**CIFAR10 – Object Recognition in Images**

GR5242 Advanced Machine Learning

Final Project

Shiqi Duan (sd3072)

Chenyang Li (cl3505)

Ji Shen (js5006)

Jihan Wei (jw3447)

**Abstract**

In this project, we built networks to solve an image classification problem with 10 classes. Starting from literature review, we implemented several existing algorithms and make comparisons. Based on existing algorithms, we developed our own networks. We also included feature visualization tools for people to better understand how our Neural Network works.

1. **Dataset Description**

CIFAR10is an established computer-vision dataset used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing 10 object classes, with 6000 images per class. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here is some data information in this project:

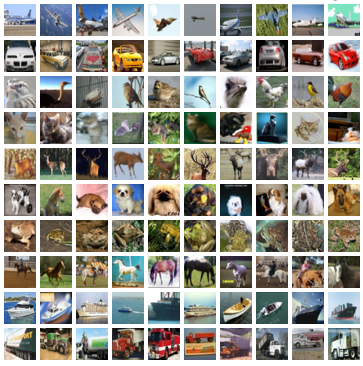
1. Training/Testing Data: There are 50000 training images and 10000 test images.

2. The Label Classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. (The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, and things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.)

3. Dimension of the Inputs: 32\*32\*3 (pixel height\*width\*number of color channels)

4. Dimension of the Outputs: 1\*10

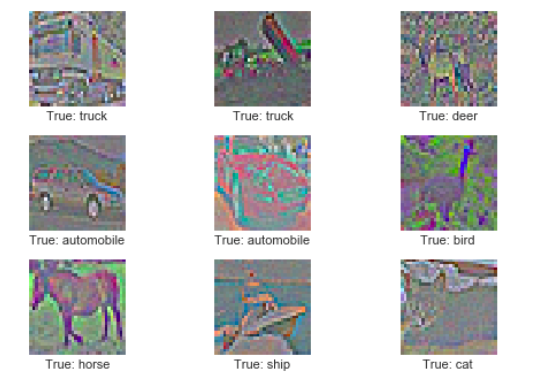
5. Sample Images:



1. **Preprocessing**
   1. **ZCA Whitening**

The term "ZCA" has been introduced in Bell and Sejnowski (1997), *Edges are the 'Independent Components' of Natural Scenes,* in the context of independent component analysis, and stands for "zero-phase component analysis". For image processing, when applied to a bunch of natural images (pixels as features, each image as a data point), ZCA transforms the data to reduce correlation between features. Unlike PCA reducing the dimensionality of data, ZCA results in whitened data that is as close as possible to the original data (in the least squares sense). Due to these properties of ZCA Whitening, we used it to preprocess our data to improve the efficiency of our algorithms.

Sample Images After ZCA Whitening:



* 1. **Data Augmentation**

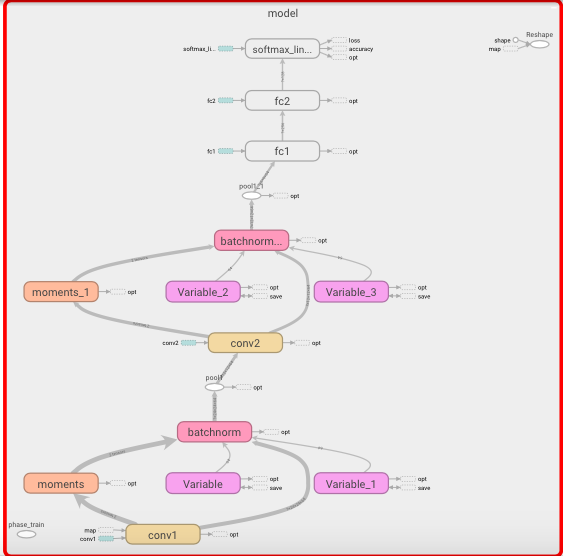
Besides ZCA Whitening, we also applied Data Augmentation on the image data. In this step, the pre-processing is different for training and testing of the neural network. For training, the input images are randomly cropped, randomly flipped horizontally, and the hue, contrast and saturation is adjusted with random values. This artificially inflates the size of the training set by creating random variations of the original input images. Examples of distorted images are shown further below. For testing, the input images are cropped around the center and nothing else is adjusted. After Data Augmentation, the dimension of data becomes 24x24x3.

Sample Images After Preprocessing:

1. **Methods**
   1. **CNN (Convolutional Neural Networks)**

As CNNs have fewer connections and parameters compared to other neural networks with similar layer sizes, they are easier to train and control. So we first built a traditional CNN algorithm to solve the problem. Our net contains five layers with weights: two Convolutional layers and three Fully Connected layers. Then the output of the last Fully Connected layer is fed to a 10-way Softmax which produces a distribution over the 10 class labels.

Here is the detailed framework of layers in our CNN in order: Convolutional Layer1, Batch Normalization Layer, Max-Pooling Layer, Convolutional Layer2, Batch Normalization Layer, Max-Pooling Layer, Fully Connected Layer1, Fully Connected Layer2, and Softmax Layer.



* + 1. **Convolutional (Conv) Layer**

The Conv layer’s parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. During the forward pass, we convolve each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Each filter produces a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

In our CNN, we used 64 5x5x3 filters padding 2 pixels broader in Conv layer 1 for an input with 24x24x3 dimensions. It gave us an output of 24x24 pixels with 64 channels. In Conv layer 2, we used 64 5x5x64 filters padding 2 pixels broader for an input with 12x12x64 dimensions and obtained an output with 12x12x64 dimensions.

* + 1. **Batch Normalization**

After each Conv layer, we did batch normalization, in which we performed normalization for each training mini-batch to achieve higher learning rates and recover the potential problems about layer initializations.

* + 1. **Max-Pooling Layer**

The Pooling layer makes the representations smaller and more manageable. It operates over each activation map independently. In our CNN, we max-pooled networks with 2x2 filters with a stride of 2. That is, we took the maximum number from each 2x2 window as our output. After each Max-Pooling layer, the height and width of pixels become half of before while the number of channels stays the same as before.

* + 1. **Fully Connected (FC) Layer**

Neurons in a Fully Connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. (In fact, the only difference between FC and Conv layers is that the neurons in the Conv layer are connected only to a local region in the input, and that many of the neurons in a Conv volume share parameters.)

In our FC layer1, the input is of 6x6x64 and the output is of a vector of length 256, which is the input of FC layer2. The output of FC layer2 is a vector of length 128.

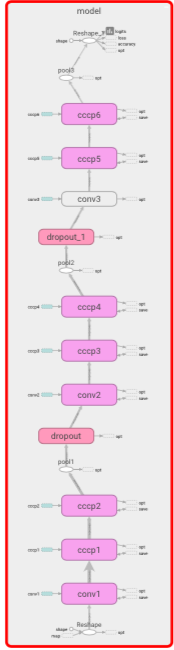
* + 1. **Softmax Layer**

The Softmax classifier is a generalization of the binary form of Logistic Regression. Softmax layer gives us probabilities for each class label. We identified an image into the class with the largest probability in the output of the Softmax layer.

* 1. **Network in Network (NIN)**

NIN is one of the most famous methods for solving CIFAR10 problem. It is a deep network structure to enhance model recognition ability. The overall structure of NIN is a stack of Multilayer Perceptron Convolution (MLPConv) layers, on top of which lie in the global average pooling and the objective cost layer. For the CIFAR10 classification task, we tuned the number of layers in both NIN and the micro network, as well as the hyper-parameters. We added some sub-sampling layers between MLPConv layers, just as CNN does. Inspired from our experiments in CNN, we also added Batch Normalization layers to accelerate the training process.

Here is the framework of layers in our NIN in order: MLPConv Layer1, Max-Pooling Layer, Dropout Layer1, MLPConv Layer2, Average-Pooling Layer1, Dropout Layer2, MLPConv Layer3, and Average-Pooling Layer2.



LEARNING DETAILS:

Optimizer: *Adam*

Batch Size: 256

Learning Rate: 1e-4

Initializer:

A zero-mean Gaussian distribution with standard deviation 5e-2 for weights

Constant zero for bias

* + 1. **MLPConv Layers**

In our NIN, each MLPConv layer consists of one Conv layer and two Cross Channel Parametric Pooling (Cccp) Layers. Cccp layers allow complex and learnable interactions of cross channel information. Each Cccp layer acts the same as a Conv layer with 1x1 kernel. We applied Batch Normalization in each sub-layer of all MLPConv layers. The multiple layers and non-linear activation in MLPConv layers distinguish data that is not linearly separable.

In MLPConv Layer1, its Conv layer deals with a 24x24x3 input using 96 5x5x3 filters padding 2 pixels broader and outputs with a 24x24x96 dimensions, following by two Cccp layers. One uses 80 1x1x96 filters and another uses 48 1x1x80 filters. Both of the two Cccp layers pad 2 pixels broader. The output after the two Cccp layers is of 24x24x48 dimensions.

In MLPConv Layer2, its Conv layer deals with a 12x12x48 input using 96 5x5x48 filters padding 2 pixels broader and outputs with a 12x12x96 dimensions, following by two Cccp layers. Both Cccp layers use 96 1x1x96 filters to produce outputs of 12x12x96 dimensions.

In MLPConv Layer3, its Conv layer deals with a 6x6x96 input using 96 3x3x96 filters padding 1 pixel broader and outputs with a 6x6x96 dimensions, following by two Cccp layers. One uses 96 1x1x96 filters and another uses 10 1x1x96 filters. The output after the two Cccp layers is of 6x6x10 dimensions.

* + 1. **Max-Pooling Layer**

We only used Max-Pooling layer after MLPConv Layer1, where we max-pooled networks with 3x3 filters padding 1 pixel to the right with a stride of 2. That is, we took the maximum number from each 3x3 window as our output. After the Max-Pooling layer, we obtained an output of 12x12x48 dimensions.

* + 1. **Average-Pooling (Global Average-Pooling) Layer**

We built Average-Pooling layers after MLPConv Layer2 and MLPConv Layer3. Average-Pooling layers replace the traditional FC layers in CNN, generating one feature map for each corresponding category in the last MLPConv layer. We took the average of each feature map and fed it into the Softmax Layer. For the reason of using Average-Pooling instead of Max-Pooling here is that Max-Pooling extracts only the most important features like edges but Average-Pooling takes all into account so that we can extract features in a smooth way, which can prevent the network from learning structures like edges and textures.

In Average-Pooling layer1, we average-pooled networks with 3x3 filters padding 1 pixel to the right with a stride of 2. That is, we took the mean from each 3x3 window as our output. After the Average-Pooling layer1, we obtained an output of 6x6x96 dimensions.

In Average-Pooling layer2, we average-pooled networks with 6x6 filters and with a stride of 1. That is, we took the mean from each 6x6 window as our output. After the Average-Pooling layer2, we obtained an output of 1x1x10 dimensions, which is the log probabilities of 10 classes.

* + 1. **Dropout Layer**

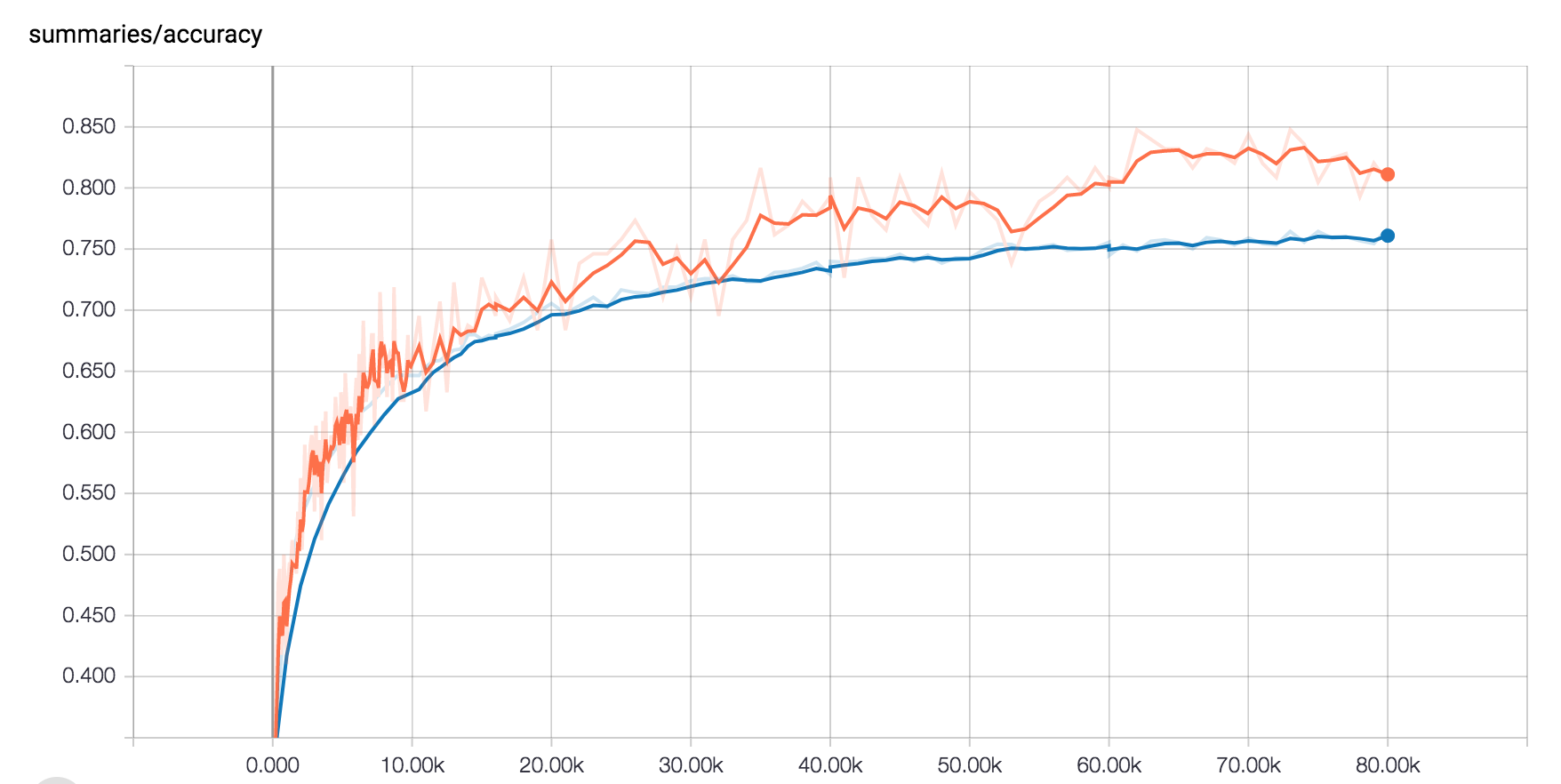
Dropout layer acts as a popular regularization method for reducing overfitting in neural networks. In most existing NINs, there is no Dropout layers as Average-Pooling can also help solve overfitting problems. In our NIN, we added two Dropout layers to see whether it can better prevent overfitting and improve models.

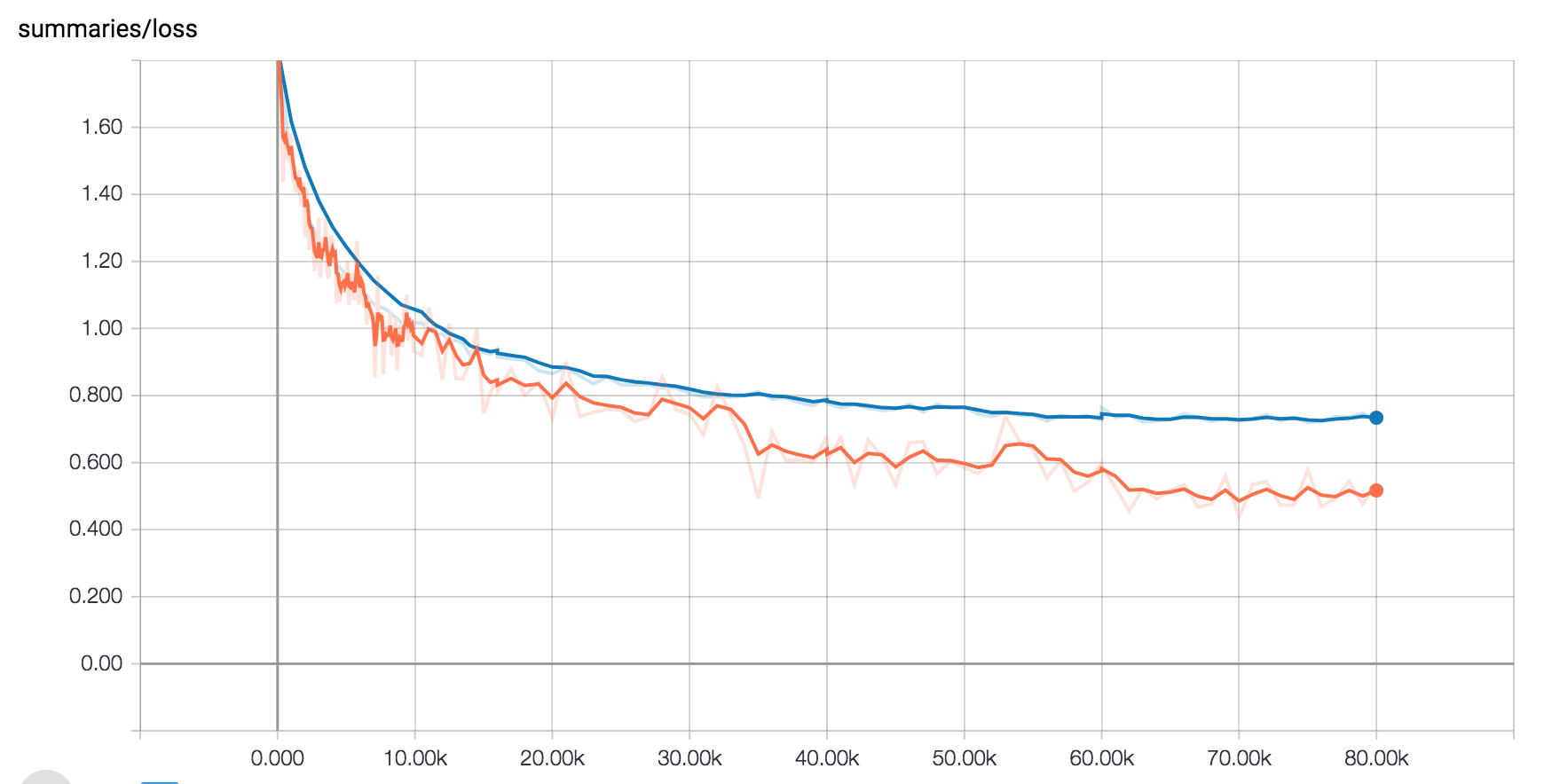
In the two Dropout layers, we used the same value for parameter *keep\_prob*, 0.5. That is, for non-zero features, our Dropout layers scale them up by 2 (=1/0.5).

1. **Results & Discussions**

To compare the performance of our CNN and NIN models, we took use of three evaluation methods: the softmax cross entropy loss, 0-1 accuracy, and confusion matrix.

Here are the results of our CNN:

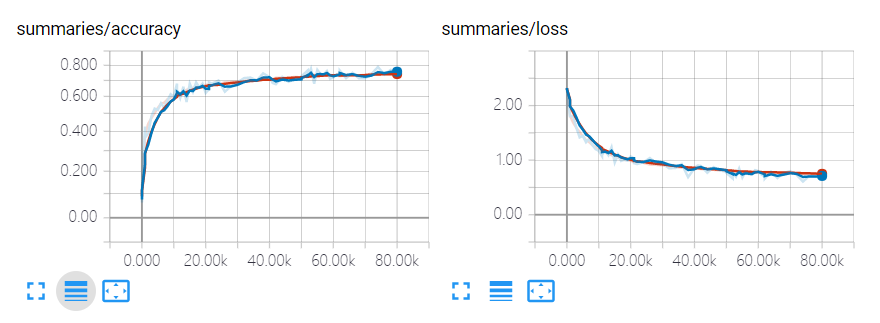




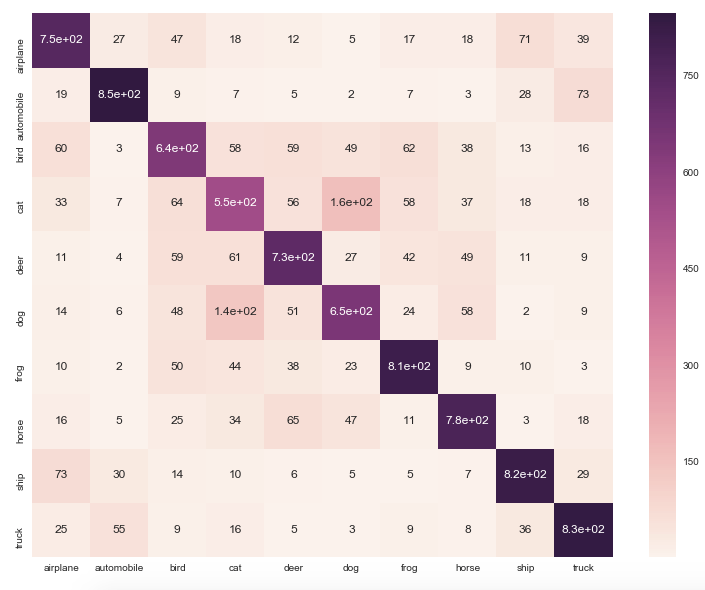
The accuracy is 0.7969 and the loss is 0.5528 on the training set, while the accuracy is 0.7668 and the loss is 0.7270 on the test set for CNN. It increases the accuracy and decreases the loss very quickly during the first 10,000 iterations and then improves the performance slower and smoother.



Here are the results of our NIN:



The accuracy is 0.7812 and the loss is 0.6100 on the training set, while the accuracy is 0.7422 and the loss is 0.7544 on the test set for NIN. It also converges quickly during the first 10,000 iterations and becomes slower then.

As we see from the results, our CNN model outperforms NIN in terms of accuracy and loss. In terms of the confusion matrices, both CNN and NIN have difficulty distinguishing images of cat and dog. CNN is better than NIN for recognizing images of airplane, automobile, bird, deer, frog, horse, and ship, while NIN is better than CNN for images of cat, dog and truck. In general, we can see that CNN misidentified fewer images than NIN.

Based on our results and discussions above, our NIN is good at distinguishing images of classes which are quite similar from many aspects, like cats and dogs, and automobile and trucks. However, our CNN is better than our NIN in general and should have generated results with higher accuracy after running more iterations. To achieve beeter results, further improvements should be added on the model as our CNN has not demonstrated a good performance on images of cat and dog yet but has almost converged.

1. **Further Improvements**

As our models have difficulty recognizing images of cats and dogs, to further improve the performance, we may need to build more networks for identifying images of these two classes.

What’s more, we learned from some papers that Global Contrast Normalization (GCN) is also a helpful preprocessing method and some people use it together with ZCA Whitening to preprocess images. If we have more time, we should try GCN or combine it with our preprocessing methods.

Last but not least, we should have obtained better results from both our CNN and NIN, if we made our networks larger and trained models longer.

**References:**

1. Hvass-Labs (2016). TensorFlow Tutorial #06 – CIFAR10. *https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/06\_CIFAR-10.ipynb*
2. Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning* (pp. 448-456).
3. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).
4. Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400.*
5. Li, F., Karpathy A., & Johnson J. (2016). Convolutional Neural Networks.[*http://cs231n.github.io/convolutional-networks/#norm*](http://cs231n.github.io/convolutional-networks/#norm)