Theory and Practice of CNN Techniques

Machine Learning Engineer Nanodegree Capstone Project
December 31, 2016
Kumiko Kashii

1 Definition

1.1 Project Overview

This project falls under the categories of Image Classification and convolutional neural network (CNN or ConvNet). Advancement in Image Classification will be useful in many domains. One of the examples is in the medical field. Once trained CNNs become capable of detecting anomalies in X-rays, MRI images, etc., it will become a powerful aid to doctors in diagnosing diseases that may otherwise go unnoticed. Another application is in identifying species of animals and plants. By simply taking a picture of an animal or a plant and running it through a trained CNN, someday we'll be able to identify what species it belongs to without in-depth knowledge in the domain.

In this project, images to be classified consist of 6 types of animals and 4 types of vehicles. 50,000 images of these 10 classes will be used to train a supervised learning model so that when a new image, which also belongs to one of these 10 classes, goes through the model, the correct class will be predicted.

1.2 Problem Statement

The goal is to create a convolutional neural network that accurately classifies a 32 pixels by 32 pixels RGB image into one of the 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck).

1.3 Metrics

1) Accuracy on the test data will be used to measure the quality of each trained model. It is defined by (# of datapoints with correct label) / (# of datapoints = 10,000). Accuracy is a good metric for this particular dataset since the dataset is well balanced (See 2.1).

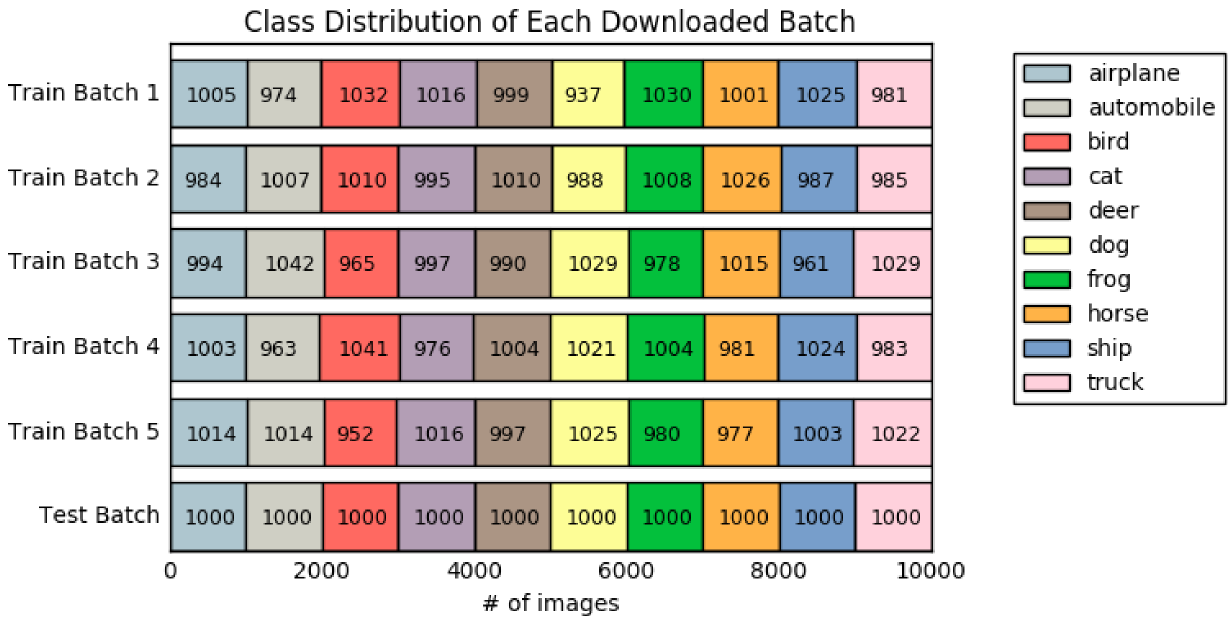
2) Training time for each model will be tracked in order to investigate tradeoffs between accuracy and efficiency.

2 Analysis

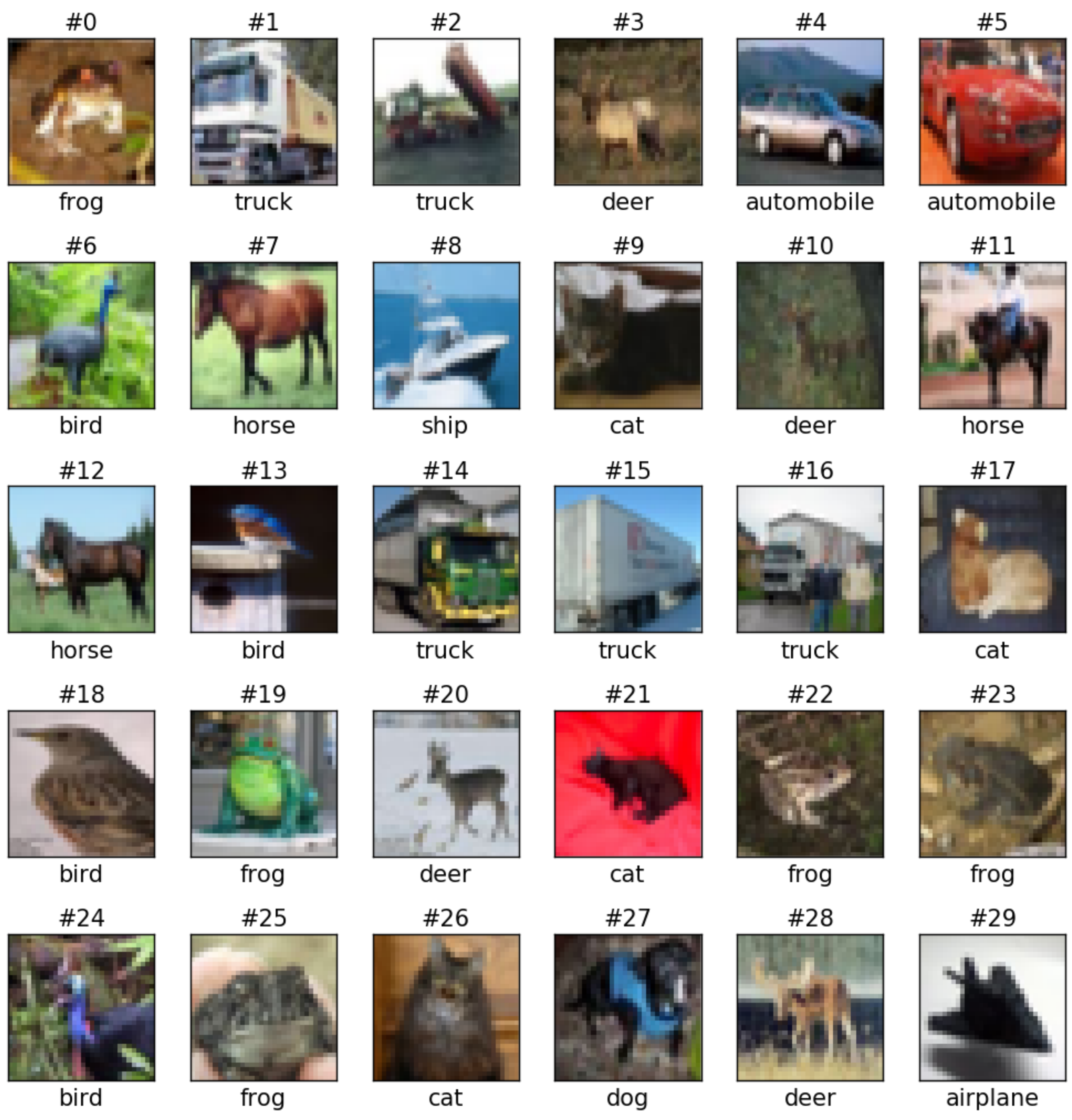
2.1 Data Exploration

The CIFAR-10 dataset will be used. It is downloadable from the official site: https://www.cs.toronto.edu/~kriz/cifar.html. It consists of 50,000 training images and labels, and 10,000 test images and labels. Each image is 32 pixels by 32 pixels in the RGB color model. The ten classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The dataset comes in 6 batches, 5 of which are training batches and 1 of which is a test batch. As shown below, each of the training batches is not exactly well balanced, with a slight offset from 1,000 images per class. This does not have a negative impact because all 5 batches will be combined into a well-balanced training set, with 5,000 images per class. Similarly, the test set is well balanced, having an equal distribution of images from each class i.e. 1,000 images per class.



The first 30 images from the training set and their classes:



2.2 Algorithms and Techniques

< Batch Normalization >

This is a model stabilization technique to normalize an input before running it through a convolutional layer or matrix multiplication layer.

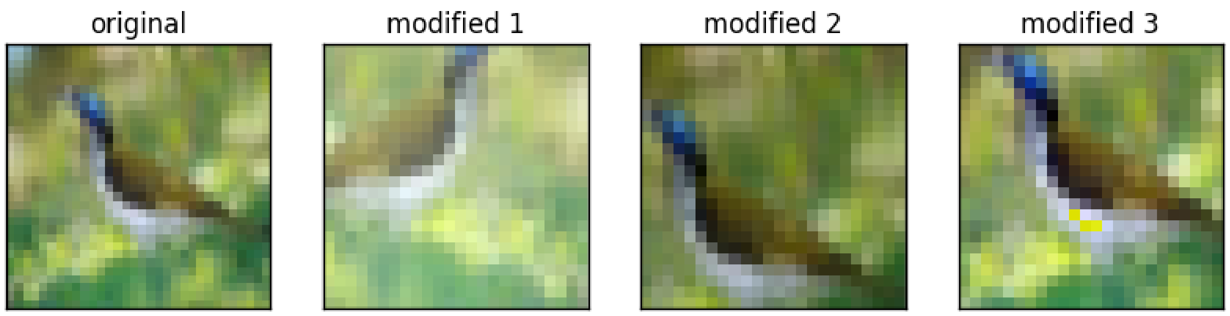
In the below example, let x be the input to the model, and all of the variables Weights_1, Biases_1, Weights_2, and Biases_2 just got updated at the end of step i. Due to the updates made to Weights_1 and Biases_1, the mean and variance of y can be significantly different between step i and i+1. However, Weights_2 and Biases_2 just got updated to work well with an input similar to y from step i i.e. there's an undesirable gap between expected y and actual y. To close this gap, y can be normalized to have the same mean and variance in every step.

```
x -> 1st Convolutional Layer w/ Weights_1 and Biases_1 -> ReLU -> y
y -> 2nd Convolutional Layer w/ Weights_2 and Biases_2 -> ReLU -> z
```

More details are found here: https://arxiv.org/pdf/1502.03167.pdf

< Data Augmentation >

This is a regularization technique to increase the number of images a model gets trained on by adding modified versions of training images to the original set. Common modifications are cropping/stretching, flipping, adjusting brightness and/or contrast, etc.



< Dropout >

This is a regularization technique to literally have some features "drop out" of each datapoint during training.

Given a dropout possibility of p, each feature has a possibility of (1 - p) to remain, and each of the remaining features is adjusted to (original value) / (1 - p). Below is an example of a datapoint after a dropout with 75% dropout possibility is applied.

```
[[1, 1, 1, 1],                                     [[4, 0, 0, 0]
 [1, 1, 1, 1],   -> 75% dropout possibility ->      [0, 0, 0, 0]
 [1, 1, 1, 1]]                                      [4, 0, 0, 4]]
```

More details are found here: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

< Local Response Normalization >

This is an adjustment technique to update each value of an input by factoring in the contrast between itself and its neighbors across the feature maps, defined as follows.

Given a position of x on the ith feature map, let y_j denote the value at the same position on the jth feature map. Let w be the window size. Then,

$$\text{lrn}(x) = x \;/\; \{\text{bias} + \text{alpha} * [\sum_{j=i-w/2}^{i-1} (y_j\verb|^|2) + x\verb|^|2 + \sum_{j=i+1}^{i+w/2} (y_j\verb|^|2)]\} \verb|^| \text{beta}$$

In the below example, observe how the originally equal values of 5 on the first feature map get updated based on the values directly below themselves.

With bias=1.0, alpha=1.0, beta=0.5,

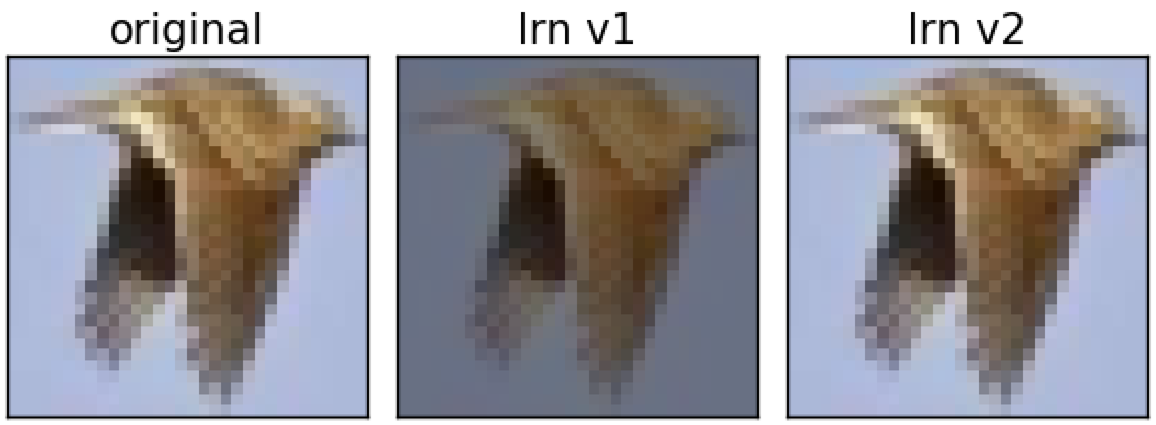
	Position 1	Position 2	Position 3
Feature map 1	5 -> 0.445	5 -> 0.700	5 -> 0.980
Feature map 2	10 -> 0.890	5 -> 0.700	0 -> 0

Adjusting the parameters can produce varying effects. In the below example, the updated values are much closer to the original values compared to the above example.

With bias=1.0, alpha=0.001 / 9.0, beta=0.75,

	Position 1	Position 2	Position 3
Feature map 1	5 -> 4.948	5 -> 4.979	5 -> 4.989
Feature map 2	10 -> 9.897	5 -> 4.979	0 -> 0

Below, the first set of parameters are used to turn the original image into the image in the middle. It has an overall darkening effect. The second set of parameters are used to turn the original image into the image on the right. There is no noticeable difference to human eyes.



More details are found in section 3.3 of here:
http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

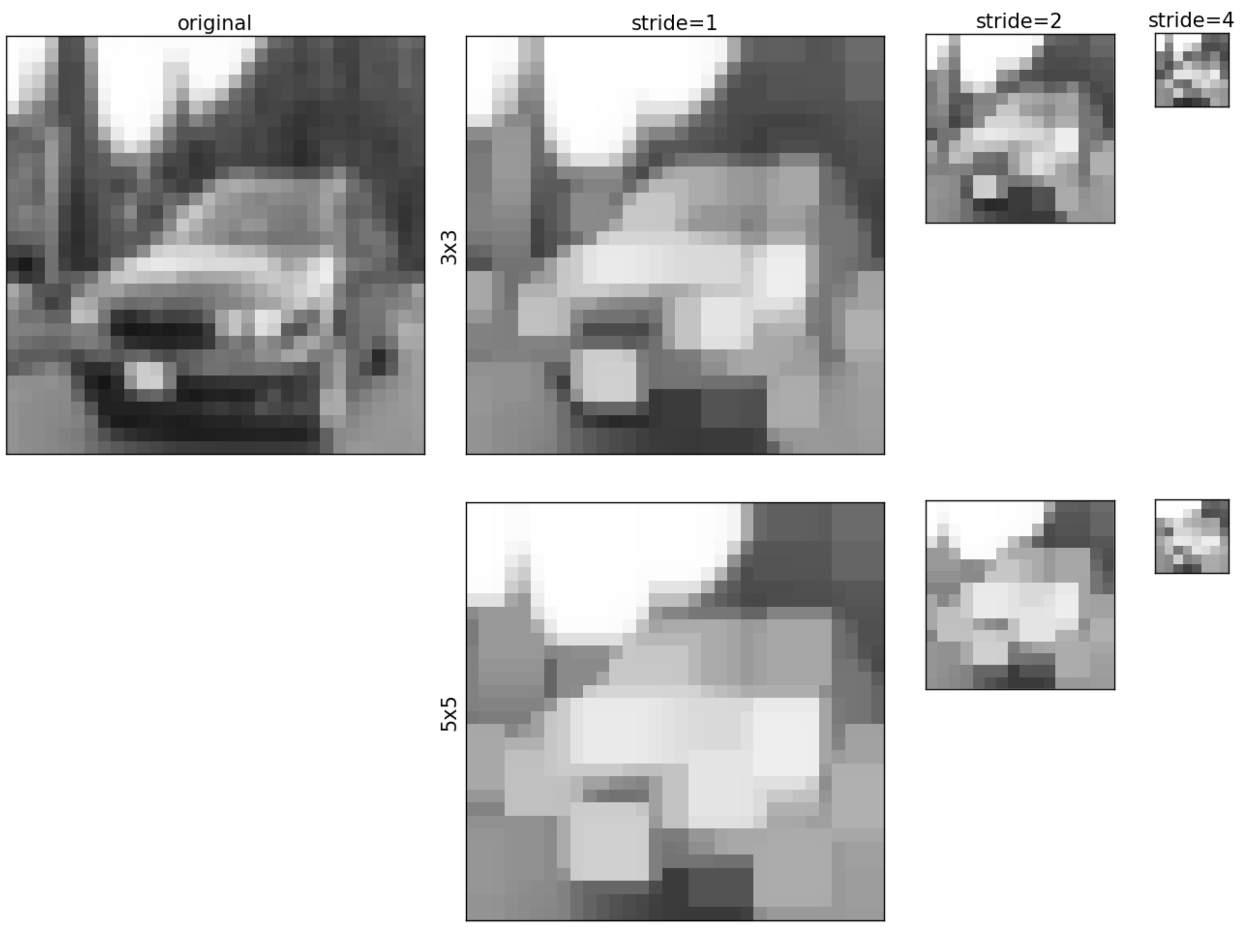
< Max Pooling >

This is a an image generalization technique to purposefully pixelate, or lose details of, an image.

Given a filter size, it outputs the maximum value from each section where the filter was applied. For example, applying a 3x3 filter to the below section outputs 13.

```
[  0 ] [  8 ] [  9 ]
[ 10 ] [ 10 ] [ 12 ]  ->  [ 13 ]
[ 12 ] [ 11 ] [ 13 ]
```

A stride size specifies how many units the filter gets shifted between pooling. If it is set to 1, the output data size is the same as the input data size. If it is set to 2, the width and the height of the output data are cut in half respectively.



More details are found here: http://www.ais.uni-bonn.de/papers/icann2010_maxpool.pdf

2.3 Benchmark

The official CIFAR-10 website mentions cuda-convnet for the benchmark results of “18% test error without data augmentation and 11% with”. With a fermi-generation GPU (GTX 4xx, GTX 5xx, or Tesla equivalent), the training time was 20 minutes without data augmentation and 75 minutes with. Unfortunately, the models used to produce these results are currently unviewable on the cuda-convnet website: https://code.google.com/p/cuda-convnet/.

3 Methodology

3.1 Data Preprocessing

1) The dataset comes in the data type of uint8 (unsigned 8-bit integer, 0 to 255). When using TensorFlow™, it is convenient to convert the dataset to float32. In addition, when using

matplotlib to display an RGB image with its values in a float data type, the values are assumed to be in the range of 0 to 1 i.e. all values need to be divided by 255.

2) The labels are integers from 0 to 9, each of which indicates a class. These labels need to be one-hot encoded such that the original label = 0 becomes a vector of [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], the original label = 1 becomes a vector of [0, 1, 0, 0, 0, 0, 0, 0, 0, 0], etc.

3.2 Implementation & Refinement

< Model 01 (Basic) >

It consists of three convolutional layers and three fully-connected layers. The code can be viewed in model_01_basic.ipynb.

#	Operation (learning rate = 0.01)	Output Size (h x w x d) or (# of features)
0.1	Input	32 x 32 x 3
1.1	3x3 convolution w/ 64 feature maps	32 x 32 x 64
1.2	relu	32 x 32 x 64
1.3	resize w/ stride=2	16 x 16 x 64
2.1	3x3 convolution w/ 64 feature maps	16 x 16 x 64
2.2	relu	16 x 16 x 64
2.3	resize w/ stride=2	8 x 8 x 64
3.1	3x3 convolution w/ 128 feature maps	8 x 8 x 128
3.2	relu	8 x 8 x 128
3.3	resize w/ stride=2	4 x 4 x 128
3.4	flatten	2048
4.1	hidden layer w/ 384 features	384
4.2	relu	384
5.1	hidden layer w/ 192 features	192
5.2	relu	192
6.1	final layer w/ 10 features	10

< Model 02 (Model 01 + Max Pooling) >

3x3 Max Pooling is applied at the end of each convolutional layer. The code can be viewed in model_02_mp.ipynb.

#	Operation (learning rate = 0.01)	Output Size (h x w x d) or (# of features)
0.1	Input	32 x 32 x 3
1.1	3x3 convolution w/ 64 feature maps	32 x 32 x 64
1.2	relu	32 x 32 x 64
1.3	3x3 max pooling w/ stride=2	16 x 16 x 64
2.1	3x3 convolution w/ 64 feature maps	16 x 16 x 64
2.2	relu	16 x 16 x 64
2.3	3x3 max pooling w/ stride=2	8 x 8 x 64
3.1	3x3 convolution w/ 128 feature maps	8 x 8 x 128
3.2	relu	8 x 8 x 128
3.3	3x3 max pooling w/ stride=2	4 x 4 x 128
3.4	flatten	2048
4.1	hidden layer w/ 384 features	384
4.2	relu	384
5.1	hidden layer w/ 192 features	192
5.2	relu	192
6.1	final layer w/ 10 features	10

< Model 03 (Model 02 + Local Response Normalization) >

Local Response Normalization is applied at the end of each convolutional layer. The code can be viewed in model_03_lrn.ipynb.

#	Operation (learning rate = 0.01)	Output Size (h x w x d) or (# of features)
0.1	Input	32 x 32 x 3
1.1	3x3 convolution w/ 64 feature maps	32 x 32 x 64
1.2	relu	32 x 32 x 64
1.3	3x3 max pooling w/ stride=2	16 x 16 x 64
1.4	local response normalization	16 x 16 x 64
2.1	3x3 convolution w/ 64 feature maps	16 x 16 x 64
2.2	relu	16 x 16 x 64
2.3	3x3 max pooling w/ stride=2	8 x 8 x 64
2.4	local response normalization	8 x 8 x 64
3.1	3x3 convolution w/ 128 feature maps	8 x 8 x 128

3.2	relu	8 x 8 x 128
3.3	3x3 max pooling w/ stride=2	4 x 4 x 128
3.4	local response normalization	4 x 4 x 128
3.5	flatten	2048
4.1	hidden layer w/ 384 features	384
4.2	relu	384
5.1	hidden layer w/ 192 features	192
5.2	relu	192
6.1	final layer w/ 10 features	10

< Model 04 (Model 03 + Batch Normalization) >

Batch normalization is applied at the beginning of each layer. The code can be viewed in model_04_bn.ipynb.

#	Operation (learning rate = 0.01, 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input	32 x 32 x 3
1.1	batch normalization	32 x 32 x 3
1.2	3x3 convolution w/ 64 feature maps	32 x 32 x 64
1.3	relu	32 x 32 x 64
1.4	3x3 max pooling w/ stride=2	16 x 16 x 64
1.5	local response normalization	16 x 16 x 64
2.1	batch normalization	16 x 16 x 64
2.2	3x3 convolution w/ 64 feature maps	16 x 16 x 64
2.3	relu	16 x 16 x 64
2.4	3x3 max pooling w/ stride=2	8 x 8 x 64
2.5	local response normalization	8 x 8 x 64
3.1	batch normalization	8 x 8 x 64
3.2	3x3 convolution w/ 128 feature maps	8 x 8 x 128
3.3	relu	8 x 8 x 128
3.4	3x3 max pooling w/ stride=2	4 x 4 x 128
3.5	local response normalization	4 x 4 x 128
3.6	flatten	2048
4.1	batch normalization	2048

4.2	hidden layer w/ 384 features	384
4.3	relu	384
5.1	batch normalization	384
5.2	hidden layer w/ 192 features	192
5.3	relu	192
6.1	batch normalization	192
6.2	final layer w/ 10 features	10

< Model 05 (Model 04 + Dropout) >

Dropout with 50% dropout possibility is applied at each fully-connected layer. The code can be viewed in model_05_dropout.ipynb.

#	Operation (learning rate = 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input	32 x 32 x 3
1.1	batch normalization	32 x 32 x 3
1.2	3x3 convolution w/ 64 feature maps	32 x 32 x 64
1.3	relu	32 x 32 x 64
1.4	3x3 max pooling w/ stride=2	16 x 16 x 64
1.5	local response normalization	16 x 16 x 64
2.1	batch normalization	16 x 16 x 64
2.2	3x3 convolution w/ 64 feature maps	16 x 16 x 64
2.3	relu	16 x 16 x 64
2.4	3x3 max pooling w/ stride=2	8 x 8 x 64
2.5	local response normalization	8 x 8 x 64
3.1	batch normalization	8 x 8 x 64
3.2	3x3 convolution w/ 128 feature maps	8 x 8 x 128
3.3	relu	8 x 8 x 128
3.4	3x3 max pooling w/ stride=2	4 x 4 x 128
3.5	local response normalization	4 x 4 x 128
3.6	flatten	2048
4.1	batch normalization	2048
4.2	dropout at 50%	2048
4.3	hidden layer w/ 384 features	384

4.4	relu	384
5.1	batch normalization	384
5.2	dropout at 50%	384
5.3	hidden layer w/ 192 features	192
5.4	relu	192
6.1	batch normalization	192
6.2	dropout at 50%	192
6.3	final layer w/ 10 features	10

< Model 06 (Model 05 + No Resize Data Augmentation) >

Each training image is randomly modified with a left-right flip, brightness adjustment, and contrast adjustment. Validation and test images are left as the original. The code can be viewed in model_06_no_resize_da.ipynb.

#	Operation (learning rate = 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input (randomly modified)	32 x 32 x 3
1.1	batch normalization	32 x 32 x 3
1.2	3x3 convolution w/ 64 feature maps	32 x 32 x 64
1.3	relu	32 x 32 x 64
1.4	3x3 max pooling w/ stride=2	16 x 16 x 64
1.5	local response normalization	16 x 16 x 64
2.1	batch normalization	16 x 16 x 64
2.2	3x3 convolution w/ 64 feature maps	16 x 16 x 64
2.3	relu	16 x 16 x 64
2.4	3x3 max pooling w/ stride=2	8 x 8 x 64
2.5	local response normalization	8 x 8 x 64
3.1	batch normalization	8 x 8 x 64
3.2	3x3 convolution w/ 128 feature maps	8 x 8 x 128
3.3	relu	8 x 8 x 128
3.4	3x3 max pooling w/ stride=2	4 x 4 x 128
3.5	local response normalization	4 x 4 x 128
3.6	flatten	2048
4.1	batch normalization	2048
4.2	dropout at 50%	2048

4.3	hidden layer w/ 384 features	384
4.4	relu	384
5.1	batch normalization	384
5.2	dropout at 50%	384
5.3	hidden layer w/ 192 features	192
5.4	relu	192
6.1	batch normalization	192
6.2	dropout at 50%	192
6.3	final layer w/ 10 features	10

< Model 07A (Model 06 + Crop) >

Each training image is randomly cropped to 24 x 24. Validation and test images are cropped to the same size at the center. Note that the size of each datapoint before flattening is now smaller than before at (3 x 3 x 128). The code can be viewed in model_07A_crop_da.ipynb.

#	Operation (learning rate = 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input (randomly modified)	24 x 24 x 3
1.1	batch normalization	24 x 24 x 3
1.2	3x3 convolution w/ 64 feature maps	24 x 24 x 64
1.3	relu	24 x 24 x 64
1.4	3x3 max pooling w/ stride=2	12 x 12 x 64
1.5	local response normalization	12 x 12 x 64
2.1	batch normalization	12 x 12 x 64
2.2	3x3 convolution w/ 64 feature maps	12 x 12 x 64
2.3	relu	12 x 12 x 64
2.4	3x3 max pooling w/ stride=2	6 x 6 x 64
2.5	local response normalization	6 x 6 x 64
3.1	batch normalization	6 x 6 x 64
3.2	3x3 convolution w/ 128 feature maps	6 x 6 x 128
3.3	relu	6 x 6 x 128
3.4	3x3 max pooling w/ stride=2	3 x 3 x 128
3.5	local response normalization	3 x 3 x 128
3.6	flatten	1152
4.1	batch normalization	1152

4.2	dropout at 50%	1152
4.3	hidden layer w/ 384 features	384
4.4	relu	384
5.1	batch normalization	384
5.2	dropout at 50%	384
5.3	hidden layer w/ 192 features	192
5.4	relu	192
6.1	batch normalization	192
6.2	dropout at 50%	192
6.3	final layer w/ 10 features	10

< Model 07B (Model 07A + Last Max Pooling Stride = 1) >

The resizing part of the last Max Pooling is removed. Now the size of each datapoint before flattening is (6 x 6 x 128). The code can be viewed in model_07B_crop_da.ipynb.

#	Operation (learning rate = 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input (randomly modified)	24 x 24 x 3
1.1	batch normalization	24 x 24 x 3
1.2	3x3 convolution w/ 64 feature maps	24 x 24 x 64
1.3	relu	24 x 24 x 64
1.4	3x3 max pooling w/ stride=2	12 x 12 x 64
1.5	local response normalization	12 x 12 x 64
2.1	batch normalization	12 x 12 x 64
2.2	3x3 convolution w/ 64 feature maps	12 x 12 x 64
2.3	relu	12 x 12 x 64
2.4	3x3 max pooling w/ stride=2	6 x 6 x 64
2.5	local response normalization	6 x 6 x 64
3.1	batch normalization	6 x 6 x 64
3.2	3x3 convolution w/ 128 feature maps	6 x 6 x 128
3.3	relu	6 x 6 x 128
3.4	3x3 max pooling w/ stride=1	6 x 6 x 128
3.5	local response normalization	6 x 6 x 128
3.6	flatten	4608

4.1	batch normalization	4608
4.2	dropout at 50%	4608
4.3	hidden layer w/ 384 features	384
4.4	relu	384
5.1	batch normalization	384
5.2	dropout at 50%	384
5.3	hidden layer w/ 192 features	192
5.4	relu	192
6.1	batch normalization	192
6.2	dropout at 50%	192
6.3	final layer w/ 10 features	10

< Model 08 (Model 06 + Enlarge + One More Convolutional Layer) >

Training, validation, and test images are enlarged to 64 x 64. A convolutional layer is added so that the size of each datapoint before flattening remains the same as before at (4 x 4 x 128). The code can be viewed in model_08_enlarge_da.ipynb.

#	Operation (learning rate = 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input (randomly modified)	64 x 64 x 3
1.1	batch normalization	64 x 64 x 3
1.2	3x3 convolution w/ 64 feature maps	64 x 64 x 64
1.3	relu	64 x 64 x 64
1.4	3x3 max pooling w/ stride=2	32 x 32 x 64
1.5	local response normalization	32 x 32 x 64
2.1	batch normalization	32 x 32 x 64
2.2	3x3 convolution w/ 64 feature maps	32 x 32 x 64
2.3	relu	32 x 32 x 64
2.4	3x3 max pooling w/ stride=2	16 x 16 x 64
2.5	local response normalization	16 x 16 x 64
3.1	batch normalization	16 x 16 x 64
3.2	3x3 convolution w/ 128 feature maps	16 x 16 x 128
3.3	relu	16 x 16 x 128
3.4	3x3 max pooling w/ stride=2	8 x 8 x 128
3.5	local response normalization	8 x 8 x 128

4.1	batch normalization	8 x 8 x 128
4.2	3x3 convolution w/ 128 feature maps	8 x 8 x 128
4.3	relu	8 x 8 x 128
4.4	3x3 max pooling w/ stride=2	4 x 4 x 128
4.5	local response normalization	4 x 4 x 128
4.6	flatten	2048
5.1	batch normalization	2048
5.2	dropout at 50%	2048
5.3	hidden layer w/ 384 features	384
5.4	relu	384
6.1	batch normalization	384
6.2	dropout at 50%	384
6.3	hidden layer w/ 192 features	192
6.4	relu	192
7.1	batch normalization	192
7.2	dropout at 50%	192
7.3	final layer w/ 10 features	10

< Model 09 (Model 08 + Crop + Last Max Pooling Stride = 1) >

Each training image is randomly cropped to 48 x 48. Validation and test images are cropped to the same size at the center. The resizing part of the last Max Pooling is removed. Now the size of each datapoint before flattening is (6 x 6 x 128). The code can be viewed in model_09_enlarge_crop_da.ipynb.

#	Operation (learning rate = 0.1)	Output Size (h x w x d) or (# of features)
0.1	Input (randomly modified)	48 x 48 x 3
1.1	batch normalization	48 x 48 x 3
1.2	3x3 convolution w/ 64 feature maps	48 x 48 x 64
1.3	relu	48 x 48 x 64
1.4	3x3 max pooling w/ stride=2	24 x 24 x 64
1.5	local response normalization	24 x 24 x 64
2.1	batch normalization	24 x 24 x 64
2.2	3x3 convolution w/ 64 feature maps	24 x 24 x 64
2.3	relu	24 x 24 x 64

2.4	3x3 max pooling w/ stride=2	12 x 12 x 64
2.5	local response normalization	12 x 12 x 64
3.1	batch normalization	12 x 12 x 64
3.2	3x3 convolution w/ 128 feature maps	12 x 12 x 128
3.3	relu	12 x 12 x 128
3.4	3x3 max pooling w/ stride=2	6 x 6 x 128
3.5	local response normalization	6 x 6 x 128
4.1	batch normalization	6 x 6 x 128
4.2	3x3 convolution w/ 128 feature maps	6 x 6 x 128
4.3	relu	6 x 6 x 128
4.4	3x3 max pooling w/ stride=1	6 x 6 x 128
4.5	local response normalization	6 x 6 x 128
4.6	flatten	4608
5.1	batch normalization	4608
5.2	dropout at 50%	4608
5.3	hidden layer w/ 384 features	384
5.4	relu	384
6.1	batch normalization	384
6.2	dropout at 50%	384
6.3	hidden layer w/ 192 features	192
6.4	relu	192
7.1	batch normalization	192
7.2	dropout at 50%	192
7.3	final layer w/ 10 features	10

4 Results

4.1 Model Evaluation and Validation

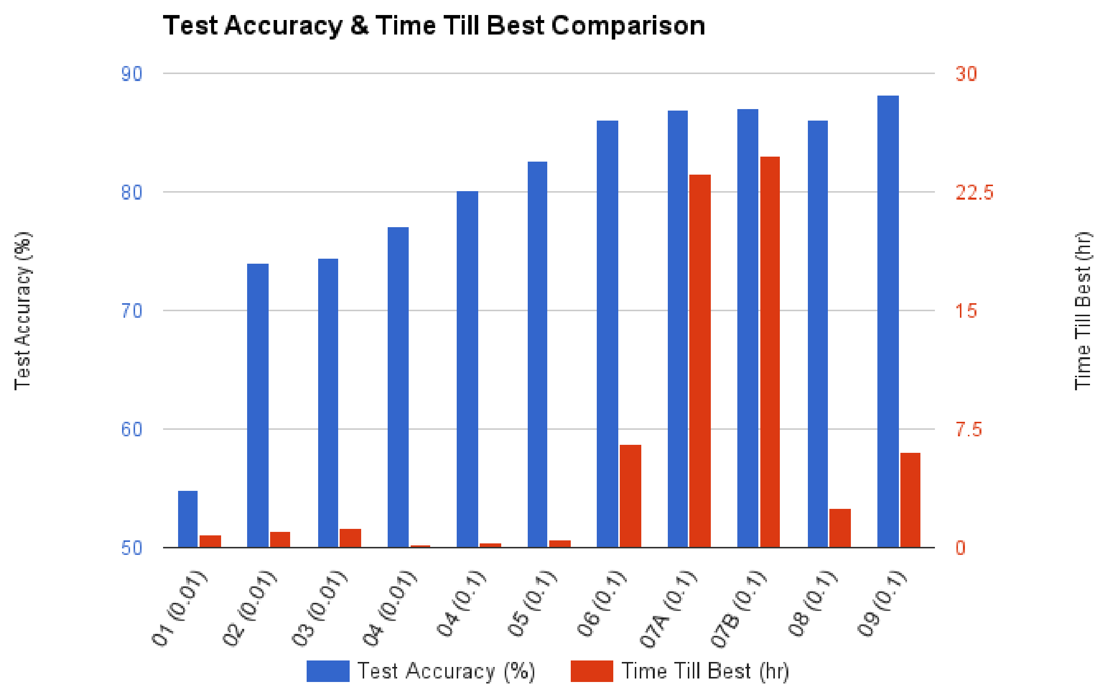
Each of the CNN techniques led to an improvement in the test accuracy when implemented incrementally. **Max Pooling** increased the test accuracy by almost 20%. **Local Response Normalization** increased the test accuracy but only by less than 0.5%. **Batch Normalization** was powerful not only to increase the test accuracy by 5.7% but also to cut the training time in quarter (See Graph #1). **Dropout** regularized the model and increased the test accuracy by almost 2.5%. At the same time, the amount of time and the number of steps to complete the training almost doubled. **Data Augmentation** without resizing was also powerful and increased the test accuracy by about 3.5%. Meanwhile, the training time jumped up to 6.6 hours from 0.5 hours. It was due to both the longer average time per step (1.5 times) and the more steps needed to complete the training (8 times). Among the four versions of resizing, the most effective was the one to first enlarge each image to 64x64, and then randomly crop it to 48x48. The resizing part of Data Augmentation increased the test accuracy by about 2%, and as a bonus, the training was completed in the nine tenths of the time.

Model	Model Description	Learning Rate	Test Accuracy (%)	Time Till Best (hr)	Steps Till Best	Average Time Per Step (sec)
01	Basic	0.01	54.82	0.8	47,400	0.058
02	01 + Max Pooling	0.01	74.07	1.0	64,500	0.058
03	02 + Local Response Normalization	0.01	74.51	1.2	67,700	0.062
04	03 + Batch Normalization	0.01	77.15	0.2	8,500	0.098
04	03 + Batch Normalization	0.10	80.20	0.3	14,600	0.077
05	04 + Dropout	0.10	82.62	0.5	28,500	0.069
06	05 + No Resize Data Augmentation	0.10	86.17	6.6	230,100	0.104
07A	06 + Crop	0.10	86.90	23.6	618,900	0.137
07B	07A + Last Max Pooling Stride = 1	0.10	87.05	24.8	692,600	0.129
08	06 + Enlarge + One More Convolutional Layer	0.10	86.15	2.5	43,200	0.204
09	08 + Crop + Last Max Pooling Stride = 1	0.10	88.16	6.0	111,200	0.193

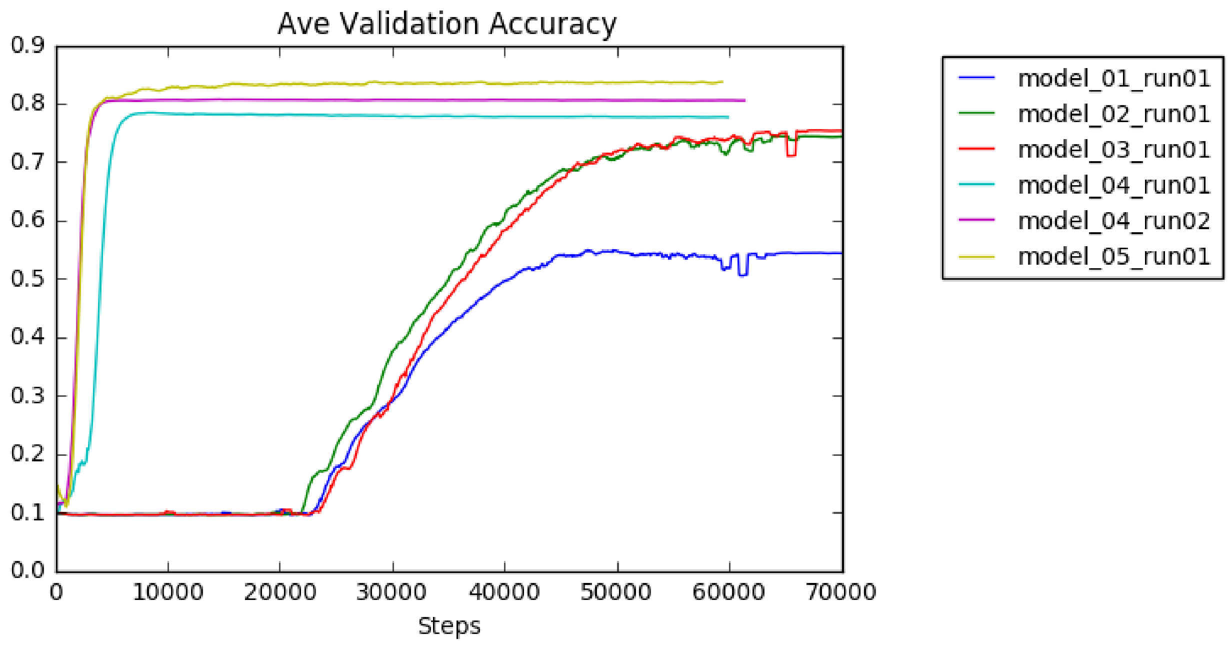
Notes:
1) During a training, a validation accuracy is recorded in a log every 100 steps.
2) Average validation accuracy is an average of the 8 most recent validation accuracies recorded in the log.
3) Test accuracy is determined by the best model i.e. the model with the highest average validation accuracy.
4) Training for Model 07A and 07B were terminated after about 24 hours even though the average validation accuracy was still increasing, but only slightly.
5) All models were trained and tested with a GPU, NVIDIA Titan X Pascal™.
6) All models were trained and tested with TensorFlow™.

As shown in the below Chart #1, the best model, 09, was not the one to take the most amount of time to complete the training. 07A and 07B stood out in a way that even after 24 hours of training, the models were still improving (See Graph #2). The slow learning for these two was probably because of the cropping of each image from 32x32 to 24x24, which reduced the amount of information to 56% of the original. Meanwhile, the test accuracy increased by about 0.9% (from 06 to 07B) due to the cropping working as a regularization technique. The same trend was observed with the models with the enlarged images (08 and 09). When each image was cropped from 64x64 to 48x48, it took about 2.4 times as long to complete the training and the test accuracy increased by almost 2%.

Chart #1

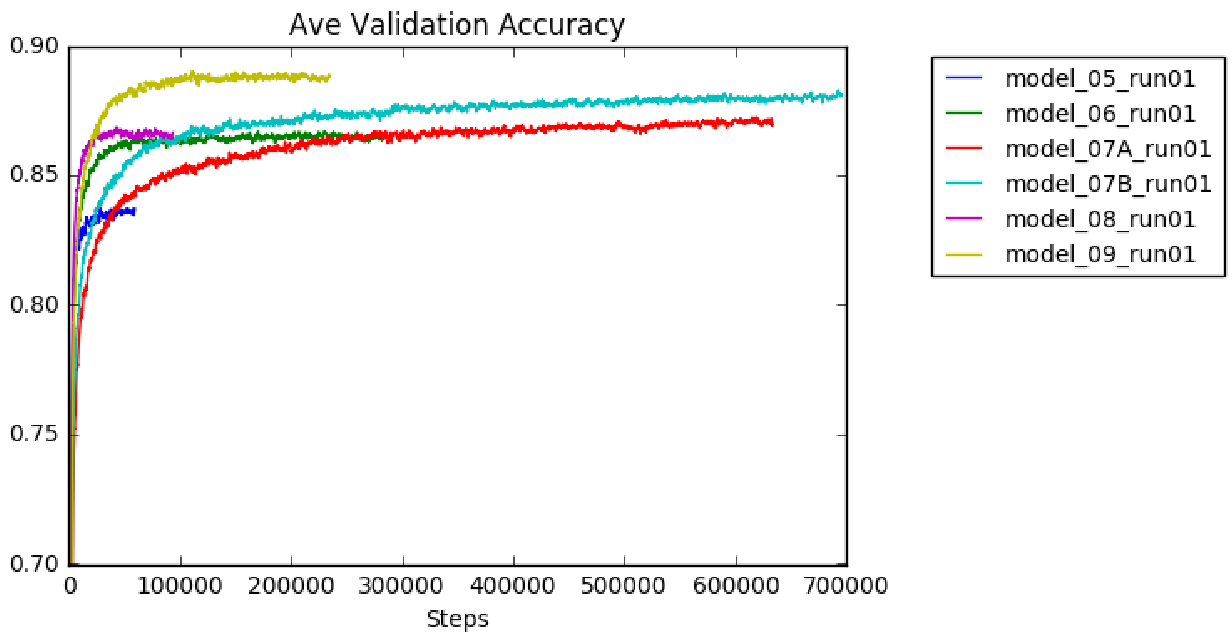


Graph #1



Notes:
model_04_run01 is with the learning rate = 0.01. model_04_run02 is with the learning rate = 0.1.

Graph #2



4.2 Justification

I almost hit the the benchmark of 89% test accuracy with data augmentation. My best model was 88%, so I can say it was a success. However, what puzzled me about the benchmark was the training time, 20 minutes without data augmentation and 75 minutes with. My models were not even close, 33 minutes without data augmentation and 358 minutes with. I wonder if they were using multiple GPUs or if not, something was fundamentally different about their model architecture.

5 Conclusion

5.1 Reflection

It was exciting to try out the various CNN techniques and see their effectiveness in the actual numbers. While learning about the intuition and algorithms behind each technique, I had no idea to what extent it would improve the model performance. So I was surprised that the first technique of Max Pooling increased the test accuracy by 20%. Local Response Normalization was the most challenging to understand theoretically because I was not expecting a CNN technique to be performed across feature maps, of course besides convolutions themselves. However, despite of it being a complicated calculation, Local Response Normalization was the least effective of all techniques with less than 1% increase in the test accuracy. The most impressive was Batch Normalization for completing the training in a quarter of the previous model's training time while reaching a higher test accuracy. Both theoretically and practically, it makes a perfect sense to use it before any convolution or matrix multiplication in a fully-connected layer. Dropout was another favorite of mine for its simplicity. I chose not to implement L2 regularization because of the time-consuming aspect of finding the optimal parameter value, but I'm glad Dropout worked well as a regularizer instead. Data Augmentation required trial and error with 4 versions. My personal twist at the time of implementation was to apply the random modification (crop, left-right flip, brightness adjustment, and contrast adjustment) at the beginning of each Stochastic Gradient Descent step so that no two training images would be exactly the same. An overall, general takeaway was that the best model did not take the longest to train. Like people with a great brain, it seems like some neural networks learn more efficiently than others, absorbing more information in less amount of time.

5.2 Improvement

Further experiments are needed to investigate how the test accuracy and training time will be affected by increasing the number of convolutional layers and feature maps. I'm also looking forward to trying out other CNN techniques such as 1x1 Convolutions and Residual Network, that reached over 95% test accuracy with the CIFAR-10 dataset.