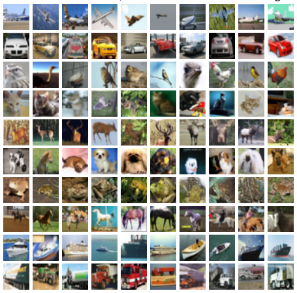
**Technical Summary**

* **Introduction: -**

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. (As this code was run in R, the numbering system in R starts from 1, so our class numbers are from 1-10, rather than 0-9)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **airplane** | **automobile** | **bird** | **cat** | **deer** | **Dog** | **frog** | **horse** | **ship** | **truck** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |



We intend to use the data set to come up with enhanced accuracy to predict the class when an input image data is given to us. With the data that we have, it is a 32\*32-pixel data with 1024-pixel boxes and each box having RGB (Red, Green & Blue), with the total being 3072 data points. So, with 3072 predictors and one response column, along with 50,000 data sets, in this flow we have implemented KNN using HOG (Histogram of Oriented Gradients) to get the best results.

* **Analysis: -**

The analysis starts with the conversion of 1024\*3 data pixels to row major order data frame. We enlist the response column and attach it to the predictor table to check if the values are assigned correctly or not. The different predictors class are numbered from 1-10 respectively. Splitting the main data set into train and test.

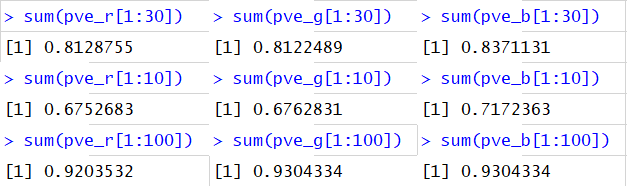
* **Preprocessing: -**

The task of image recognition is difficult to solve by a linear classifier. Every image is represented as a set of pixels, which includes the RGB representation. After some initial runs, we discovered that we need to convert the dataset to make it more relevant for our linear classifier, and hence increase the classification accuracy. As there are 3072 variables, there is a high chance that while modelling our data, we tend to overfit our data and to overcome that we use PCA, which will reduce the variables and allow us to focus on some few variables which explains most of the variance in the data. Thus, to overcome this hurdle we implemented PCA first to get our best components which will help in increasing the prediction accuracy.

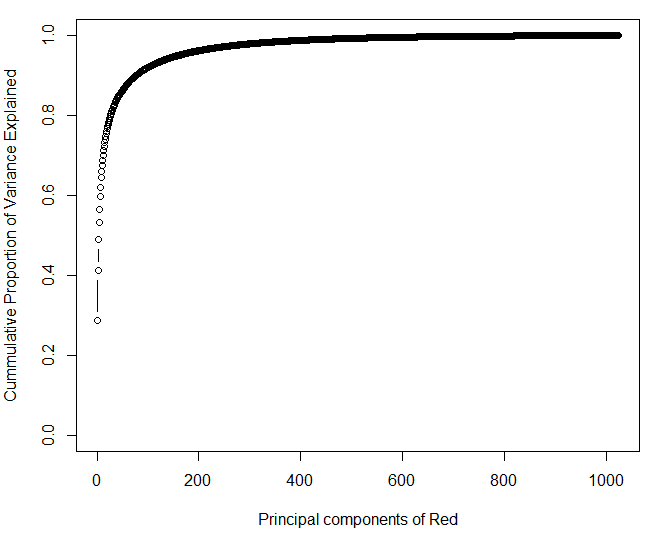
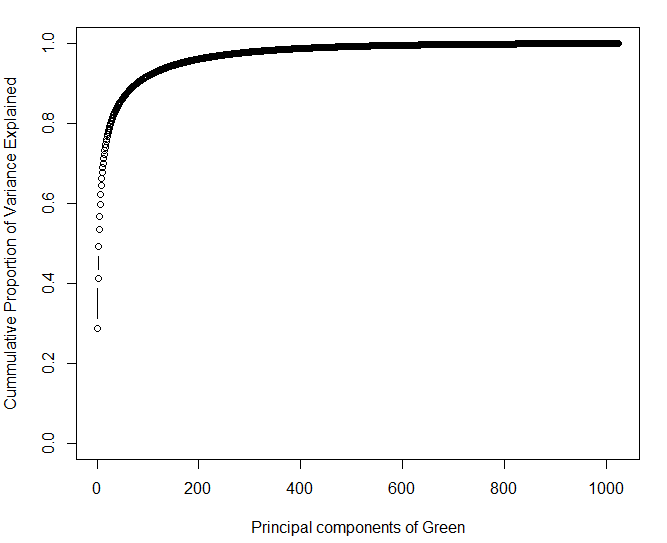
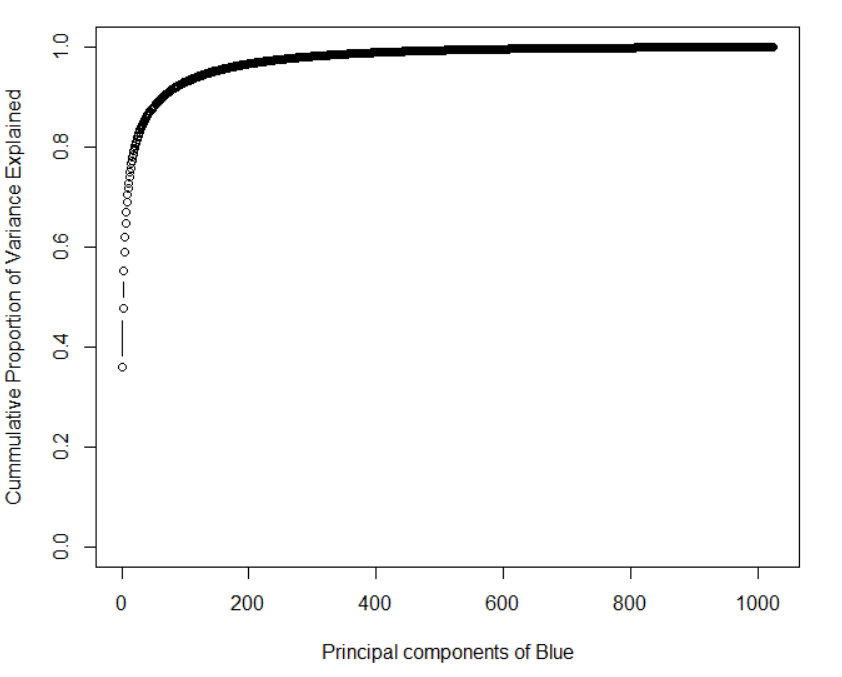
Principal component analysis (PCA) uses an orthogonal transformation to convert a set of observations of possibly corelated variables into a set of values of linearly uncorrelated variables called principal components. It gains a low-dimensional description of a data set that includes as much as possible of the variation. The concept is that each of the 'n' observations lives in p-dimensional space, but not all these dimensions are equally impressive where the concept of interesting is estimated by the amount that the observations vary along each dimension. Each of the dimensions decided by PCA is a linear combination of the 'p' variables.

* **Code Execution for PCA (with QDA)**: -

We executed the code to find the best components from the 3072 (3\*1024) variables which are nearly corelated. Listed below is the value of the variance explained by the first 10, 30 and 100 components of the Red, Green & Blue variables.

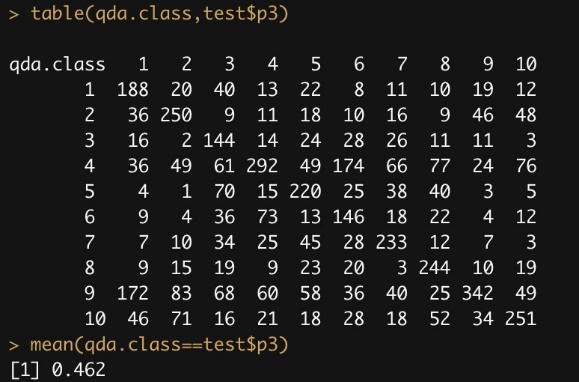
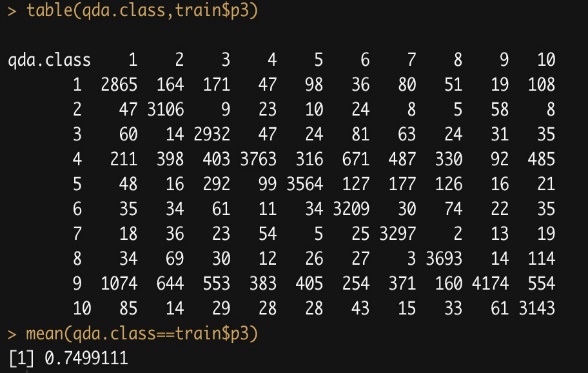
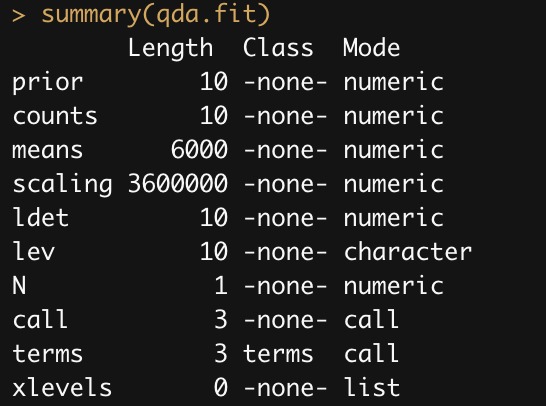
****

Similar data can be related from the graph that is listed below the values. We can observe from the graph as well as the table that nearly 30 components out of 1024 variables are responsible for about 82 percent of variance in the data.

****

Generative models for multi-class models that provide solution with multi-conditional class densities are Linear Discriminant Analysis (LDA) and Quadratic discriminant Analysis (QDA). LDA uses linear decision boundary and QDA uses quadratic decision boundary. So QDA model provides better accuracy than LDA. The data set is split into many different combinations and better combinations for test accuracy are selected. The figure B indicates the true positives and false positives for each class for train data. The figure C indicates the true positives and false positives for each class for test data.

Now implementing the PCA results using the QDA technique to get a quadratic classifier which will help us in distinguishing between the 10 classes present.



(A) (B) (C)

* Training Accuracy – **0.7499** percent; Training Error = **0.2501** percent
* Test Accuracy – **0.462** percent; Test Error = **0.538** percent
* **Code Execution for Kernel KNN with HOG**: -

The histogram of oriented gradients (HOG) is a feature descriptor applied in computer vision and image processing for object detection. The method counts incidents of gradient orientation in localized parts of an image.

The fundamental idea behind the histogram of oriented gradients descriptor is that local object representation and shape within an image can be explained by the distribution of intensity gradients or edge directions. The image is split into small associated regions called cells, and for the pixels within each cell, a histogram of gradient directions is selected. The descriptor is the sequence of these histograms.

For enhanced accuracy, the local histograms can be contrast-normalized by determining a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in more reliable invariance to changes in illumination and shadowing.

The HOG descriptor has a few key benefits over other descriptors. Since it works on local cells, it is invariant to geometric and photometric transformations, except for object orientation. Such changes would only appear in larger spatial regions.

Implementation of the HOG descriptor algorithm is as follows:

1. Divide the image into small consecutive regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels inside the cell.
2. Discretize each cell into angular bins according to the gradient orientation.
3. Each cell's pixel provides a weighted gradient to its corresponding angular bin.
4. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms.
5. A normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

Below is the code for the **HOG descriptor** and in the following table we have described the various parameters that we have taken in for consideration from our master data table.

hog = HOG\_apply(x, cells = 6, orientations = 9, rows = 32, columns = 96, threads = 6)

|  |  |  |
| --- | --- | --- |
| **Object** | **X** | a matrix, a data frame, a 3-dimensional array or a path to a folder of files (images) |
| **Cells** | **6** | the number of divisions (cells) |
| **orientations** | **9** | number of orientation bins |
| **Rows** | **32** | a value specifying the number of rows of each image-row of the matrix (required if object is a matrix) |
| **Columns** | **96** | a value specifying the number of columns of each image-row of the matrix (required if object is a matrix) |
| **Threads** | **6** | the number of parallel cores to use |

Similarly, the code for **Kernel KNN cross validation function** is described below and the following table describes the values corresponding to our model.

fit\_hog = KernelKnnCV(hog, y, k = 20, folds = 4, method = 'braycurtis',

weights\_function = 'biweight\_tricube\_MULT',

regression = F, threads = 6, Levels = sort(unique(y)))

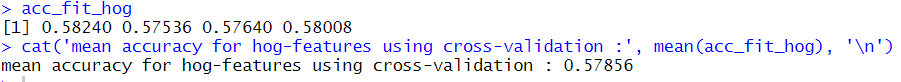
|  |  |  |
| --- | --- | --- |
| **Arguments** | | |
| **data** | **Hog** | a data frame or matrix |
| **y** | **Y** | a numeric vector (in classification the labels must be numeric from 1:Inf) |
| **k** | **20** | an integer specifying the k-nearest-neighbors |
| **folds** | **4** | the number of cross validation folds (must be greater than 1) |
| **method** | **braycurtis** | a string specifying the method. Valid methods are 'euclidean', 'manhattan', 'chebyshev', 'canberra', 'braycurtis', 'pearson\_correlation', 'simple\_matching\_coefficient', 'minkowski' (by default the order 'p' of the minkowski parameter equals k), 'hamming', 'mahalanobis', 'jaccard\_coefficient', 'Rao\_coefficient' |
| **weights\_function** | **biweight\_tricube\_MULT** | there are various ways of specifying the kernel function. |
| **regression** | **F** | a boolean (TRUE, FALSE) specifying if regression or classification should be performed |
| **threads** | **6** | the number of cores to be used in parallel (openmp will be employed) |
| **Levels** | **sort(unique(y)** | a numeric vector, because this is classification data, the unique levels of the response variable are necessary |

We ran the code for different cells and analyzed the prediction accuracy that we were achieving. We observed that if we alter the cells, we were able to achieve high accuracy.

* **For cell value = 6, 4 – fold cross validation**

* **For cell value = 8, 4 – fold cross validation**



* Therefore, from the above data we have accuracy of **57.856** percent.
* **Code Execution for Convolutional Neural Network**: -

In order to handle the multi class classification problem, one hot encoding must be done for image.lab. So, images.lab list is converted to one hot vector with 10 classes and saved as array of dimension (50000,10). Now, we split our 50000 images into 3 parts: train (90%), validation (5%) and test data (5%).

After splitting the data, we feed the training data into a convolutional Neural Network. The architecture of the network is as follows:

1. The whole network is sequential that means output of previous layer is taken as input in next layer.
2. The first layer is a ConV2\_d layer with 32 filters of dimension 3 \* 3. A stride of 1 with padding is used. Activation function used for this is Relu. This layer takes input images of dimension 32 \* 32 \* 3.
3. The second layer is a ConV2\_d layer with 32 filters of dimension 3 \* 3. Activation function used for this is Relu.
4. Followed by a max pooling layer with pool size 2 \* 2. This pooling layer is to reduce the dimension of feature map in the model.
5. Then a drop out of 0.25 percent is used.
6. Then a layer of ConV2\_d with 32 filters of dimension 3 \* 3 is used. A stride of 1 with padding is used. Activation function used for this is Relu.
7. Again, a layer of ConV2\_d layer with 32 filters of dimension 3 \* 3 is used. Activation function used for this is Relu.
8. This is followed by a layer of batch normalization and a layer of max pulling with pool size 2 \* 2.
9. A dropout of 0.25 percent is used.
10. Now all the inputs are flattened into a large vector and passed to a fully connected layer with 512 hidden units and activation function as Relu.
11. Again, a dropout of 0.5 percent is used.
12. Now, as a last layer, fully connected layer is used with hidden unit as 10, which is equivalent to number of classes. The activation function used here is Softmax.

Here we have used Adam Optimizer with different learning rate and decay, to train a model. Loss function used here is categorical cross entropy loss. Hyper-parameter tuning for the following parameters were done in the following manner: -

* Epoch
* Batch
* Learning rate

One Epoch is when a full dataset is passed forward and backward through the neural network only once. In the neural network terminology:

* **one epoch:** one forward pass and one backward pass of all the training examples
* **batch size:** the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.
* **number of iterations**: number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes). E.g. if you have 200 training examples, and your batch size is 50, then it will take 4 iterations to complete 1 epoch.

And then finally our model was trained with the training dataset (90%) and further it was validated and tested on validation and test data (5 percent each). The accuracy was measured by counting the measure in test data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Batch size** | **Learning rate** | **Training Accuracy** | **Validation accuracy** |
| 20 | 32 | 0.1 | 0.4813 | 0.4641 |
| 20 | 32 | 0.001 | 0.5598 | 0.5372 |
| 20 | 10 | 0.1 | 0.5024 | 0.4863 |
| 20 | 10 | 0.001 | 0.5692 | 0.5329 |
| 40 | 32 | 0.1 | 0.6875 | 0.6634 |
| 40 | 32 | 0.001 | 0.7778 | 0.7345 |
| 40 | 10 | 0.1 | 0.7024 | 0.6813 |
| 40 | 10 | 0.001 | 0.7892 | 0.7392 |

**Reported Test accuracy: 0.7187** for epoch 40, 10 batch size and learning rate 0.001.

There is scope of improvement with following case:

* The model can be run with more epoch.
* Data augmentation can be done for input images.

**Comparison of models and specifying best model for testing**

We tried different models in the whole project duration, and we concluded that we are getting better accuracy with three models which are Convolutional Neural Network (CNN), Kernel K- nearest neighbors with Histogram of Oriented Gradients (HOG) and Quadratic Discriminant Analysis (QDA) with Principal Component Analysis (PCA).

In the QDA model we first performed dimension reduction using Principal component Analysis. PCA seeks a small number of dimensions that are as interesting as possible, where the concept of interesting is measured by the amount that the observations vary along each dimension. Though we were able to reduce the dimension of the data set and our accuracy also increased but it was not good enough. Classification accuracy of Quadratic Discriminant Analysis (QDA) with Principal Component Analysis (PCA) is ranged between 46.9% and 50.2%.

Then, we found out the concept of Histogram of oriented Gradients (HOG) which is feature descriptor used with kernel KNN CV for image processing where HOG looks for group of cells rather than comparing cell by cell. This enables us to look for group of cells which are similar in characteristics and gives results based on comparison. By increasing cells, we can increase the classification accuracy of model. Classification accuracy of Kernel KNN with Histogram of Oriented Gradients (HOG) is ranged between 55% to 58%.

Finally, our best model was Convolutional Neural Network, because of the advanced learning model, neural network was able to bring down the current error rate and enhance the accuracy to a higher level. The neural network can reduce the error rate with every epoch and back propagation, in which the model learns and is able to predict the class with a higher accuracy. So, with the neural network model we were able to achieve an accuracy of 71.87 percent.

**Results**: -

We concluded that Convolutional Neural Network works best with our image processing dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **QDA with PCA** | **KNN with HOG** | **CNN** |
|
| **Accuracy** | 46.20% | 57.85% | 71.87% |
|