ISEN 613 - PROJECT REPORT

Bhaskar Shenoy (729007540),

Srujan Jabbireddy (430000474),

Parthasarathy Ramamoorthy (529005616),

Jayesh Patil (230002375),

Siddhant Deshmukh (430000713)

Executive summary

From facial recognition to unlock our smartphones to self-driving car technology, from automatic image tagging in social media technology to detecting abnormalities in the manufacturing products, there is a high focus on building machine learning techniques for image recognition. In an effort to understand the concepts of image recognition and explore various methodologies behind that, the team was tasked to build three models to predict the object in a set of images. The data set given for this project has been divided into 10 class objects in random order with 5000 images per class totaling up to 50,000 color images. The team has made an effort beyond the concepts taught in the class and create a better algorithm for image classification.

The team has researched extensively to learn about how the machines can read images and how to convert the images into machine-readable language. Using R, a statistical analysis freeware, various concepts taught in the curriculum namely Linear Discriminant Analysis, Quadratic Discriminant Analysis (QDA), Support vector machines, Random forest techniques were implemented. Out of the three methods reported, through QDA and Random Forest an accuracy of 44.5% and 46.5% was achieved respectively. On further research, the team came across various other techniques and learned that neural networks are able to provide better results and hence proceeded to use Convolutional Neural Networks (CNN). **With CNN, training accuracy of 87.4% and test accuracy of 84.87% was achieved**.

CNN has become an important fundamental deep learning technique and there has been intensive research going on to develop more for image classification and object recognition. The advantages of CNN are that they are able to study each part of the images and capture minute details of the images. They extract important and useful attributes and be able to predict the images with higher accuracy. Even though they have advantages, they still have disadvantages. They need a lot of training data and have a high computational cost. Without proper GPU, they even take a lot more memory and are quite slow to train.

More advanced techniques have been developed for image classification and object recognition, which have been able to achieve more than 95% accuracy on this data set. With our limited time and computation power, we have been able to implement the fundamental techniques used in image recognition and achieve an accuracy of nearly 85%. We further hope to improve this model using advanced techniques and learn them in more depth.

All the team members have contributed equally in all aspects of the project.

Thank you

TECHNICAL SUMMARY

Overview

This report explains in detail the approach towards solving the image classification problem that was presented to the group. The report will describe the preprocessing of data, the three best methods used for solving the problem, the way different parameters were chosen and the estimated test accuracy for each method.

The three best models explained in the report are:

1. Quadratic Discriminant Analysis (QDA)
2. Random Forest
3. Convolutional Neural Network (CNN)

Dataset

The group was presented with 50000 labeled images of 10 different classes in random order. Each image is made up of 32X32 pixels with RGB channel for each. When expanded 3 levels of red, green and blue is obtained for each pixel. Hence each image can be represented with 3072 variables. This is a very large number and carrying out any training on this is computationally expensive.

Preprocessing

Since our data contains 3072 variables, Principal Component Analysis (PCA) comes in handy to reduce the dimensionality of this data. Before implementing PCA the Red, Green and Blue levels are stored on different data frames. PCA is performed separately on these three data frames. This ensures that the variance with terms of Red, Green and Blue are captured separately.

The *prcomp* method in R is used to implement PCA. This gives 1024 principal components for each Red, Green and Blue data frame. Each component explains certain amount of variance in the data in a decreasing order such that the first principal component explains maximum variance and the last principal component explains minimum variance. The scree plot is one way to select the number of principal components to be chosen for training.

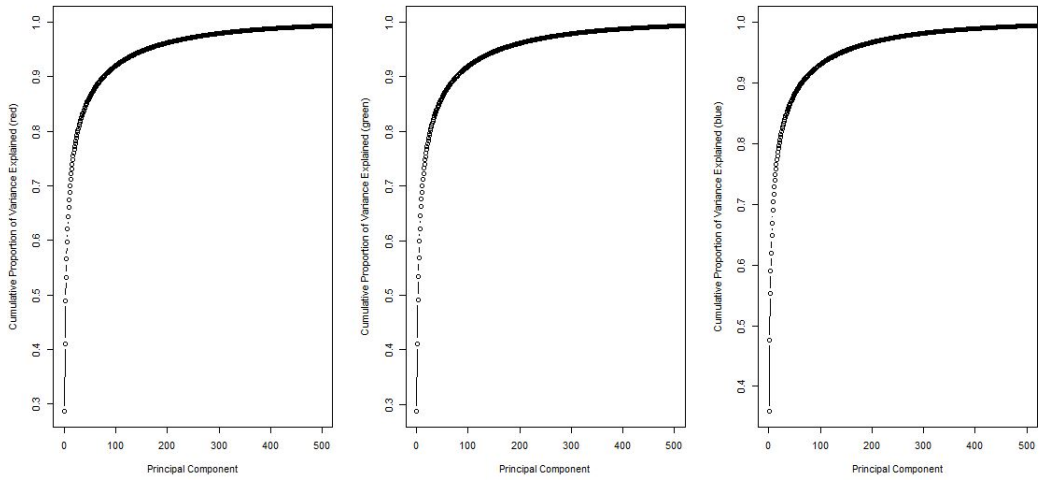


Fig. 1: scree plot for Red, Green and Blue PCA

Fig. 1 shows the scree plot i.e. the cumulative proportion of variance explained versus the number of principal components considered. From the scree plots it was evident that first 75 principal components explain more than 90% of the variance in Red, Green and Blue channels. Hence, we use the first 75 components of each channel to form a new dataset with 225 variables for training our models.

Statistical Models

Several classification models like one-vs-all Logistic Regression, LDA, QDA, SVM, Classification trees, CNN were considered and the summary of best thee models based selected based on its prediction accuracy and training time is reported below.

Quadratic Discriminant Analysis

As the training data contains images already differentiated into different classes, discriminant analysis is one of the most inexpensive learning methods with respect to training time, that can be easily deployed for classification.

QDA separates classes with a non-linear surface. Although the number of variables considered are 75*3 i.e. 225, there is still a possibility of the number of principal components used for building the model affecting the accuracy of the model. Hence the QDA model built was trained for different number of principal components from red, green and blue channels each.

Due to the limitation of computations the model was tested on 10, 20, 30 40, 50, 60, 70, 80, 90 & 100 principal components from each channel. The results are summarized in fig.2.

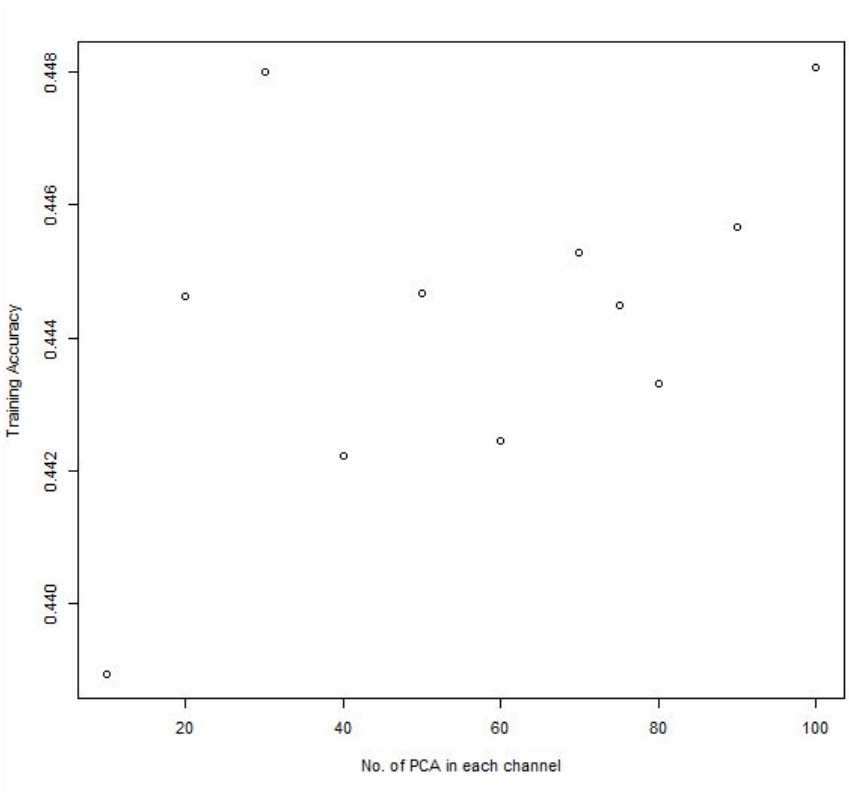


Fig 2: CV Accuracy vs. No. of PCA in each channel

It can be seen from fig. 2 that the Cross Validated accuracy is almost same when 30 PCA from each channel are considered and when 100 PCA from each channel are considered. The final model is built by using 30 PCA from each channel as it is computationally fast and also provides similar or higher accuracy than the models with more PCAs.

Increasing the number of PCAs considered to build the model has a similar effect as increasing the predictors in regression, it reduces the bias but increases the variance. Hence when an optimal amount of PCAs are considered there is a decrease in variance with slight increase in bias. This trade off helps in increasing the accuracy.

The model trained by considering the factors above had a cross validated error rate of 55.22%. The time for training the model with 10-fold cross validation in a system with i5 9^{th} Gen Processor with 8 logical cores and 12 GB RAM was 13 seconds.

Scope for improvement:

QDA can be further tuned by performing a ROC analysis to set the prior probabilities. The multi-class ROC analysis in the current scenario is slightly complicated as there are 10 classes and 10 prior probabilities to deal with.

Random Forest

Random forest is an ensemble of individual decision trees. Each tree predicts a class value for input data, and the final class prediction is the class with a majority vote. The major advantage of Random Forest is it is based on numerous uncorrelated trees. Hence it outperforms a single decision tree and also bagged tree where the trees tend to be highly correlated.

Random forest can be tuned on several parameters, out of which we have considered *mtry* and *ntree* in our model. *mtry* is the number of predictors to be considered when each tree is built with the random forest and *ntree* is the total number of trees in the ensemble. It is always better to tune the model on the complete data available for training. But due to the processing constraints a small dataset of 2000 datapoints are sampled out of the training dataset for tuning.

Initially the random forest was analyzed for different values of *mtry*. The algorithm by default takes p/3 i.e. 75 in our case as the *mtry* value. We built several models with *mtry* values in the range 1 to 30. The results are shown in Fig 3.

It was observed that *mtry* value of 18 had the least prediction error. Once the *mtry* value was set to 18, the random forest model was tuned for *ntree*. The values of *ntree* considered were: 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600. The results of the tuning are shown in Fig.4. It was observed that the random forest with ntree value 300 had the least prediction error rate.

The final model is built on whole data by taking *mtry*=18 and *ntree*=300, on cross validation error rate obtained for the model is 57.78%. The time for training the model in a system with i5 9th Gen Processor with 8 logical cores and 12 GB RAM was 15 minutes and 33 seconds.

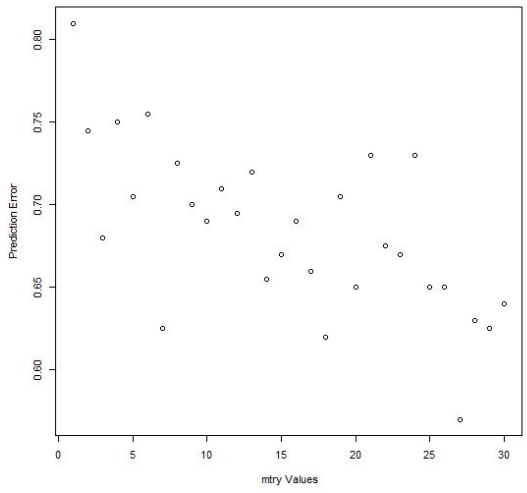


Fig. 3: Prediction error rate vs. mtry values.

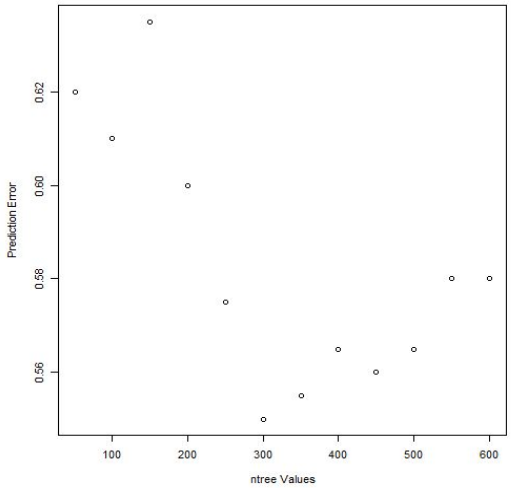


Fig. 4: Prediction error rate vs. ntree values.

Convolutional Neural Network

Convolutional Neural Networks is one of the major methods used in image classification & image recognition. In this method, it takes in the images as input & classifies it into classes according to their RGB labels. Neural networks consist of an input layer, a hidden layer, and an output layer. Each neuron receives several inputs, takes a weighted sum and bias over them, passes it through an activation function and responds with an output.

In the CNN, the algorithm looks for low-level features like edges and curves and starts building more abstract concepts through a series of convolution layers. CNN's are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, passes it through an activation function and responds with an output. An image passes through a series of convolutional pooling and connected layers to get an output.

The following image gives a clear idea behind the working of the CNN:

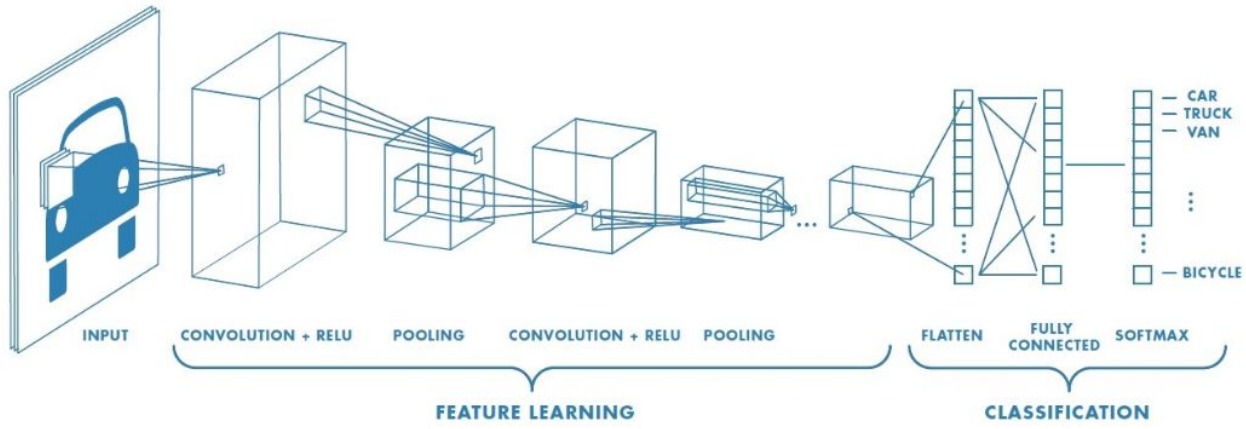


Fig. 5: Working of a Convolutional Neural Network

Source: medium.com

For the current problem Keras library is used. Models in Keras come in two forms, Sequential and Functional API. The Sequential Model is used as it allows us to stack layers like recurrent in order from input to output.

Next, a 2D convolutional layer is added to process the image inputs. The first layer is provided with 32 output channels with kernel size 3x3 [filter X layer] with Rectified Linear Unit (ReLU) activation. Another 2D convolution layer with a 3x3 kernel size is provided. Next, a 2D max-pooling layer is added with a 2x2 size of pooling in X & Y directions in order to reduce the dimensionality.

The output from the above layer is 32x32x32. Next, another 2D convolution and pooling layers with 64 output channels. The output from this layer is 32x32x64. Next, another 2D convolution

and pooling layers with 128 output channels. In the view of experimenting with our model and tuning for a higher frequency, we have used batch normalization techniques in an effort to stabilize the learning and accelerate the process. After every convolution layer, in an effort to increase the regularization we have provided with a dropout rate in increasing rate in order to prevent overfitting, where the mentioned rate of fraction nodes of units will be dropped.

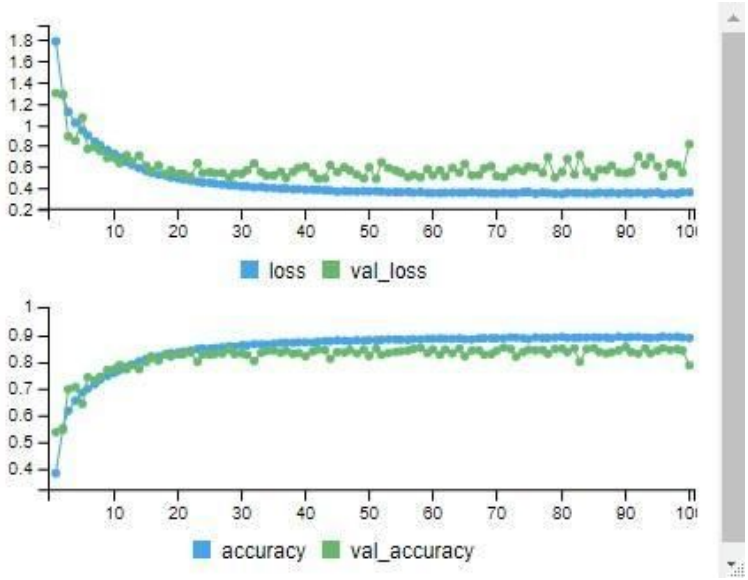
We have built six convolution layers with kernel_size = 3x3 with growing number of filters (32,32,64,64,128,128). Next, we have to flatten the output fully to enter fully connected layers. We define our connected layer with Dense() layer with size 512 nodes, each activated by a ReLU function and softmax() classification with the number of classes size in our output layer.

After building the architecture, we have to build a compiler. Keras provides many loss functions and we preferred to use the standard categorical_crossentropy. For the optimizer, we preferred to use RMSprop (which divided the gradient by a running average of its recent magnitude) with a learning rate = 0.001. We have also enabled the accuracy calculating operation in order to assess the accuracy.

After building the Keras CNN model, it was time to train our model. We divided the training and testing data in 80:20 ratio with a batch size =28, the number of training epochs = 100 and setting verbose =2 for viewing the progress of model fitting.

After computing for 100 epochs which took around 7hrs of run time, the average classification val_accuracy achieved was 83% and training classification accuracy was 86%. These can be seen in the below figure.

The loss and accuracy graphs for training and validation sets can be seen below:



Summary of the architecture built can be seen below:

```
> summary(model)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 30, 30, 32)	9248
activation_1 (Activation)	(None, 30, 30, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 30, 30, 32)	128
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_2 (Conv2D)	(None, 15, 15, 64)	18496
activation_2 (Activation)	(None, 15, 15, 64)	0
batch_normalization_2 (BatchNormalization)	(None, 15, 15, 64)	256
conv2d_3 (Conv2D)	(None, 13, 13, 64)	36928
activation_3 (Activation)	(None, 13, 13, 64)	0
batch_normalization_3 (BatchNormalization)	(None, 13, 13, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
conv2d_4 (Conv2D)	(None, 6, 6, 128)	73856
activation_4 (Activation)	(None, 6, 6, 128)	0
batch_normalization_4 (BatchNormalization)	(None, 6, 6, 128)	512
conv2d_5 (Conv2D)	(None, 4, 4, 128)	147584
activation_5 (Activation)	(None, 4, 4, 128)	0
batch_normalization_5 (BatchNormalization)	(None, 4, 4, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
dropout_2 (Dropout)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
activation_6 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130
activation_7 (Activation)	(None, 10)	0

```
Total params: 556,586
Trainable params: 555,690
Non-trainable params: 896
```

Comparison

The three best models summarized above are compared with each other in the following aspects:

1. Pre-Processing
2. Training time
3. Cross Validated Error Rate

Pre-Processing: Whenever a preprocessing is used before building a model, during testing, the test data needs to be transformed in the same way in order to use it for prediction. If it is not done, the model provides very inaccurate predictions. So a model that requires less pre-processing is always preferred.

Training Time: Training time is a huge factor to be considered if a model has many parameters to be tuned. It would be really cumbersome to train a model if it takes a huge amount of time each time a parameter is changed. hence a model that takes less time to train is preferred.

Cross-Validated Error Rate: Most of the time the ultimate goal of a statistical learning method is to produce accurate results. hence a model with the lowest error rate is always preferred.

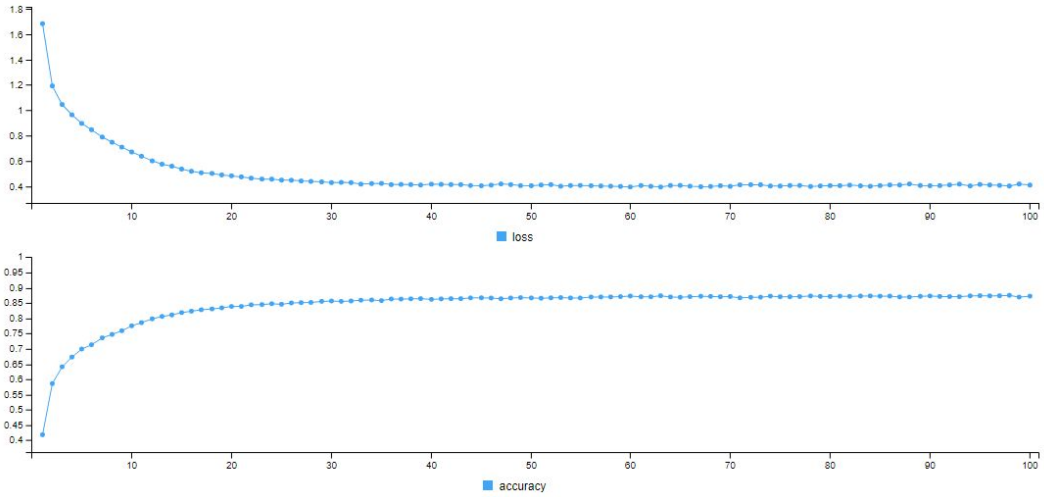
Method	**Preprocessing Method**	**Time for pre-processing 50000 datapoints**	**Pre-processing on test data**	**Training Time**	**CV Error Rate**
QDA	PCA	~5 Minutes	Required	13s	55.32%
Random Forest	PCA	~5 Minutes	Required	15.5 Min	57.80%
Convolutional Neural Network	Not Required	-	Not Required	7 Hours	17%

Considering all the factors mentioned above we can say that CNN is the best method for this problem. Although the training time is very high, the accuracy that it provides overshadows it. If a GPU is used then the train time can be further reduced.

Evaluating the test points on our best model and reporting of test error

The best model among our 3 trained models is the Convolution neural network with 6 layers. We have achieved a training accuracy of 84% on 40000 images and testing accuracy of 83% with the initial data set provided.

When the test data has been provided we trained the CNN model again using all the 50000 initial data set. This took us around 5 hrs of computation time on i7 laptop with 12GB ram. We achieved an 87% training accuracy on this data. Below shown in the image is the training accuracy and the loss graph ran for 100 epochs with 32 batch sizes.



Using this trained model, we have tested on the new data and achieved an accuracy of 84.87%.

```
> testdata<- timages.rgb[,,,]
> x_test<- testdata[,,,]/255
> testlabels<- timages.lab[,]
> y_test <- as.matrix(testlabels[])
> y_test <- to_categorical(y_test, num_classes = 10)
> model <- load_model_hdf5("C:\\Users\\bhask\\Desktop\\Fall19\\ISEN613\\Project\\Data For Project\\Data For Project\\first_cnn_model.h
5")
2019-12-06 16:54:57.170607: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is optimized with Intel(R) MKL-D
NN to use the following CPU instructions in performance critical operations:  AVX AVX2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2019-12-06 16:54:57.180105: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with default inter op settin
g: 8. Tune using inter_op_parallelism_threads for best performance.
> results <- model %>% evaluate(x_test_1, y_test_1, verbose =2)
Error in is_tensorflow_dataset(x) : object 'x_test_1' not found
> results <- model %>% evaluate(x_test, y_test, verbose =2)
10000/1 - 9s - loss: 0.4532 - accuracy: 0.8487
> print(paste0("test loss:",results$loss))
[1] "test loss:0.542934631347656"
> print(paste0("test loss:",results$accuracy))
[1] "test loss:0.848699986934662"
>
>
```

```
> print(paste0("test loss:",results$loss))
[1] "test loss:0.542934631347656"
> print(paste0("test accuracy:",results$accuracy))
[1] "test accuracy:0.848699986934662"
>
>
```

Steps to improve our best model

We have built 6 layer Convolutional Neural Network. We have fixed various parameters like using gradient layer dropout, using Refined Linear Unit activation later, with no padding and batch normalization at the end of every two neural layers.

In the initial phases of building this CNN model we started with one layer, two layers experimented with various parameters. All this has been tested on small databases like 5000 images to 10000 images. Our model in the phase-1 submission was one of the best models we could build after various iterations.

But still, there are more improvements which can be done in our model. For optimizer, we have used RMSprop. This is supposed to work better for smaller machines and smaller models. There many other optimizers like Adam, Stochastic Gradient Descent, AdaGrad. These are supposed to take less computational time and even predict with better accuracy.

For better visualization of the test error, we can build a confusion matrix at the end of the testing. A confusion matrix can help in figuring out which images have been predicted wrong and how many times and what they have been predicted wrongly. This helps in learning what the machine has assumed and we can build better models.

There has been continuous research on this data set and recently they have achieved an accuracy of 99% using Giant Neural networks using Pipe parallelism. There have been other methods like Stochastic pooling for the regularization of deep neural networks, recurrent neural network and a combination of RNN and CNN model. CNN is a forward neural network whereas RNN works on the principle of saving the output of layer and feeding it back to the input. RNN can memorize the previous input due to internal memory usage. Combining these models will help the algorithm to capture every minute detail of the image and able to predict with higher accuracy.