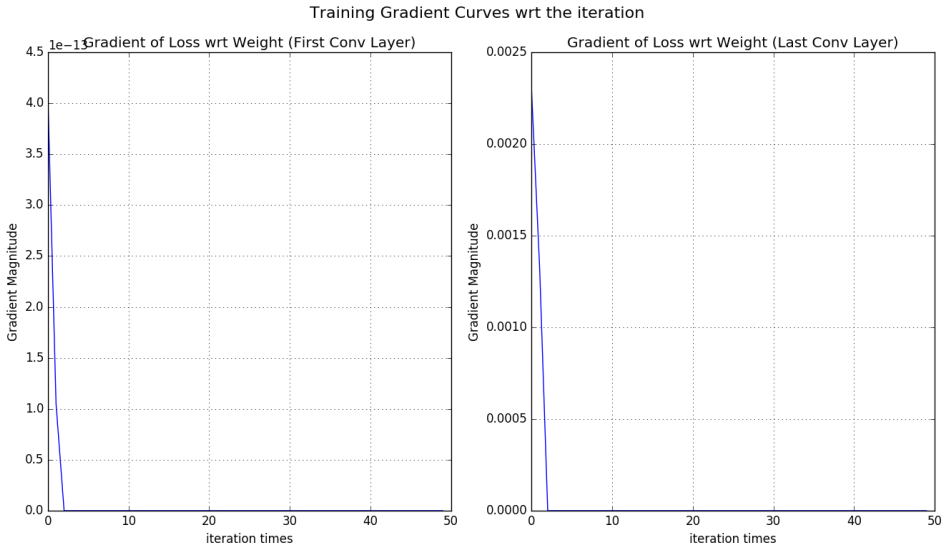

Figure 8: Loss Gradient wrt the Weight Parameter

After adding 20 more convolutional layers, there happens very distinguishable vanishing gradient problem, loss cannot decrease anymore after 5 or 6 epoch iterations, due to the gradient totally disappear both in the first and last layer, that is, the network is dead. Finally, the model just guess randomly for the test images.

2.4 Solve the Vanishing Gradient Problem

Theoretically, it's reasonable to use the DSN or batch normalization strategies to deal with the vanishing gradient problem. Due to the deeper layer in this case, I apply the ResNet structure which is also called as

the skip connections method, in order to re-activate the network. Below figure shows my modifications on the additional layers, all other layers keep the same as the problem 2.1.

```
def forward(self, x):
  # conv1 + relu + avg pool
  x = F.relu(self.conv1(x))
  x = self.avgPool(x)

  # additional conv nets
  identity1 = x
  x = F.relu(self.convAdd1(x))
  x += identity1

  identity2 = x
  x = F.relu(self.convAdd2(x))
  x += identity2

  identity3 = x
  x = F.relu(self.convAdd3(x))
  x += identity3

  identity4 = x
  x = F.relu(self.convAdd4(x))
  x += identity4

  identity5 = x
  x = F.relu(self.convAdd5(x))
  x += identity5

  identity6 = x
  x = F.relu(self.convAdd6(x))
  x += identity6
```

Figure 9: ResNet Structure

After modifications, the training process becomes reasonable again, with non-zero gradient values in both the first and last layers, the final result shows below.

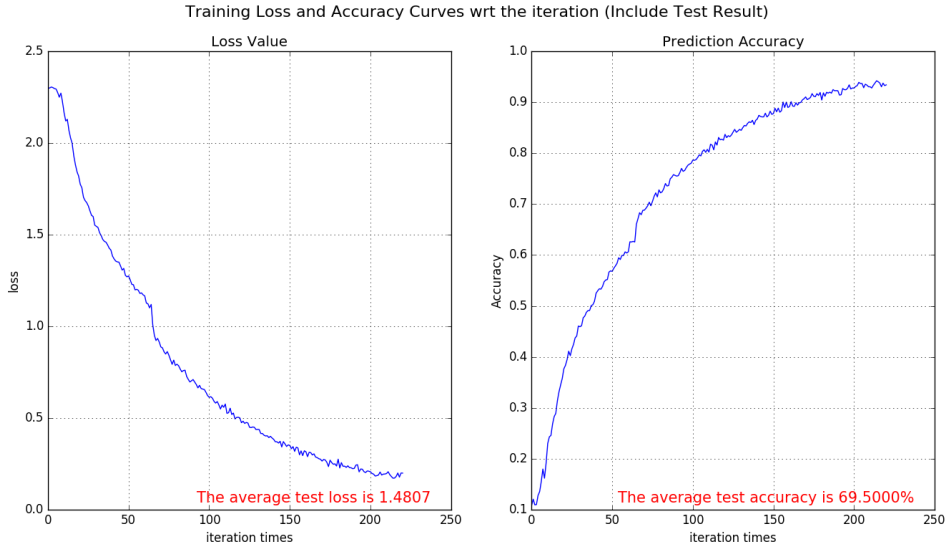

Figure 10: Training loss and accuracy curves (normalized images)

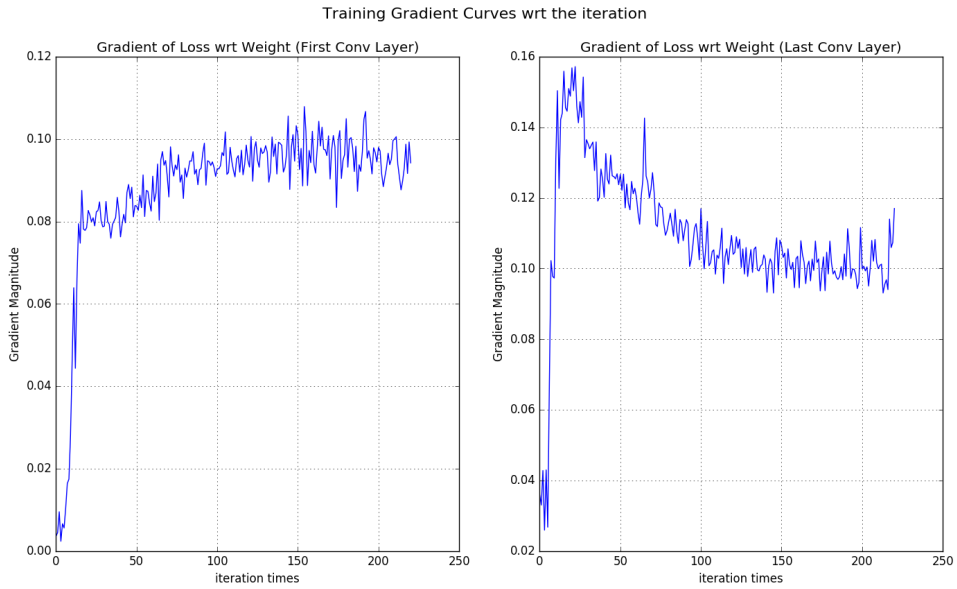

Figure 11: Loss Gradient wrt the Weight Parameter

The traditional idea is, much deeper the network, more powerful the network can achieve, however, there exists many many problems of deeper stack structure network and adding more layers that stacking one by one will just harm the network such as causing the vanishing gradient problem. In a word, deeper is not equal to the better, and not all systems and layers are easy to be optimized.

The idea of ResNet is pretty straightforward, that is, based on the original plain stacked layers, add additional residual layers to help learning and optimization. In terms of the architecture, for each input data x, it has two branches, one just feed forwards through the stacked layers to get next level feature map, meanwhile, the other branch just let input x reaches the output side of layers directly.

The intuition of ResNet is, the identity x can be set as a prior information for the next layer training. While in the traditional plain deep network, it can always achieve optimal model parameters at some certain number of layers, and then for the following layers, they contribute very little for the global optimization, sometimes they even do harm to the previous results. Now by adding the identity directly to layers output side, if x has been optimized well, the solver will simply drive the weights of the multiple nonlinear layers towards zero in order to approach the identity mappings. Therefore, those identities can make those additional layers redundant and useless for the whole structure optimization.

3 Adversarial Images

The data augmentation is a good way to prevent the overfitting problem of the network. What's more, in order to enable the network to learn instead of memorizing, introducing adversarial images to the dataset is a good approach, which can also make the network robust to the image perturbations.

3.1 Generate Adversarial Images

The algorithm just follow the pipeline directly, feed an image into the network, compute the loss with some random wrong ground truth, fix the network parameters and use the loss to update input image to perturb it, finally, once the prediction confidence of arbitrary class reaches above 99%, done. Below figures shows original image, the perturbations and the generated adversarial image.

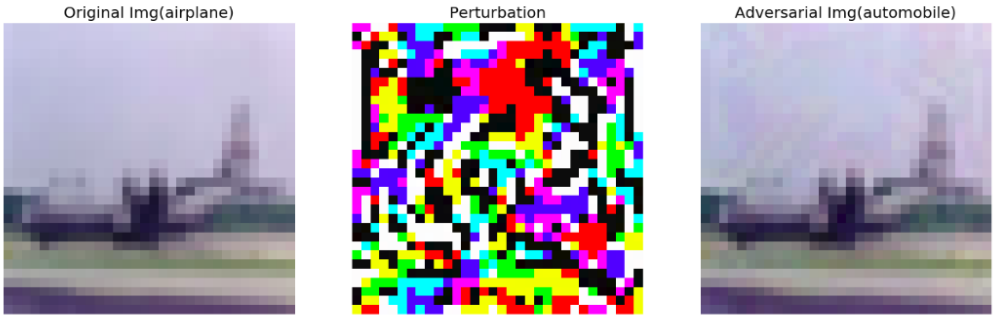

Figure 12: Airplane to Automobile 1

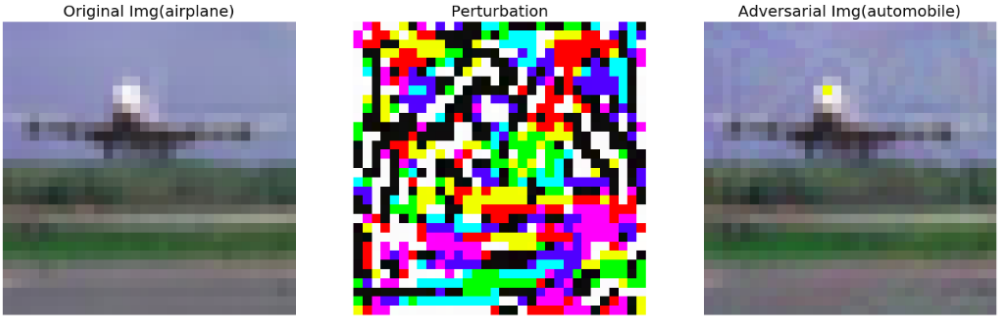

Figure 13: Airplane to Automobile 2

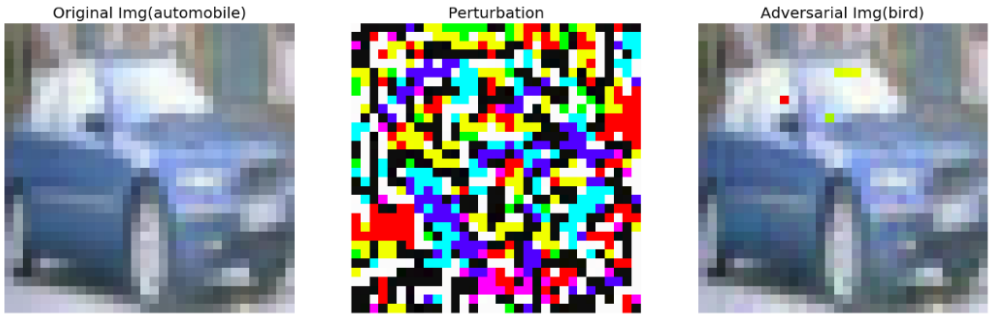

Figure 14: Automobile to Bird 1

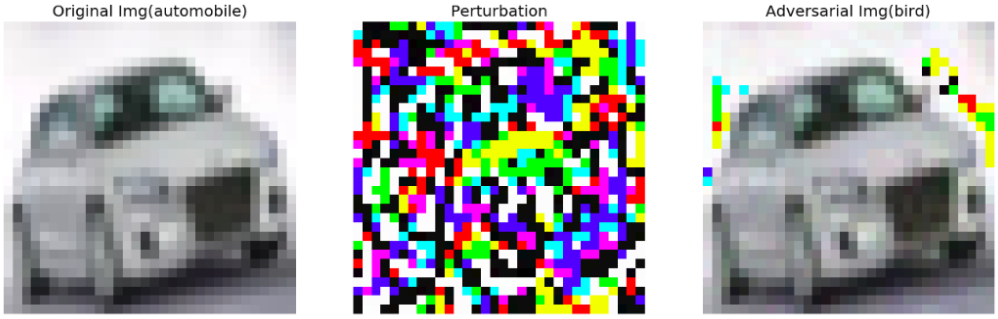

Figure 15: Automobile to Bird 2

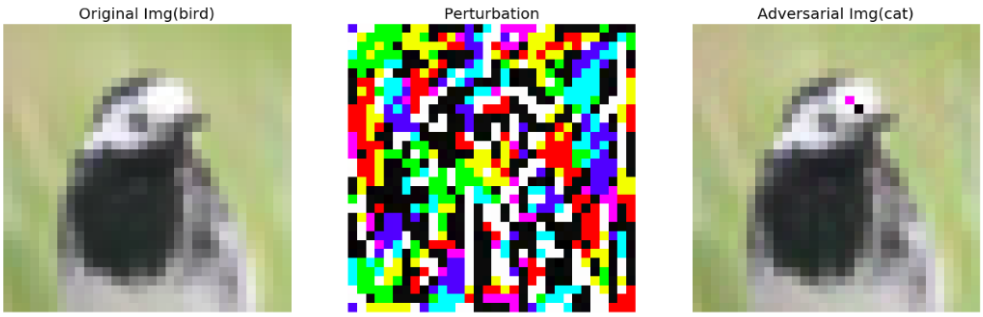

Figure 16: Bird to Cat 1

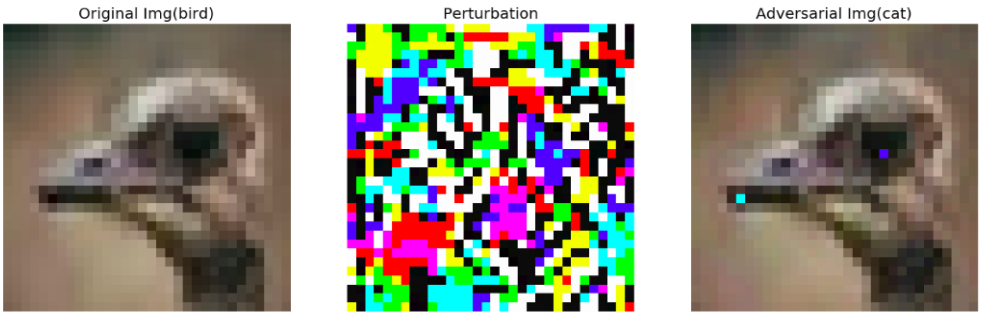

Figure 17: Bird to Cat 2

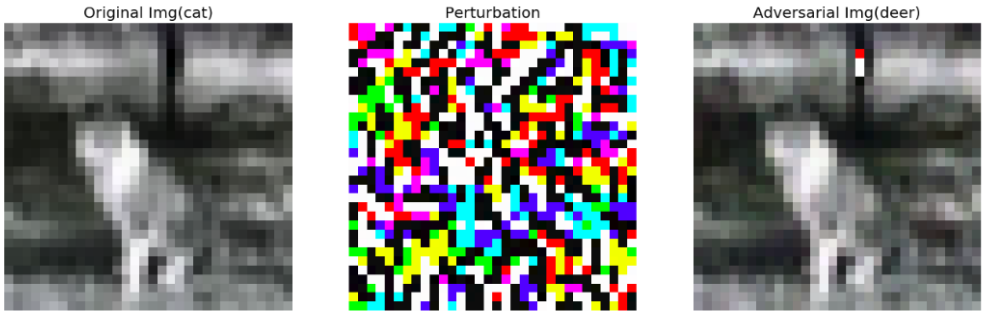

Figure 18: Cat to Deer 1

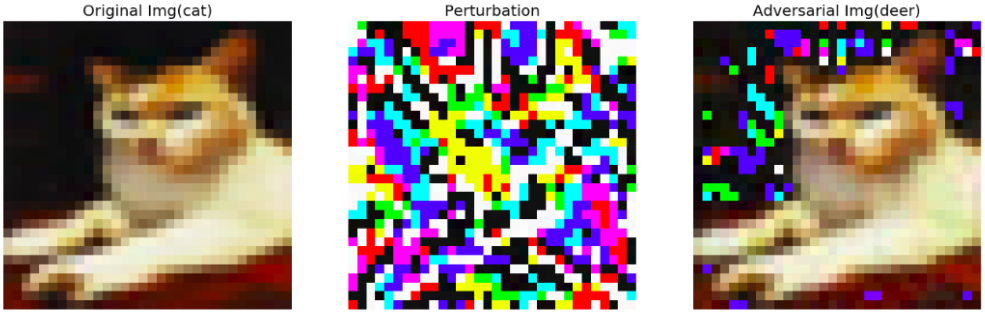

Figure 19: Cat to Deer 2

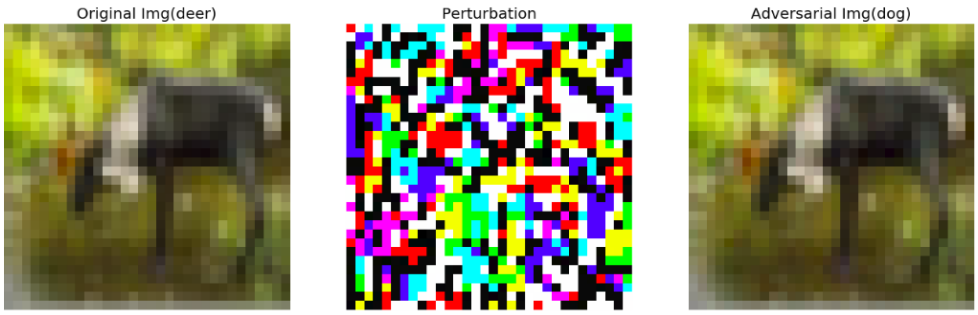

Figure 20: Deer to Dog 1

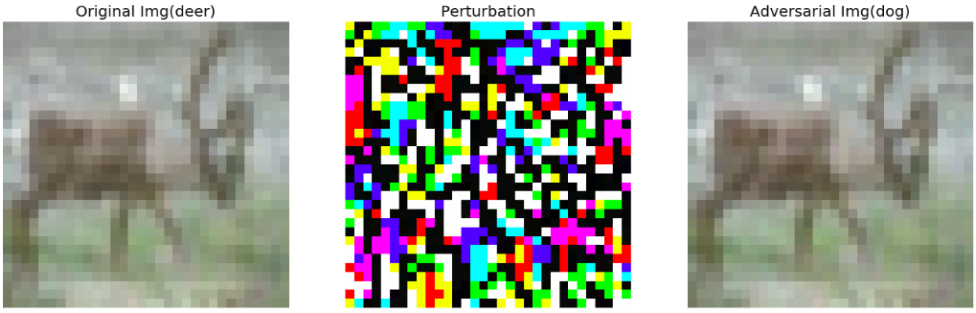

Figure 21: Deer to Dog 2

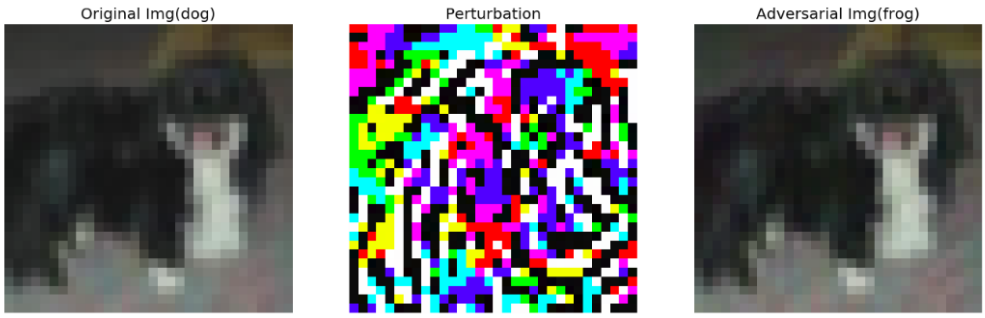

Figure 22: Dog to Frog 1

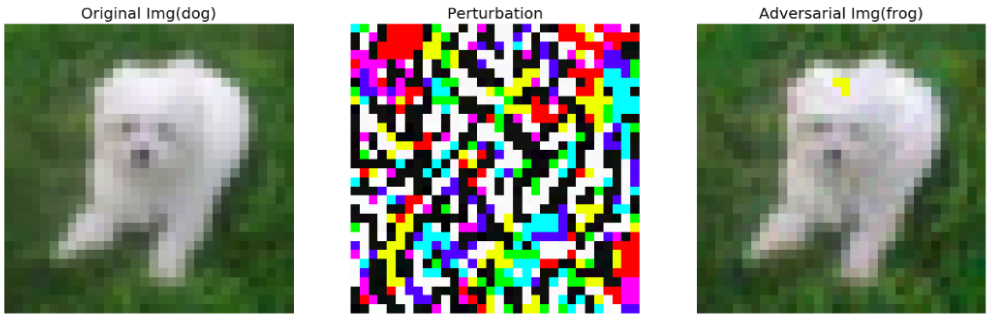

Figure 23: Dog to Frog 2

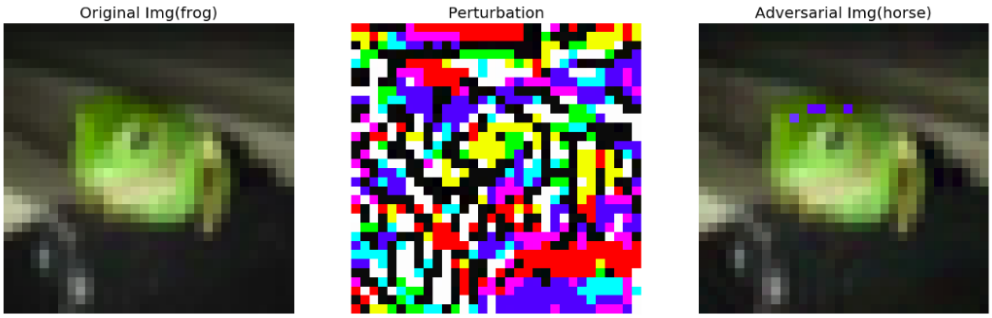

Figure 24: Frog to Horse 1

Figure 25: Frog to Horse 2

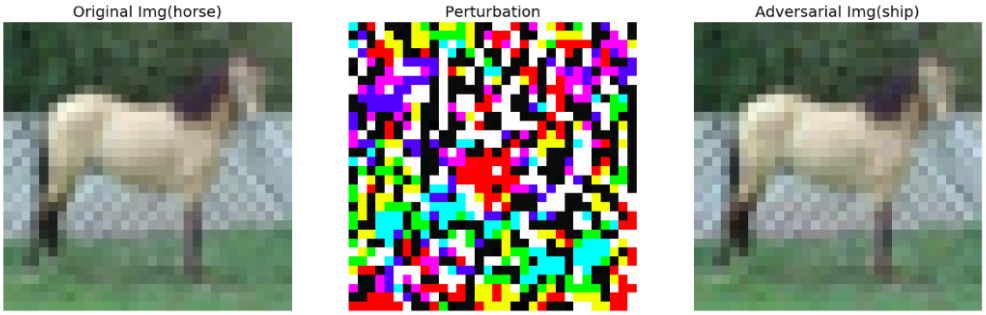

Figure 26: Horse to Ship 1

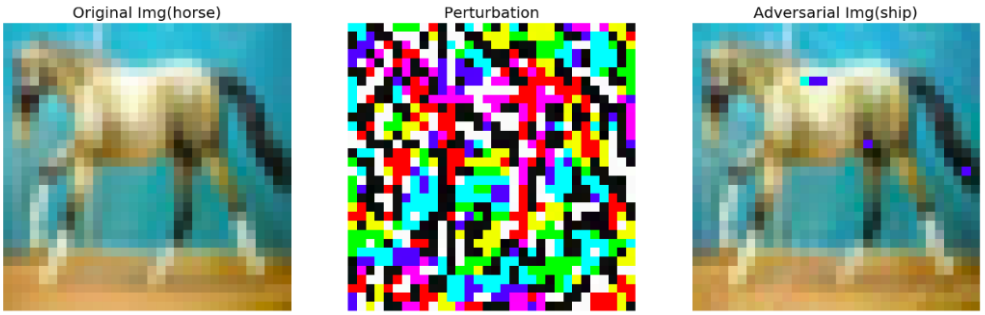

Figure 27: Horse to Ship 2

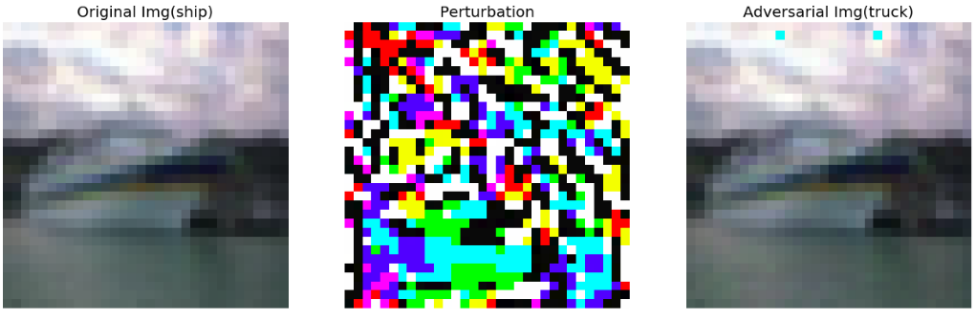

Figure 28: Ship to Truck 1

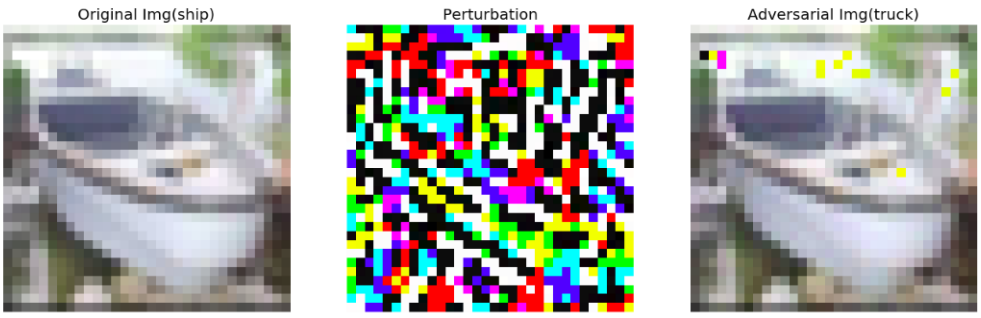

Figure 29: Ship to Truck 2

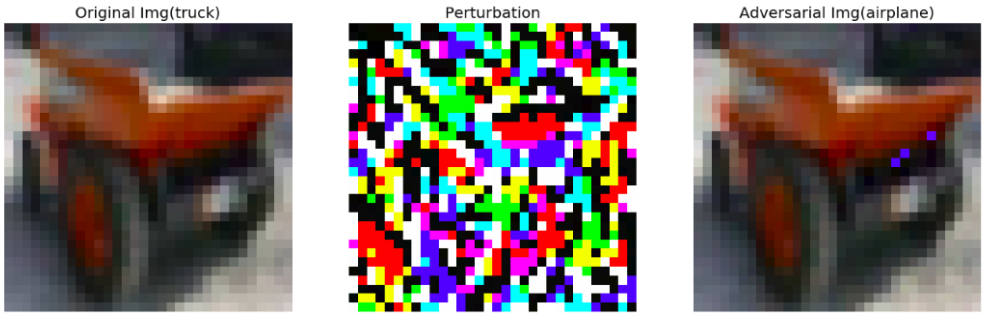

Figure 30: Truck to Airplane 1

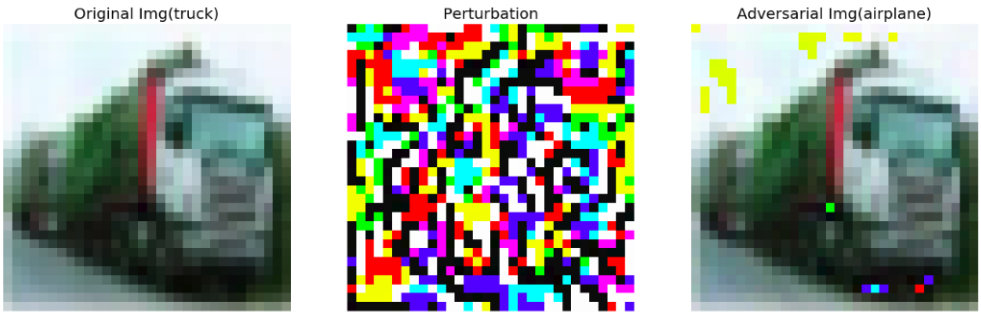

Figure 31: Truck to Airplane 2

According to the results above, almost all adversarial images are same as the original images according to human views, only one or two of them have some perturbations in background space. However, for the network, they are total different objects and even predict with the confidence as high as almost 100%. So that's why the network is easy to fool.

3.2 Train Model with Adversarial Images

Introduce those generated adversarial images into the training dataset but with correct label, so that trying to make the model robust to the perturbation influence. Below show the training accuracy, gradient curve and test accuracy(The input data is normalized only and use model trained in part 2).

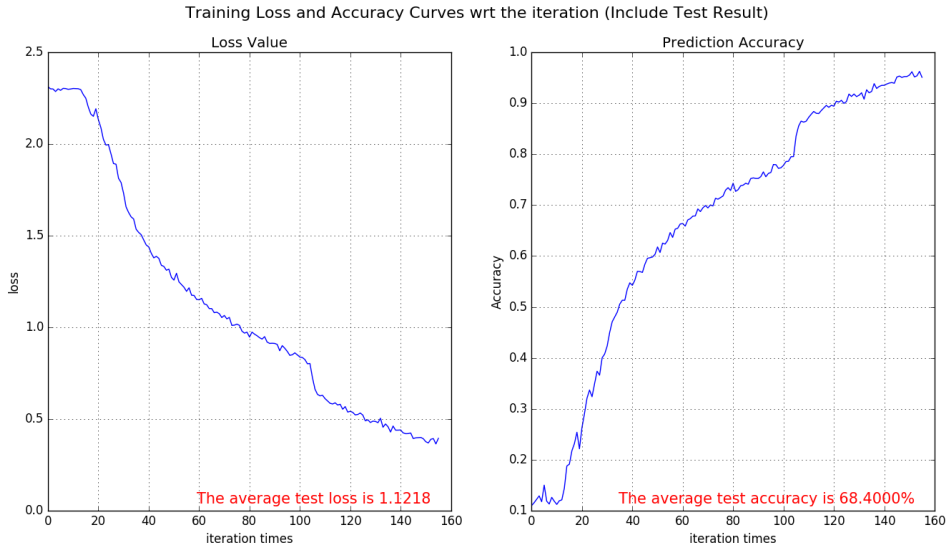

Figure 32: Training loss and accuracy curves (normalized images)

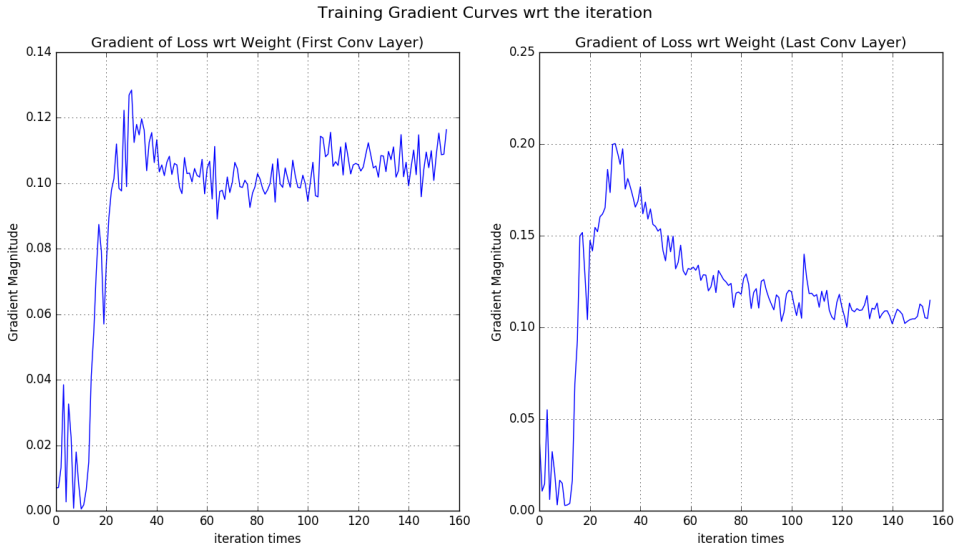

Figure 33: Loss Gradient wrt the Weight Parameter

The overall training process looks reasonable again, except some relative large fluctuations, the final test accuracy is 68.4%, which is lower than the result in part 2 with same data form and network model.

For the lower testing accuracy, the reason might be, in the training data we include those adversarial images that will modify the model such that trained model may become more general to those kind of perturbations. However, our test images are original ones without any perturbation, after being applied into this perturbation-robust model, the accuracy should be lower than previous one.