# University of Idaho

# Final Report


**Adhar Partap Singh**

**Sing5830@vandals.uidaho.edu**

**Statistics 517 Statistical Learning and Predictive Modeling**

**Stephen Lee**

**Dec 10, 2018**

**Abstract**

In this project, a Deep neural network is proposed and simulated to classify different classes of color images using the CIFAR-10 dataset. Distinct sets of neural networks (i.e with different architectures and different hyper-parameters) were applied on the CIFAR-10 dataset to improve the classification accuracy.

## 1 Introduction

Deep learning in general terms, implies a learning technique that learns features in layers. It is a technique that enables computers/intelligent systems to improve with experience and data. Deep learning is a class of machine learning algorithms that:

• Uses a series of multiple layers for feature extraction and transformation. Each successive layer uses the output from the previous layer as input[5].

• Use some form of gradient descent for training via backpropagation.

## 2 Problem

In this section, the dataset used to implement the Neural Network, different deep learning architectures are discussed.

### 2.1 CIFAR-10 Dataset

Organization of the CIFAR-10 Dataset:

It consists of 60,000 32X32 color images in 10 classes(each image consists of 3 color channels-R,G,B), with 6000 images per class. The classes are labelled as: "'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'".

• There are 60,000 total images

• The dataset is divided into 40,000 training images, 10,000 validation images and 10,000 test images.

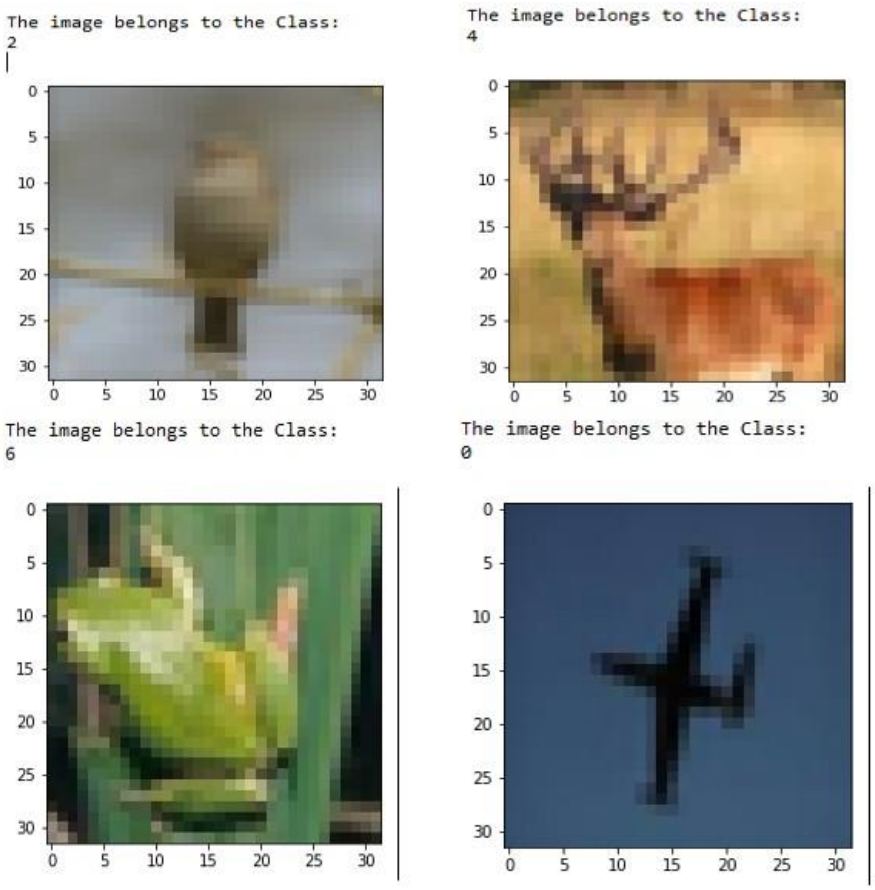


Figure 1: Sample images from the CIFAR10 dataset

### 2.1.1 Data Processing

• The dataset (in the raw format) is divided into five training batches and one test batch, each with 10,000 images and corresponding labels. The test batch is kept as it is. Four out of the five training batches are concatenated and used as the final training image set for the project. The fifth batch is used as the validation image set.

• The images and labels are zipped together using the 'zip' function so as to associate each image with its corresponding label, namely: Training data=(images,labels), Validation data=(images,labels) and Test data=(images,labels)

• The image is reshaped to form a 3072 X 1 vector.

### 2.1.2 Data Normalization

Image:

The pixel values are in the range of 0 to 255 for each of the red, green and blue

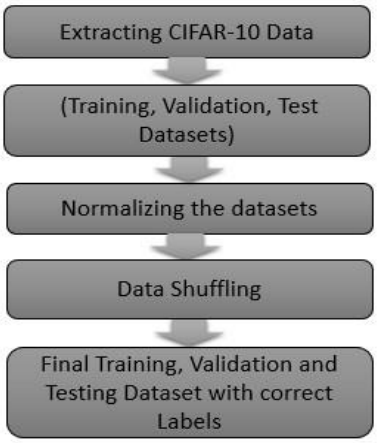


Figure 2: Data Processing Flowchart

channels. The input values are normalized to the range 0 to 1 by dividing each value by the maximum observation which is 255.

Since the data is loaded as integers, the input values are converted to floating point values in order to perform the division. The output variables are defined as a vector of integers from 0 to 1 for each class.

Labels:

The output label lies between 0 to 9 each integer indicating a different class.

### 2.1.3 Data Shuffling

The data is then shuffled using the 'shuffle' function so as to randomize the input pattern.

## 2.2 Feed-Forward Neural Network

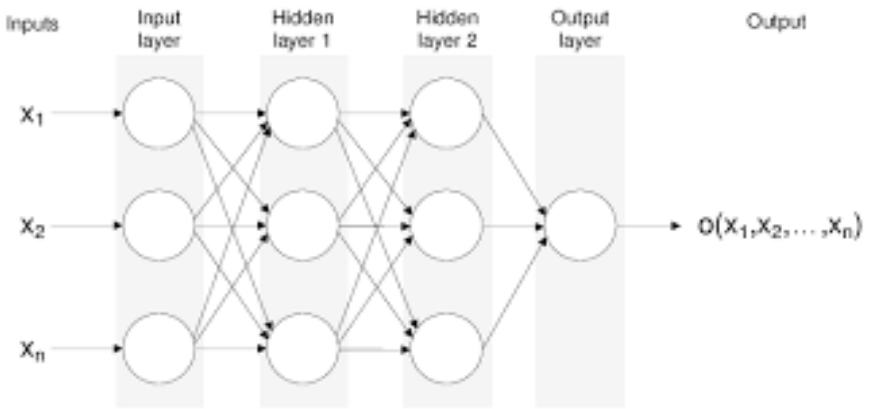


Figure 3: Multilayer Neural Network [8]

A feed-forward neural network comprises of one or more hidden layers, excluding the input and the output layer. The input layer accepts the data and passes it on to the network, while the output layer gives the prediction information of the input (eg, an image).

Training the feed-forward network essentially means tuning the weights so as to optimize a predetermined loss function with respect to the dataset. A commonly used loss function is cross-entropy which defines how well a model fits on the dataset.[6]

Gradient descent is generally used to optimize this loss function.

| Input Layer Neurons | Hidden Layer 1 Neurons | Hidden Layer 2 Neurons | Output Layer Neurons | Learning Rate (eta) | Regularization (Lambda) | Classification Accuracy (%) |
|---|---|---|---|---|---|---|
| 3072 | 30 | - | 10 | 0.5 | 0.05 | 34.41 |
| 3072 | 30 | - | 10 | 0.5 | 0.5 | 37.67 |
| 3027 | 30 | 30 | 10 | 0.5 | 0.5 | 36.63 |
| 3072 | 30 | 30 | 10 | 0.5 | 0.05 | 36.90 |
| 3072 | 30 | 30 | 10 | 0.05 | 0.05 | 36.23 |

Figure 4: Feedforward Network Results

### 2.2.1 Feed-Forward Neural Network Result

The Neural Network is first implemented using the Feed-forward network. Fig- ure 4 provides the information about the classification accuracy obtained using different architectures.

In the first iteration, I use the input layer with , 3*32*32=3072, neurons, one hidden layer (30 neurons) and the output layer (10 neurons representing the 10 classes). The hyperparamteres used are : eta=0.5 and lambda=0.5. The classification accuracy obtained was equal to 37.67. To increase the accuracy I tried using different combinations of hyperparameters, but the best result (with only one hidden layer) was obtained with eta=0.5,lambda=0.05. A second hid- den layer (30 neurons) was also added to improve the performance. But the performance didnot

increase much as the network now suffers the problem of over fitting since it has too many parameters to optimize. The best result obtained was around 37 percent. I implemented the Convolution neural network next to enhance the performance.

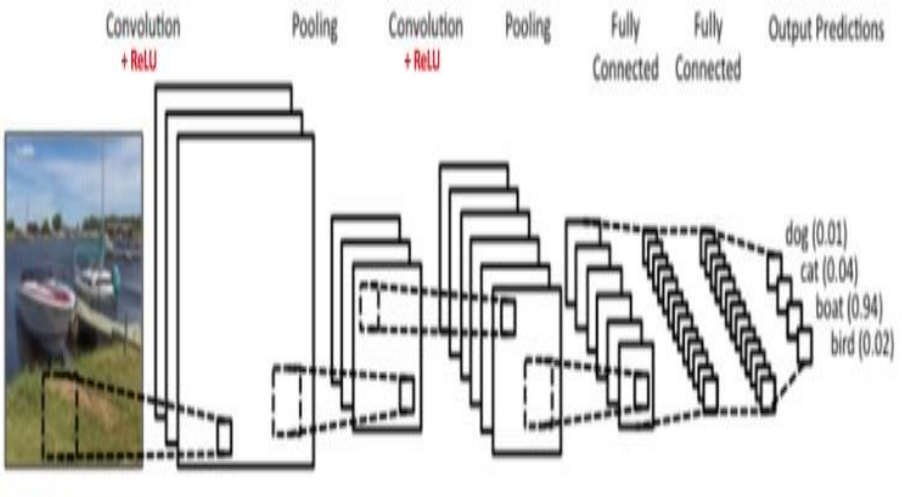


Figure 5: Convolutional Neural Network [7]

### 2.3 Convolutional Neural Netork(CNN)

A Convolutional Neural Net (CNN) neuron collects the output from all the neu- rons from all the neurons in the previous layer. It uses a window of fixed size on a region of data and computes the weighted sum over the region. The window is then moved horizontally and vertically and applied to all possible regions of data and the corresponding weighted sum is calculated. This method of extracting information is known as 'Convolution' [9].

### 2.4 Theano

There are many open source tools available which are used for deep learning algorithms to reduce the computational effort such as keras, tensorflow, theano, pyTorch, Caffe etc. Here, theano was chosen. [4]

## 3 Methodology and Results

A convolutional network was and implemented on the CIFAR-10 data. This net- work used three convpool layers, three max-pooling layers, , ,two fully-connected layers, one softmax layer form a multi-layer network with 1000 hidden neurons and 10 outputs. The details for the convolutional layers are tabulated below.

**Classification**

• Network architecture 1:

I started the CNN using only one fully-connected and one softmax with the default activation function (sigmoid) for simplicity. The accuracy was
38.54 percent.

• Network architeture 2: I added one convolutional layer. I tried using various combinations of different activation functions and hyperparameters.

When I used sigmoid as the activation function the accuracy touched the lowest possible percentage (10 percent). I changed the activation function to ReLU and tried different hyperparameters. The best accuracy obtained was 64.88 percent. A change in the number of filters used and filter size resulted in a marginal increase in accuracy over the previous iterations (65.77 percent).

| Convpool1 | Filter1 | Pool1 | Fully-connected1 | Softmax | Act. Func | Eta | Lambda | Dropout | %Accuracy |
|---|---|---|---|---|---|---|---|---|---|
| | | | 3072 | 1000 | Sigmoid | 0.03 | 0.1 | | 37.38 |
| | | | 3072 | 1000 | Sigmoid | 0.03 | 0.01 | | 38.54 |
| 32 | 20,3,3 | 2,2 | 20*15*15 | 1000 | Sigmoid | 0.03 | 0.1 | 0.5 | 9.87 |
| 32 | 20,3,3 | 2,2 | 20*15*15 | 1000 | Sigmoid | 0.03 | 0.01 | 0.5 | 10.03 |
| 32 | 20,3,3 | 2,2 | 20*15*15 | 1000 | ReLU | 0.03 | 0.1 | 0.5 | 64.88 |
| 32 | 20,3,3 | 2,2 | 20*15*15 | 1000 | ReLU | 0.03 | 0.01 | 0.5 | 63.11 |
| 32 | 20,3,3 | 2,2 | 20*15*15 | 1000 | ReLU | 0.03 | 0.001 | 0.5 | 63.61 |
| 32 | 40,3,3 | 2,2 | 40*15*15 | 1000 | ReLU | 0.03 | 0.1 | 0.5 | 63.76 |
| 32 | 40,5,5 | 2,2 | 40*14*14 | 1000 | ReLU | 0.03 | 0.1 | 0.5 | 65.77 |

Figure 6: One CNN Layer Results

**• Network architecture 3:**

To elicit a better performance from the network, I further added a second convolutional layer.

| Conv Layer 1 | Filter1 | Pool1 | Conv Layer 2 | Filter2 | Pool2 | FC1 | FC2 | Softmax | Act. Func | Eta | Lmda | Dropout | %Acc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 40,5,5 | 2,2 | 14 | 20,40,5,5 | 2,2 | 20*5*5 | - | 1000 | ReLU | 0.03 | 0.1 | 0.5 | 66.75 |
| 32 | 40,5,5 | 2,2 | 14 | 20,40,5,5 | 2,2 | 20*5*5 | 1000 | 1000 | ReLU | 0.03 | 0.1 | 0.5 | 71.07 |
| 32 | 40,5,5 | 2,2 | 14 | 40,40,5,5 | 2,2 | 40*5*5 | 1000 | 1000 | ReLU | 0.03 | 0.1 | 0.5 | 72.42 |
| 32 | 40,5,5 | 2,2 | 14 | 40,40,5,5 | 2,2 | 40*5*5 | 1000 | 1000 | ReLU | 0.03 | 0.01 | 0.5 | 71.21 |

Figure 7: Two CNN Layers Results

I kept the filter window size the same as before (5,5). I tried the network with one fully-connected layer initially. The result was 66.75 percent accuracy. Addition of another fully-connected layer increased the accuracy to 71.07 percent. Increasing the number of feature maps did not affect the accuracy too much. It marginally increased to 72.42 percent. Changing the hyperparameters in fact lowered the performance. So I chose the network configura- tion which gave the best result when two convolutional layers were used (i.e eta=0.1,lambda=0.03,filter size=5,5 with 40 feature maps)

### 3.1 Final Design

- Final Network architecture:

At the expense of computational effort, I decided to add yet another convolutional layer. The network structure now:

The input size for the first convolutional layer is 3072 (3*32*32). I have used

| |
|---|
| **ConvPoolLayer 1**: (image shape=3, 32, 32), filter shape= (40, 3, 5, 5), pool size= (2,2), activation func=ReLU) |
| **ConvPoolLayer 2**: (image shape=40, 14, 14), filter shape= (40, 40, 3, 3), pool size= (2, 2), activation func=ReLU) |
| **ConvPoolLayer 3**: (image shape=40, 6, 6), filter shape= (40, 40, 3, 3), pool size= (2, 2), activation func=ReLU) |
| **Fully-Connected Layer 1**: (N in=40*2*2, N out=1000, activation func=ReLU, p dropout=0.5 |
| **Fully-Connected Layer 2**: (N in=1000, N out=1000, activation func=ReLU, p dropout=0.5 |
| **Softmax Layer**: (N in=1000, N out=10) |
| eta=0.03, lambda=0.1 |

Figure 8: Final Network Design

40 feature maps of size 5,5 for the first layer with a Max pooling of 2,2. This results in a downsampled image of size 14,14. I use 40 feature maps of 3,3 size for the second convolutional layer with a Max pooling of 2,2. This produces a downsampled image of size 6,6. 40 feature maps of size 3,3 were used for the third convolutional layer which produced a 2,2 downsampled image size. Then I use a Fully-connected layer which accepts an input of size 40*2*2. The number of hidden neurons for this layer are 1000 with a droupout rate of 0.5. The second fully-connected layer accepts an input of size 1000. It also has 1000 hidden neurons. The droupout is maintained at 0.5. The activation function for both these layers is ReLU (since using sigmoid produces unfavorable results, as previously seen). A single Softmax layer with a log-likelihood cost function is used. Mini-batch size was equal to 10.
The best result was obtained for this case (73.84 percent). Increasing the mini- batch size to 50 from 10 tends to decrease the accuracy (66.83 percent).

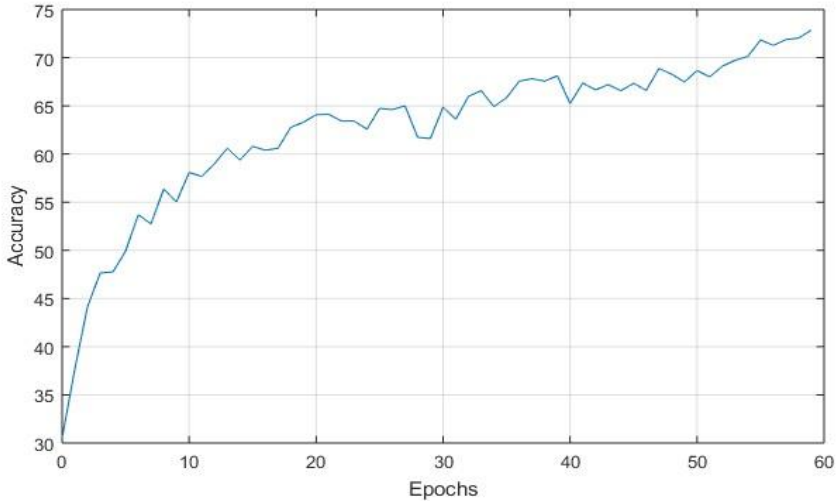


Figure 9: Final Accuracy Plot

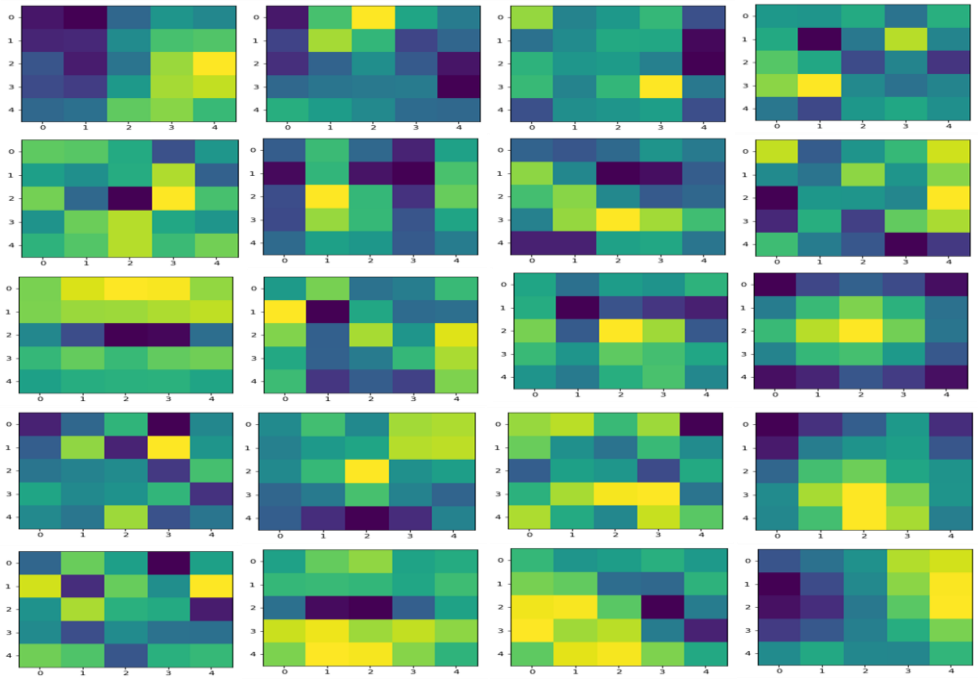

Figure 10: First Convolutional Layer Weights(First 20 maps)

```
datadict= pickle.load(f)
        X = datadict['data']
        Y = datadict['labels']
        X = X.reshape(10000, 3, 32,
32).transpose(0,2,3,1).astype('float32')
```

```
        Y = np.array(Y).astype('int32')
        return X, Y

def load_CIFAR10(directory):
    '''Load all of CIFAR'''
    xs = []
    ys = []
    for k in range(1,6):
        f = os.path.join(directory, "data_batch_%d" % k)
        X, Y = load_CIFAR_file(f)
        xs.append(X)
        ys.append(Y)
    Xtr = np.concatenate(xs)
    Ytr = np.concatenate(ys)
```

The weight maps categorizing features of 10 classes on bases of there edge, wheel, legs, height, length and other many more. Different colors weigh different features of a image in data set. If we look carefully we can see that both models produce artifacts on the borders of the reconstructed images; this is due to the initial padding operation that forces the network to learn a padded, thus not real, representation of the input.

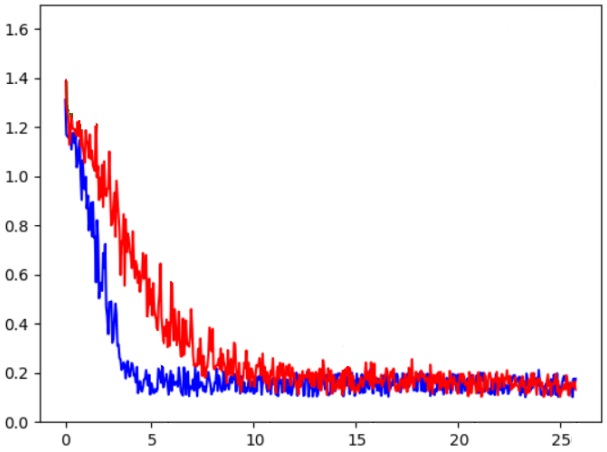


Figure 11: Training Testing Curve

The x-axis represents the number of epochs which in turn reflect the number of time the whole dataset is passed through the neural network for training. The y-axis represents the loss of the neural network which is calculated using the mean squared loss of the target value and the predicted value for the target. The curve follows the normal trend

i.e. the loss is decreasing with increasing the number of epochs. This reflects that the network is trained properly by adjusting the appropriate weights and can be used for testing and validation.

**Regression**

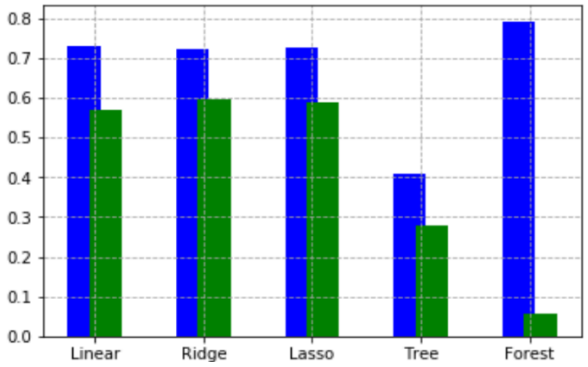


Figure 12: Regression

for n in range(1,6):

```
forest = RandomForestClassifier(n_estimators=n, random_state=2).fit(X_train, y_train)
  train_scores['forest' + str(n)] = forest.score(X_train, y_train)
  test_scores['forest' + str(n)] = forest.score(X_test, y_test)
  print('Random Forest with ', n, ' estimators training set score: ', train_scores['forest' + str(n)])
  print('Random Forest with ', n, ' estimators testing set score: ', test_scores['forest' + str(n)])
```

```
Random Forest with  1  estimators training set score:  0.6518
Random Forest with  1  estimators testing set score:  0.0548
Random Forest with  2  estimators training set score:  0.6532
Random Forest with  2  estimators testing set score:  0.0516
Random Forest with  3  estimators training set score:  0.792
Random Forest with  3  estimators testing set score:  0.056
Random Forest with  4  estimators training set score:  0.8948
Random Forest with  4  estimators testing set score:  0.0666
Random Forest with  5  estimators training set score:  0.9472
Random Forest with  5  estimators testing set score:  0.0718
```

for n in range(1,6):

```
  tree = DecisionTreeRegressor(max_depth=n).fit(X_train, y_train)
  train_scores['tree' + str(n)] = tree.score(X_train, y_train)
  test_scores['tree' + str(n)] = tree.score(X_test, y_test)
  print('Decision Tree of depth ', n, ' training set score: ', train_scores['tree' + str(n)])
```

```
print('Decision Tree of depth ', n, ' testing set score: ', test_scores['tree' + str(n)])
```

```
Decision Tree of depth  1  training set score:  0.0793931387685397
Decision Tree of depth  1  testing set score:  0.07731104414428347
Decision Tree of depth  2  training set score:  0.15509533885781013
Decision Tree of depth  2  testing set score:  0.14628498384961497
Decision Tree of depth  3  training set score:  0.24521832339141192
Decision Tree of depth  3  testing set score:  0.2267411593333222
Decision Tree of depth  4  training set score:  0.3420422777513519
Decision Tree of depth  4  testing set score:  0.2798405029964942
Decision Tree of depth  5  training set score:  0.407878341435725
Decision Tree of depth  5  testing set score:  0.27789375368190716
```

```
lasso = Lasso().fit(X_train, y_train)
train_scores['lasso'] = lasso.score(X_train, y_train)
test_scores['lasso'] = lasso.score(X_test, y_test)
print('Lasso training set score: ', train_scores['lasso'])
print('Lasso testing set score: ', test_scores['lasso'])
```

```
Lasso training set score:  0.7252724536372157
Lasso testing set score:  0.588836739126878
```

```
ridge = Ridge().fit(X_train, y_train)
train_scores['ridge'] = ridge.score(X_train, y_train)
test_scores['ridge'] = ridge.score(X_test, y_test)
print('Ridge Regression training set score: ', train_scores['ridge'])
print('Ridge Regression testing set score: ', test_scores['ridge'])
```

```
Ridge Regression training set score:  0.7230868904933O5
Ridge Regression testing set score:  0.5958352577790762
```

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
train_scores['linear'] = lr.score(X_train, y_train)
test_scores['linear'] = lr.score(X_test, y_test)
print('Linear regression training set score: ', train_scores['linear'])
print('Linear regression testing set score: ', test_scores['linear'])
```

```
Linear regression training set score:  0.7288183814202877
Linear regression testing set score:  0.5672493726619512
```

The x-axis represents different models regression. The Y-axis represent the accuracy score. Highest training score is achieved by random forest method whereas highest testing is achieved in ridge regression. The overall appropriate method is ridge regression. This is because of features of image were specified more appropriately which make it visible and recognized.

**Association**

Association rule transform the dataset into binary figures. By using frequent patterns it is easy to find frequent item sets, and association rules to find antecedents and consequents, association analysis can look for rules of individual choice. I'm still running through the association analysis but some of the common rules are fly, jump, and run . For example, different classes of precipitation are associated with particular feature like fly airplane and bird both can fly but there classifications are different. Exploring the rules allows for particular variables of interest to be searched. For example, what make NN to recognize fly object is airplane or bird?

```
treed=pd.get_dummies(image, columns=["Airplane", "Automobile", "Bird"])
treed['elevation']=pd.cut(treed.elev_ft,bins=[0,1000,2000,3000,4000,5000,6000,7000,8000,9000,
10000],
labels=["less_1000","btw_1_2thou","btw_2_3thou","btw_3_4thou","btw_4_5thou","btw_5_6tho
u","btw_6_7thou","btw_7_8thou","btw_8_9"btw_9_10,thou"thou""great_10000"])
treed=pd.get_dummies(treed, columns=["elevation"])
treed=pd.get_dummies(treed,columns=["temp"])
treed['NS_Aspect']=pd.cut(treed.cos_aspect,bins=[-1,0,1],labels=["wings","edge"])
treed=pd.get_dummies(treed,columns=["NS_Aspect"])
treed['EW_Aspect']=pd.cut(treed.sin_aspect,bins=[-1,0,1],labels=["long","wheels"])
treed=pd.get_dummies(treed,columns=["EW_Aspect"])
treed['fly']=pd.cut(treed.fly,bins=[0,100,200,300,401],labels=["airplane","automobile","ship","tru
ck","bird"])
treed=pd.get_dummies(treed,columns=["Slope"])
```

```
testrules=rules[consequents]=={cond_fly}
```

```
testrule_sort_values([lift],acending=false)
```

| | antecedents | consequents | antecedents support | consequents support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 5177 | (airplane) | (cond_fly) | 0.009466 | 0.009449 | 0.000116 | 0.980466 | 8.009466 | 0.004156 | inf |
| 24619 | (automobile) | (cond_fly) | 0.003694 | 0.005006 | 0.001166 | 0.000801 | 34.009466 | 0.004278 | 1.126366 |
| 1469 | (ship) | (cond_fly) | 0.000052 | 0.000066 | 0.000173 | 0.000061 | 55.009466 | 0.004365 | 1.279425 |
| 3157 | (truck) | (cond_fly) | 0.000019 | 0.000016 | 0.000006 | 0.000019 | 87.009466 | 0.001452 | 1.000801 |
| 419 | (bird) | (cond_fly) | 0.009066 | 0.000746 | 0.009266 | 0.925651 | 5.009466 | 0.002598 | inf |

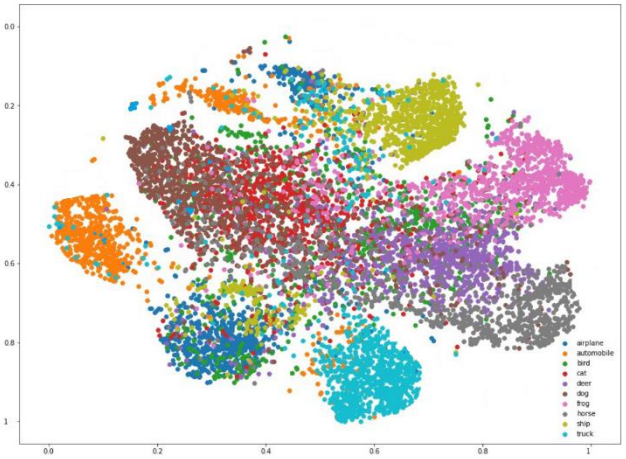


Figure 13: Cluster

```
'airplane',
    'automobile',
    'bird',
    'cat',
    'deer',
    'dog',
    'frog',
    'horse',
    'ship',
    'truck'
train, test = chainer.datasets.get_cifar10()

print('len(train), type ', len(train), type(train))
print('len(test), type ', len(test), type(test))
```

```
print('train[0]', type(train[0]), len(train[0]))

x0, y0 = train[0]
print('train[0][0]', x0.shape, x0)
print('train[0][1]', y0.shape, y0, '->', CIFAR10_LABELS_LIST[y0])

def plot_cifar(filepath, data, row, col, scale=3., label_list=None):
    fig_width = data[0][0].shape[1] / 80 * row * scale
    fig_height = data[0][0].shape[2] / 80 * col * scale
    fig, axes = plt.subplots(row,
                             col,
                             figsize=(fig_height, fig_width))
    for i in range(row * col):
        # train[i][0] is i-th image data with size 32x32
        image, label_index = data[i]
        image = image.transpose(1, 2, 0)
        r, c = divmod(i, col)
        axes[r][c].imshow(image)  # cmap='gray' is for black and white
picture.
        if label_list is None:
            axes[r][c].set_title('label {}'.format(label_index))
        else:
            axes[r][c].set_title('{}: {}'.format(label_index,
label_list[label_index]))
        axes[r][c].axis('off')  # do not show axis value
    plt.tight_layout()   # automatic padding between subplots
    plt.savefig(filepath)

testrules=rules[consequents]=={cond_fly}
testrule_sort_values([lift],acending=false)
```

The above curve shows K-mean clustering curves for different classes like airplane, automobile, bird and many more. The curve seems to be successfully classifying different classes into different clusters, except few data point which are misclassified in different classes. Overall, K-mean classifier did a great job in classifying different data point in appropriate classes.

## 4 Conculsion

The convolutional network working on the MNIST dataset was adapted to work with CIFAR-10 dataset.

Feed forward network is not recommended for this dataset as it achieves a

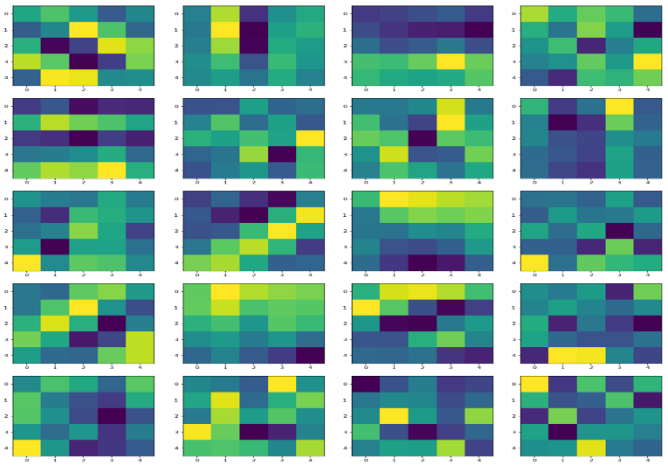

Figure 14: First Convolutional Layer Weights(Last 20 maps)

performance accuracy of 37 percent at best. Since CIFAR-10 data contains color images it is better to use a convolutional neural net since it contains multiple feature maps. This project achieves a best result close to about 74 percent accuracy using three convolutional layers. It still has quite a scope for improvement since Keras and Tensorflow libraries have been used to increase the performance upto 95 percent. While I did not achieve the full results I had originally intended , I have successfully managed to implemented multiple neural network architectures (including CNN) and successfully applied them to object classification problems with good results.

References

[1] https://www.cs.toronto.edu/ kriz/cifar.html

[2] “Fractional Max-Pooling”, Graham, Benjamin, arXiv:1412.6071

[3] ”UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION”, Chiyuan Zhang, Samy Bengio, Moritz Hardt, arXiv:1611.03530v2

[4] http://deeplearning.net/software/theano/

[5] ”Signal Processing and Networking for Big Data Applications”, Zhu Han, Mingyi Hong, Dan Wang

[6] ”michaelnielsen.org ”

[7] ”http://www.wildml.com/2015/11/understanding-convolutional-neural- networks-for-nlp/”

[8] ”https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/ch04.html”

[9] ”https://www.slideshare.net/DaleiLEE/applying-neural-networks-to- cifar10-dataset”

[10] ”http://cs229.stanford.edu/proj2013/ReesmanMcCann- Vehicle20Detection.pdf”