

Image Classification using Convolution Neural Network

Zhongyu Kuang

1. Introduction

1.1 Overview

Image classification is one of the fundamental building blocks for solving visual object recognition problem in the computer vision domain. Extracting information classes according to images' visual contexts (for instance, whether this is an image of a boat or an airplane), as it seems like such a natural task for us human, however, it has proven to be a challenge for computers. With the blooming research in deep learning and the development of fast computation in the past decade, our community has achieved drastic progress in image classification problem. Ever since the annual ImageNet competition[1] first started in 2010, the top-5 error rate for classifying 1000 object categories of 150,000 images has dropped from 28% (winning result in 2010 competition) to 3.08% (winning result in 2016 competition). Prior to the success of AlexNet[2] in 2012, the popular algorithm for approaching the image classification problem is through the visual descriptors[3]: the winning team of the ImageNet competition in 2010 used descriptor coding with support vector machine to achieve the 28% top-5 error rate. In the ImageNet competition 2012, Krizhevsky et al.[2] applied deep convolutional neural network (DCNN) in the image classification problem and brought down the top-5 error rate from 26.2% to 15.3% (Figure 1 shows the winning results of ImageNet classification competition from 2012 to 2016 using DCNN based algorithms). The successful application of DCNN in visual object recognition has motivated a lot of research in applying deep learning technique to solve computer vision problems.

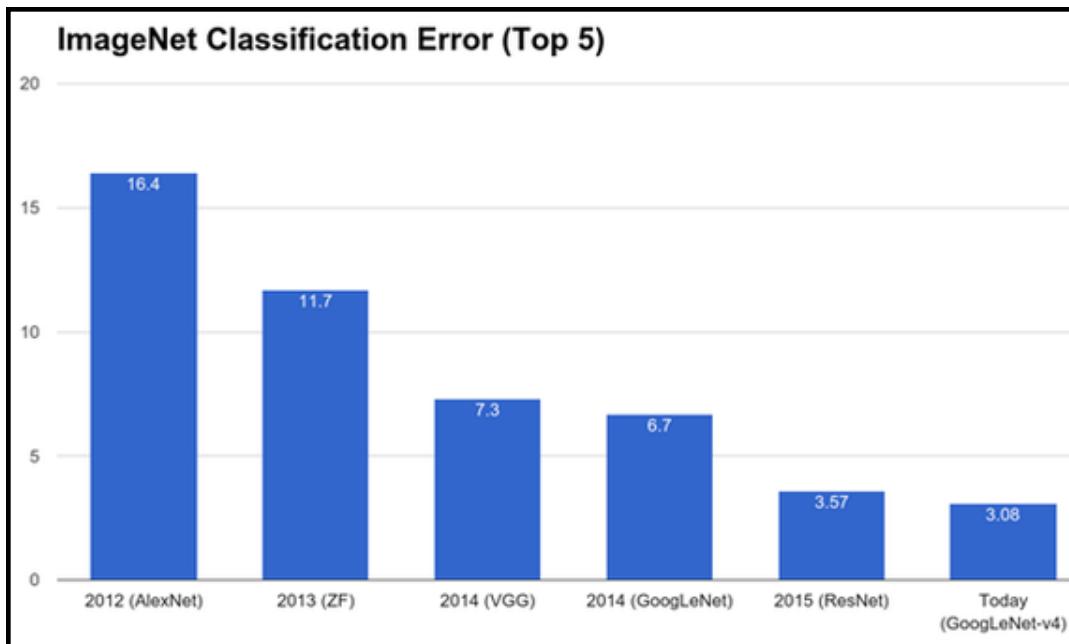


Figure 1. ImageNet competition winning team top-5 error rate

The purpose of this project is to study DCNN algorithm and apply it to solve natural image classification problem in practice. Among the various datasets available[4] for deep learning research, CIFAR-10[5] is chosen for this project for the following reasons:

- (a) CIFAR-10 has a fairly large set of samples (50,000 training images and 10,000 test images) with well-balanced object categories (10 classes total, 6000 images per class)
- (b) CIFAR-10 dataset is suitable and complex enough to employ a DCNN algorithm, however, it's simple (sample resolution: 32 pixels by 32 pixels) enough such that a less computationally expensive architecture could undertake.
- (c) CIFAR-10 has been widely used in deep learning research so that there are many benchmark results available for comparison.

1.2 Problem Statement

The objective of this project is to develop a CNN network for correctly classifying images in the CIFAR-10 dataset. Even though recent research indicates that complex connected CNN architecture has great potential in performing image classification tasks very accurately (i.e. Inception[6], ResNet[7] and GoogLeNet-v4[8]), due to the limitation of computation resources, a vanilla flavored rail-road stacking architecture is used in this project. In order to better monitor the training process, in addition to the testing images, another 10,000 images are randomly selected from the 50,000 training images to form a validation set. The category labels are marked by numbers ranging from 0~9, which each unique number corresponds to one unique category. All classes are mutually exclusive and non-overlapping. Before feeding the dataset into the designed network, one-hot encoding is applied to convert the label information into a vector format.

1.3 Metrics

The classification accuracy, where the accuracy is calculated as in equation 1, is used as the performance metric for evaluating the CNN network. The metric is chosen for its simplicity to interpret, its effectiveness for evaluation and its wide usage in similar deep learning research using the CIFAR-10 dataset. Although this is a multi-class classification problem, the reason for not using a more complex metrics (such as f1-score, precision, recall, etc.) is because they are less meaningful in the context of the general purpose of image classification.

$$\text{accuracy} = \frac{\text{number of correctly classified instances}}{\text{total number of instances}} \quad (\text{equation 1})$$

2. Analysis

2.1 Dataset Exploration

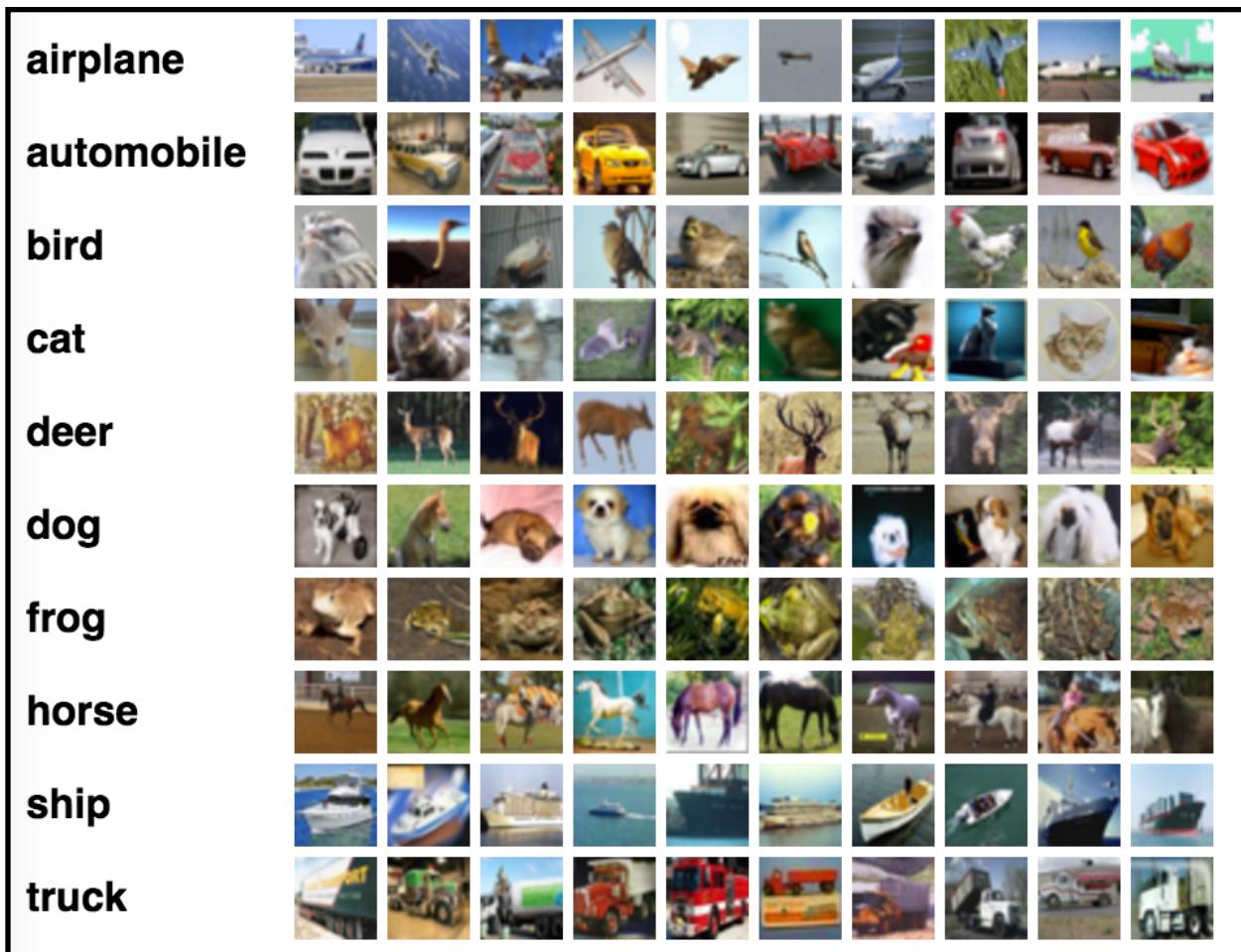
A quick summary of the CIFAR-10 dataset is provided in Table 1. As it is shown, the CIFAR-10 dataset has an equal amount of samples in each category and 10 mutually exclusive categories in total. Each sample in the dataset is a 32-pixel by 32-pixel by 3 RGB channels image. The value of a sample at any pixel locations of any color channels ranges from 0~255.

Table 1. Summary of CIFAR-10 Dataset for this project

	Training Set	Validation Set	Testing Set
sample size	40000 samples	10000 samples	10000 samples
total categories	10 categories (4000 samples per category)	10 categories (1000 samples per category)	10 categories (1000 samples per category)
sample resolution	32 pixels by 32 pixels	32 pixels by 32 pixels	32 pixels by 32 pixels
color channels	3 RGB	3 RGB	3 RGB

2.2 Data Sample Visualization

Figure 2 lists out all of the 10 classes and provides 10 random images from each class. A random sample from the frog category is plotted in Figure 3 as an example for stand-alone sample visualization.

**Figure 2. All 10 classes of the CIFAR-10 dataset and 10 random images for each class**

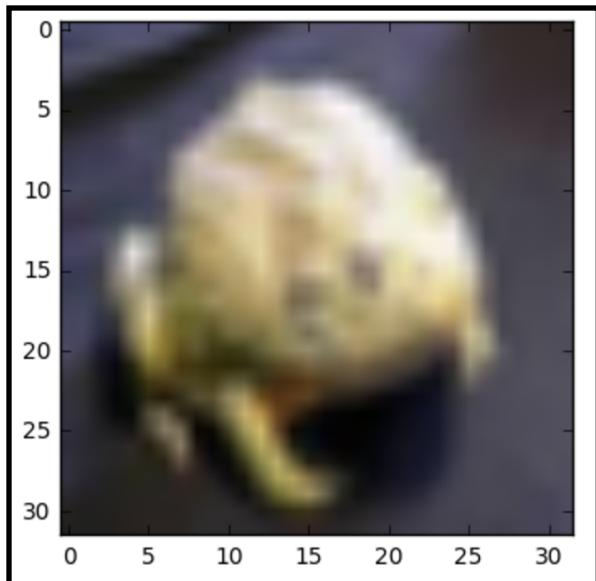


Figure 3. A random sample from the frog category for stand-alone sample visualization

2.3 Algorithms and Techniques

As the convolutional neural network has shown great success in image classification, it is used in this project for classifying the CIFAR-10 dataset. The basic building blocks for constructing a CNN includes convolutional layers, pooling layers, and fully-connected layers.

As the core building block for CNN, a convolutional layer is computed by sliding certain fixed size (i.e. 3x3, 5x5, 7x7) two-dimensional filters/windows across the input images to obtain different feature maps. Suppose the input data has the dimension of [32x32x3] (input width:32, input height:32, input depth: 3) and a [3x3] filter size is chosen to construct the convolutional layer. During the forward pass, a [3x3x3] (filter width: 3, filter height: 3, filter depth: 3) filter is sliding across the width and height of the input sample and computing the dot product followed by pre-specified activation function (commonly used activation functions include ReLu activation, tanh function and sigmoid function) to form the output feature map (Figure 4, adapted from [9]). During the backpropagation pass, the gradient of each neuron in the filter with respect to the final output is computed and the weight parameters are adjusted in the next step of training accordingly to reduce the total loss function of the network. The number of filters to use per convolutional layer, the size of the filter, the stride to use when slides the filter across the input and the padding choices around the border are all hyper-parameters. The design of convolutional layers enables parameter sharing and the statistical invariant of neurons by sliding filters across the height and the width dimension. Additionally, it reduces the computational costs since it has fewer parameters to train as compared to fully connected layers.

Pooling layers are commonly inserted in between convolutional layers to gradually reduce the spatial size of feature maps. The operation mechanism (Figure 5, adapted from [9]) is very similar to convolutional layers: with the size of filters and the stride to use for sliding procedure as hyperparameters, a pooling layer down-samples input feature maps based on the choice of

specified pooling method (commonly used pooling methods: average and maximum). It is worth noting that there are no trainable parameters in pooling layers and the main purpose of pooling layers are for down sampling activated feature maps.

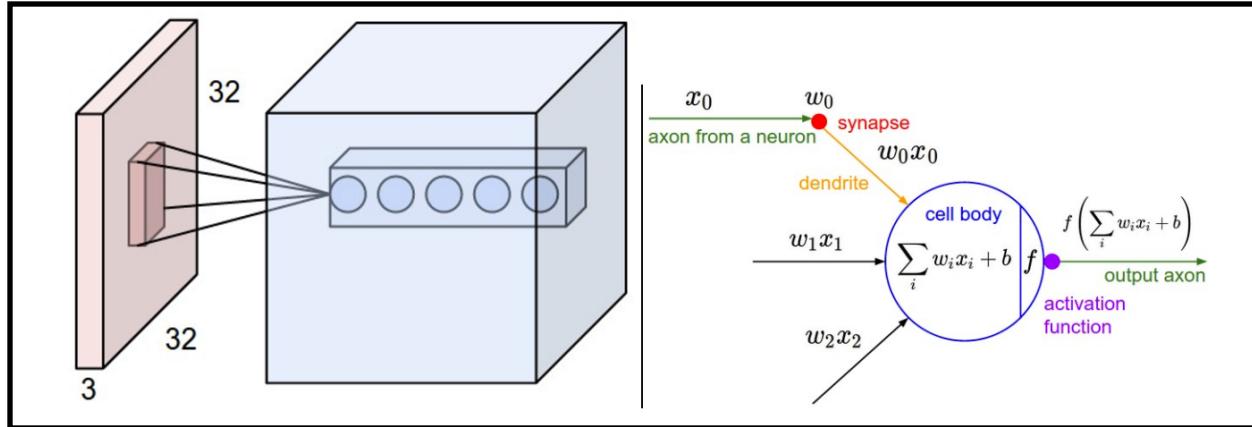


Figure 4. Diagram illustrates convolutional layers. Adapted from [9]

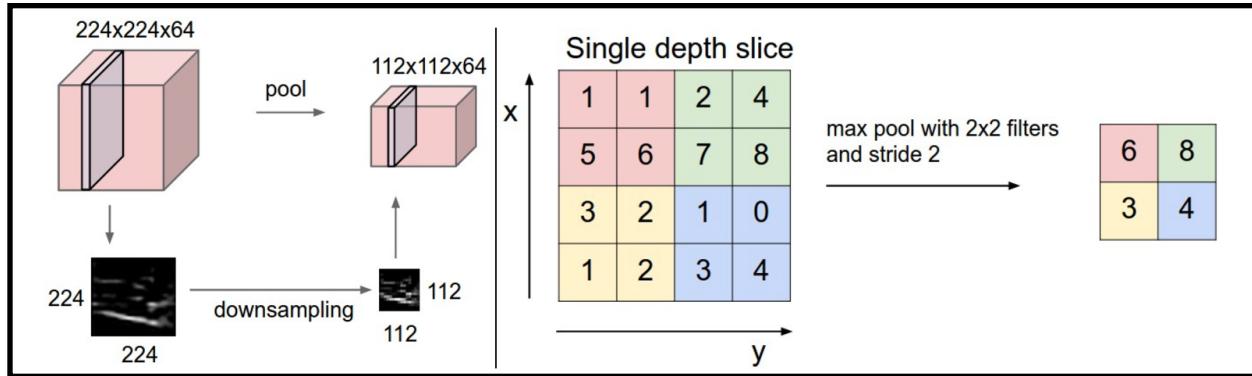


Figure 5. Diagram illustrates max pooling layers. Adapted from [9]

Different from convolutional layers, the neurons in fully-connected layers are fully connected pair-wisely to all the elements in the previous input layer and the next layer (Figure 6, adapted from [10]). Just like the convolutional layers, activation functions are applied after matrix multiplication to introduce non-linearity. With ReLu, sigmoid and tanh functions as popular activation function choices, practical experiences indicating that ReLu function performs well with cheap computation costs. Often the outputs of the last fully connected layer are fed into a classifier to transform the values into the range of 0~1, which is interpreted as the probability of the certain sample being as the certain class label. Sigmoid and soft-max have been two commonly used functions in the last classification step, although sigmoid has gradually fallen out of favor in recent years and soft-max is becoming the default choice in this step.

Besides the fundamental building blocks discussed above, a technique called Dropout[11] is often applied to fully-connected layers for regularization. With a specified keep probability as a

hyperparameter to tune, each neuron has (1-keep probability) that it'd be turned off during the training phase (Figure 7, adapted from [11] Figure 1). While during the testing phase, the results of all “thinned” networks would be averaged to come up with the final classification. The dropout technique is proven to be computationally cheap and effective in practice. It prevents deep neural networks from overfitting and can be thought of as a kind of ensemble techniques.

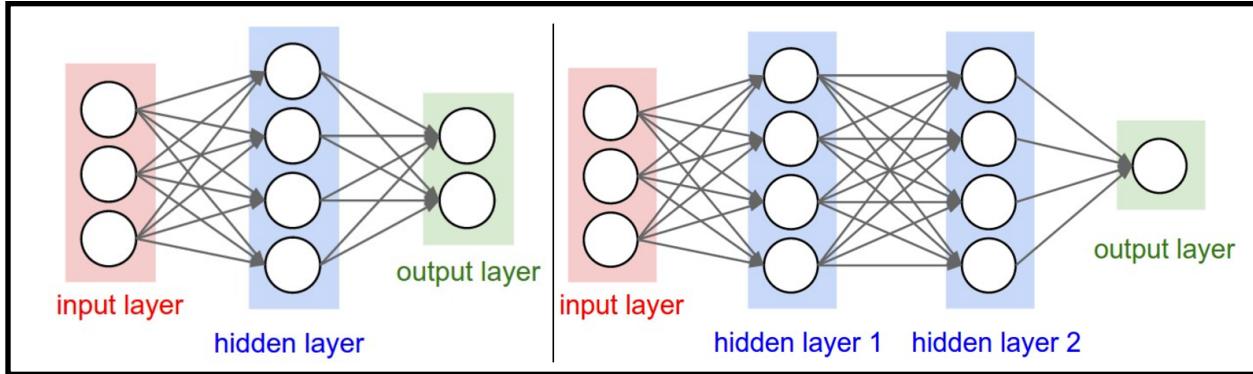


Figure 6. Diagram illustrates fully-connected layers. Adapted from [10]

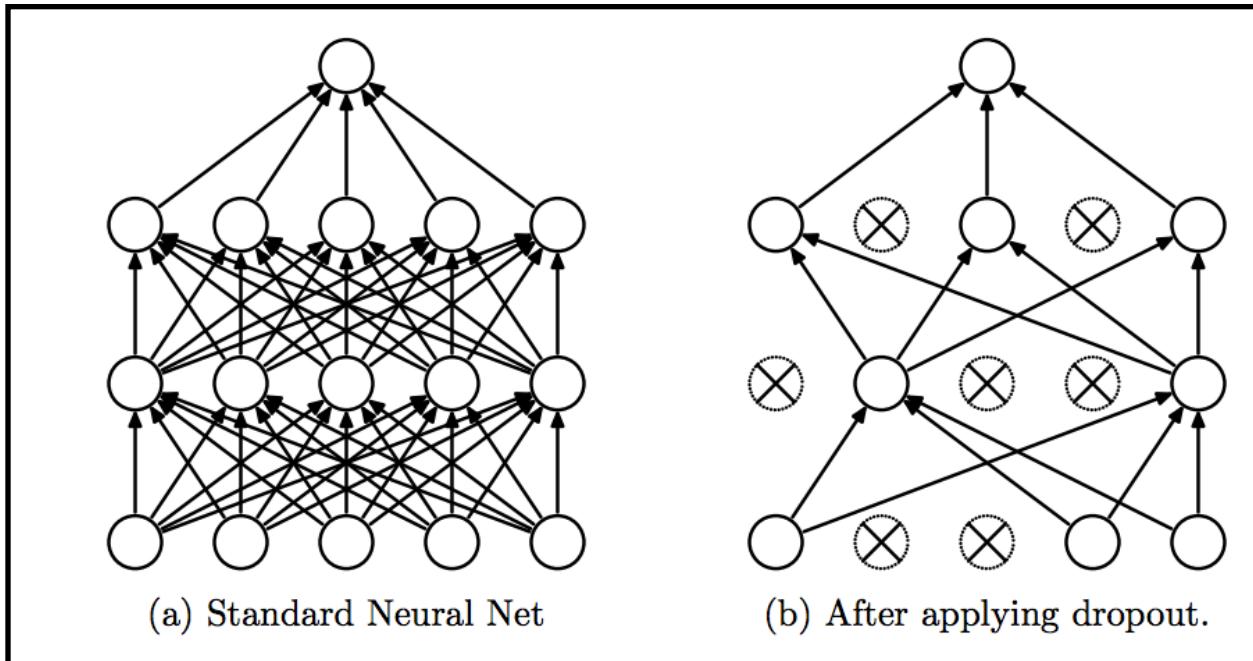


Figure 7. Left: without dropout. Right: with dropout. Adapted from [11]

2.4 Benchmark

As the CIFAR-10 dataset is widely used in a lot of deep learning researches, the accuracy performances on test set are reported[12] to range from 75.86% to the state of art performance of 96.53%[13].

3. Methodology

3.1 Data Preprocessing

As past research indicates that whitening input data and data augmentation are beneficial for training neural networks, several data preprocessing techniques are explored and among which a few are applied to this project.

(a) Data augmentation technique: random flipping. For each sample in the training set, there is 50% of the chance that it augmented by flipping upside down and 50% of the chance it is augmented through mirror flipping (Figure 8). Hence the training set size is doubled to 80,000 before training.

(b) Data augmentation technique: Principle Component Analysis (PCA) based color jittering[2]. As the technique proposed and used in AlexNet[2], it is explored in this project (Figure 9). However due to lack of fast implementation, eventually this data augmentation technique is not applied to the full dataset.

(c) All the data (training set, validation set, testing set) has been centered around zero means. No scaling is performed on each element since the scales of all values are in the same range (0~255)

(d) One-hot encoding is applied to convert the numerical labels into a vector format.

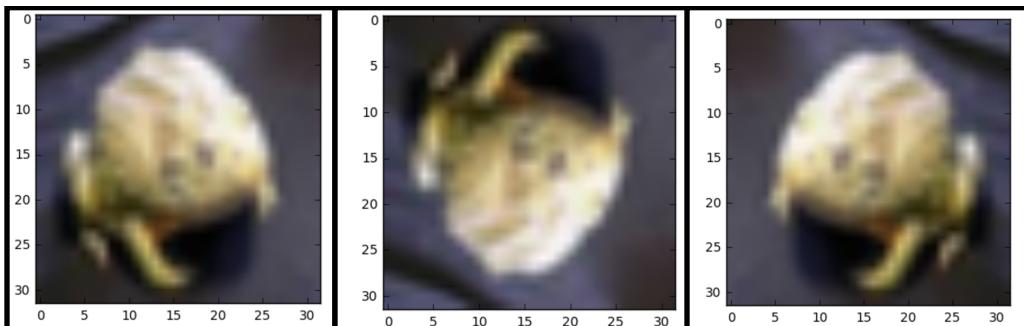


Figure 8. Data Augment: random flipping. Left: Original sample. Middle: 50% probability of up side down flip. Right: 50% probability of mirror flip

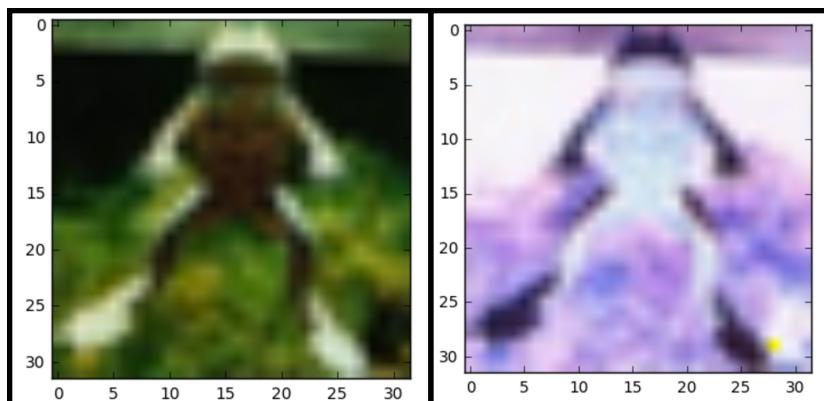


Figure 9. Data Augment: PCA color jitter. Left: Original sample. Right: PCA color jittered

3.2 Shallow CNN Architecture

A shallow CNN is constructed using the building blocks and techniques discussed in section 2.3. A diagram of the designed shallow CNN architecture is provided in Figure 10. Some of the selected parameter choices pertaining to the convolutional and pooling layers are provided in Table 2. ReLu activations are used after each convolutional layer (before max pooling layer) and after each fully-connected layer. The implementation of the designed CNN is carried out employing python TensorFlow framework (the explicit code can be found at [author's GitHub CNNplayground Repository](#)).

Table 2. Selected Hyper-parameter Settings for Convolutional Layers and Pooling Layers

	Filter/Window size	Stride	Padding	Number of Filters
Convolutional Layer 1	5 x 5	1	SAME	64 or 16
Convolutional Layer 2	3 x 3	1	SAME	64 or 16
Convolutional Layer 3	5 x 5	1	SAME	64 or 16
Max Pooling Layers	2 x 2	2	SAME	N/A

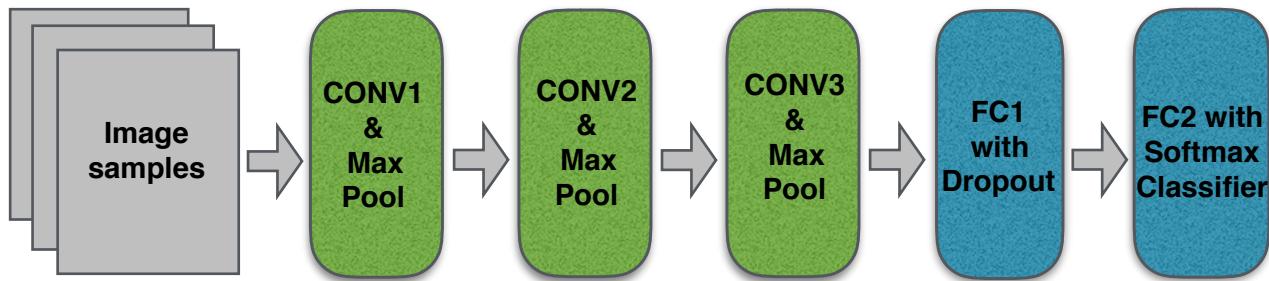


Figure 10. Shallow CNN architecture for this project

3.3 Hyperparameters Tuning and Refinement

The designed CNN architecture in Figure 10 is initially trained on random sets of small samples with a wide range of hyperparameter settings and validated on random sets of small samples. The small sets of instances are sampled from the whole training set using bootstrap sampling. The choice of hyperparameter settings is first tuned using a random search approach through the log space range of the hyper-parameters, then refined into a narrower range based on the performances measured during the random search through the hyper-parameter spaces. A few training experiments with hyper-parameters picked from the refined ranges/choices are then used to train the full dataset on an AWS 16GB RAM CUP server. The coarse, refined and the further refined hyperparameter ranges/choices are shown in Table 3. The results using slightly different hyper-parameter settings picked out from Table 2 and Table 3 are demonstrated, compared and analyzed in section 4.

Table 3. Random Search through Hyperparameter Spaces

Hyperparameters	Coarse ranges/choices	Refined ranges/choices	Further refined choices
weights initialization	(1) Xavier Initializer [14] (2) ReLu Xavier Initializer [15] (3) Random normal initializer with zero mean and standard deviation ranges from $10^{-5} \sim 10^{-1}$	Random normal initializer with zero mean and a standard deviation range of $0.01 \sim 0.02$	Random normal initializer with zero mean and a standard deviation of 0.015
biases initializer	(1) constant zeros (2) constant ones	constant ones	constant ones
dropout keep probability	0.1 ~ 0.8	0.3 ~ 0.5	0.5
initial learning rate	$10^{-6} \sim 10^{-1}$	0.0001 ~ 0.0007	0.0003 or 0.0004
optimizer	(1) Stochastic Gradient Descent (2) AdamOptimizer[16]	AdamOptimizer	AdamOptimizer

4. Results - Model Evaluation, Validation and Justification

The results achieved through employing the CNN architecture designed in Figure 10 with slightly different hyperparameters settings picked out from Table 2 and Table 3 are shown in Table 4 and Figure 11.

As shown in Table 4 and Figure 11, CNN model 2 only uses 16 filters per convolutional layer and achieves the lowest test set accuracy of 67.94% after 3000 training steps. This indicates that the learning power with a less complex CNN model is weaker as compared to a CNN model used a larger number of convolutional filters. On the other hand, with the filter size per convolutional layer (3 convolutional layers total) decreases by a factor of 4, the runtime for one training step decreases by a factor of 3.6.

Without data augmentation, CNN model 1 produces a test set accuracy of 73.00%, while with data augmentation, the test set accuracy trained by CNN model 3 increases 0.40% as compared to CNN model 1. It is also worth noticing that in Figure 11, the gap between training accuracy and validation accuracy in the top left subplot (CNN model 1) starts to appear after 600 training steps. However in the middle left subplot (CNN model 3) there is no noticeable gap between training and validation accuracy. This indicates that data augmentation has a mild effect in improving CNN model performance and by increasing the complexity and variance of training samples it delays overfitting.

Among the all five CNN models, model 5 produces the highest test set accuracy of 74.27% after 2500 training steps and model 4 produces the second highest test set accuracy of 73.51% after 1700 training steps. In the bottom left subplot (CNN Model 5) in Figure 11, a small gap between

the training and the validation line emerges after about 1500 training steps and grows gradually. In the contrast, in the middle right subplot (CNN Model 4) in Figure 11, with a smaller dropout keep probability, there is no apparent sign of overfitting after 1500 steps of training.

In summary, CNN model 5 achieves the highest accuracy performance in the test set and hence is selected as the final model. The achieved test accuracy of 74.27% is at the edge of the lower bound of the various deep learning research benchmark records[12]. Perhaps there are potentials for further improvement upon this performance (more details are discussed in the following conclusion section). In addition to that, the analysis of all five model performances indicate that:

- (a) Data augmentation helps increase the model performance and delays the emergence of overfitting;
- (b) Complex CNN models are computationally more expensive to train, however, it has bigger learning power than simple CNN models;
- (c) Bigger dropout rate helps regularize model training and prevents overfitting.

Table 4. Five CNN model Performances with their Hyper-Parameter Settings

CNN Model I.D.	CNN Model 1	CNN Model 2	CNN Model 3	CNN Model 4	CNN Model 5
No. of Filters per CONV	64	16	64	64	64
Data Augmentation	No	Yes	Yes	Yes	Yes
Initial Learning Rate	0.0004	0.0004	0.0004	0.0004	0.0004
Dropout Keep Probability	0.50	0.50	0.50	0.33	0.5
Weight Initializer std dev	0.015	0.015	0.015	0.015	0.015
Time cost per training step	72 secs	20 secs	77 secs	72 secs	72 secs
Total training steps	1200	3000	1200	1700	2500
Final Training Accuracy	0.8633	0.6719	0.7623	0.7695	0.8809
Final Validation Accuracy	0.7395	0.6809	0.7334	0.7421	0.7465
Test Accuracy	0.7300	0.6794	0.7340	0.7351	0.7427

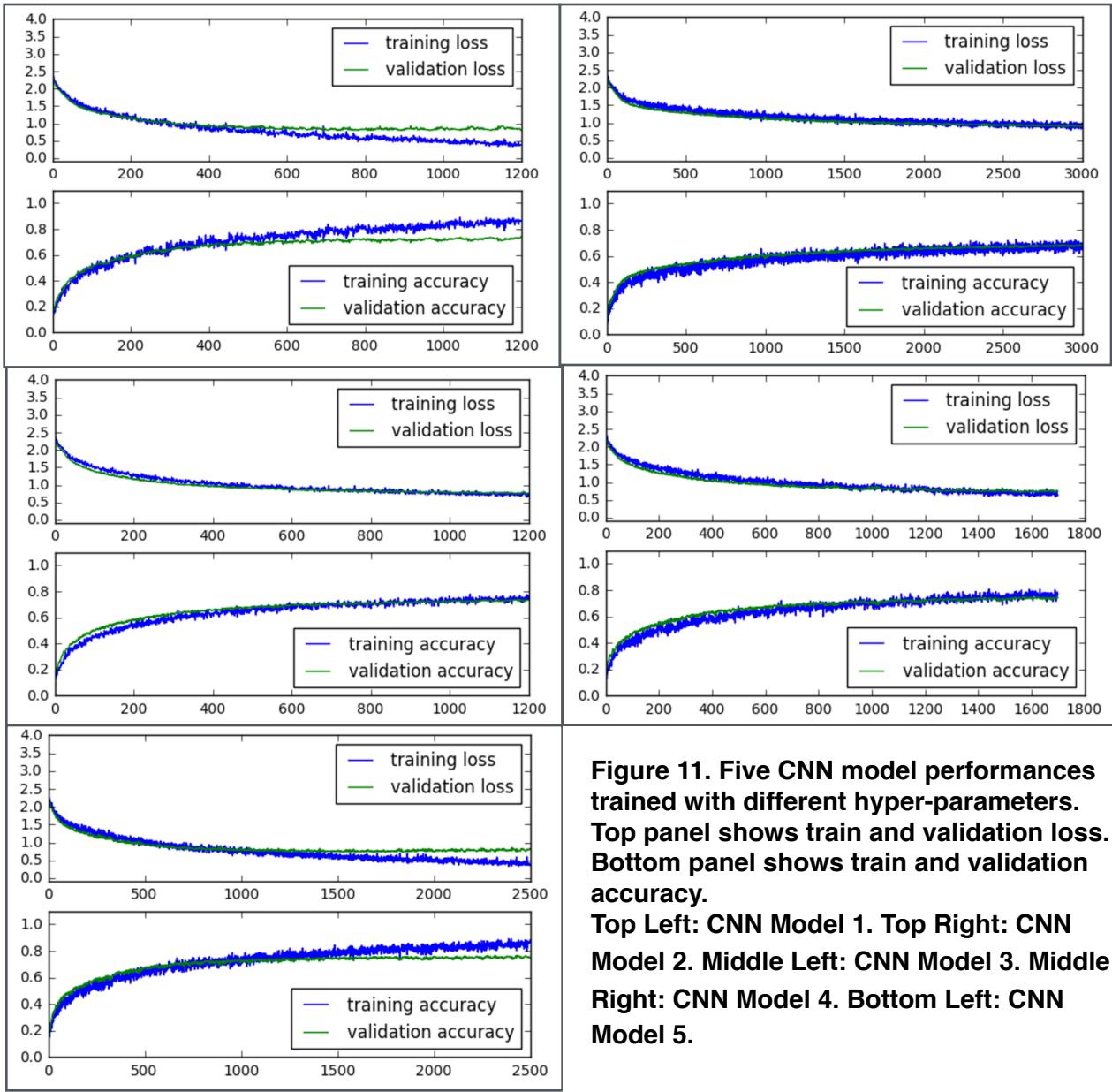


Figure 11. Five CNN model performances trained with different hyper-parameters.
Top panel shows train and validation loss.
Bottom panel shows train and validation accuracy.
Top Left: CNN Model 1. Top Right: CNN Model 2. Middle Left: CNN Model 3. Middle Right: CNN Model 4. Bottom Left: CNN Model 5.

5. Conclusion

In this project, a simple vanilla shallow CNN network is designed and trained using the CIFAR-10 dataset. The final selected CNN model is able to achieve an accuracy score of 74.27% on the test set after 2500 steps of training. Even though this performance is comparable to some of the reported benchmark results, it is still fairly far away from the state of art performance of 96.53% accuracy score.

In Figure 11 the last subplot, the appearing gap between the training loss and the validation loss indicates that the CNN model 5 suffers from overfitting after 1500 steps of training. Therefore

perhaps addressing this existing issue in CNN model 5 by either decreasing the dropout keep probability or using L2 regularization is the first step for further improving the performance of the CNN model 5. Additionally, batch normalization[17] is another technique known for helping network converging faster, preventing overfitting and improving performance. Perhaps implementing batch normalization is one of the approaches which could potentially improve the performance of CNN model 5 in the next step. (As a matter of fact, the author has attempted to insert batch normalization layers into the designed CNN architecture in this project. However, unfortunately, there appears to be some bug in the batch normalization implementation which leads to strange training performances and will need more time to resolve and facilitate.)

6. Reference

- [1] <http://image-net.org/>
- [2] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
- [3] Lin, Y., Lv, F., Zhu, S., Yang, M., Cour, T., Yu, K., Cao, L. and Huang, T., 2011, June. Large-scale image classification: fast feature extraction and svm training. In Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on (pp. 1689-1696). IEEE.
- [4] <http://deeplearning.net/datasets/>
- [5] <http://www.cs.utoronto.ca/~kriz/cifar.html>
- [6] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).
- [7] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385.
- [8] Szegedy, C., Ioffe, S. and Vanhoucke, V., 2016. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261.
- [9] <http://cs231n.github.io/convolutional-networks/>
- [10] <http://cs231n.github.io/neural-networks-1/>
- [11] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), pp.1929-1958.
- [12] [Collected CIFAR-10 Benchmark Results](#)
- [13] Graham, B., 2014. Fractional max-pooling. arXiv preprint arXiv:1412.6071.
- [14] Glorot, X. and Bengio, Y., 2010, May. Understanding the difficulty of training deep feedforward neural networks. In Aistats (Vol. 9, pp. 249-256).
- [15] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1026-1034).

- [16] Kingma, D. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
- [17] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.