REPORT

DL HOMEWORK2

Rashid ALI

Rashid Ali
12-4-2018

1- Using Convolutional Neural Network for Image Recognition

1.1 Network Architecture

Input->Block1[Conv1->Relu->Pool1]->Block2[Conv2->Relu->Pool2]->Block 3[Conv3->Relu->Pool3]->Block4->[FC1-
Relu]->Block5[FC2->Relu]->Softmax

Dimensions of Input, weight, bias and output for each Block:

Input data shape:

Shape of Train_x: (50000, 28, 28, 1)
Shape of Train_y: (50000, 10)

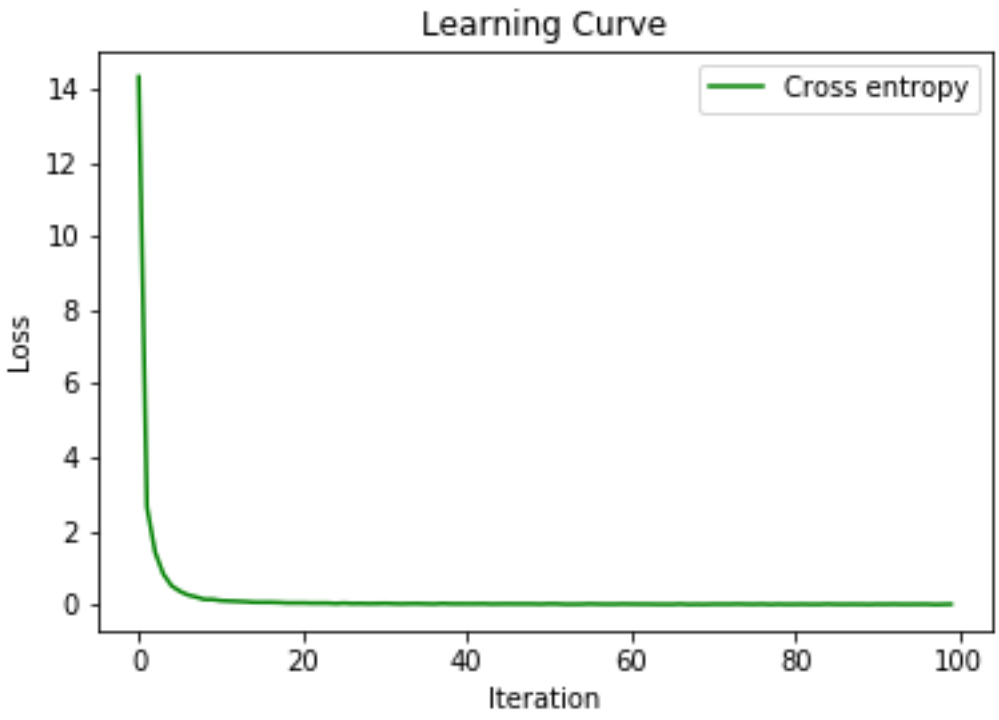
Block1: Input [.,28,28,1], W_conv1[5,5,1,32], B_conv1[32],
Output [.,14,14,32]
Block2: Input [.,14,14,32], W_conv2[5,5,32,32], B_conv2[32], Output [.,7,7,32]
Block3: Input [.,7,7,64], W_conv2[3,3,32,64], B_conv2[64], Output [.,4,4,64]
Block4: Input [.,4,4,64], W_fc1[4*4*64,500], B_fc1[500], Output [.,500]
Block5: Input [.,500], W_fc2[500,10], B_fc2[10], Output [.,10], Softmax->y_pred

Dimension of Max_Pool and Strides for each Layers: Max_Pool[1,2,2,1], Strides[1,2,2,1]
Type of Padding for each Layer: Same

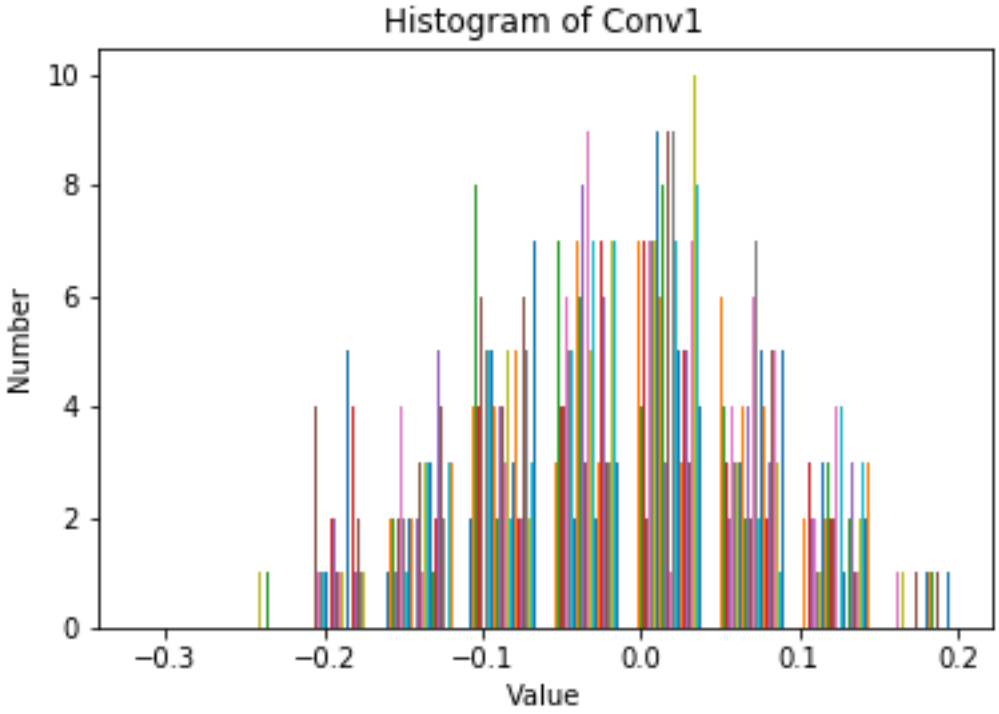
Effect of different stride and filter size: The experiments are done using stride of [1,2,2,1], [1,4,4,1] and [1,8,8,1]. The 4 pixels are skipped for stride of [1,4,4,1] and the 8 pixels are skipped for stride of [1,8,8,1] each time the window is shifted during the convolution operation. So, the stride of 4,4 and 8,8 waste a lot of information during the down sampling. Therefore, the model showed on 98% accuracy on training set but only 60% accuracy on test set for stride of [1,4,4,1] and [1,8,8,1]. The latest model uses stride of [1,2,2,1] for all layers. It performed really well with accuracy of 97% on test data.

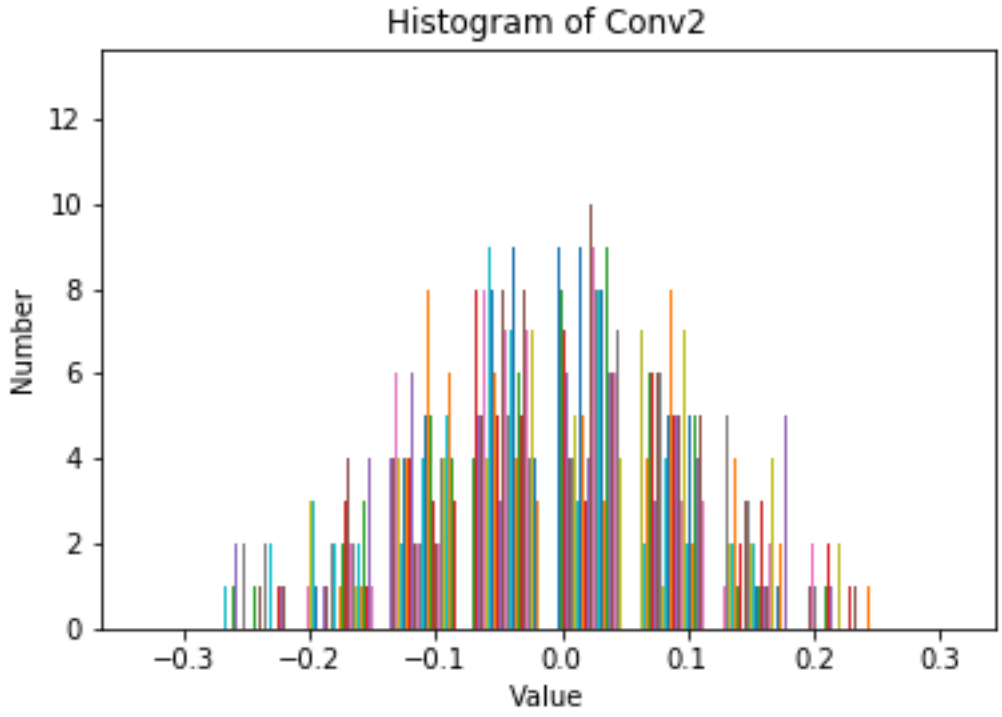
The first experiment is done using filter sizes: w_conv2 [3,3,32,64], w_conv3[3,3,64,128] and fc1[3*3*128,1024]. It was noticed that it produced large number of parameters which takes more time for training. The latest model uses 32 and 64 depths instead of 64 and 128 which significantly decreases the number of parameters.

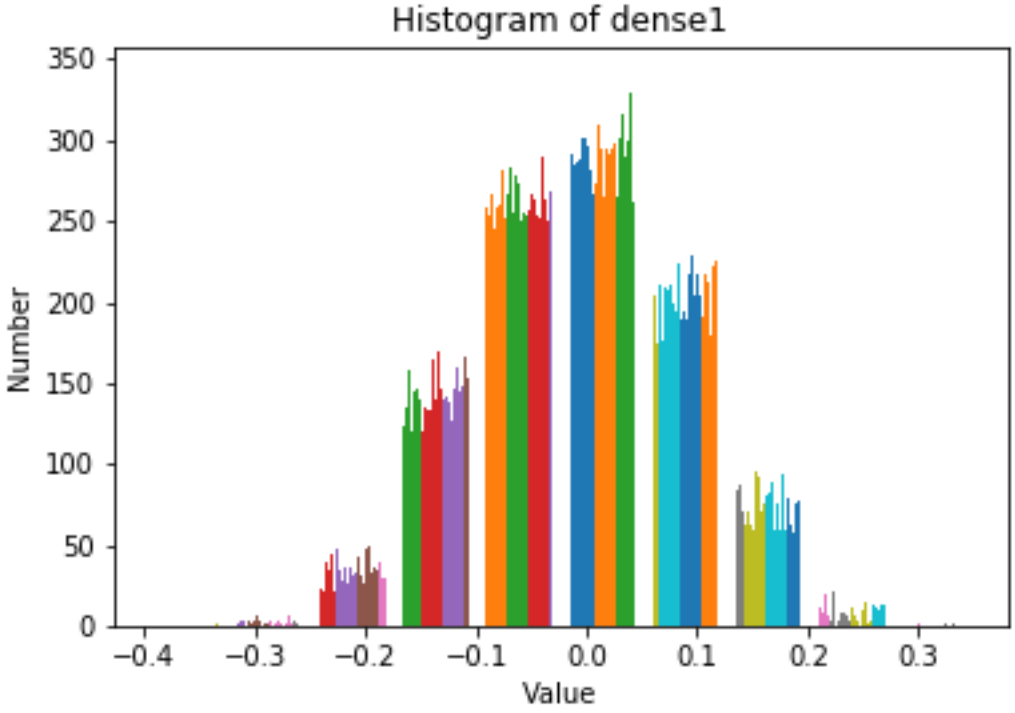
Learning rate = 0.0001
Epochs = 100
Minibatch size = 16
Cost of training data after epoch 95: 0.016345
Train Accuracy: 0.999059999999965
Validation Accuracy: 0.9814
Test Accuracy: 0.9835

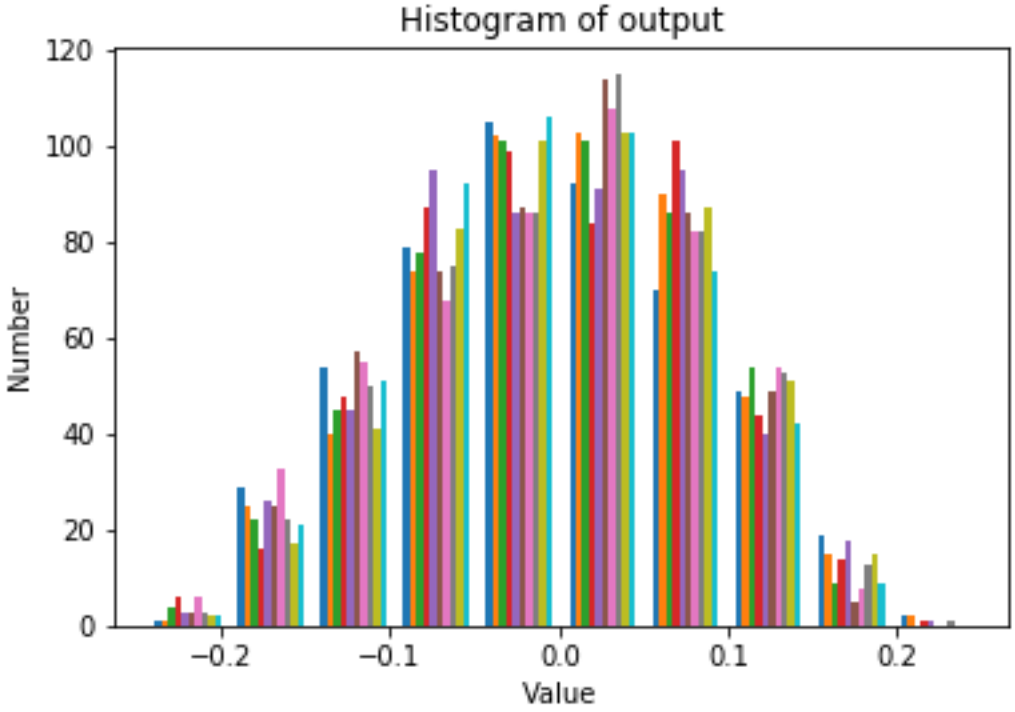



Training Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Accuracy Rate
0
20
40
60
80
100
Iteration
Training Accuracy
Validation Accuracy
Test Accuracy


Histogram of Conv1
10
8
6
4
2
0
Number
−0.3
−0.2
−0.1
0.0
0.1
0.2
Value


Histogram of Conv2
Number
Value
0
2
4
6
8
10
12
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

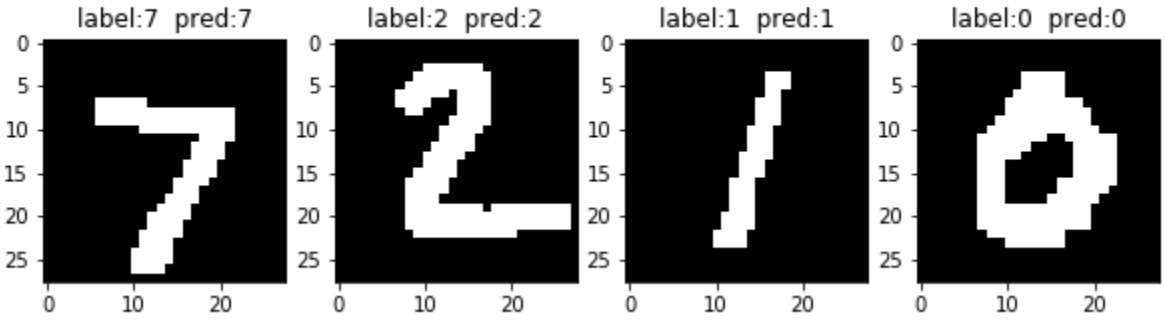

Histogram of dense1
Number
Value
0
50
100
150
200
250
300
350
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3



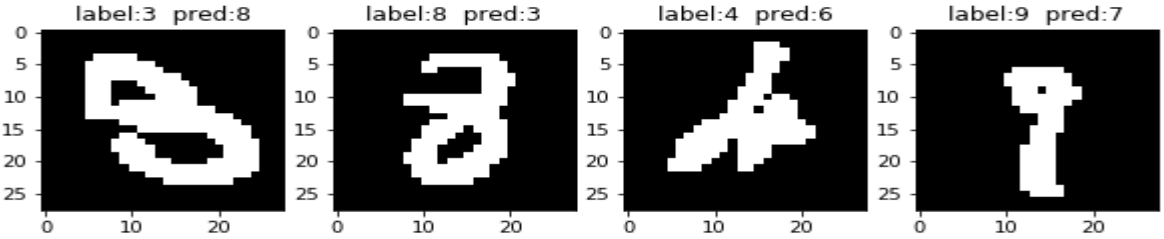
1.2 Correctly classified and miss-classified Images

False predictions: 165 out of 10000 on test data of size 10000
True predictions: 9835 out of 10000 on test data of size 10000

Correctly classified Images:



Miss-classified Images:



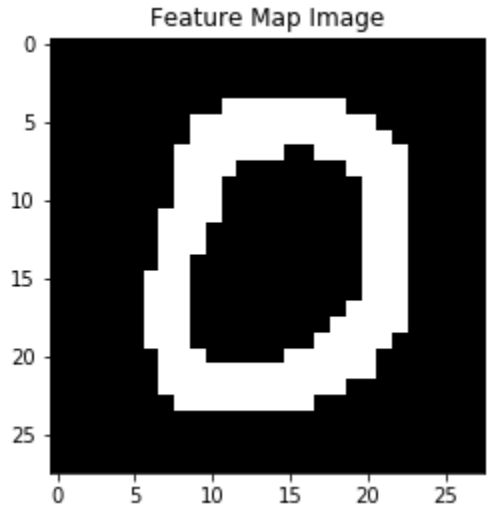
1-3 Feature Maps

The following figures displays 9 feature maps per convolutional layer. Conv1 actually contains 32 feature maps, since we have 32 filters in that layer. But we are only visualizing the first 9 features in conv1. In the same way, we visualize 9 feature maps in conv2 and conv3. There are some interesting observations about the feature maps as we progress through the layers. The feature maps for zero (0) and six (6) images are visualized.

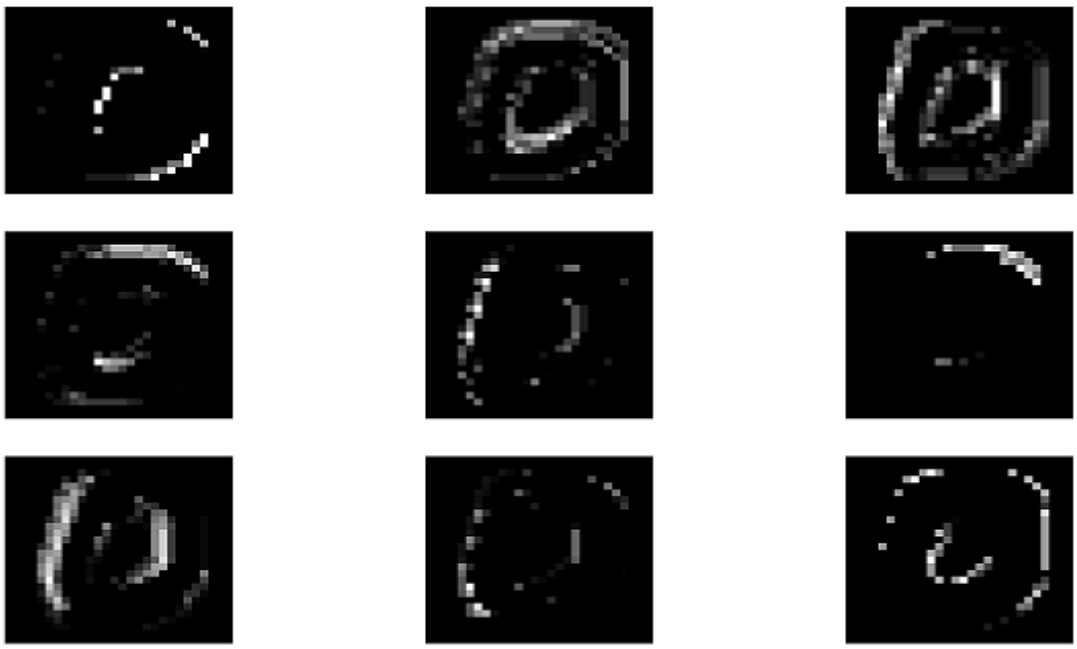
- The first layer conv1 feature maps retain the most of the information present in the image. In CNN architectures the first layers usually act as edge detectors.

- As we go deeper into the network the feature maps look less like the original image and more like an abstract representation of it. As you can see in conv2 the zero (0) and six (6) is somewhat visible, but after that it becomes unrecognizable. The reason is that deeper feature-maps encode high level concepts like structure of zero (0) and six (6) while lower level feature maps detect simple edges and shapes. That's why deeper feature maps contain less information about the image and more about the class of the image. They still encode useful features, but they are less visually interpretable by us.

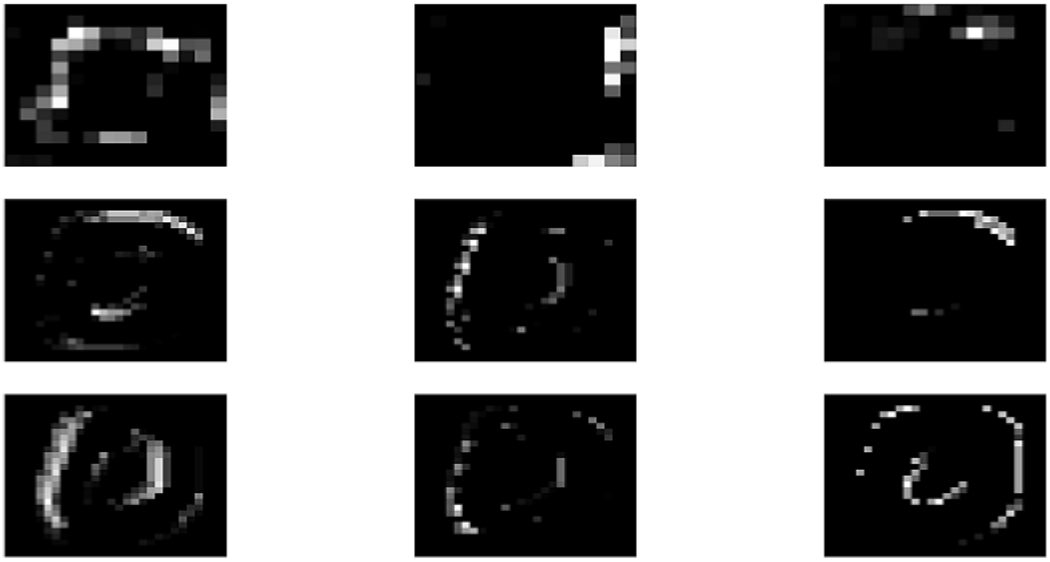
- The feature maps (conv3) become sparser as we go deeper, meaning the filters detect less features. It makes sense because the filters in the first layers detect simple shapes, and every image contains those. But as we go deeper we start looking for more complex stuff like the structure of each digit.



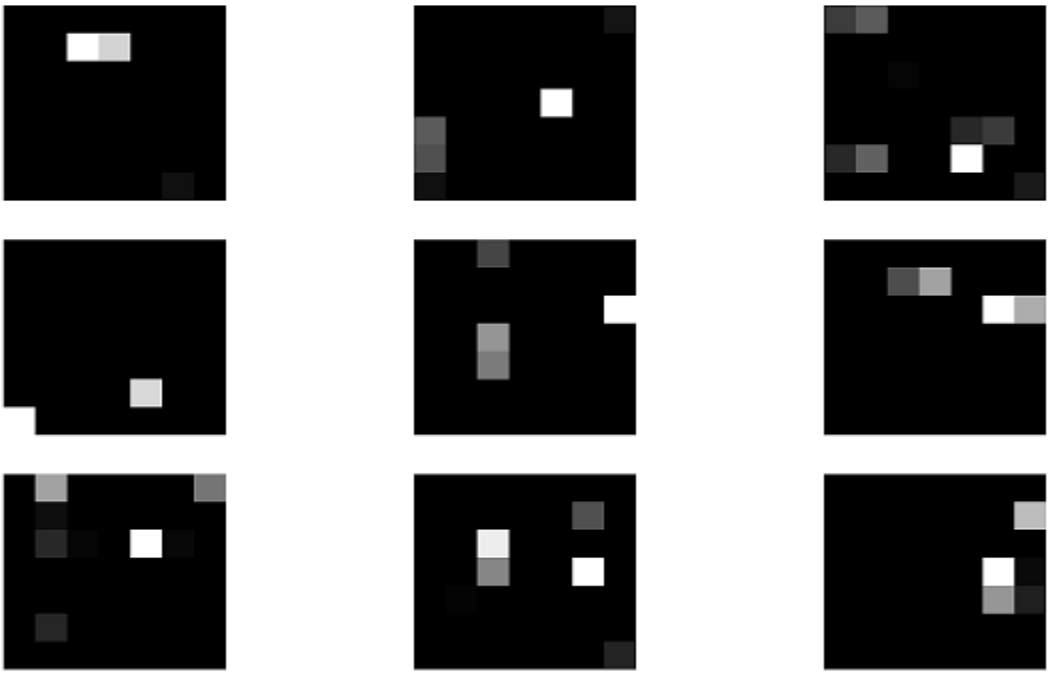
Conv1 Feature Map

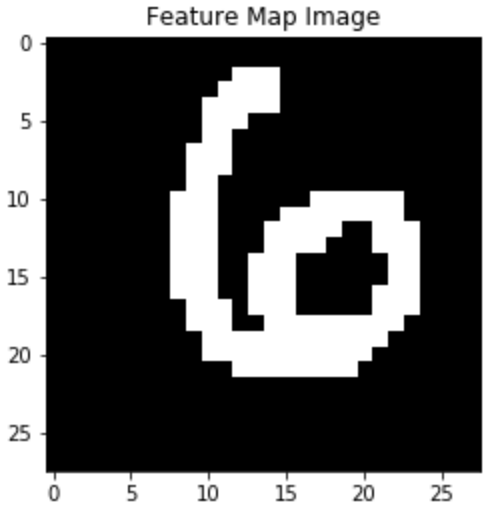


Conv2 Feature Map

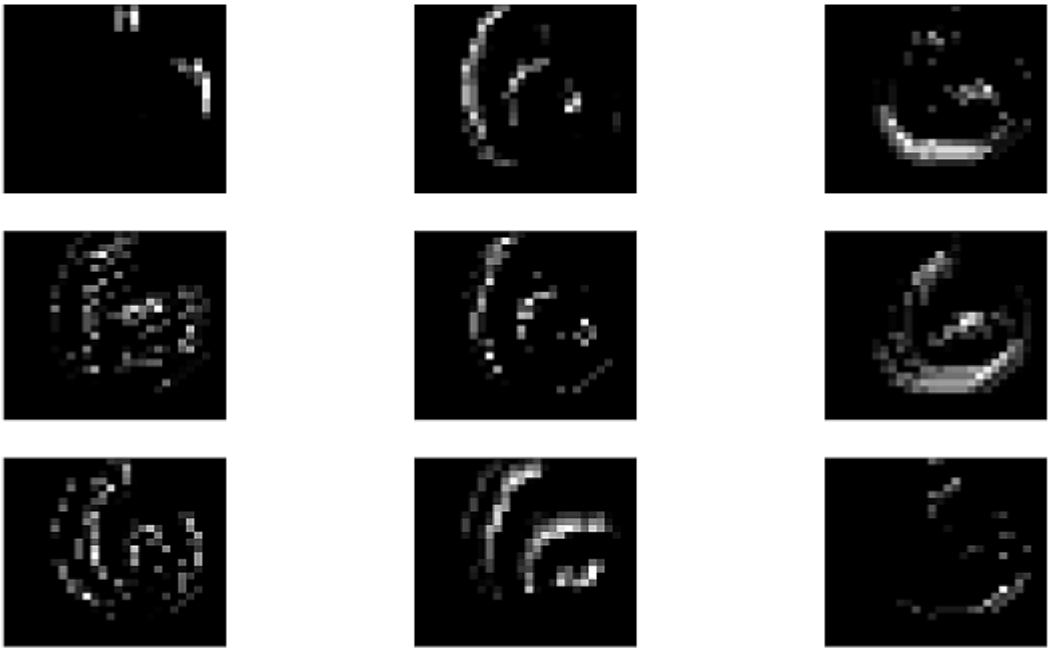


Conv3 Feature Map

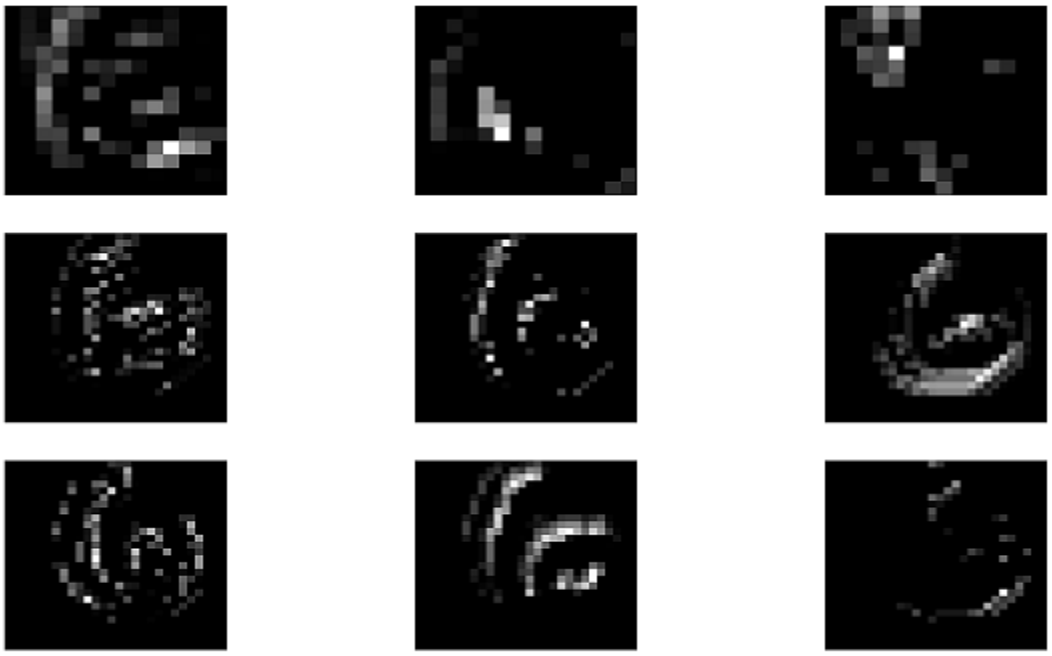




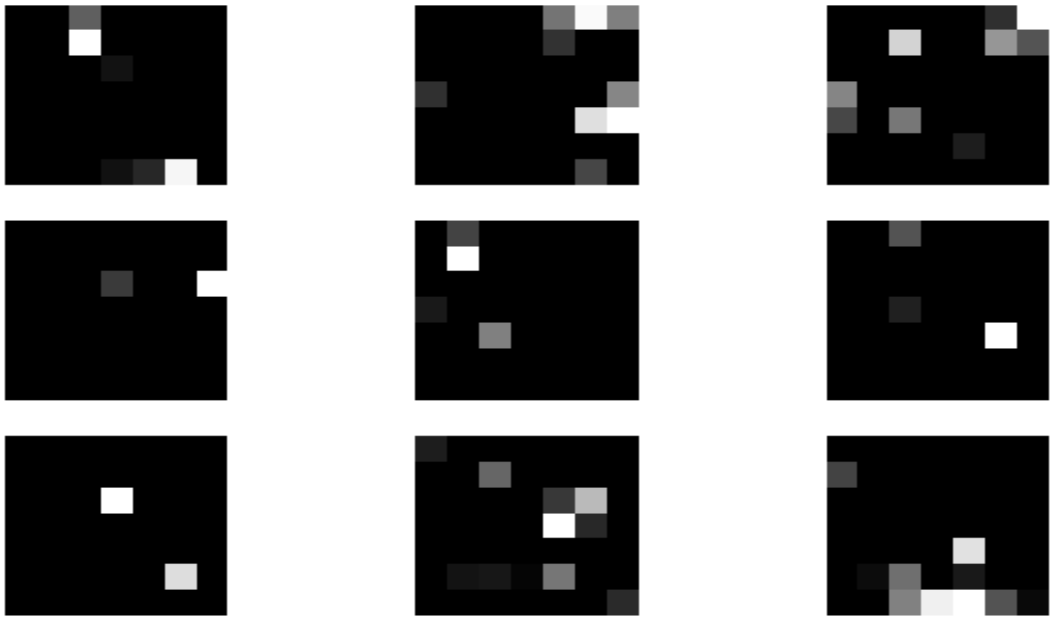
Conv1 Feature Map



Conv2 Feature Map



Conv3 Feature Map



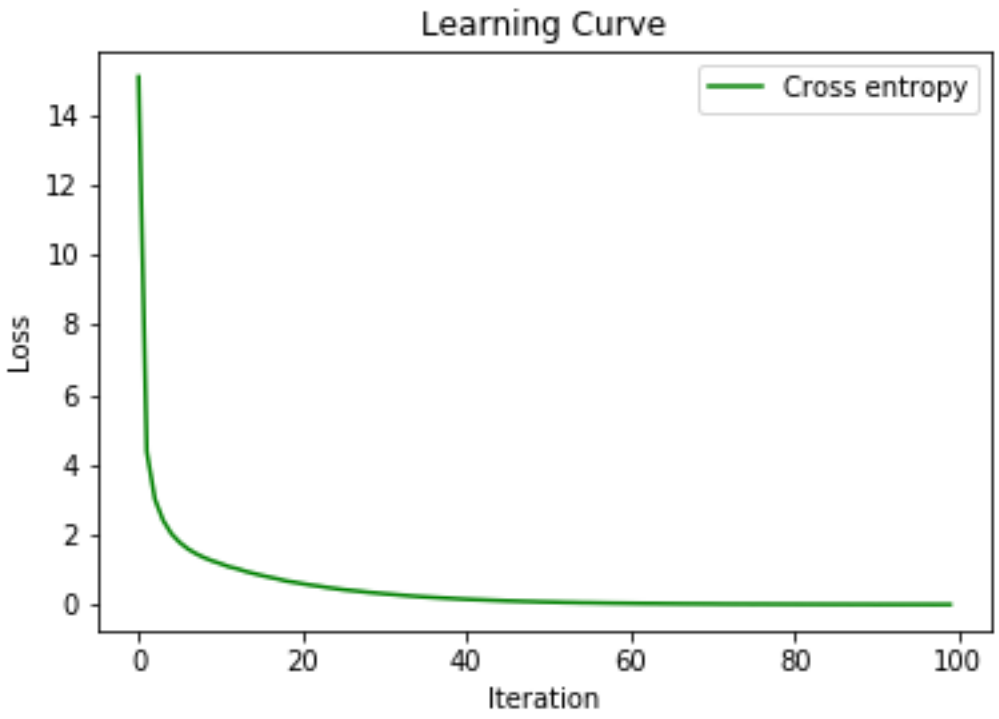
1-4 Effect L2 Regularization

L2 Regularization is applied to all the layers of network and Alpha=9e-4

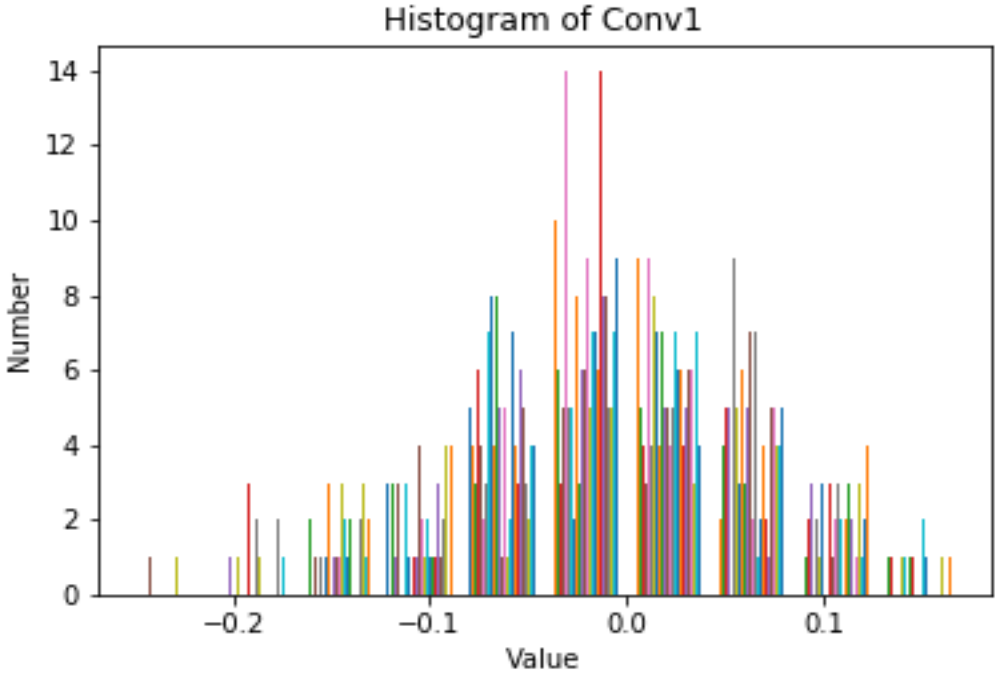
The results after applying regularizations are:

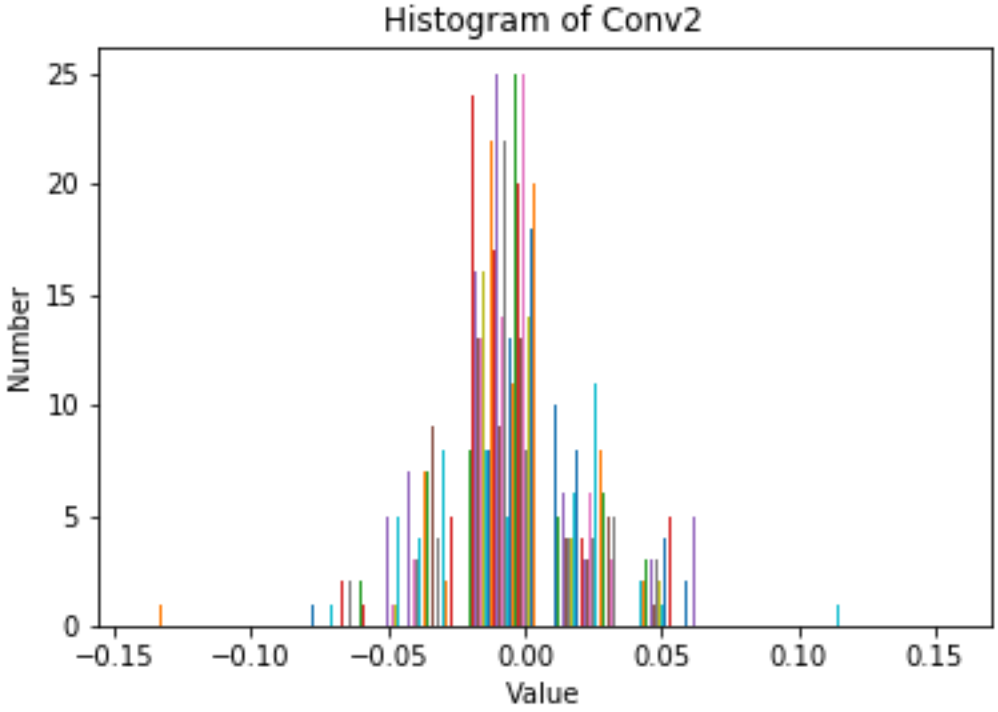
Cost of training data after epoch 95: 0.023876
Train Accuracy: 0.9979799999999647
Validation Accuracy: 0.9828
Test Accuracy: 0.9886
False predictions: 114 out of 10000
True predictions: 9886 out of 10000

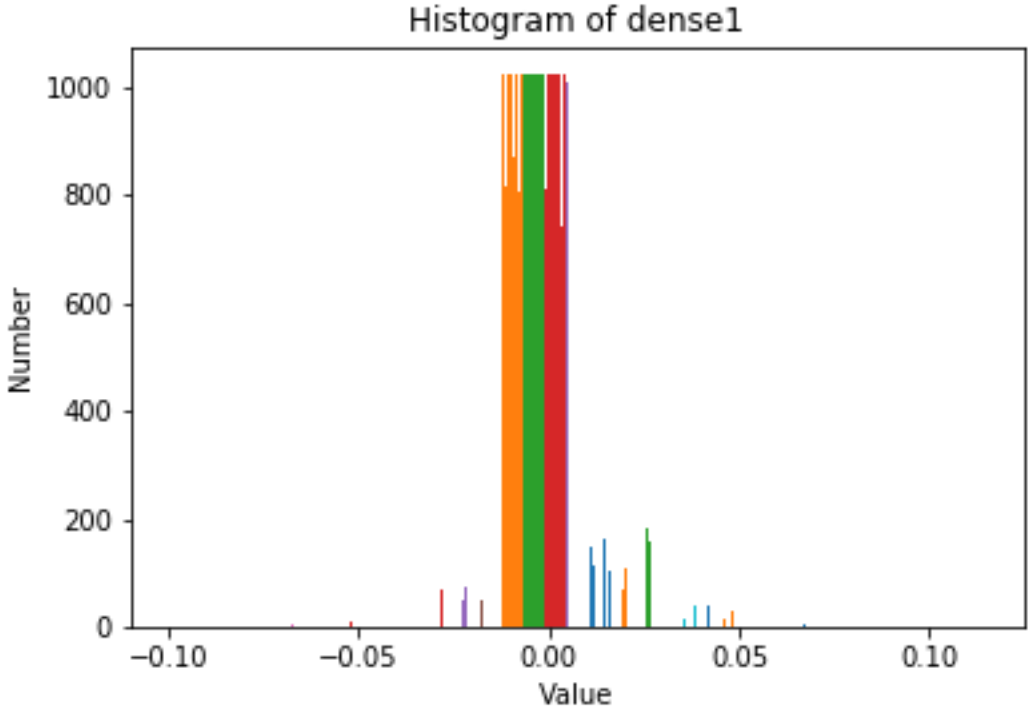
The number of false predictions decreased from 165 to 114 after applying the L2 regularization to model. So, clearly the test accuracy is increased.

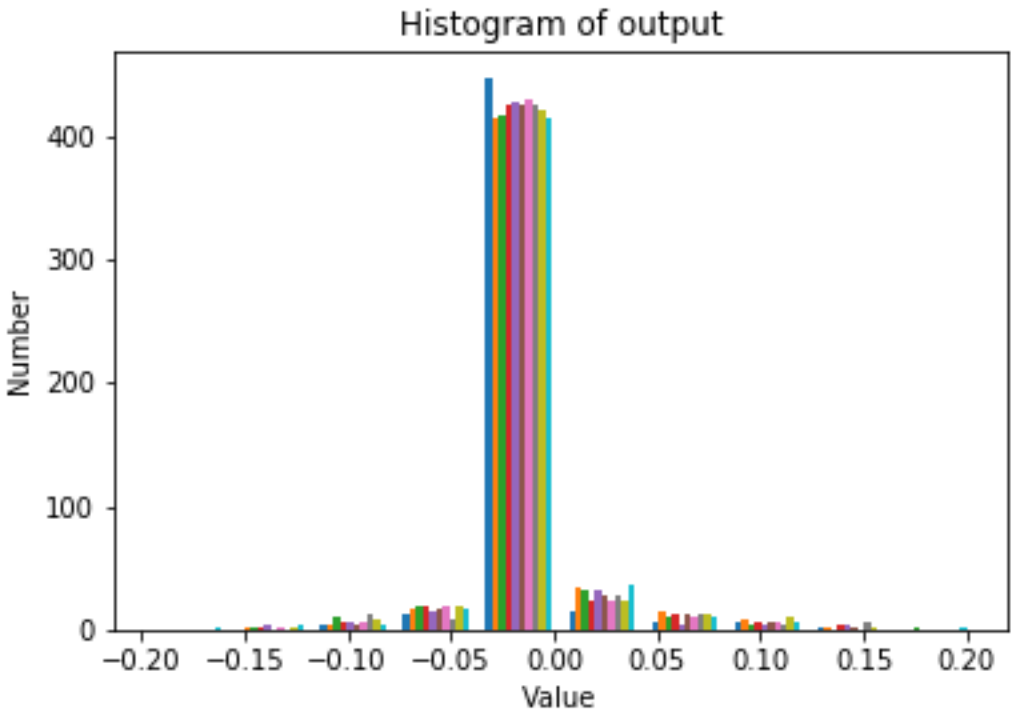



Training Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Accuracy Rate
0
20
40
60
80
100
Iteration
Training Accuracy
Validation Accuracy
Test Accuracy


Histogram of Conv1
14
12
10
8
6
4
2
0
Number
−0.2
−0.1
0.0
0.1
Value


Histogram of Conv2
Number
Value
25
20
15
10
5
0
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15


Histogram of dense1
Number
Value
1000
800
600
400
200
0
−0.10
−0.05
0.00
0.05
0.10



2- Preprocessing Before Using Convolutional Neural Network For Image Recognition

2-1 Network Architecture

Input->Block1[Conv1->Relu->Batch_Norm->Pool1]->Block2[Conv2->Relu->Batch_Norm->Pool2]->Block3[Conv3->Relu->Batch_Norm->Pool3]->Block4->[FC1->Relu->Batch_Norm]->Block5[FC2->Relu->Batch_Norm]->Softmax

Dimensions of Input, weight, bias and output for each Block:

Input data shape:

Shape of Train_x: (49000, 32, 32, 3)

Shape of Train_y: (49000, 10)

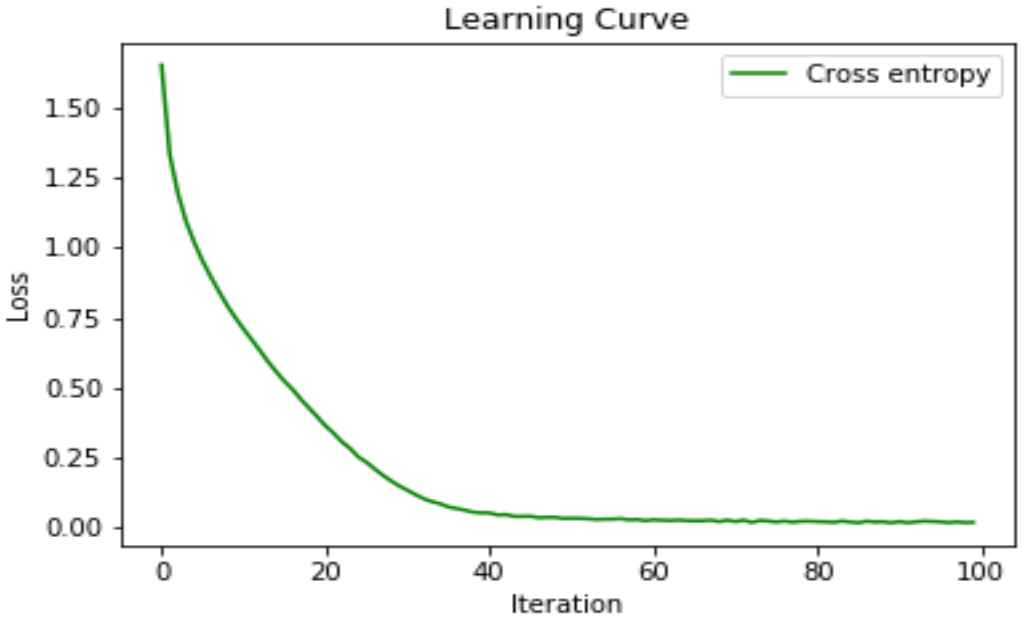
Block1: Input [.,28,28,1], W_conv1[5,5,3,32], B_conv1[32],
Output [.,14,14,32]
Block2: Input [.,14,14,32], W_conv2[5,5,32,32], B_conv2[32], Output [.,7,7,32]
Block3: Input [.,7,7,64], W_conv2[3,3,32,64], B_conv2[64], Output [.,4,4,64]
Block4: Input [.,4,4,64], W_fc1[4*4*64,500], B_fc1[500], Output [.,500]
Block5: Input [.,500], W_fc2[500,10], B_fc2[10], Output [.,10], Softmax->y_pred

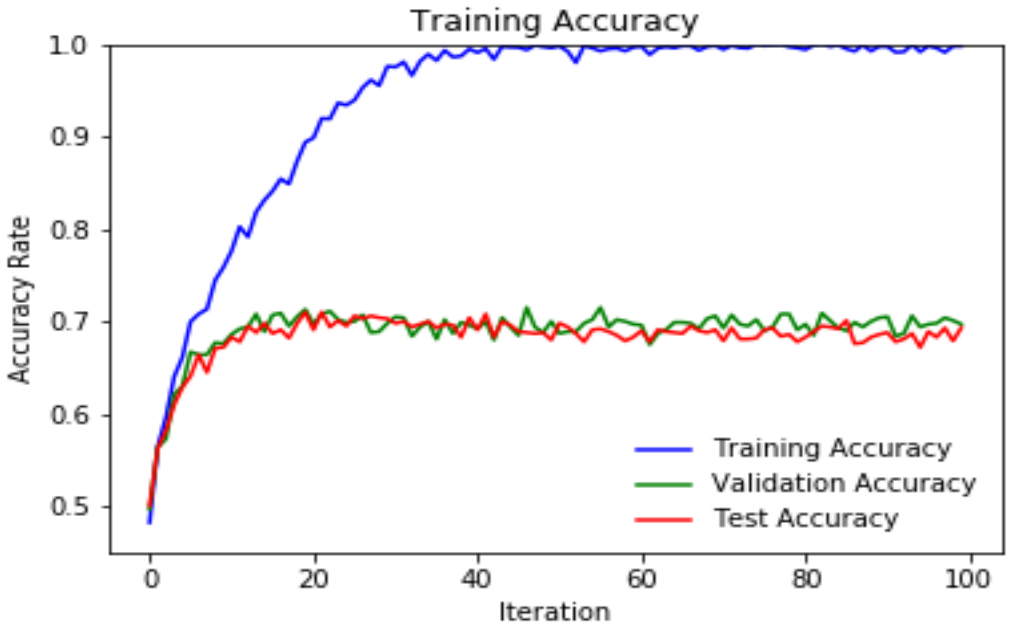
Dimension of Max_Pool and Strides for each Layers: Max_Pool[1,2,2,1], Strides[1,2,2,1]
Type of Padding for each Layer: Same

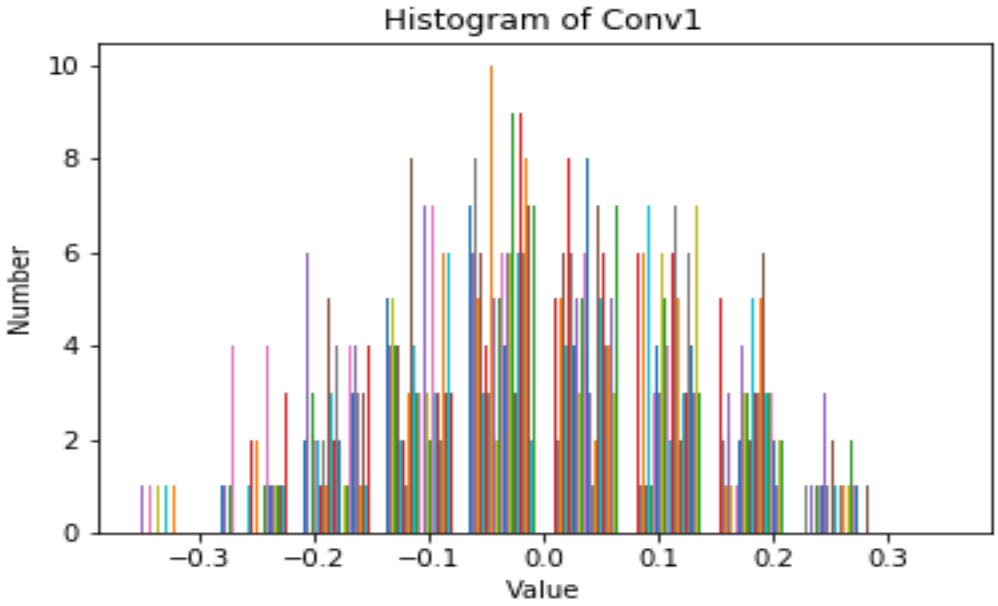
Effect of different stride and filter size: The experiments are done using stride of [1,2,2,1], [1,4,4,1] and [1,8,8,1]. The 4 pixels are skipped for stride of [1,4,4,1] and the 8 pixels are skipped for stride of [1,8,8,1] each time the window is shifted during the convolution operation. So, the stride of 4,4 and 8,8 waste a lot of information during the down sampling. Therefore, the model showed on 99% accuracy on training set but only 40% accuracy on test set for stride of [1,4,4,1] and [1,8,8,1]. The latest model uses stride of [1,2,2,1] for all layers. It performed really well with accuracy of 69% on test data.

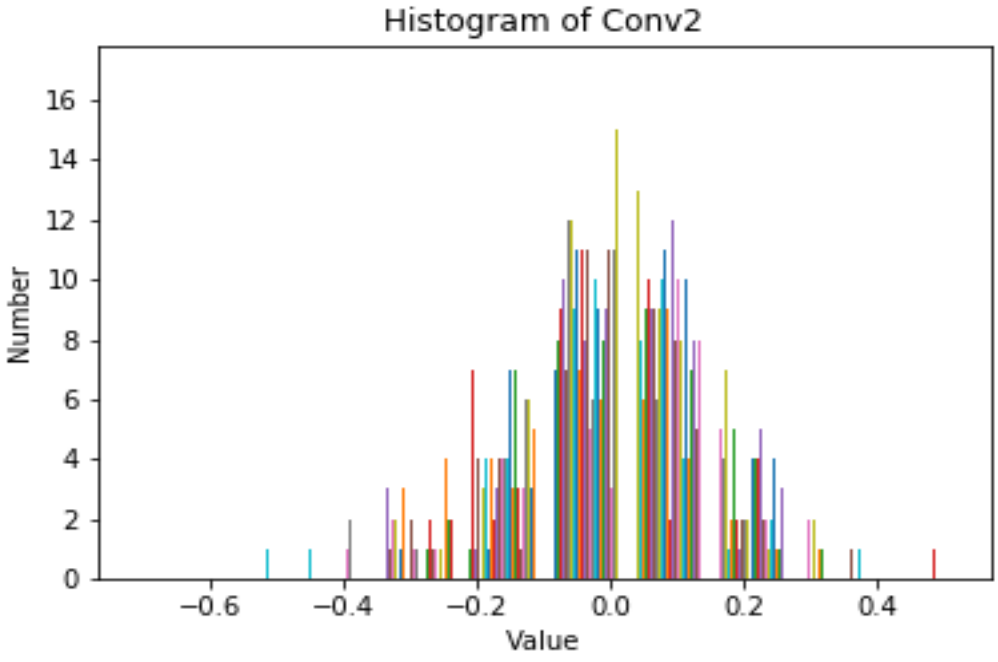
The first experiment is done using filter sizes: w_conv2 [3,3,32,64], w_conv3[3,3,64,128] and fc1[3*3*128,1024]. It was notices that it produced large number of parameters which takes more time for training. The latest model uses 32 and 64 depths instead of 64 and 128 which significantly decreases the number of parameters.

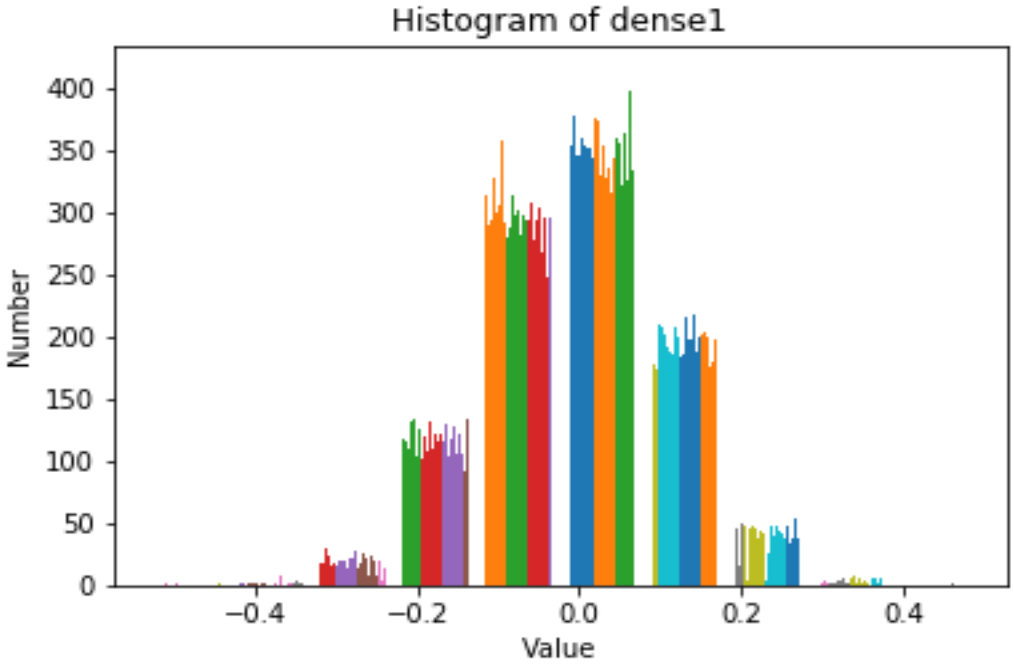
Learning rate = 0.0001
Epochs = 100
Minibatch size = 16
Cost of training data after epoch 95: 0.018467
Train Accuracy: 0.9976322664924507
Validation Accuracy: 0.697
Test Accuracy: 0.693

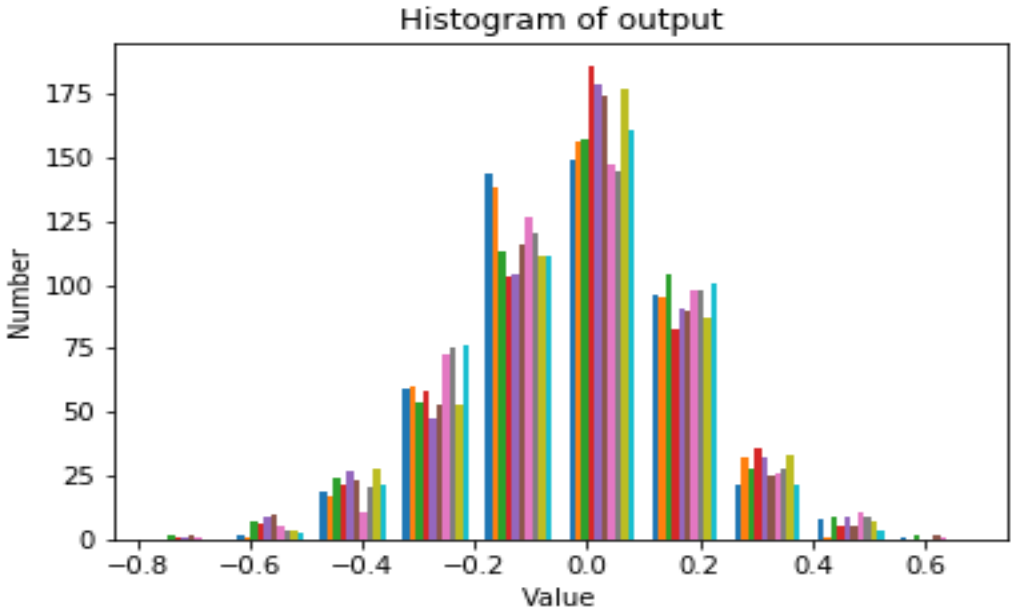

Learning Curve
Cross entropy
Loss
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
20
40
60
80
100
Iteration


Training Accuracy
Accuracy Rate
1.0
0.9
0.8
0.7
0.6
0.5
Training Accuracy
Validation Accuracy
Test Accuracy
0
20
40
60
80
100
Iteration


Histogram of Conv1
Number
Value
0
2
4
6
8
10
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3


Histogram of Conv2
Number
Value
0
2
4
6
8
10
12
14
16
−0.6
−0.4
−0.2
0.0
0.2
0.4

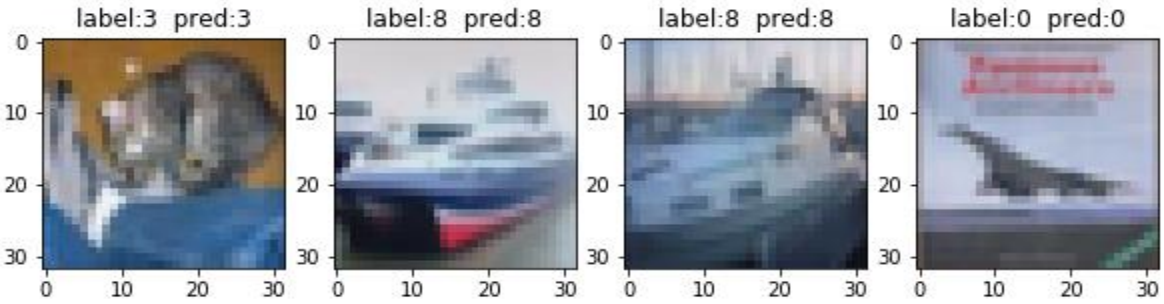




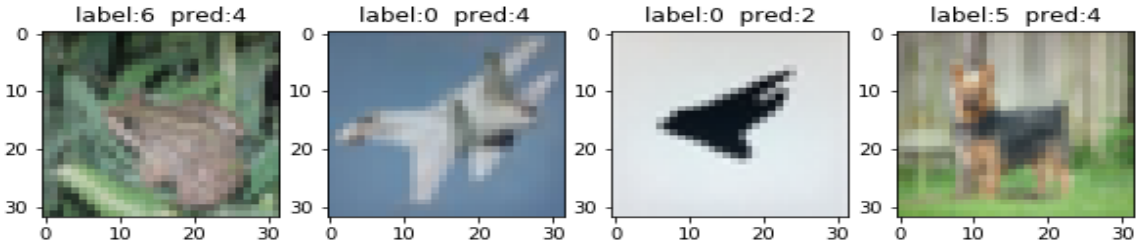
2-2 Correctly classified and miss-classified Images

False predictions: 307 out of 1000 on test data of size 1000
True predictions: 693 out of 1000 on test data of size 1000

Correctly classified Images:



Miss-classified Images:



2-3 Feature Maps

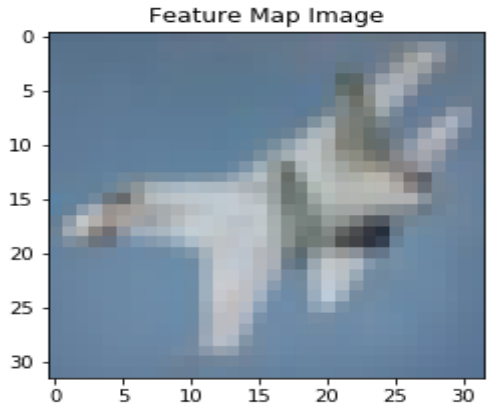
The following figures display 9 feature maps per convolutional layer. Conv1 actually contains 32 feature maps, since we have 32 filters in that layer. But we are only visualizing the first 9 features in conv1. In the same way, we visualize 9 feature maps in conv2 and conv3. There are some interesting observations about the feature maps as we progress through the layers. The feature maps of plane and deer images are visualized.

- The first layer conv1 feature maps retain the most of the information present in the image. In CNN architectures the first layers usually act as edge detectors.

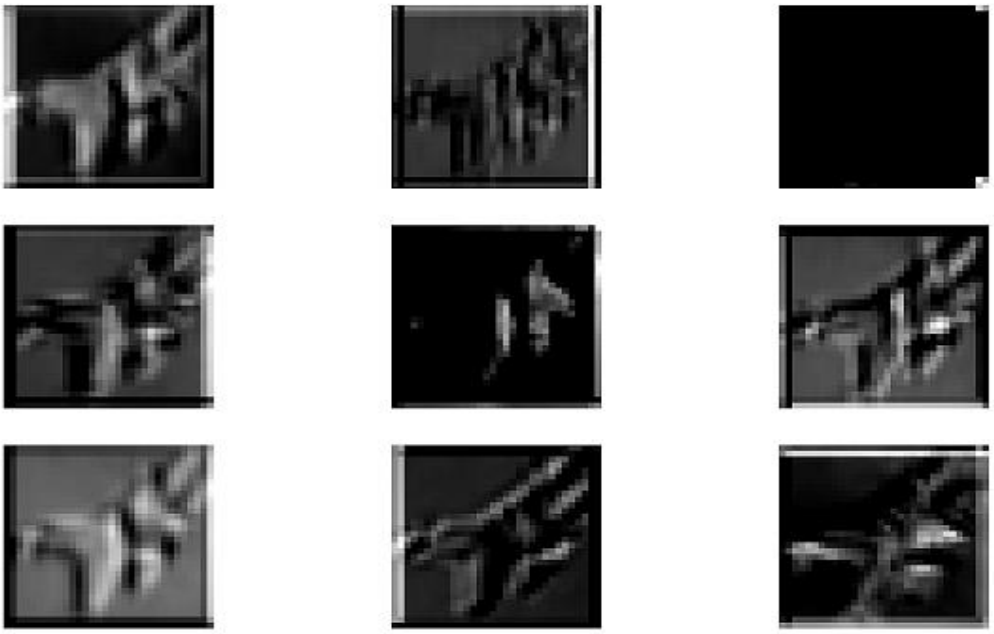
- As we go deeper into the network the feature maps look less like the original image and more like an abstract representation of it. As you can see in conv2 the plane and deer is somewhat visible, but after that it becomes unrecognizable. The reason is that deeper feature maps encode high level concepts like wings of plane and legs of deer while lower

level feature maps detect simple edges and shapes. That's why deeper feature maps contain less information about the image and more about the class of the image. They still encode useful features, but they are less visually interpretable by us.

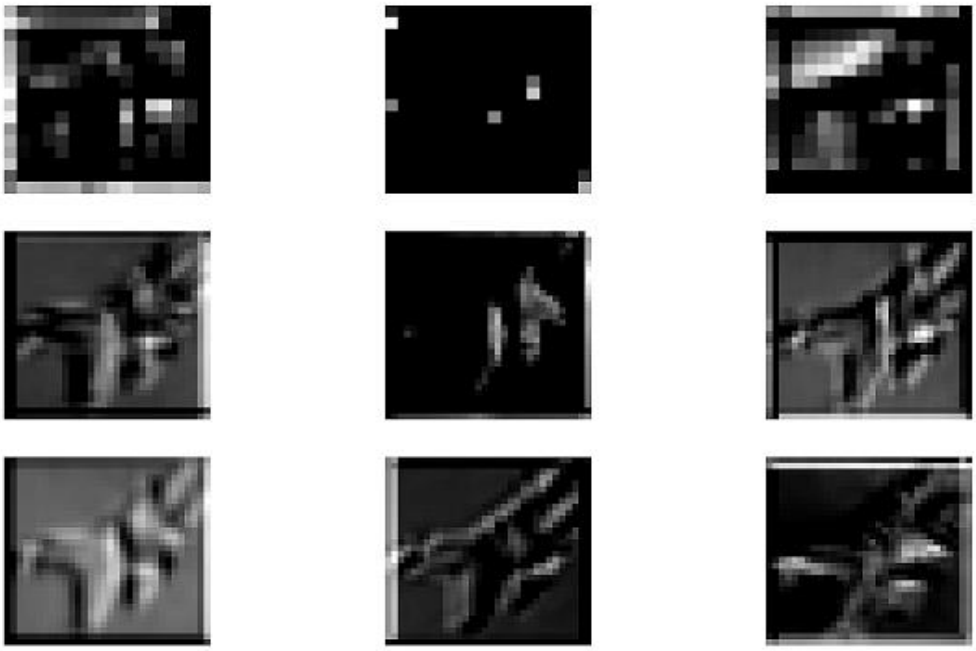
- The feature maps (conv3) become sparser as we go deeper, meaning the filters detect less features. It makes sense because the filters in the first layers detect simple shapes, and every image contains those. But as we go deeper we start looking for more complex stuff like the wings of plane and tail of deer.



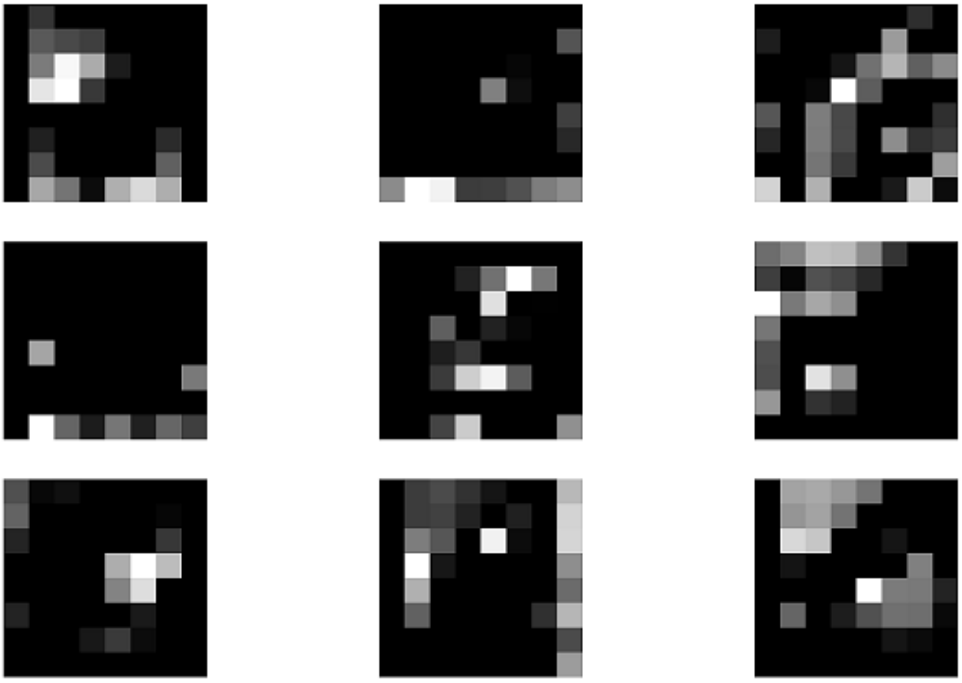
Conv1 Feature Map

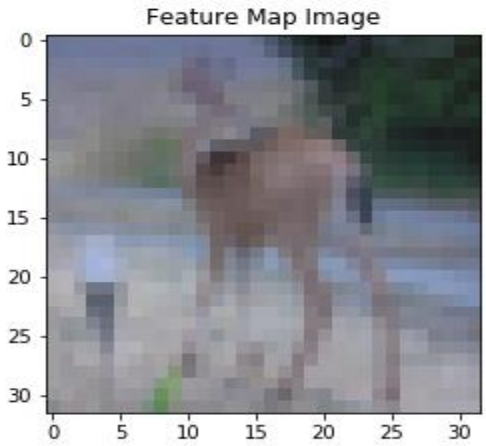


Conv2 Feature Map

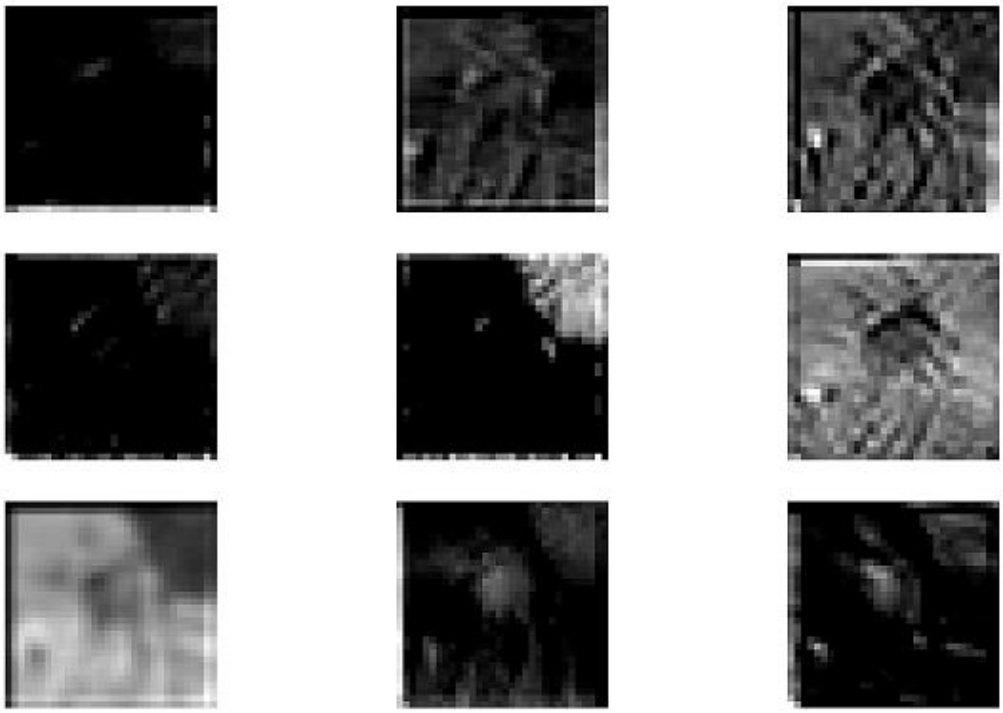


Conv3 Feature Map

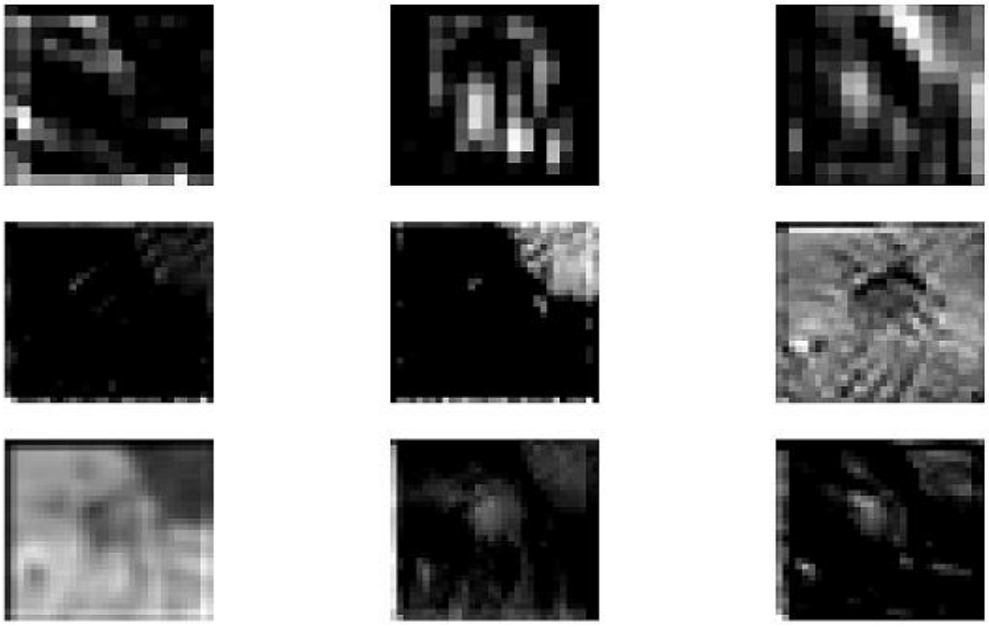




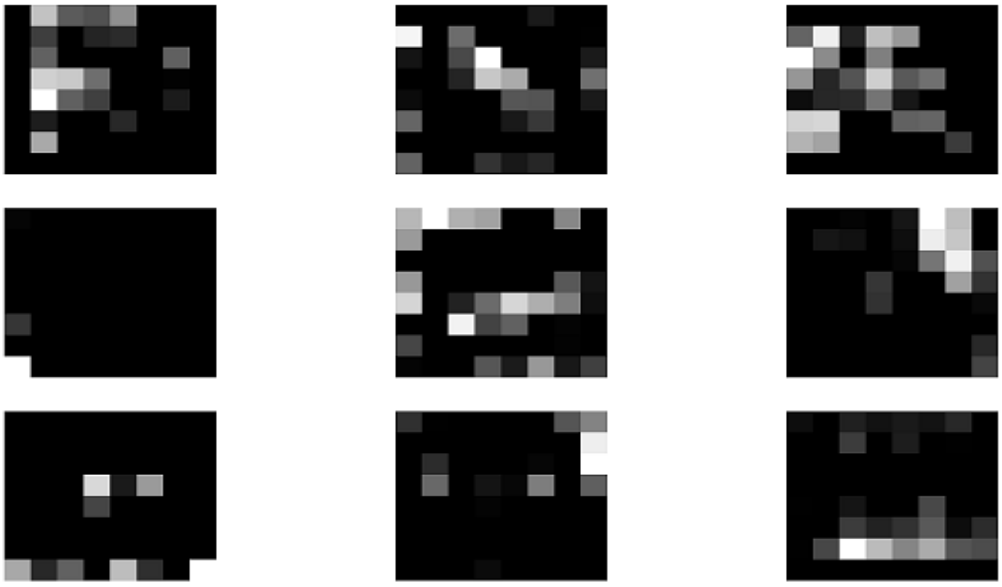
Conv1 Feature Map



Conv2 Feature Map



Conv3 Feature Map



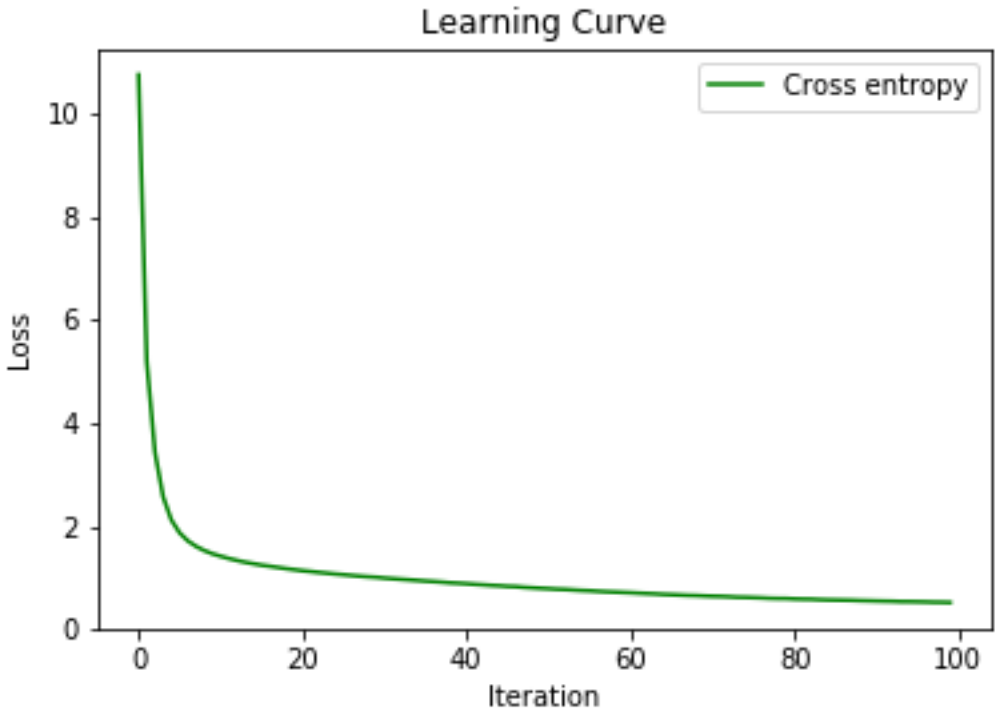
2-4 Effect L2 Regularization

L2 Regularization is applied to all the layers of network and Alpha=9e-4

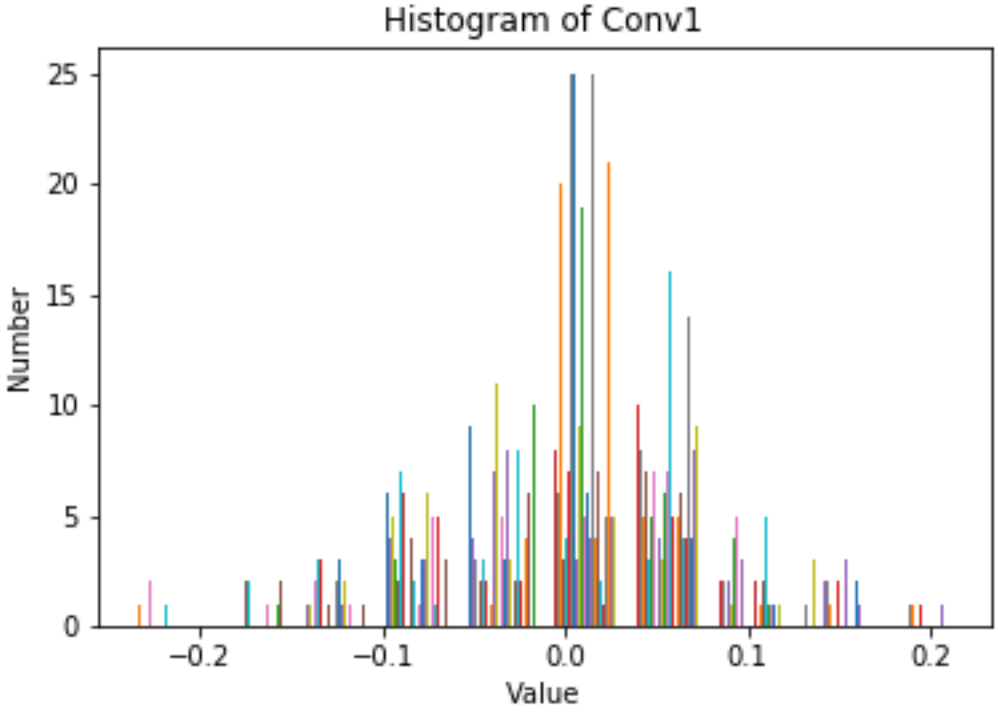
The results after applying regularizations are:

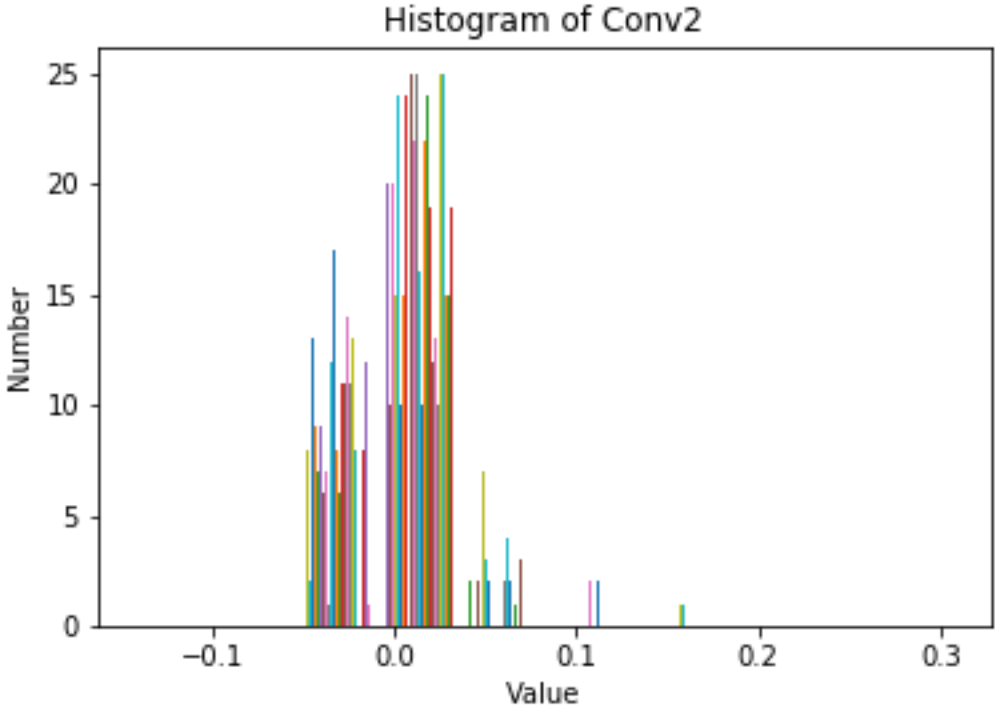
Cost of training data after epoch 95: 0.535060
Train Accuracy: 0.9657903331155958
Validation Accuracy: 0.757
Test Accuracy: 0.768
False predictions: 232 out of 1000
True predictions: 768 out of 1000

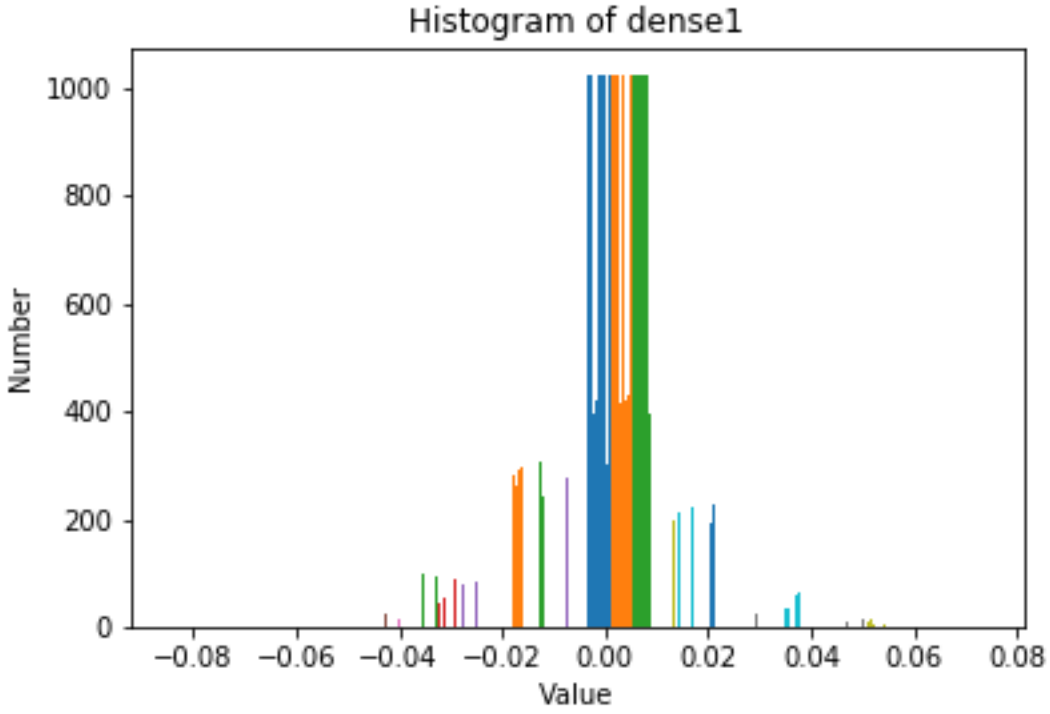
The number of false predictions decreased from 307 to 232 after applying the L2 regularization to model. So, clearly the test accuracy is increased.

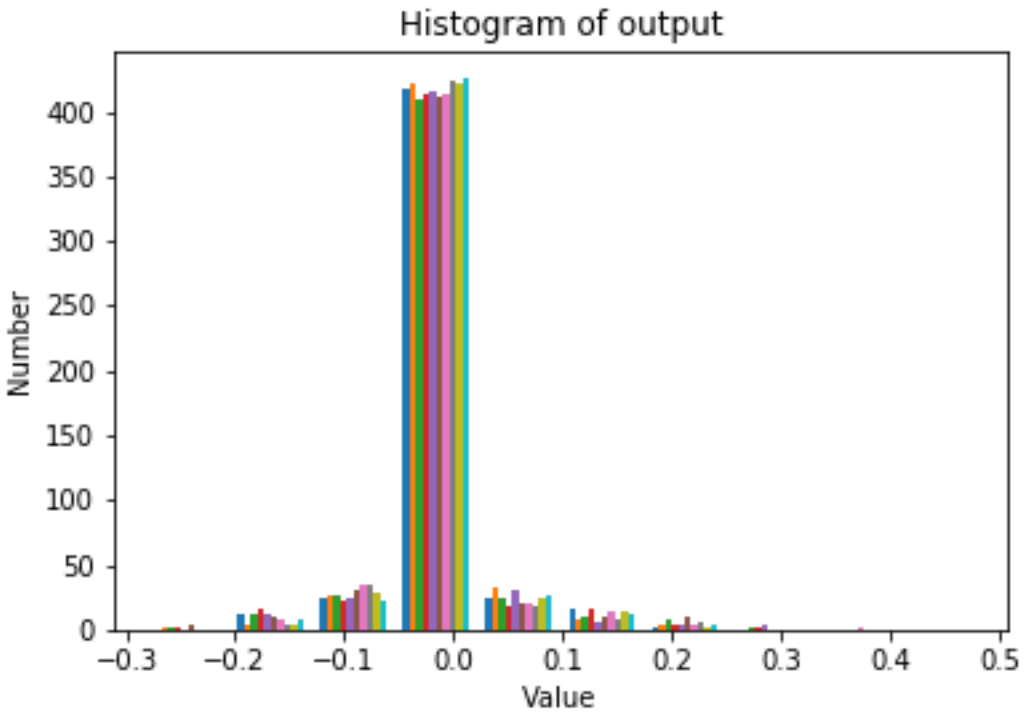



Training Accuracy
Accuracy Rate
Iteration
Training Accuracy
Validation Accuracy
Test Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100


Histogram of Conv1
Number
Value
25
20
15
10
5
0
−0.2
−0.1
0.0
0.1
0.2


Histogram of Conv2
Number
Value
25
20
15
10
5
0
−0.1
0.0
0.1
0.2
0.3


Histogram of dense1
Number
Value
1000
800
600
400
200
0
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
0.08



2-5 CIFAR Preprocessing

2-5.1 Center the data around mean

The mean of training data is calculated around the row axis. Then the mean is subtracted from training data, validation data and test data to center the three data sets around the mean. Subtracting the mean centers, the input around zero.

2-5.2 Scale feature range using minimax normalization

The data is normalized to scale the feature of values between the min and max range of data examples.

Why do we need to preprocess data? Consider how a neural network learns its weights. CNNs learn by continually adding gradient error vectors (multiplied by a learning rate) computed from backpropagation to various weight matrices throughout the network

as training examples are passed through. The thing to notice here is the "multiplied by a learning rate".

If we didn't scale our input training vectors, the ranges of our distributions of feature values would likely be different for each feature, and thus the learning rate would cause corrections in each dimension that would differ (proportionally speaking) from one another. We might be over compensating a correction in one weight dimension while undercompensating in another. This is non-ideal as we might find ourselves in an oscillating (unable to center onto some better manima in cost(weights) space) state or in a slow moving (traveling too slow to get to a better manima) state.

2-5.3 Get one-hot vectors of labels

The labels are converted in one-hot vectors.

3 Implementation of CNN operations using Numpy

I have implemented parts of forward propagation and backward propagation using Numpy just for understanding the process of forward pass and backward pass for convolutional nets but I completed the homework tasks using core Tensorflow library which low level library as compared to TFlearn, Slim. The operation which I have implemented includes: zero padding, forward convolution, forward pooling (max and average), backward propagation and backward pooling.

Reference: https://www.coursera.org/learn/convolutional-neural-networks