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.

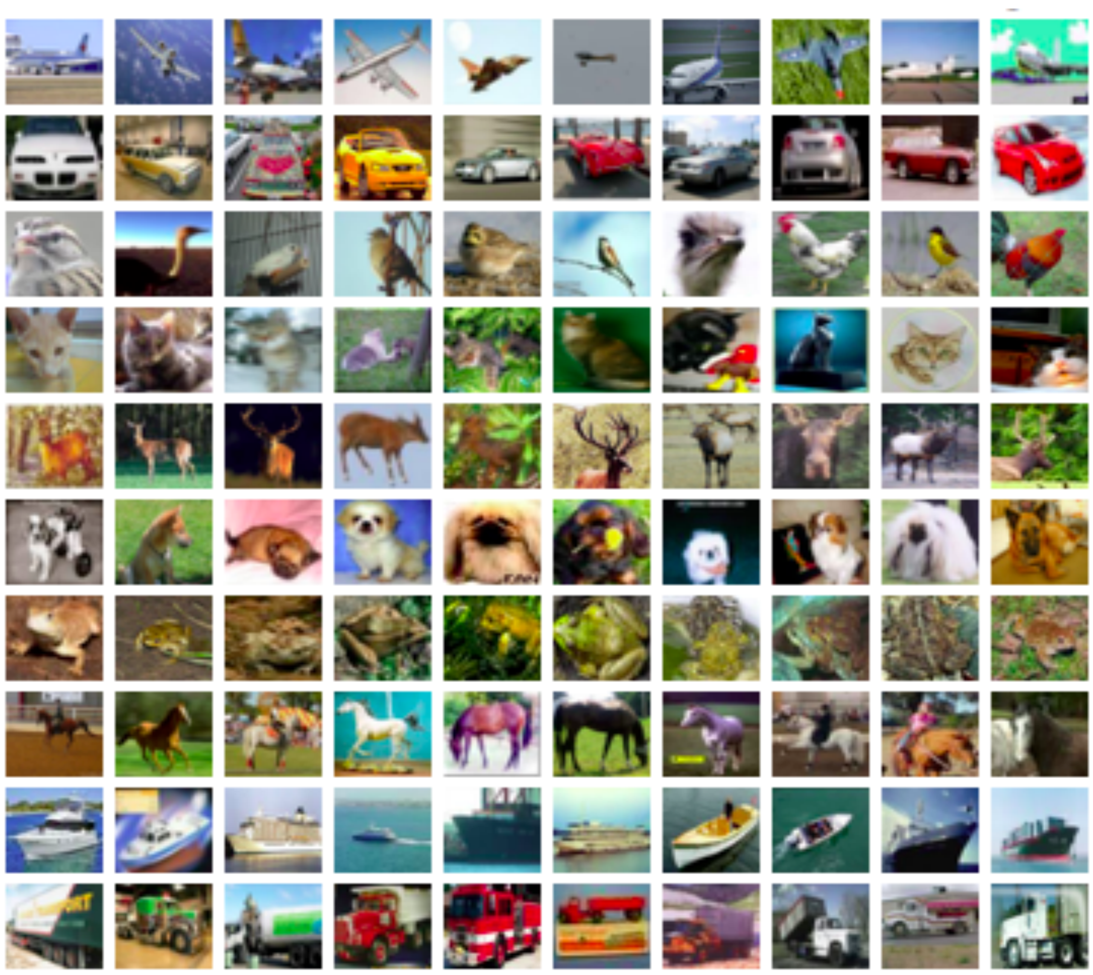
1. Dataset Description

CIFAR10 is an established computer-vision dataset used for object recognition. According to the official website, 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. The detailed information can be found in the link: *https://www.cs.toronto.edu/~kriz/cifar.html*.

Here is some data information in this project:

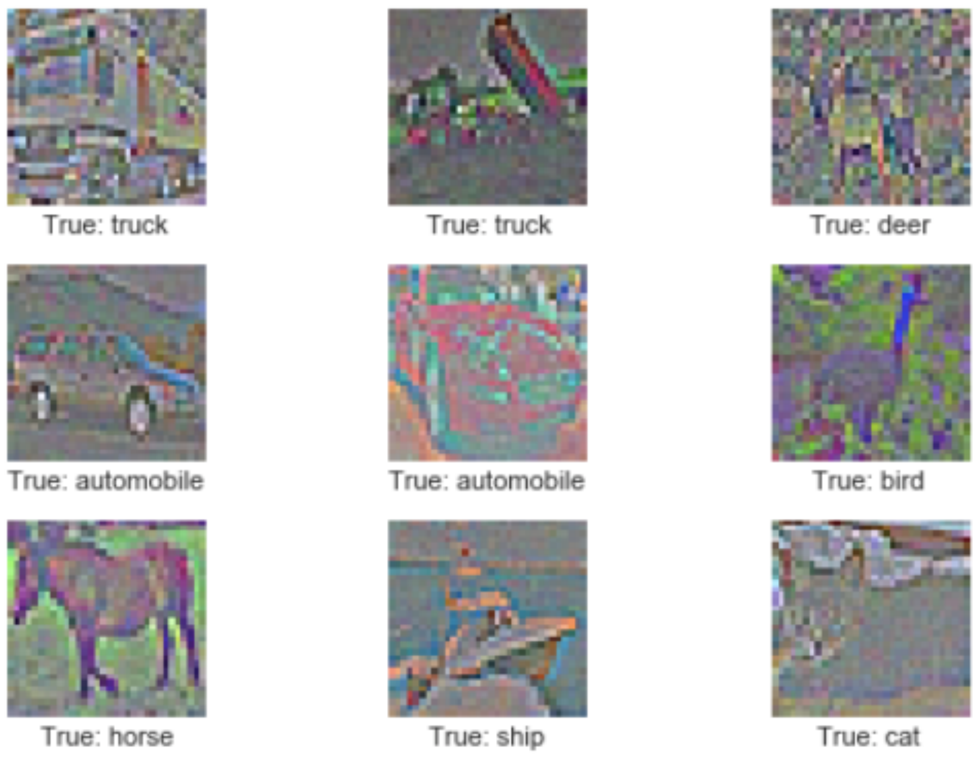
1. Training/Testing Data: There are 50000 training images and 10000 test images.
2. The Label Classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. (The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, 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:



2. Preprocessing

2.1 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:

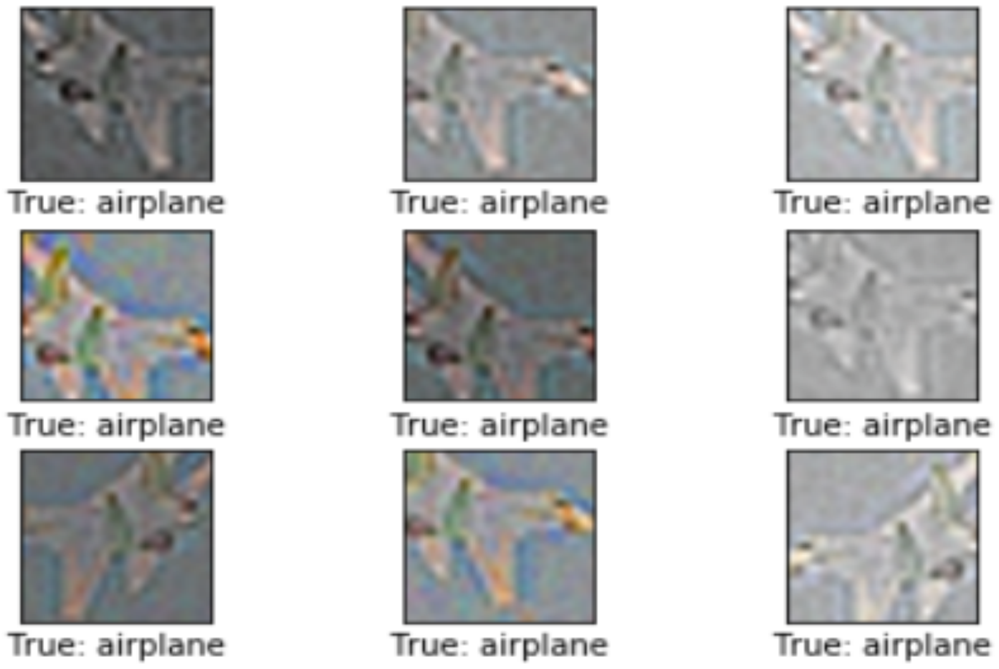


2.2 Data Augmentation

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 Image of a Dog after Data Augmentation (from Original):

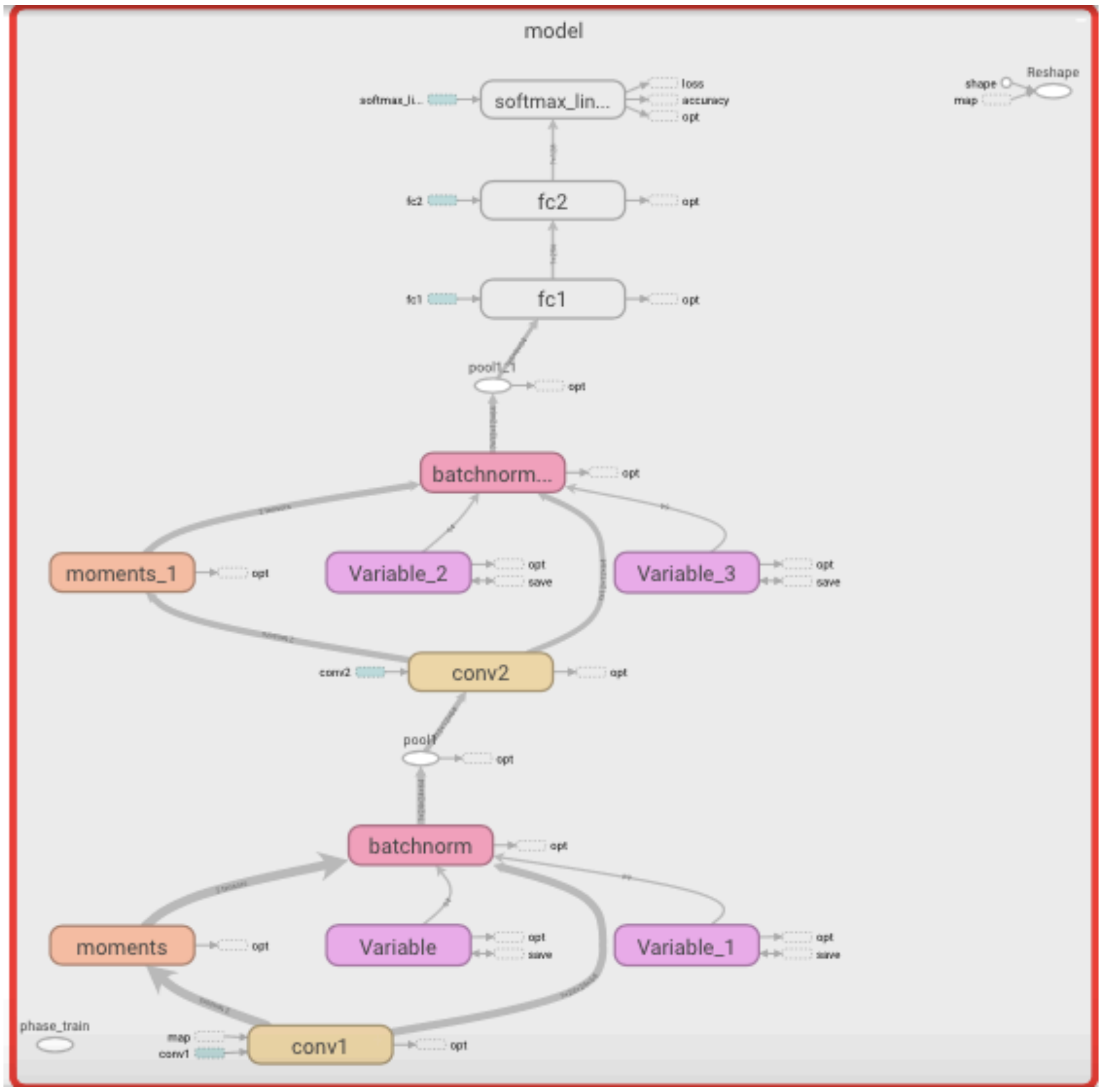
Sample Image of an Airplane after Preprocessing:



3. Methods

3.1 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.



3.1.1 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 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.

3.1.2 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.

3.1.3 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.

3.1.4 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.

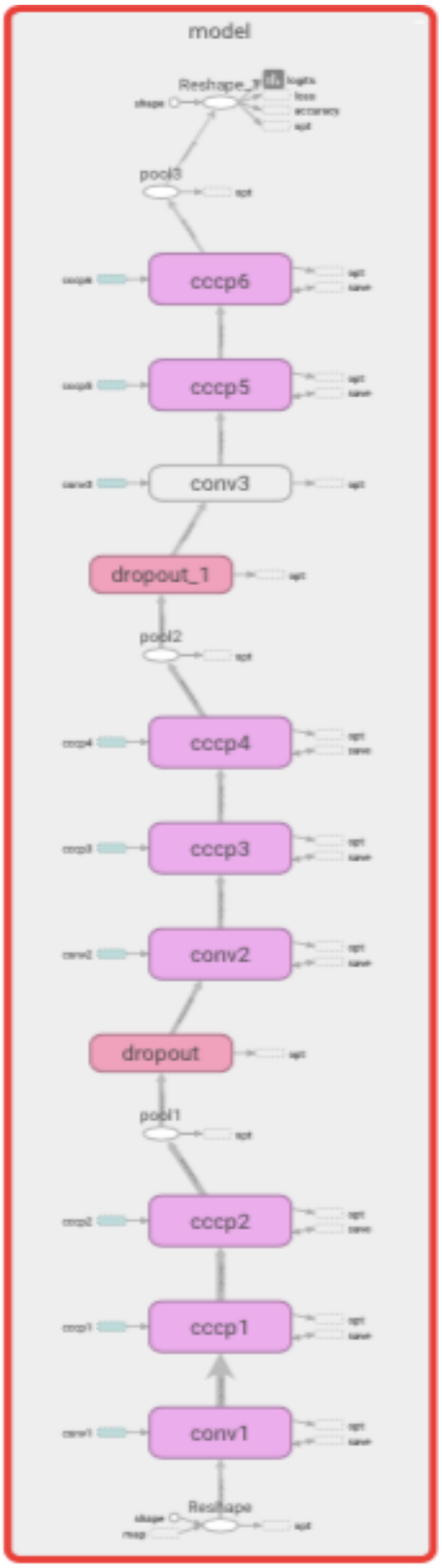
3.1.5 Softmax Layer

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.

3.2 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.



3.2.1 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.

3.2.2 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.

3.2.3 Average-Pooling (Global Average-Pooling) Layer

We built Average-Pooling layers after MLPConv Layer2 and MLPConv Layer3. The last Average-Pooling layer replaces 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 probability of 10 classes.

3.2.4 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).

4. Details of learning

We trained our models using Adam optimization with a batch size of 256 examples. We initialized the weights in each layer from a zero-mean Gaussian distribution with standard deviation 1e-5. We initialized the neuron biases with constant 0.0. The learning rate was initialized at 1e-4. We used an equal learning rate for all layers throughout the training.

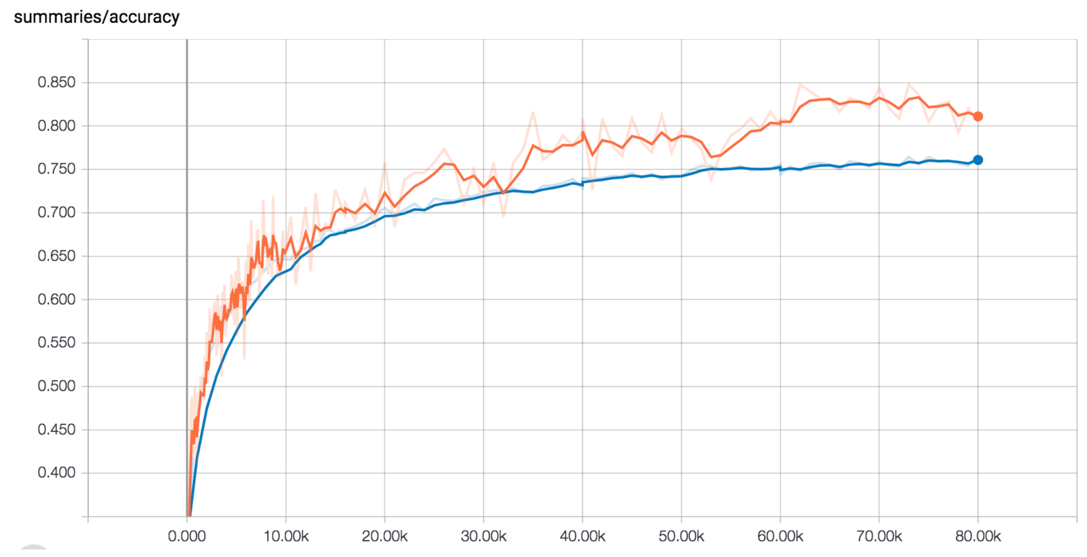
We trained the networks CNN and NIN for 80,000 iterations, respectively, which went through the training set of 50,000 images for around 200 cycles. Each of the networks took two to three days on our PCs.

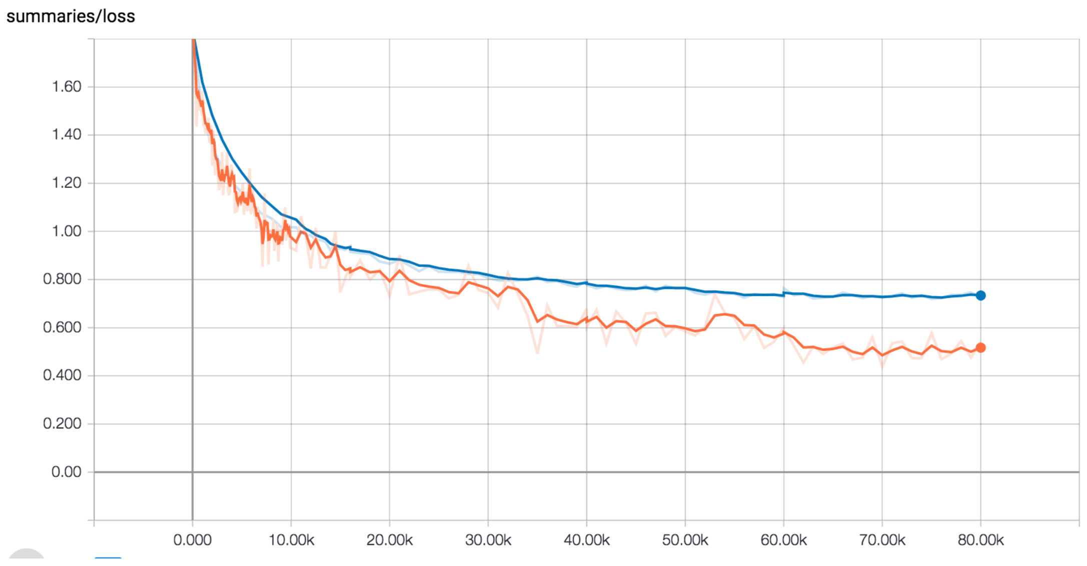
5. Results & Discussions

5.1 Accuracy & Loss

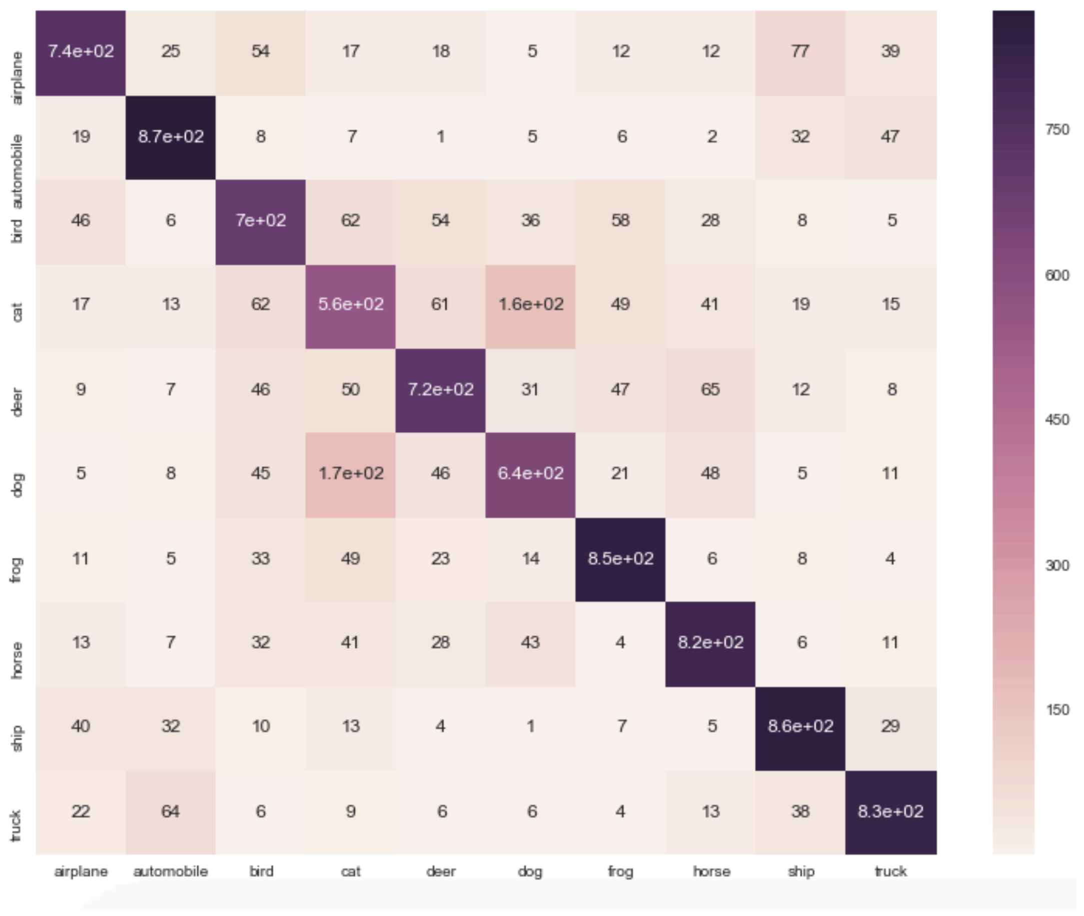
To compare the performance of our CNN and NIN models, we took use of three evaluation methods: the softmax cross entropy loss, 0-1 accuracy, and the confusion matrix.

Here are the results of our CNN:



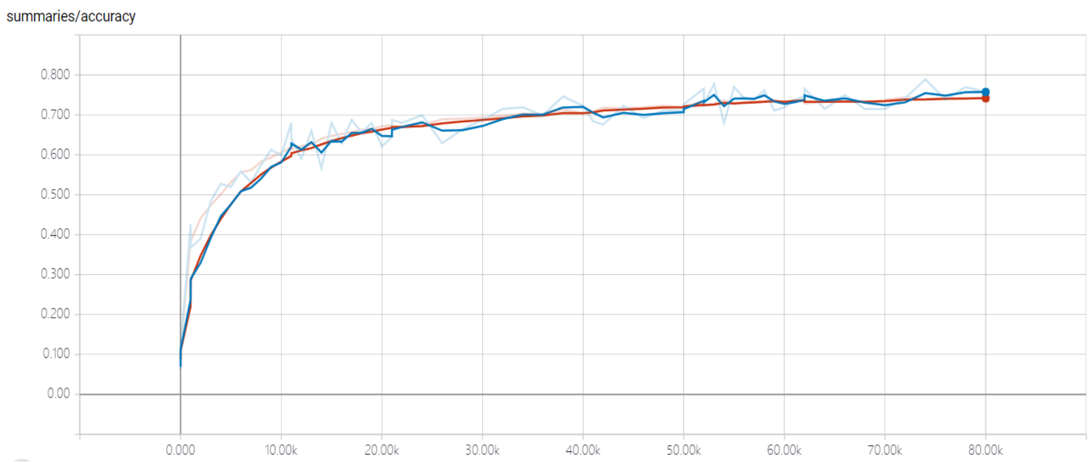


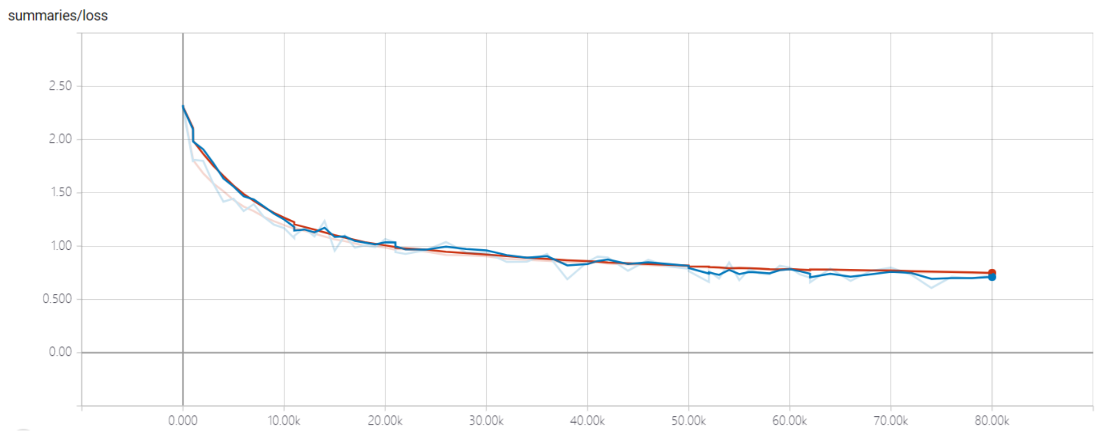
The accuracy is 0.8438 and the loss is 0.4995 on the training set, while the accuracy is 0.7578 and the loss is 0.7224 on the test set for CNN. It increases the accuracy and decreases the loss very quickly during the first 10,000 iterations and then improves the performance slower and smoother.



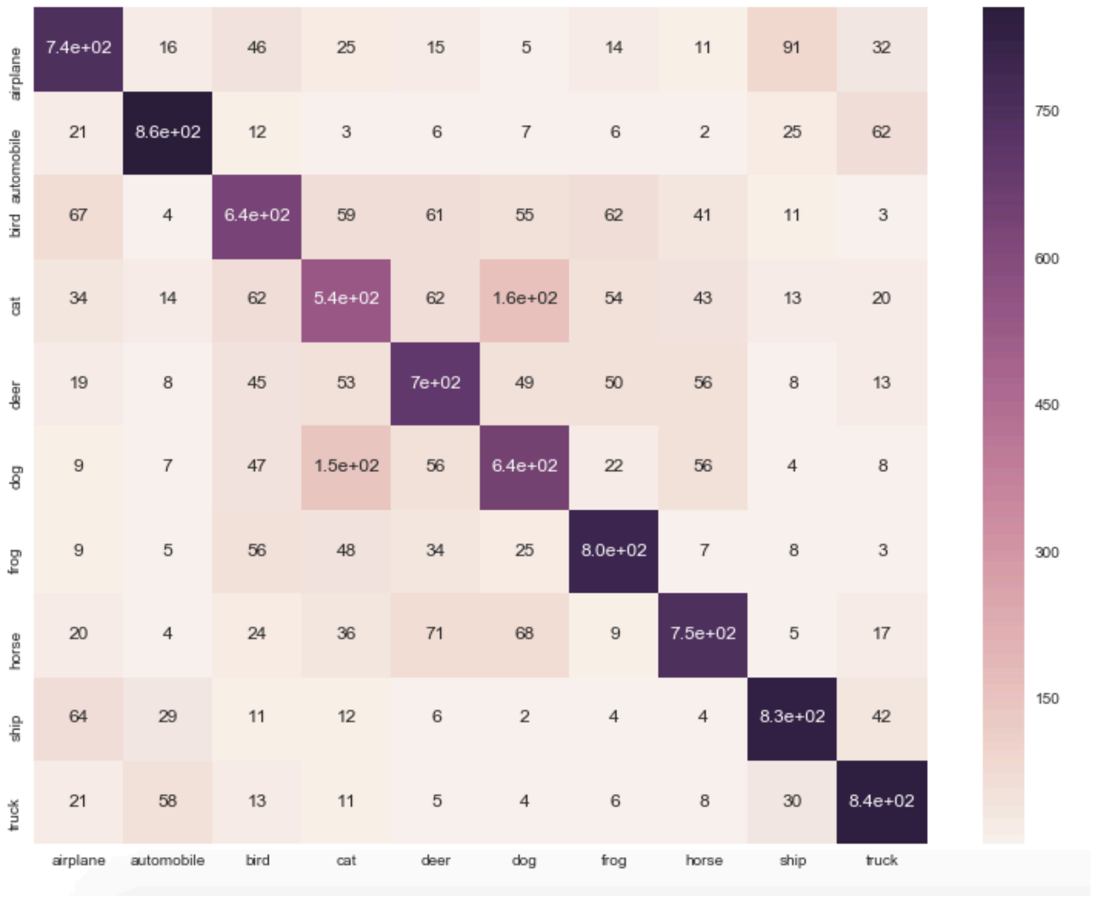
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	7.4e+02	25	54	17	18	5	12	12	77	39
automobile	19	8.7e+02	8	7	1	5	6	2	32	47
bird	46	6	7e+02	62	54	36	58	28	8	5
cat	17	13	62	5.6e+02	61	1.6e+02	49	41	19	15
deer	9	7	46	50	7.2e+02	31	47	65	12	8
dog	5	8	45	1.7e+02	46	6.4e+02	21	48	5	11
frog	11	5	33	49	23	14	8.5e+02	6	8	4
horse	13	7	32	41	28	43	4	8.2e+02	6	11
ship	40	32	10	13	4	1	7	5	8.6e+02	29
truck	22	64	6	9	6	6	4	13	38	8.3e+02

Here are the results of our NIN:





The accuracy is 0.7617 and the loss is 0.7089 on the training set, while the accuracy is 0.7433 and the loss is 0.7520 on the test set for NIN. It also converges quickly during the first 10,000 iterations and becomes slower then.



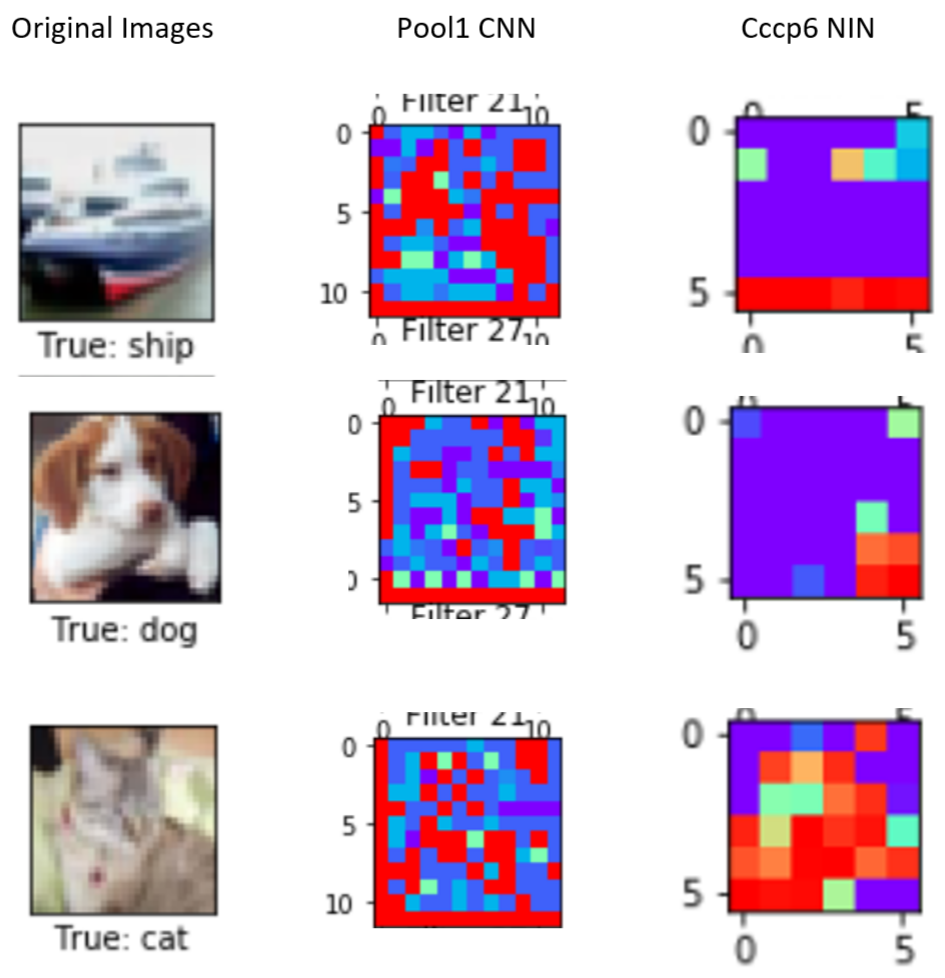
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	7.4e+02	16	46	25	15	5	14	11	91	32
automobile	21	8.6e+02	12	3	6	7	6	2	25	62
bird	67	4	6.4e+02	59	61	55	62	41	11	3
cat	34	14	62	5.4e+02	62	1.6e+02	54	43	13	20
deer	19	8	45	53	7e+02	49	50	56	8	13
dog	9	7	47	1.5e+02	56	6.4e+02	22	56	4	8
frog	9	5	56	48	34	25	8.0e+02	7	8	3
horse	20	4	24	36	71	68	9	7.5e+02	5	17
ship	64	29	11	12	6	2	4	4	8.3e+02	42
truck	21	58	13	11	5	4	6	8	30	8.4e+02

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. Based on analysis of

confusion matrices in our many trials, CNN is better than NIN for recognizing images of automobile, bird, deer, frog, horse, and ship, while NIN is better than CNN for images of airplane, cat, dog and truck. In general, we can see that CNN misidentified fewer images than NIN.
One point worth mentioning in NIN is that it to some degree fixes CNN's problem of misclassifying truck as automobile. MLPConv layers model the local patches better, which might explain for the improvement. What's more, Global Average-Pooling acts as a structural regularizer that prevents overfitting globally. We can obviously see this from the figures above.

5.2. Visualization

In this part, we will show some of the intermediate layers of our CNN and NIN to further understand the networks. The followings are the original images, together with their pool1 layer (Filter NO. 21) from our CNN and pool1 layer (Filter NO. 91) from our NIN.



From CNN, we can roughly see their profiles and shapes in the pool1 layer. Also, we can find that the intermediate layers for the cat and the dog look more similar, which explains why our model sometimes misclassifies cat as dog. As for NIN, even though the shapes are not clear, the cccp6 layer for cat and dog are alike while the cccp6 for ship looks different.

6. Further Improvements

Both our networks show difficulty in distinguishing images of dogs from those of cats. To address this problem, we proposed several possible solutions. 1) Add additional images of dogs and cats to training samples. 2) Train another deep convolutional network to classify dogs and cats. Feed images with prediction cat/dog from our original network into this new network. Predict the category using a combination of the two networks.

To simplify our experiments, we did not use any unsupervised pre-training or weight decay during our training. However, our codes can be easily adjusted to add weight decay and we expect it will help.

We used two Convolutional layers and three Fully Connected layers in our CNN, and three MLP Convolutional layers, each with two Cross Channel Parametric Pooling layers in our NIN. The numbers of layers in both CNN and NIN are flexible and we believe that a larger and deeper network should provide more helpful information. Besides, if we train the networks longer, our results can be further improved.

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*