# Advanced Machine Learning Final Report

Liangchen Xing, lx2212, section 002
Kuo Yang, ky2368, section 001
Yuanyuan Lei, yl3828, section 001

December 17, 2018

**Abstract**

We selected Project 1 as our final project. The main task is to build our own convolution neural networks and implement the classifier on the dataset CIFAR-10. First, we started from a simple architecture ConvNet and chose an optimization algorithm with the best performance. Second, we improved the classification accuracy and reduced overfitting by applying data augmentation techniques. Third, in order to further prevent overfitting, we added Batch Normalization between the layers of the model and increased the classification accuracy. Finally, we continued to improve our model by using dropout method and explored the influence of the dropout ratio. The final convolution network we constructed has truly good performance and the classification accuracy on the test dataset reached over 90 percent.

## 1 Data Introduction and Preprocessing

### 1.1 Data Introduction

The data we use to build convolutional neural networks classifier is called CIFAR-10. The data consists of 60000 32*32*3 colour images in 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. Among which 50000 are included in the training set and 10000 are included in the testing set. The test data contains exactly 1000 random-selected images from 10 classes and train data contains remaining 5000 images in each class. The classes are completely mutually exclusive, meaning there is no overlap between different classes.
In order to better tune the parameters in our ConvNet models, we split the original training dataset into two parts: a validation set consisting of 10000 random-selected images, and a training set consisting of the remaining 40000 images. We will use the training set and the validation set to train the classifiers, and the original test dataset to evaluate models.

## 1.2 Data Preprocessing

Data preprocessing is an important step in data mining process, including cleaning, Instance selection, normalization, transformation, feature extraction and selection, etc. Building an effective neural network model requires careful consideration of the network architecture as well as the input data format. We need to transform raw data into a format that our network can understand, also a format that benefits the learning process.

### 1.2.1 Categorical Data Preprocessing

In our dataset, class label is a categorical variable with 10 classes, denoted by integer 0, 1, . . . , 9. If we don't change its format, the model would think the values in our 10-class categorical variable have relational order. To prevent this, we can change it into dummy variables.

### 1.2.2 Image Data Preprocessing

Common image preprocessing techniques include uniform aspect ratio, image scaling, normalization, dimensionality reduction, data augmentation and etc. We will discuss the necessity of these techniques in the following paragraphs.
Out image data is rather neat, size 32 by 32, with 3 channels. Thus we don't need to worry about the uniformness of aspect ratio and image scaling.
To ensure each input pixel has a similar distribution, we will normalize the input images. This can also make convergence faster while training the network.
We will skip the dimensionality reduction step, which collapses the RGB channels into a single grayscale channel. Because we think the color is important for classification, as we can see from the mean images of different classes below.

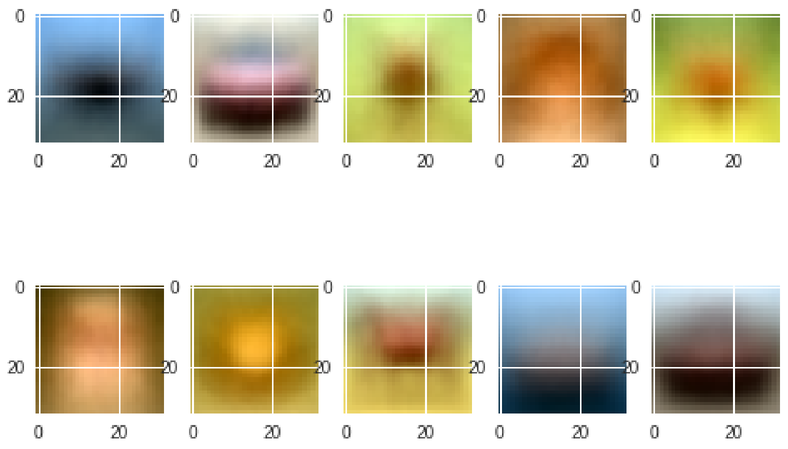


Figure 1: data preprocessing - mean images of classes

## 2 Model One - Starting Model

### 2.1 Understanding how ConvNet works

**What is ConvNet:**
Convolutional Neural Networks are a class of Neural Networks that have proven very effective in areas such as image recognition and classification.
ConvNets derive their name from the "convolution" operator. The primary purpose of Convolution is to extract features from the input image while preserving the spatial relationship between pixels.

**ConvNet terminology:**
*Filters and Feature Maps*: In general, after input layer, Convolution use *Filters* (always a $3 * 3$ matrix) to detect features from the original input image. Convolution slides the filters over the image and computes the dot product to get *Feature Maps*, saving activations of detected features.
*Activation function*: often applied after every Convolution. Most common used function is Rectified Linear Unit. ReLU is an element wise operation which replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to add non-linearity to the ConvNet which is a linear operation.
*Spatial Pooling*: a widely used method to reduce the dimensionality of feature maps but retains the most important information. In the case of max-pooling, a filter (often a $2 * 2$ matrix) slides over the feature maps and takes the largest element within the window. Pooling reduces the number of parameters, thus can help control overfitting and decrease computation costs. That's why convolution layers are always followed by pooling layers.
*Fully connected layer*: connects every neuron in the previous layer to every neuron on the next layer. Thus we need to flatten the feature maps before we feed them into this layer. The output from previous convolutional and pooling layers usually represents high-level features of the input image. The Fully Connected layer can learn the non-linear combination of output features from previous layers, which would benefit the classification.

### 2.2 Starting Model Architecture

We created our starting model based on the architecture of VGG-16 model with the following characteristics:

- Layers combinations: Two convolutional layers or three convolutional layers combined with one max-pooling layer.
- The number of feature maps: 64 feature maps for the convolutional layers close to the input layer, 256 or 512 feature maps for each layer close to the output layer.

We used a similar structure as our starting model. First, we use a combination of two convolutional layers and one max-pooling layer to extract major spatial

features with only 64 feature maps. Then, we added another two combinations to extract more detailed spatial features with 128 and 256 feature maps. At last, we added two full connection layers in order to transfer spatial features from convolutional layer to signals in order to make predictions with softmax activation function.
When building the ConvNet model, we set *epoch* number to be 156 and *batch size* to be 256. During the training process, our model trains on the whole training set 156 times and uses the validation set to adjust the parameters. Within each epoch, the optimization algorithm randomly selects 256 data observations to update the parameters until using out all of the 40000 training data. After the training process, we use the test set to predict images classes and get test accuracy to evaluate model performance.

## 2.3 Optimizer Choice

During the training process, we need a sound optimization algorithm to learn the value of the parameters and minimize the loss function as accurate and as fast as possible. There are six possible optimization algorithms we are able to use in Keras model: RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam. They are all adaptive learning rate methods and proved to be more intelligent and have better performance than ordinary stochastic gradient descent method. In the training process, these optimization algorithms adapt the learning rate to the parameters, performing smaller updates for parameters with frequently occurring features and larger updates for parameters with infrequent features.
In order to select the best performance optimization algorithm, we tried all those optimizers within our starting model and compared their performances by validation accuracy and convergence speed. The results are shown in the picture below, the left one is classification accuracy of validation dataset and the right one is the loss function value with respect to the epoch increase.

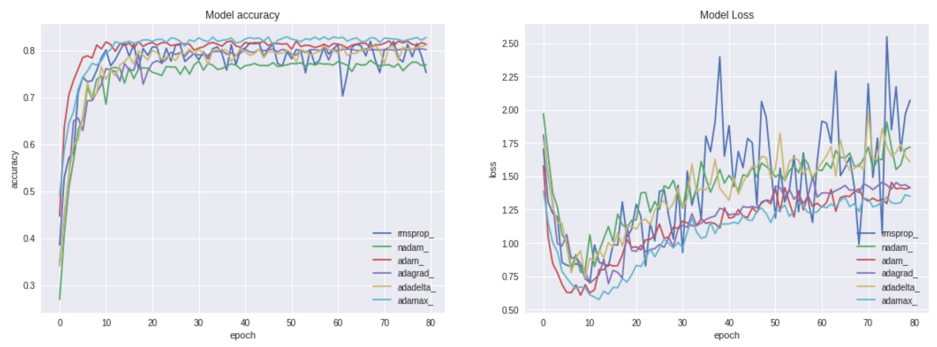


Figure 2: Comparison of Different Optimization Algorithms

From the comparison pictures above, we can see that the AdamMax algorithm has the highest classification accuracy for the validation set and lowest loss function as epoch becomes larger. The convergence speed of AdamMax is also the second fastest among these six candidates, only slightly slower than Adam method. For this reason, we decide to use AdamMax optimizer in our ConvNet model.

### 2.4 Model Performance

After building our own starting convolution neural network model with the architecture stated above and implementing it using Keras package in Python, we get the testing classification accuracy is **0.8026**. We also plot out the accuracy ratio and loss function value of training and validation sets with respect to epoch increases during the training process as follows:

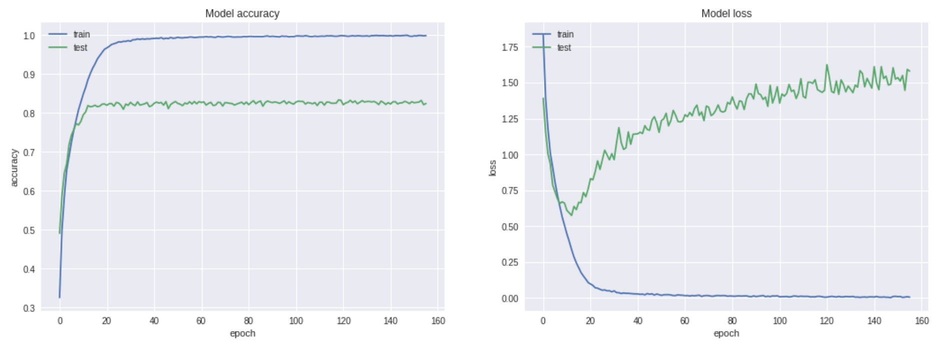


Figure 3: Model One Performance - Starting Model

## 3 Model Two - Data Augmentation

We can see from the plot above that there exists a severe overfitting problem in our starting model: the training accuracy starts to outperform the validation accuracy in very early epochs. Overfitting tells us the network we built is complex enough for CIFAR-10 classification problem and we need to prevent overfitting either by refining the ConvNet architecture or applying regularization techniques. Besides, there is another way to reduce overfitting: increase the sample size by Data Augmentation method.

### 3.1 Data Augmentation

Data augmentation is a technique that generates images by transforming the actual training images by rotation, crop, shifts, flip, reflection, zoom and so on. There are two main benefits of this technique:

- **Enlarge training set**: Our training and validation set is not a relatively large dataset: only 5000 images in each class. However, our model is of high complexity and have large numbers of parameters (167k). In order to capture more features, we want to maintain the complexity of the model and thus need more data to feed our model. Data augmentation helps us generate large quantity of data and also cover a larger portion of the data distribution.

- **Avoid irrelevant features**: Data augmentation can prevent the network from learning irrelevant features, like specific viewpoints, so as to reduce overfitting. Take the bird category as an example, as we can see in the following picture, some images contain the whole body of the bird, some only contain the head. And the bird can be anywhere in the image, not necessarily in the middle. Techniques in data augmentation like shift and scale can prevent the network from learning these irrelevant characteristics.

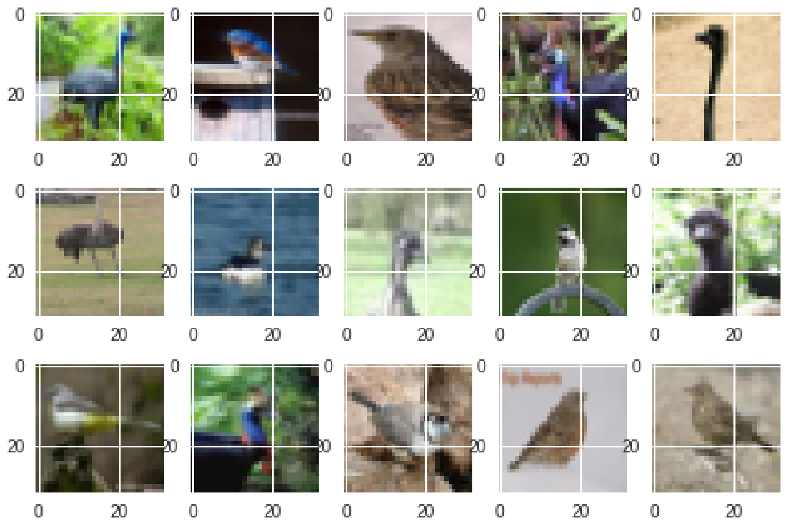


Figure 4: Data Augmentation: prevent from learning irrelevant features

To improve the starting model one, we used three kinds of data augmentation methods and form the model two: rotation (0-20 degree), horizontal flip, width and height shift (within 10 percent of original size). We will find that this technique truly helps us improve the classification performance.

## 3.2 Model Performance

After applying the data augmentation techniques stated above, we amplified our data quantities. Training the ConvNet model on the data after augmentation, the testing accuracy has been increased to **0.8764** by over 7 percent. We also plot out training and validation accuracy and loss function value as epoch increases as follows:

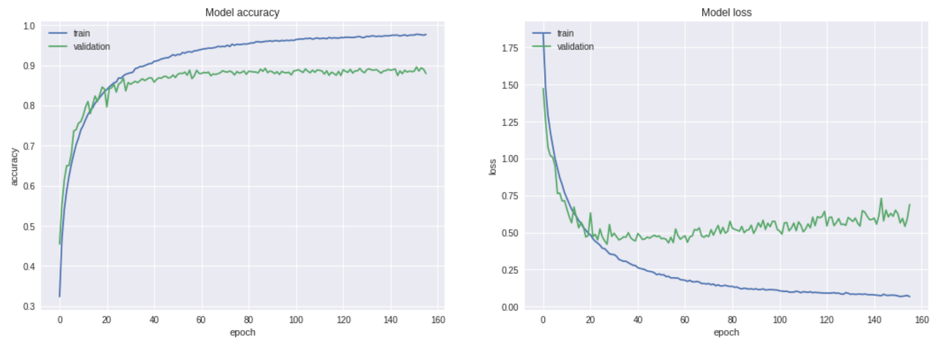


Figure 5: Model Two Performance - Data Augmentation

From the performance result of model two, we can see that Data Augmentation truly increase classification accuracy a lot. Comparing the trend of training loss and validation loss as epoch becomes larger, we can see that Data Augmentation also help solve the overfitting problem to a great extent.

## 4 Model Three - Batch Normalization

However, as we can see in the result picture of model two above, our model still suffers from overfitting problem despite using Data Augmentation. In this part, we are going to make some improvement in the network architecture to further reduce overfitting.

### 4.1 Batch Normalization

The concept of Batch Normalization is rather easy to understand. It is typically added between layers to normalize the training data in each mini-batch with 256 observations. Batch Normalization can reduce internal covariate shift, accelerate learning and make the network more tolerant about initialization.
But in practice, there is a debate about "Batch Normalization before or after ReLU", so we implemented both architectures to compare the results. We added seven BN layers in the starting model, before or after ReLU activation layer. See results below (We use BN to denote Batch Normalization, ReLU-BN and BN-ReLU to denote the two architectures.)
We can see ReLU-BN (red) performs better than BN-ReLU (green) architecture, as the red curve not only converges faster with a higher accuracy and a lower loss but with less fluctuation rate. Comparing ReLU-BN (red) with the model without BN (blue), ReLU-BN (red) starts to outperform the model without BN (blue) after epoch 60 with a higher accuracy and a lower loss.

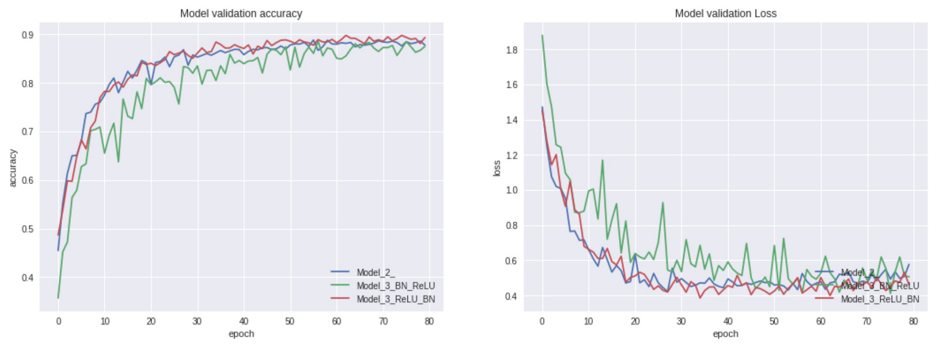


Figure 6: Comparison of using Batch Normalization before or after ReLU

Thus we come to two major conclusions: 1. BN can help prevent overfitting and accelerate the learning process, so we should use it to refine the architecture. 2. we should add BN after ReLU activation to improve performance.

## 4.2 Model Performance

After using Batch Normalization and adding it after activation function ReLU, we retrained our model and the testing accuracy has been increased to **0.8992** by over 2 percent. The following result is again the accuracy ratio and loss function value of the training set and validation set during the training process as epoch becomes larger:

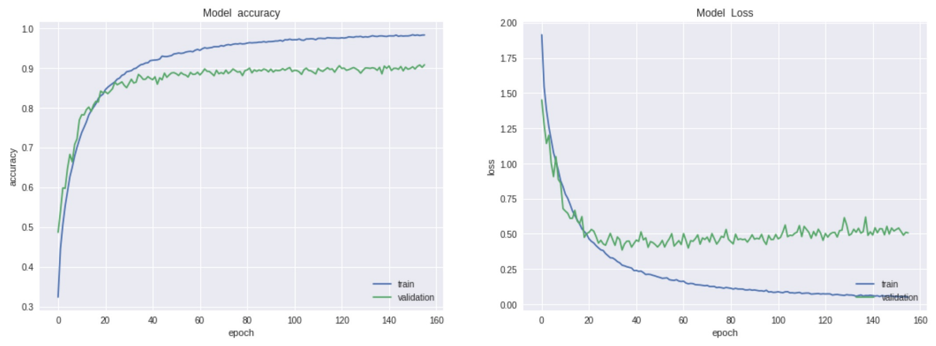


Figure 7: Model Three Performance - Batch Normalization

From the result above, our current model already has a sound performance. In the next step, we will continue to refine it by adding Dropout layers.

# 5 Model Four - Dropout

## 5.1 Dropout Method

If we have unlimited computational power, in order to prevent overfitting, we can fit the data in different convolutional models, and then combine them and average the results to make predictions. While in practice, we encounter the limitation of our computational power. The average model will take too long to make predictions. That is why we want to introduce Dropout method, which can also reduce overfitting by teaching to a neural networks to average all possible sub-networks with a lower computational cost.

### 5.1.1 Averaging sub-networks trained by each mini batch

The dropout layer will randomly drop some hidden unit by a probability. This will make each hidden units be able to generalize the input signal instead of the noise. In the entire training process, a dropout layer could be treated as a random sampling for the parameters of each sub-network (convolutional networks between two dropout layers). During one epoch, based on each training mini-batch, we only update the parameters within our new sub-network (we drop different parameters every time). Note that the massive sub-networks are not independent of each other – they share the weights. Therefore, we can treat the dropout as an average process for the prediction base on each sub-network.

### 5.1.2 Reducing the parameters of each layer

Each time, when updating parameters base on back-propagation, we only consider the subset of parameters in each layer. Therefore, our computational cost will be reduced significantly.

In our Model, we use:
Module-1: two convolutional -layers + one max-pooling layer
Module-2: two convolutional-layers + one max-pooling layer
Module-3: two convolutional-layers + one max-pooling layer

The difference between Model 4 and Model 3 is that, in Model 4, we add one dropout layer in Module-2 (with probability 0.5) and two dropout layers in Module-3 (with probability 0.5), and removed several Batch Normalization functions in Module-1, Module-2, and Module-3.

We made these changes based on the consideration that there is a disharmony between Dropout and Batch Normalization, which is also proved theoretically that Dropout would shift the variance of a specific neural unit when we transfer the network from training to test, while batch Normalization makes effort in maintaining the variance. So we have to be very careful about where to add Dropout layers.

By experiment, we find that directly adding dropout layers on Model 3 does not make the model perform better. Thus we tried several possibilities to remove some of the Batch Normalization functions. Finally, by experiment, we come to our Model 4 by adding three dropout layers in total and achieve a prediction accuracy above 90 percent.

## 5.2 Model Performance

After applying dropout method as stated above and modifying our model structure, we get to our final result. The testing accuracy has been increased to **0.9033** which competes a lot of other classifiers. The training and validation accuracy and loss function as epoch increases are plotted as below:

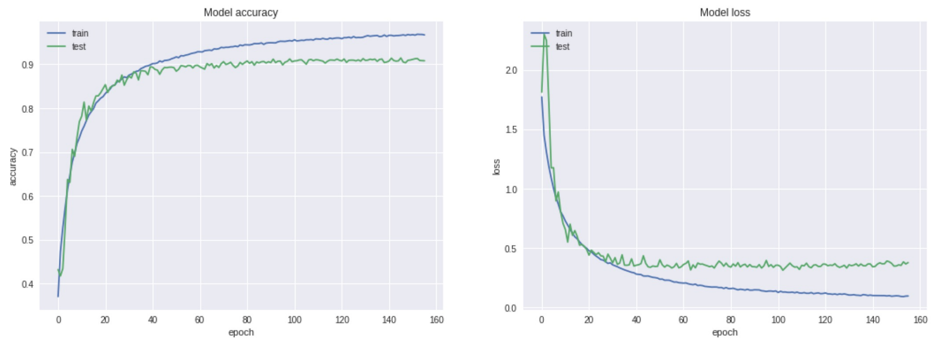


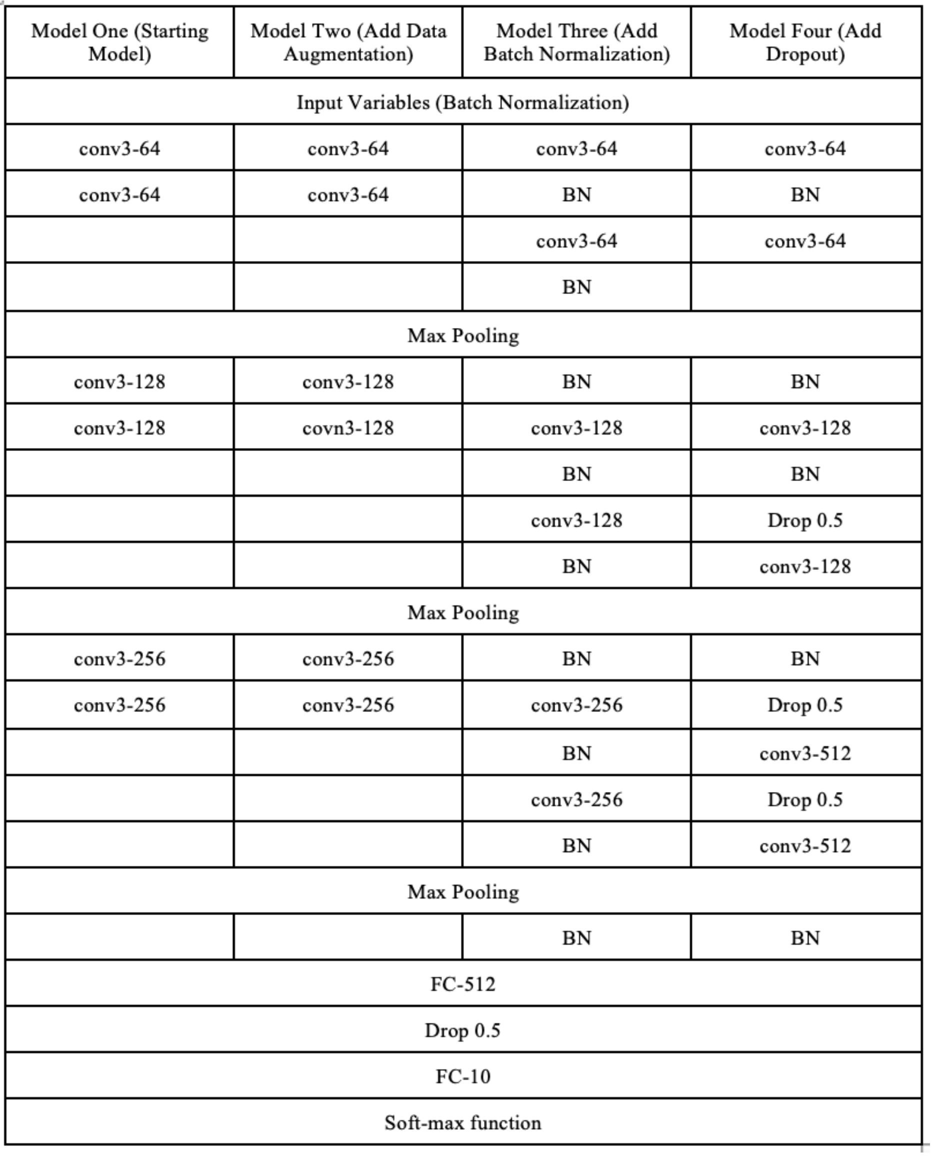
Figure 8: Model Four Performance - Dropout

From the result of our final model four, we can see that classification performance is satisfying and the huge overfitting problem at the beginning has also been solved a lot. The convergence speed is also very fast.

# 6 Model Summary and Conclusion

## 6.1 Models Architecture Summary

We constructed our classification ConvNet model first based on the characteristics of VGG-16 model. Then we improved the model step by step: Model two adds Data Augmentation on Model one, Model three adds Batch Normalization method based on Model two, Model four adds Dropout on Model three.
We summarize our models architectures here and compare their differences as shown in the table below. "conv3-64" means a convolution layer with 64 3*3 filters, "BN" means batch normalization, "Drop 0.5" means drop out 50 percent of the parameters, "FC-512" means a fully connected layer with 512 units.



| Model One (Starting Model) | Model Two (Add Data Augmentation) | Model Three (Add Batch Normalization) | Model Four (Add Dropout) |
|---|---|---|---|
| Input Variables (Batch Normalization) | | | |
| conv3-64 | conv3-64 | conv3-64 | conv3-64 |
| conv3-64 | conv3-64 | BN | BN |
| | | conv3-64 | conv3-64 |
| | | BN | |
| Max Pooling | | | |
| conv3-128 | conv3-128 | BN | BN |
| conv3-128 | covn3-128 | conv3-128 | conv3-128 |
| | | BN | BN |
| | | conv3-128 | Drop 0.5 |
| | | BN | conv3-128 |
| Max Pooling | | | |
| conv3-256 | conv3-256 | BN | BN |
| conv3-256 | conv3-256 | conv3-256 | Drop 0.5 |
| | | BN | conv3-512 |
| | | conv3-256 | Drop 0.5 |
| | | BN | conv3-512 |
| Max Pooling | | | |
| | | BN | BN |
| FC-512 | | | |
| Drop 0.5 | | | |
| FC-10 | | | |
| Soft-max function | | | |

Figure 9: Model Architecture Summary

### 6.2 Results and Conclusions

We summarize the classification accuracy on testing dataset of each model as shown in the table below:

| Steps | Model One | Model Two | Model Three | Model Four |
|---|---|---|---|---|
| Accuracy | 0.8026 | 0.8764 | 0.8992 | 0.9033 |

Figure 10: Models Accuracy Results

And we also plot out the accuracy and loss function value on validation sets as epoch increases of the corresponding four models, as shown in the picture below:

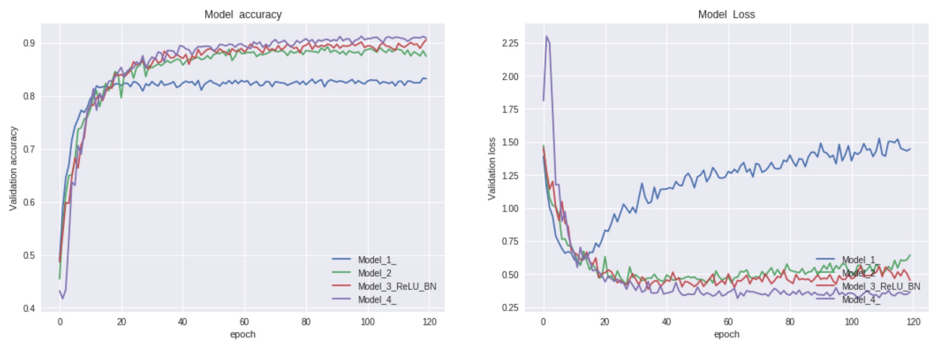


Figure 11: Models Comparison

Thus we come into the final conclusions here:

- Within each step of refining the model architecture, the test accuracy was improved steadily and the overfitting problem was weaken step by step.
- Data Augmentation, Batch Normalization, Dropout can indeed improve the model performance and reduce overfitting.
- The final model we use can classify CIFAR-10 well with **90.33%** accuracy

## 7 Success, Insights, Challenge and Future steps

### 7.1 Key Success

- We designed a logical experiment design process: adding method step-by-step and gradually improved our model (from 82 percent to 87 to 89 percent to 90 percent)

- We searched significant number of papers on convolutional network improvement. And we put focus on preventing over-fitting situation by introducing those methods to our models in each step
- By using convolutional module from VGG-16 model, we construct a very good starting model, and achieve 82 percent prediction accuracy
- By correctly understanding and applying different optimizer methods, augmentation method, batch normalization method, and dropout method, we continuously improved our model prediction performance
- We realized that adding batch normalization function and dropout method together might results contradiction. Therefore, by doing experiment, we successfully switch the model structure from model 3 to model 4—by remove most batch normalization functions and add three dropout method to each convolutional module

## 7.2 Key Insights

- The structure of the starting model may have an impact on the best accuracy we could reach after improvement
- A good optimizer can dramatically accelerate learning rate and increase accuracy. But the choice of optimizer may vary according to different datasets and model architecture
- The order of Batch Normalization and activation functions is an interesting thing to explore. For in some cases, these two architectures don't have much difference in performance, while in our case one outperforms the other
- Batch Normalization and Dropout are both powerful tools to reduce over-fitting, but chances are the performance become much worse by combining them together. Model accuracy is quite sensitive to different combinations and experiments have to be taken to solve this problem

## 7.3 Challenge

In our experiment, the biggest challenge we met is that, each time we change the model architecture, we need to switch the location of batch normalization function. Otherwise, the model may not perform better.
In addition, when we directly add dropout from our Model-3 to Model-4, the model initially performs badly. Batch Normalization will minimize the variance among each mini-batch subset. However, the dropout method will generate difference sub-network structure for each mini-batch subset. This will make prediction unstable. Therefore, we make experiment: trying to remain a little batch normalization function in model-4 in order to prevent this situation. Finally

we found that, by adding dropout function, we achieve prediction accuracy above 90 percent on model-4.
In this project, we also tried fine-tuning with the pre-trained VGG-19 network, which pre-trained on ImageNet dataset. However, since the resolution between these two datasets is too different from each other, fine-tuning does not perform very well.
We also tried to use more than 10 layers structure to the cifar-10 dataset. By taking a long time training process, we only achieve accuracy around 75 percent. Due to time limitation, we cannot improve our model by change parameters. Then we give up and turn to use model with less layers.

### 7.4 Future Steps

In this experiment, we also implements 20 layers ResNet. Although the paper shows Resnet-34 improves prediction accuracy to 93 percent, ResNet-20 does not preform that well. It takes longer training time than model-4 and performs only 73 percent validation accuracy. Also, in the epochs after 50, the validation loss increases. This indicates the model is over-fitting.
Due to the time limitation, we did not continue this model. Maybe this model could perform even better than the Model-4 by change the parameters of each convolutional layer.

## References

[1] https://en.wikipedia.org/wiki/Data-pre-processing
[2] Karen, Simonyan, and Zisserman Andrew. "Very Deep Convolutional Networks for Large-Scale Image Recognition." [Astro-Ph/0005112] A Determination of the Hubble Constant from Cepheid Distances and a Model of the Local Peculiar Velocity Field, American Physical Society, 10 Apr. 2015, arxiv.org/abs/1409.1556.
[3] Li, Xiang, et al. "Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift." [Astro-Ph/0005112] A Determination of the Hubble Constant from Cepheid Distances and a Model of the Local Peculiar Velocity Field, American Physical Society, 16 Jan. 2018, arxiv.org/abs/1801.05134.
[4] Srivastava, Nitish, et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." CiteSeerX, June 2014
[5] Howard, Andrew G, Zhu, Menglong, Chen, Bo, Kalenichenko, Dmitry, Wang, Weijun, Weyand, Tobias, Andreetto, Marco, and Adam, Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.
[6] Hannun, Awni, Case, Carl, Casper, Jared, Catanzaro, Bryan, Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, et al. Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567, 2014.