5242 Final Project-CIFAR10

Wei, Xiaoya (Section1, xw2513)
Dong, Jiaqi (Section1, jd3418)
Yu, Wenting (Section2, wy2294)

December 2018

1 Project Description

The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class. Due to training efficiency, we construct our model under keras.
The main goals of this project are to:

- Learn to preprocess image data.
- Implement the best deep learning classifier we can create on this data
- Compare different implementations and better understand their architectures.

Github Link: https://github.com/wxya2017/GR5242-CIFAR10/tree/master

2 Data Preprocessing

We do the following steps to get the data prepared for training:

- Reshape each row into a (32,32,3) numpy array, with one inner array as one pixel with three channels: red, green and blue. The shape of the reshaped training data is (50,000, 32, 32, 3). The shape of the reshaped test data is (10,000, 32, 32, 3).
  Here are the first 16 images in test set.

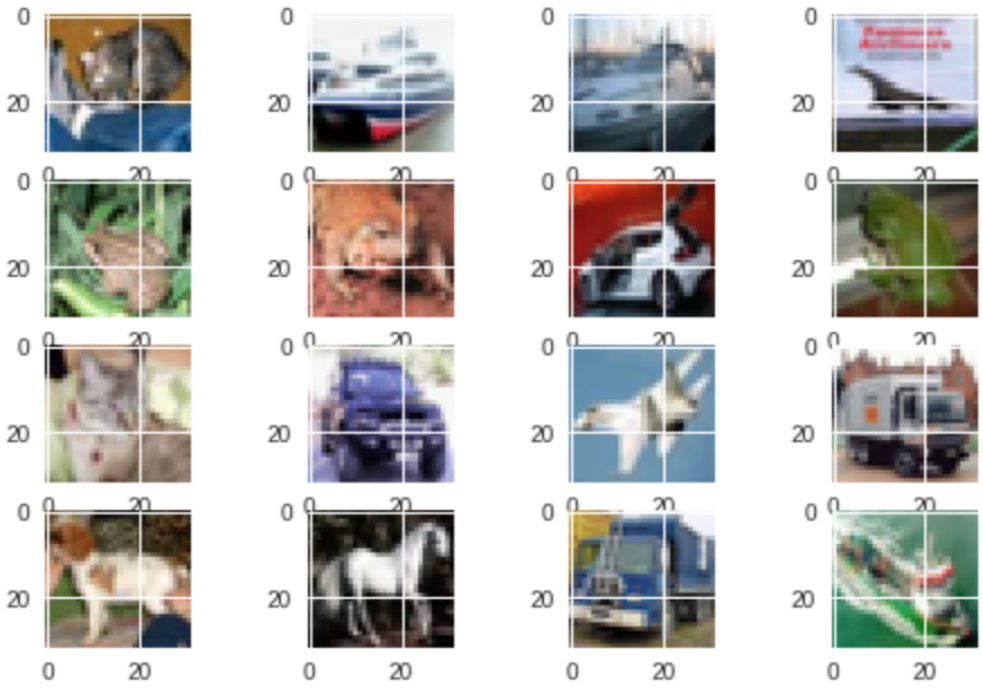


Figure 1: Reshaped Images

- Split the training data into two datasets: The new training dataset of shape (40000, 32, 32, 3) and the validation dataset of shape (10000, 32, 32, 3).
- Z-score normalization of images. This is important because it results in similarly-ranged feature values and that the gradients don't go out of control.
- Convert image labels to one-hot-encoding.
- Data augmentation. We inflate the size of the training dataset by adding randomly distorted images which are rotated, width-shifted, height-shifted or horizontally flipped. This way of distorting images will include different variation of images in training set, and will therefore make the CNN model we trained to generalize better in the test dataset.

Here are 9 examples of the a randomly-chosen image we get after data preprocessing.

Figure 2: Data Augmentation

3 Model

3.1 CNN baseline

First, we construct a CNN baseline model with only few layers to fit the data. We also take dropout layer into model to get rid of overfitting and finally the test accuracy is about 80.6%. Through visualizing the convolution layer, we can better understand how the network work with the pictures.

3.1.1 Structure

- Batch size: 128
- Epoch: 250
- Input: $32 \times 32 \times 3$ preprocessed image
  - Conv2D with ReLu: kernel size: 3×3 filter numbers: 32
  - Conv2D with ReLu: kernel size: 3×3 filter numbers: 32

- MaxPooling2D: pool size: 2×2
- Dropout: 0.5
- Conv2D with ReLu: kernel size: 3×3 filter numbers: 64
- Conv2D with ReLu: kernel size: 3×3 filter numbers: 64
- MaxPooling2D:pool size: 2×2
- Dropout: 0.3
- Dense with Softmax: 10 units

- Output: 10 probabilities
- Optimizer: Adam

3.1.2 Result

The test accuracy of this baseline model is about **80.6%** and the plots below show the training process and as we can see the trend of training and validation is almost same. Therefore the model can be improved to increase the test accuracy.

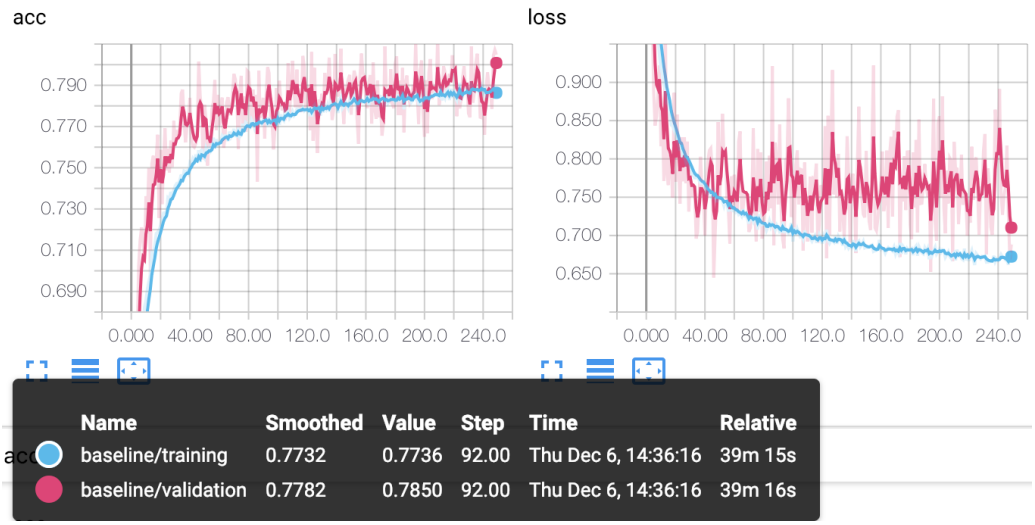


Figure 3: Baseline Training

3.1.3 Comment

- It is obvious that even though the network is not deep and wide, we can still get a relatively high accuracy with short training time. This result proves the usefulness and efficiency of neural network.
- We also investigate the inner layer output in order to better understand how the 'black box' handle the features of input images. For example, we take the image of ship and investigate the filters output of first convolution layer. The filters with different weights will focus on different parts of the image which helps to do classification.

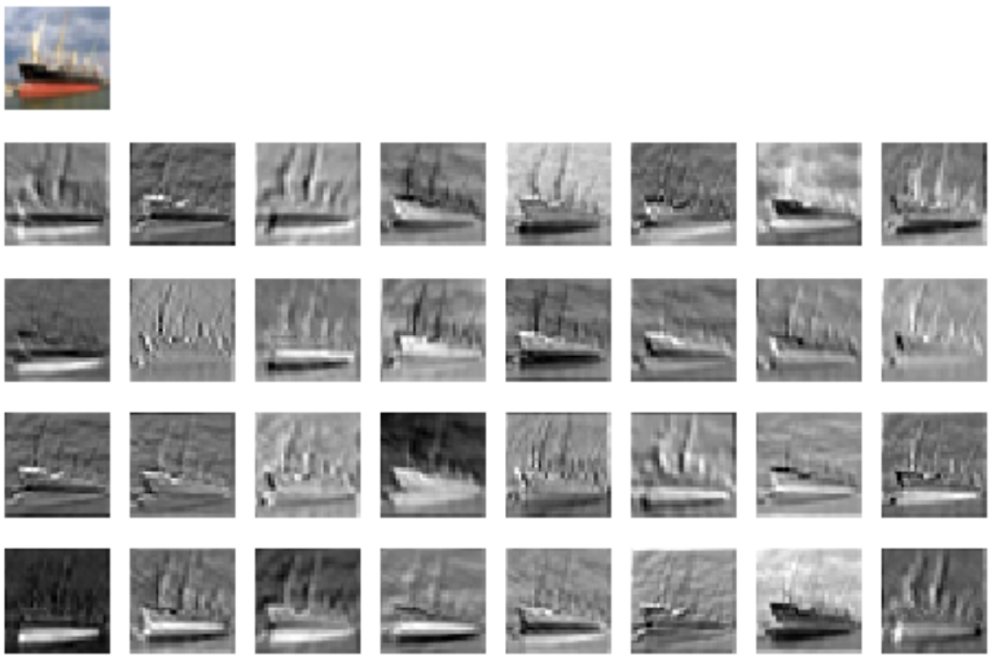


Figure 4: Baseline Conv1

3.2 CNN

In order to come up with an improved CNN model, we added more convolutional layers into the baseline described as above, and played it with different parameters and empiricism. Fortunately, we finally ended up with a satisfactory accuracy around 86.63%. The detailed structure and result of our final CNN model are as following.

3.2.1 Structure

- Batch size: 64
- Epoch: 250
- Input: $32 \times 32 \times 3$ preprocessed image
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 32
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 32
  - MaxPooling2D: pool size: 2×2
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 32
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 64
  - MaxPooling2D: pool size: 2×2
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 64
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 64
  - MaxPooling2D: pool size: 2×2
  - Conv2D with eLu and BatchNormalization with zero-padding: kernel size: 3×3 filter numbers: 128
  - MaxPooling2D: pool size: 2×2
- Dense with Softmax: 10 units
- Output: 10 probabilities
- Optimizer: RMSprop

3.2.2 Result

The test accuracy of this improved model is about **86.63%** and and the test loss has been reduced to **0.598**. The plots below show the training process of this model. As we can see from the picture, the training accuracy keeps increasing while the validation accuracy fluctuates around 0.87 with more and more epochs.

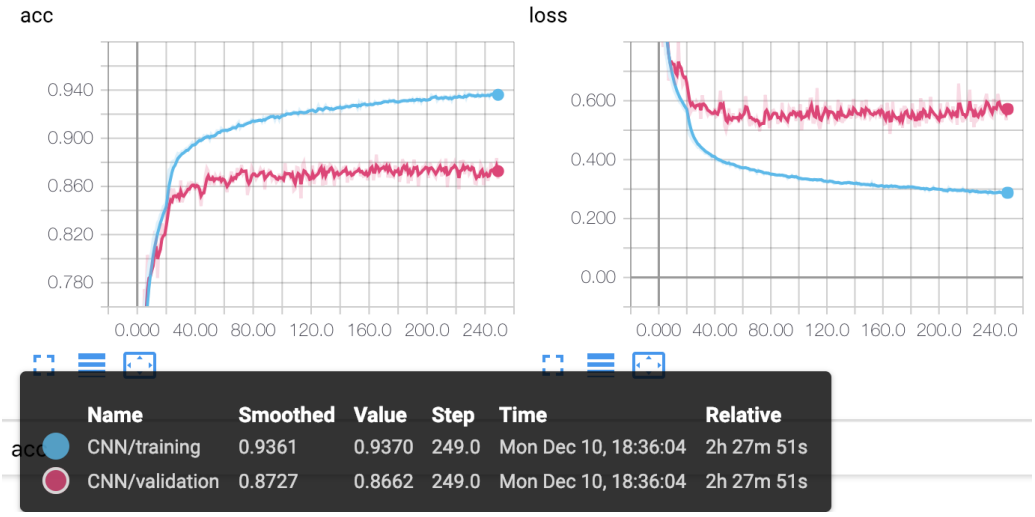


Figure 5: Improved CNN Training

3.2.3 Comment

- As we see, after playing with more numbers of layers and different combination of parameters, the final CNN model has come up with better test accuracy than the baseline model.
- Similarly as what we did in the baseline, we also investigate the inner layer output here in order to better understand how the 'black box' handle the features of input images. Taking the image of ship as an example,

Figure 6: Improved CNN Conv1

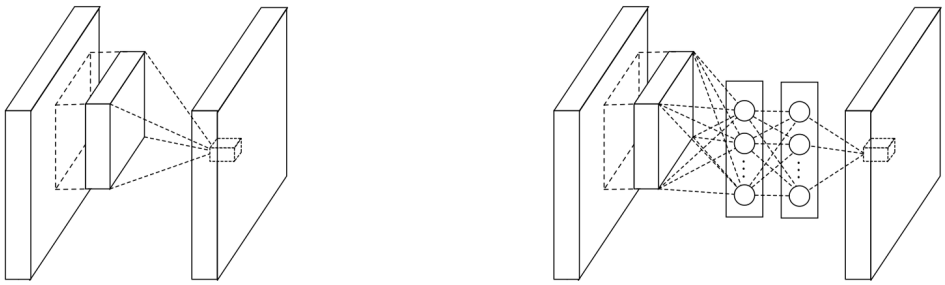
- The filter in this improved cnn model extracts different features and details from the image (compared with the baseline model).

3.3 NIN

NIN, Network in Network, is a deep network structure proposed in 2014 which enhances model discriminability for local patches within the receptive field. The final test accuracy we got is about 91.1% which is pretty high compared to previous two models.

There are mainly 2 changes in NIN:

1. **MLP Layer** maps the input local patch to the output feature vector with a multilayer perceptron (MLP) consisting of multiple fully connected layers with nonlinear activation functions followed by a classic convolution layer with activation function. This kind of a more potent nonlinear function approximator can enhance the abstraction ability of the local model.



(a) Linear convolution layer (b) Mlpconv layer

Figure 7: MLP Structure

2. **Global Average Pooling Layer** replaces the traditional fully connected layers for classification in CNN. NIN directly outputs the spatial average of the feature maps from the last MLP layer as the confidence of categories via a global average pooling layer, and then the resulting vector is fed into the softmax layer.This kind of structure will help us to better interpret the how the model to do classification (We will discuss more in comment.) and also reduce a huge amount of parameters without fully connected layer.

3.3.1 Structure

Due to MLP layer, we choose the filter size 5 at beginning instead of a smaller one in order to handle the more general features of a image. And comparing to the model with smaller filter, the accuracy of our final model proved this choice reasonable.

- Batch size: 128
- Epoch: 250
- Input: $32 \times 32 \times 3$ preprocessed image
- MLP-1:
  - Conv2D with BatchNorm and ReLu: kernel size: 5×5 filter numbers: 192
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 160
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 96
  - MaxPooling2D: pool size: 3×3, stride: 2×2
- MLP-2:
  - Conv2D with BatchNorm and ReLu: kernel size: 5×5 filter numbers: 192
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 192
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 192
  - MaxPooling2D: pool size: 3×3, stride: 2×2
- MLP-3:
  - Conv2D with BatchNorm and ReLu: kernel size: 5×5 filter numbers: 192

  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 192
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 192
  - MaxPooling2D: pool size: 3×3, stride: 2×2
- MLP-3:
  - Conv2D with BatchNorm and ReLu: kernel size: 3×3 filter numbers: 160
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 160
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 160
  - MaxPooling2D: pool size: 3×3, stride: 2×2
- MLP-4:
  - Conv2D with BatchNorm and ReLu: kernel size: 3×3 filter numbers: 160
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 160
  - Conv2D with BatchNorm and ReLu: kernel size: 1×1 filter numbers: 10
- GlobalAveragePooling2D with Softmax
- Output: 10 probabilities
- Optimizer: SGD with self-design Learning Rate

3.3.2 Result

The test accuracy of NIN model is about **91.1%** which is much higher then CNN model. The plots below show the training process

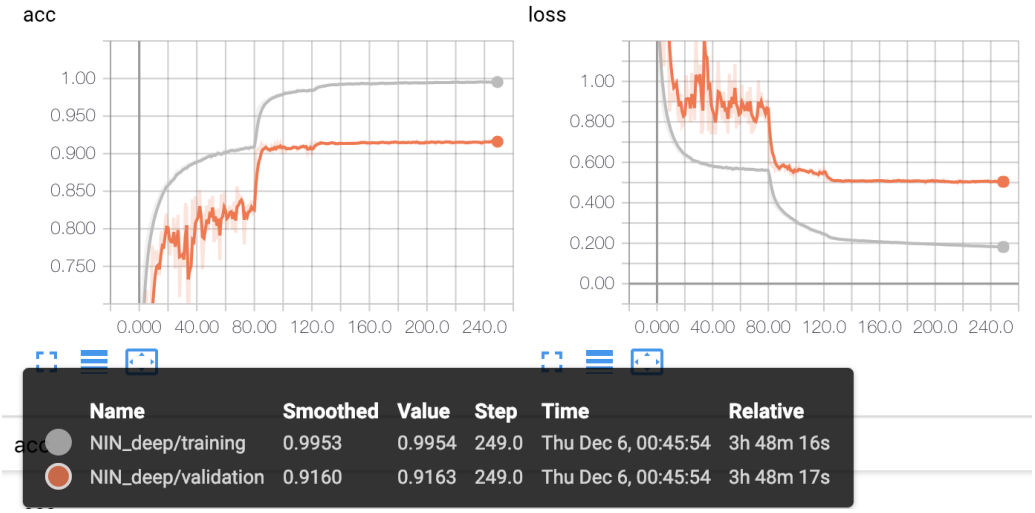


Figure 8: NIN Training

3.3.3 Comment

- Besides the data augmentation we used in our final model, we also tried different process with different rotation angle and flip, the accuracy of training changes significantly. The plot below is the comparison of final model with different learning rate scheduler and data augmentation process. *(NIN_deep is final model, NIN_deep_old is same structure but with default setting of augmentation and simple learning rate design.)*

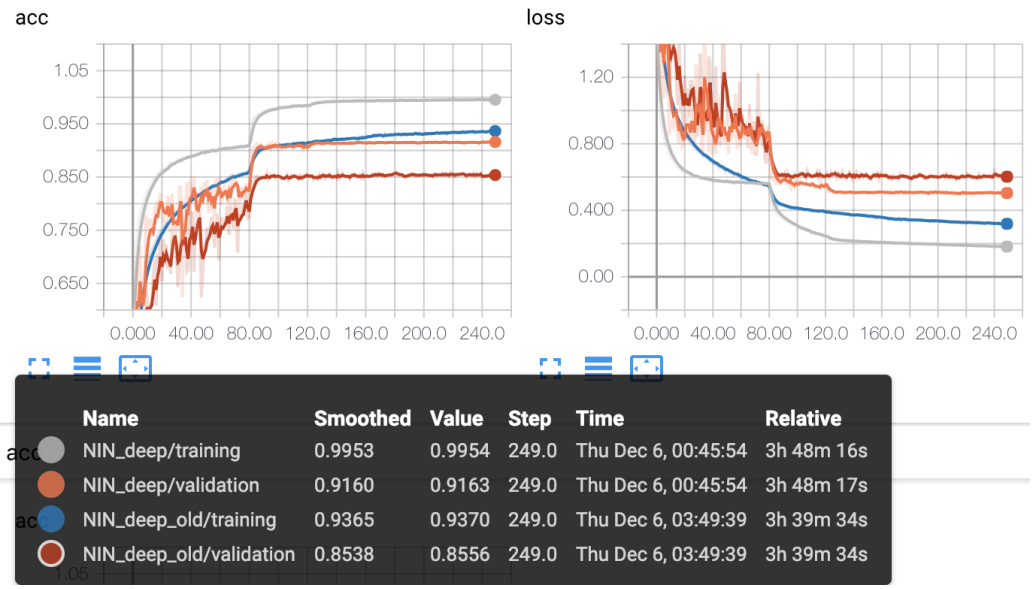


Figure 9: NIN Preprocess Compare

- From the training process image, it is obvious that there is a step at epoch 80 where is actually the place learning rate changes. This phenomenon may due to the bounce around the optimal point. So the design of learning rate is essential in this model. Therefore besides designing learning rate scheduler ourselves, we also tried the optimizer AdaDelta method which is an adaptive learning rate method. We compared the result on the NIN base model and found SGD with self-design learning rate scheduler works better here. *(NIN_adad uses optimizer AdaDelta and NIN_sgd uses self-design learning rate scheduler.)*

Figure 10: NIN Optimizer Compare

- NIN model helps better interpret the how the model to do classification because of global average pooling, so we want to know how this work. By investigating the last layer of MLP-4 we can obvious get the point. *(We take one image of each class from test set and plot the 10 filters for each image. As the predict class is same as true class, the diaganal filters should be more lighter than those off diaganal.)*

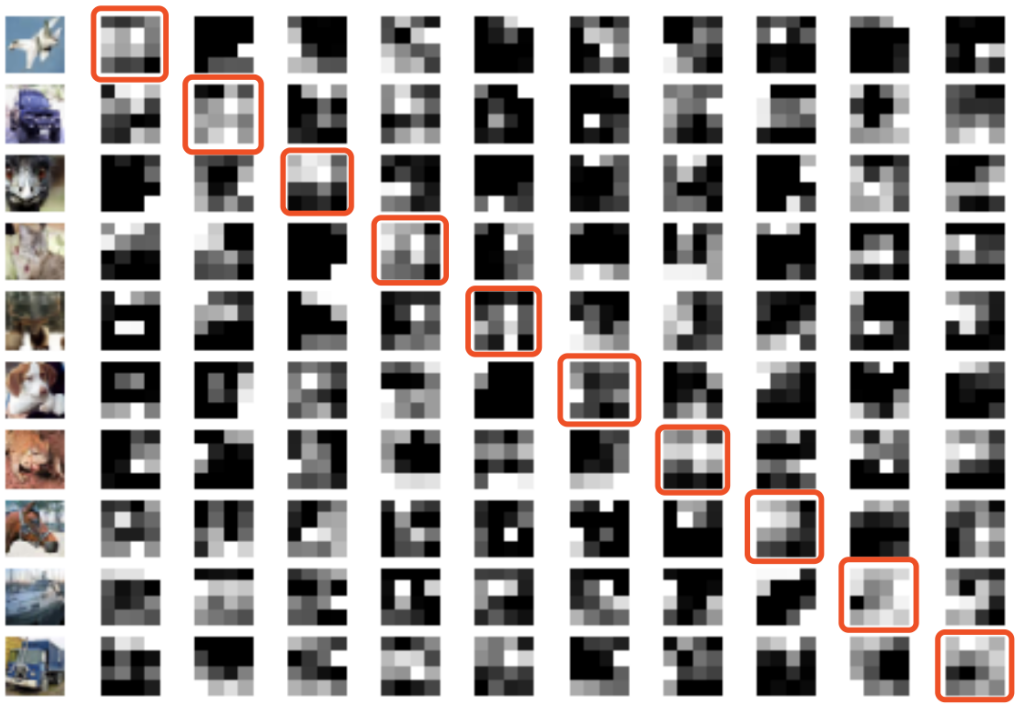


Figure 11: NIN Last Layer

4 Conclusion

- **Baseline** According to the heatmap of test data on baseline model, we can find that it does not work well on distinguishing animals like bird, cat, deer and dog. There are many images being false predicted to frog. So we need to adjust the model structure to enhance the ability of distinguishing the features between animals, which can be improved through improved CNN model and NIN model.

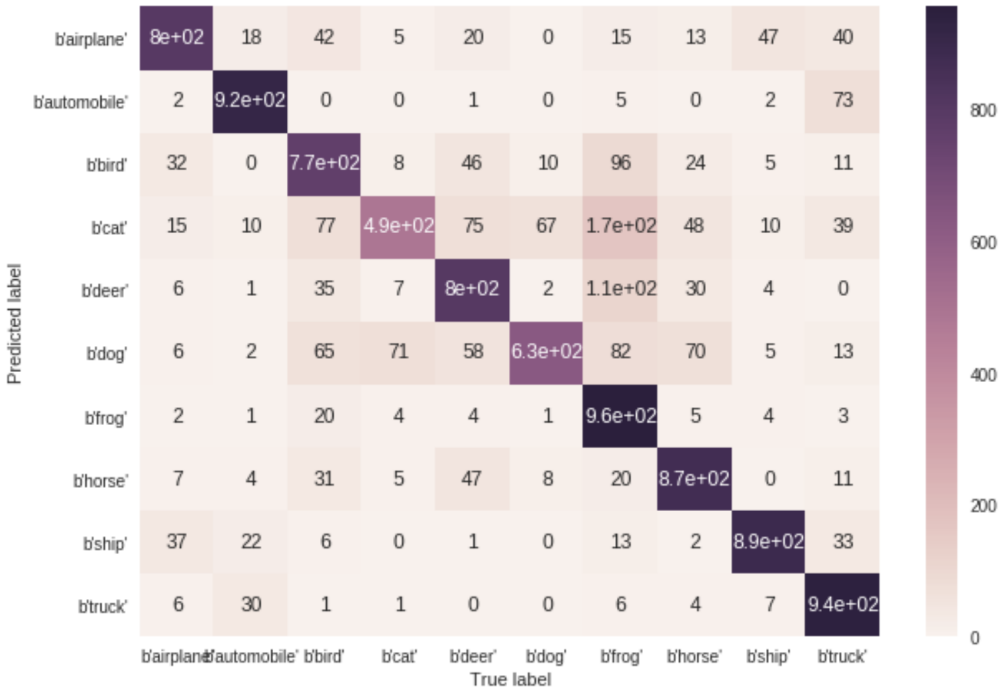


Figure 12: Baseline Test Heatmap

- **Improved CNN vs NIN** It is obvious that in the heatmap of Improved CNN model, there are many false predict between the dog and cat categories but we may can think this problem is solved in the NIN model, which may contribute to the significant increase in test accuracy.

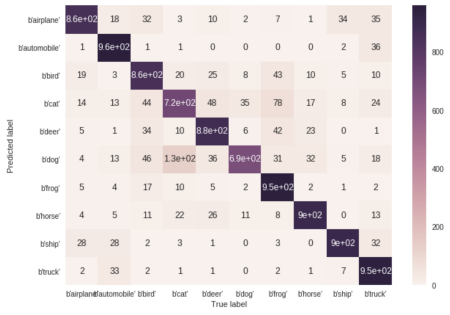


Figure 13: Improved CNN Test Heatmap

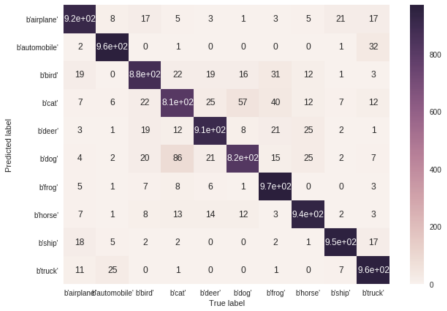


Figure 14: NIN Test Heatmap

5 Future Work

- Try more learning rate scheduler to increase the test accuracy.
- Try different data augmentation combinations.
- Add more layers to handle more features.
- Explore more deep learning models like ResNet, ImageNet, etc.

6 Reference

[1] Magnus Erik Hvass Pedersen. TensorFlow Tutorial 06. https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/06$_CIFAR-10.ipynb$
[2]$MinLin, QiangChen, ShuichengYan.NetworkInNetwork.arXiv$: 1312.4400$v3[cs.NE], 4Mar2014$.
[3]$MatthewD.Zeiler.ADADELTA : ANADAPTIVELEARNINGRATEMETHOD.arXiv$: 1212.5701$v1$
$[cs.LG], 22Dec2012$.
[4]$ParkChansung : CIFAR-10ImageClassificationinTensorFlow, TowardsDataScience, 2018, Apr, 17,$
$https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c$