# Executive Summary

**To**: President X

**From**: Data Consultant Y

**Subject**: Analysis of color-coded images to predict the actual object in it, Executive Summary

Image detection is a very vital feature that computers these days are able to perform that have eliminated the risks of spam, identify most important interactions and help marketers to focus on the most significant data. It is also important in robotics such as driverless vehicles and face recognition where the images are expected to be classified better, faster and accurately. This can also be used by various brands and marketers to offer products according to the human type and create better experience for them. In this project also, the team was assigned the tasks to analyse the color-coded images in RGB, examine the accuracy of different classification models and predict the object in there so that this system can be used on other images to predict the other items.

To achieve this many different algorithms were used. There were 50,000 images available that were needed to be trained. The images were known to contain the images of at least one of the following objects: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Among all the models used CNN classified the data with small number of misclassifications. Simpler models like trees and Logistic Regression were able to classify some classes relatively well such as airplanes and ships but strongly misclassified cats. Our best model was Convolutional Neural Networks, which has a highly distinguishable diagonal on confusion matrix, indicated high classification rates.

One of the problems faced by the team was the amount of dataset and then lack of computational power to fit the models and hence the models took more than required time to run for the training data. One of the tasks we made easy was to use neural networks to automatically extract features from images by using adjacent pixel information with a window matrix and creating a convolution layer to learn patterns in images to better classify the dataset. Other than that decision trees are good classifiers because of their robust nature to outliers, accuracy and do not suffer from over fitting. But they definitely take a long time to run.

The contribution of each team member is as follows. Bhavishya developed a model using decision trees, Joel developed a model using Neural Networks, Zaid developed a model using Linear Discriminant Analysis and Xin developed a model using k nearest neighbour and kernel knn. Each one of us ran the training data for our models and we unanimously decided the 3 models that produced the least errors as our recommended models. The comparison of all the models was written by Xin which included which model did what and our reason for choosing CNN as the best model after comparing the all 3 models. Bhavishya wrote the executive summary. Joel and Zaid performed testing on the best model with the data provided for testing.

# Model - CNN

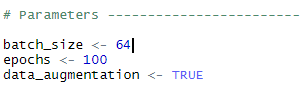
CIFAR-10 is one of the most widely used dataset with 50000 images as training data and 10000 images as the test data. For processing image data, Convolutional Neural networks are without doubt the go to modelling technique. The advantage of CNN is it automatically detects important features on the images without human intervention and hence gives very good accuracy scores.

Keras allows us to use Tensorflow backend in r with an API. We installed Keras and Tensorflow and imported their respective libraries. Keras also has easy access to the CIFAR-10 dataset within its libraries that helps us directly import the training set for our model.

The shape of the training data is as following: 

Next we will normalize the training dataset by dividing the RGB codes by 255 so that we scale down the values between 0 and 1 such that no value gets higher weight than required to prevent exploding gradients.

We set the parameters of the neural network to have the required step size and epochs:



Our input image is a tensor whose width is 32 pixels and height is 32 pixels with 3 channels representing RGB (red, green, blue) color intensities. Thus, we need to define a model which takes (None, 32, 32, 3) input shape and predicts (None, 10) output with probabilities for all classes. None in shapes stands for batch.

Simple feed-forward networks in Keras can be defined in the following way:

We will Stack 4 convolutional layers with kernel size (3, 3) with growing number of filters (16, 32, 32, 64) with padding size of same so input and output images will have same dimensions.

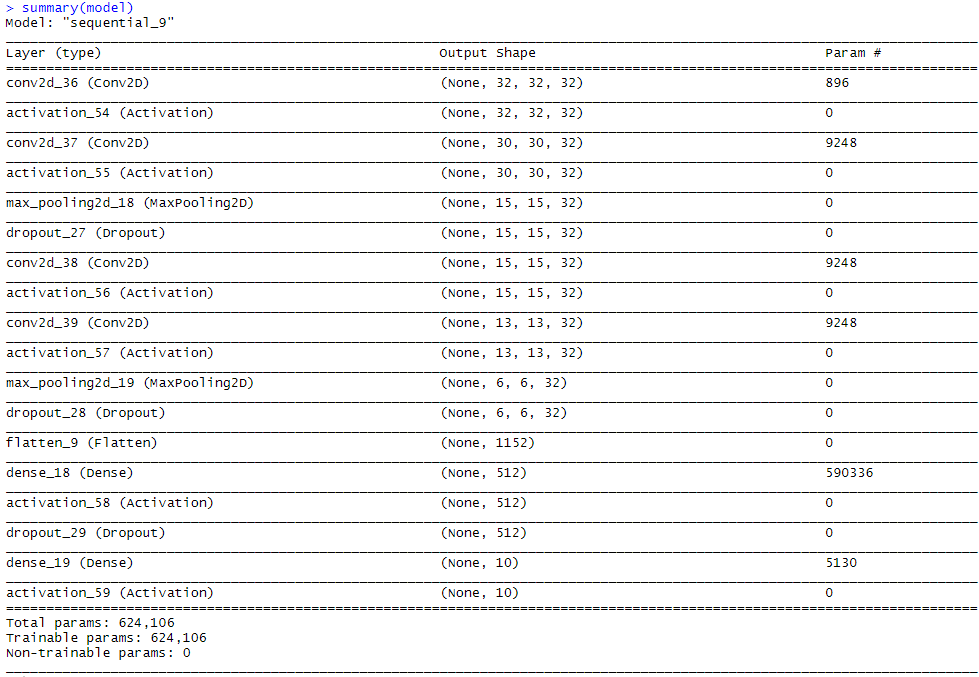
We will add 2x2 pooling layer after every 2 convolutional layers (conv-conv-pool scheme). We use the ReLU activation function.

After adding all conv-conv-pool scheme layers we will add a dense layer with 512 neurons and a second dense layer with 10 neurons for classes. We used a Flatten layer before first dense layer to reshape input volume into a flat vector.

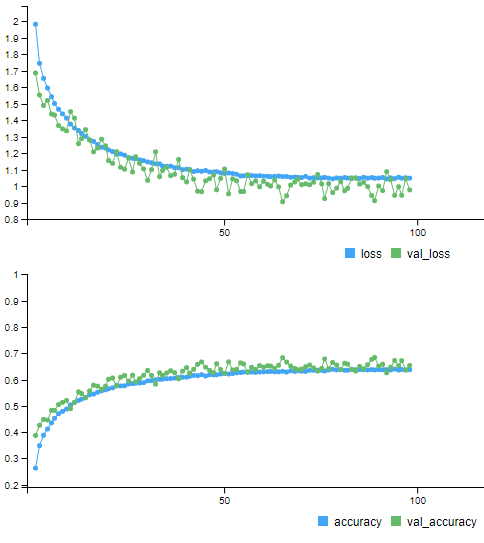
Dropout layer after every pooling layer will be used to reduce overfitting in the model with setting the fraction value to 0.25 so that 0.25 fraction of units will be dropped.

We train the data with our respective hyperparameters along with RMSprop optimizer as opposed to gradient descent as this optimizer avoids vertical oscillations.

The model summary is as below with all the necessary layers and their corresponding values:



The accuracy and loss of the model is given below along with the validation set



We get maximum accuracy of 0.6866 for the 74th epoch and minimum cross entropy loss of 0.9139.

# Model: KNN and KernelKnn

**K-Nearest Neighbor**

Firstly, we considered K-Nearest Neighbor (KNN) as our target model, since it could lead to lower bias at the cost of higher variance. The downside to KNN is that the high dimensionality of the dataset affects the predictive power of the model. Also, KNN is non-parametric, which means the model does not explain the relationship between the predictors and the response. Missing data might introduce high bias to the predictors.

After running the KNN model, the accuracy for each k values remains at around 29%, which are shown below. We choose K = 10, as it produce the high accuracy with reasonable less computational work.

图片包含 文字, 地图

描述已自动生成

Fig.1 Accuracy for KNN Models with Different K Values

Even though KNN is considered as one of the simplest algorithms in machine learning, this accuracy is low.

We then use HOG features of the cifar-10 data to train the model, which gives us the accuracy of k =10 as shown below. According to the results, the highest accuracy for k=10 is about 41%.

图片包含 文字, 地图

描述已自动生成

Fig.2 Accuracy for KNN Models with K = 10.

**KernelKnn**

Then we found that there is a released package named KernelKnn on CRAN, which can give us a better result than KNN. The simple KNN algorithm can be extended by giving different weights to the selected k nearest neighbors, while the KernelKnn package is to use different weight functions (kernels) in order to optimize the output predictions in both regression and classification.

After running the KernelKnn model with HOG features, we have the accuracy improved from 41.7% to around 59.4% with K = 10.



Fig.3 Accuracy for Predicting Cat with K = 10 KNN Model.



Fig.4 Accuracy for Predicting Cat with K = 10 KernelKnn Model.

# Model: Decision Trees

We considered decision trees as our target model because it would split the pixels into regions and thereby help us leverage multiclass classification. We started with splitting the data into 2 parts: training 75% and testing 25%. The Rpart function is used to train the data into multiple classes with 10 fold cross validation.

A screenshot of text

Description automatically generated

A screenshot of a cell phone

Description automatically generated

We visualized the fitted model using Rpart and it lets us know the nodes, including the leaf nodes, how the data is spitted, Y value and the corresponding probabilities.

Below is the picture of the fitted model.

A close up of text on a white background

Description automatically generated

Also, please find below the prediction model. The accuracy of the model is around 23% by comparing predicted values with actual values.

A screenshot of a cell phone

Description automatically generated

# Best Model Comparison

CIFAR-10 is one of the most widely used dataset with 50000 images as training data and 10000 images as the test data. Since the images in CIFAR-10 are low-resolution (32x32), this dataset can allow researchers to quickly try different algorithms to see what works.

Among all the models tested, ***Convolutional Neural Network*** (CNN), ***K-Nearest Neighbor*** (KNN) and ***Decision Tree*** models stand out as the top 3 models based on the performance measures of the models. We considered the *accuracy* as the standard for comparing the models, since it is the most intuitive way to show the prediction quality of the model.

Table 1 shows the three models’ accuracy that can be used for comparison. From the table blow we can see that the CNN model has the highest accuracy, while random forest model produces the lowest.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **CNN** | **KNN** | **Decision Tree** |
| **Accuracy** | 0.653 | 0.417 | 0.23 |

**Table 1. Accuracy Comparison for all Three Models**

We also consider the complexity of operating the model as one standard for selecting the best model. The CNN model provides easy feature extraction automatically without any human supervision. And as compared to other neural networks, CNN also provides a feature called weight sharing, which allows it to provide near human accuracy at the cost of longer computation time. Also, features such as max pooling, helps in reducing the number of neurons required for classification.

Since the CNN provides the best accuracy score, we choose CNN as our best model.

# Test Point on Best Model with Test Errors

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Final Test Accuracy after the 100th epoch is 0.6530

Final test loss after 100th epoch is 1.0014 (Cross Entropy Loss)

# Improvements:

For improvements we improved the epoch to 200 and reduce batch siez to 32 but it did not show significant improvement apart from taking a long time. Also, we tried different optimizers like RMSProp, Adam Optimizer, AdaBoost but the RMRProp performed best.