**CS 6301.004**

**Project1 (CIFAR 10 - Image Classification)**

**Team Members:**

Prathyusha Kanmanth Reddy(pxk166230)

Teja Kiran Chunduri(txc163430)

Shree Charan Varma Enugandla(sxe170530)

Sneha Nandigala(sxn171430)

Srinivasa Rajesh Ayachitula(sxa172830)

**Image classification on CIFAR- 10 dataset**

In this project we will do a supervised learning on the CIFAR – 10 data. It consists of 60,000 32x32 colour images in 10 classes, with 6000 images per class. There are 50,000 training data images and 300,000 test data images of which 10,000 images are actual test data, rest of the 290,000 images are junk data to prevent manual labelling.

The following report focuses on the methods used for classifying airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck, gauging the performance by calculating the accuracy for different parameters.

**Implementation**

Our test data consists of 300,000 images of which only 10,000 images are authentic, and rest are trash introduced by Kaggle to prevent cheating by manual labels. Importing 300,000 images to UTD server takes a lot of time (approx. estimated 18 hours). But, if we decide to download a zip instead, the uploading the 300,000 images to cloud would take approximately 2 – 3 hours just for the job to get submitted. Owing to these limitations, we have decided instead to split the train data into train and test (45,000 train images and 5,000 test images).

The images are of 32 x 32 format. Since they are small enough already, we did not see any reason to resize into smaller images.

**Normalizing pixels**: Performed linear normalization of RGB image, to make each pixel to have similar distribution. This will help the Stochastic Gradient Descent to descend faster because a single learning rate will have a proportional influence in all regions of Image. Also normalizing would remove the noise and accountable variance in an image.

The categories of images are of varied nature, to reduce the number of attributes, we have converted each image into grey scale image from RGB images. The data is then converted into vectors, divided into train data and train labels, then transformed into a matrix suitable to be fed into deep CNN network.

**Hyper parameter tuning**: To leverage the computation speed and efficiency of cloud. We have decided to implement a grid search to get the best accuracy possible in the given range of parameters. Surprisingly, this took a significant amount of time and failed unable to handle the computation intensity. We then proceeded with giving an array of values, feeding different parameters each time.

The rest of the report concentrates on the results obtained and the limitations and challenges we’ve faced with the data (like data loading).

1. **Results**

|  |  |  |
| --- | --- | --- |
| Iteration | Parameters | Train and Test Accuracy |
| 1 | |  |  | | --- | --- | | Convolution 2D Layer 1 Filter | 32 | | Convolution 2D Layer 1 Kernel Size | 3 | | Convolution Layer 1 Activation | "relu" | | Convolution 2D Layer 2 Filter | 32 | | Convolution 2D Layer 2 Kernel Size | 3 | | Convolution Layer 2 Activation | "relu" | | Max Pool Layer 1 Size | 2 | | Dropout Layer 1 | 0.25 | | Convolution Layer 3 Filter | 32 | | Convolution Layer 3 Kernel Size | 3 | | Convolution Layer 3 Activation | "relu" | | Convolution Layer 4 Filter | 32 | | Convolution Layer 4 Kernel Size | 3 | | Convolution Layer 4 Activation | "relu" | | Max Pool Layer 2 Size | 2 | | Dropout Layer 2 | 0.25 | | Dense Final | 512 | | Activation Final | "relu" | | Dropout Final | 0.5 | | Train = 95.33%  Test = 76.79% |
| 2 | |  |  | | --- | --- | | Convolution 2D Layer 1 Filter | 32 | | Convolution 2D Layer 1 Kernel Size | 2 | | Convolution Layer 1 Activation | "relu" | | Convolution 2D Layer 2 Filter | 32 | | Convolution 2D Layer 2 Kernel Size | 2 | | Convolution Layer 2 Activation | "relu" | | Max Pool Layer 1 Size | 2 | | Dropout Layer 1 | 0.3 | | Convolution Layer 3 Filter | 32 | | Convolution Layer 3 Kernel Size | 2 | | Convolution Layer 3 Activation | "relu" | | Convolution Layer 4 Filter | 32 | | Convolution Layer 4 Kernel Size | 2 | | Convolution Layer 4 Activation | "relu" | | Max Pool Layer 2 Size | 2 | | Dropout Layer 2 | 0.3 | | Dense Final | 512 | | Activation Final | "relu" | | Dropout Final | 0.4 | | Train = 94.94%  Test = 74.7% |
| 3 | |  |  | | --- | --- | | Convolution 2D Layer 1 Filter | 32 | | Convolution 2D Layer 1 Kernel Size | 4 | | Convolution Layer 1 Activation | "relu" | | Convolution 2D Layer 2 Filter | 32 | | Convolution 2D Layer 2 Kernel Size | 4 | | Convolution Layer 2 Activation | "relu" | | Max Pool Layer 1 Size | 2 | | Dropout Layer 1 | 0.15 | | Convolution Layer 3 Filter | 32 | | Convolution Layer 3 Kernel Size | 4 | | Convolution Layer 3 Activation | "relu" | | Convolution Layer 4 Filter | 32 | | Convolution Layer 4 Kernel Size | 4 | | Convolution Layer 4 Activation | "relu" | | Max Pool Layer 2 Size | 2 | | Dropout Layer 2 | 0.1 | | Dense Final | 512 | | Activation Final | "relu" | | Dropout Final | 0.25 | | Train = 94.43%  Test = 72.07% |
| 4 | |  |  | | --- | --- | | Convolution 2D Layer 1 Filter | 16 | | Convolution 2D Layer 1 Kernel Size | 3 | | Convolution Layer 1 Activation | "relu" | | Convolution 2D Layer 2 Filter | 16 | | Convolution 2D Layer 2 Kernel Size | 4 | | Convolution Layer 2 Activation | "relu" | | Max Pool Layer 1 Size | 2 | | Dropout Layer 1 | 0.25 | | Convolution Layer 3 Filter | 16 | | Convolution Layer 3 Kernel Size | 4 | | Convolution Layer 3 Activation | "relu" | | Convolution Layer 4 Filter | 16 | | Convolution Layer 4 Kernel Size | 3 | | Convolution Layer 4 Activation | "relu" | | Max Pool Layer 2 Size | 2 | | Dropout Layer 2 | 0.25 | | Dense Final | 256 | | Activation Final | "relu" | | Dropout Final | 0.5 | | Train = 91.47%  Test = 75.43% |
| 5 | |  |  | | --- | --- | | Convolution 2D Layer 1 Filter | 16 | | Convolution 2D Layer 1 Kernel Size | 3 | | Convolution Layer 1 Activation | "relu" | | Convolution 2D Layer 2 Filter | 32 | | Convolution 2D Layer 2 Kernel Size | 2 | | Convolution Layer 2 Activation | "relu" | | Max Pool Layer 1 Size | 2 | | Dropout Layer 1 | 0.25 | | Convolution Layer 3 Filter | 32 | | Convolution Layer 3 Kernel Size | 2 | | Convolution Layer 3 Activation | "relu" | | Convolution Layer 4 Filter | 16 | | Convolution Layer 4 Kernel Size | 3 | | Convolution Layer 4 Activation | "relu" | | Max Pool Layer 2 Size | 2 | | Dropout Layer 2 | 0.25 | | Dense Final | 512 | | Activation Final | "relu" | | Dropout Final | 0.5 | | Train = 94.67%  Test = 74.84% |

**Interpreting the 5 x 5 image labelling:**

The image displayed in the grid is not the original image. It is the pre-processed version of the original image.

We’ve first converted the original image into grayscale image.

Then we resized the image to 32 x 32 pixels.

Each image has the following information: (Probability, Actual label, Predicted label)

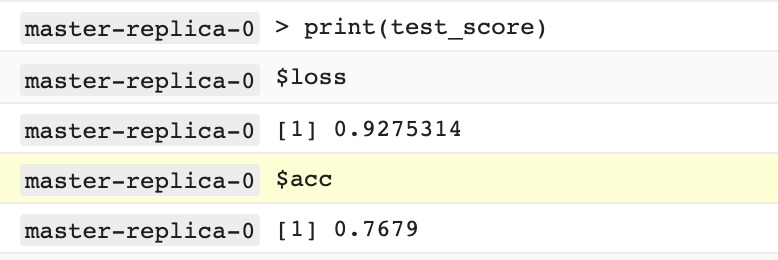
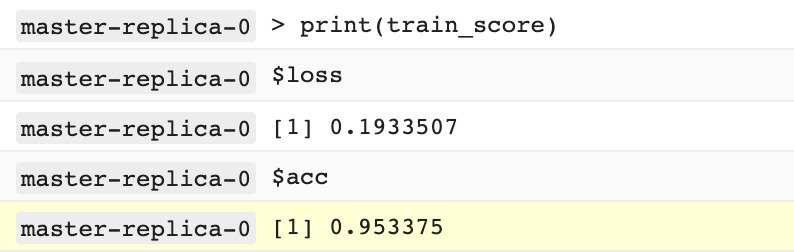
If the classifier has predicted correctly then the probability and predicted label are displayed in green colour else, they are displayed in red colour.

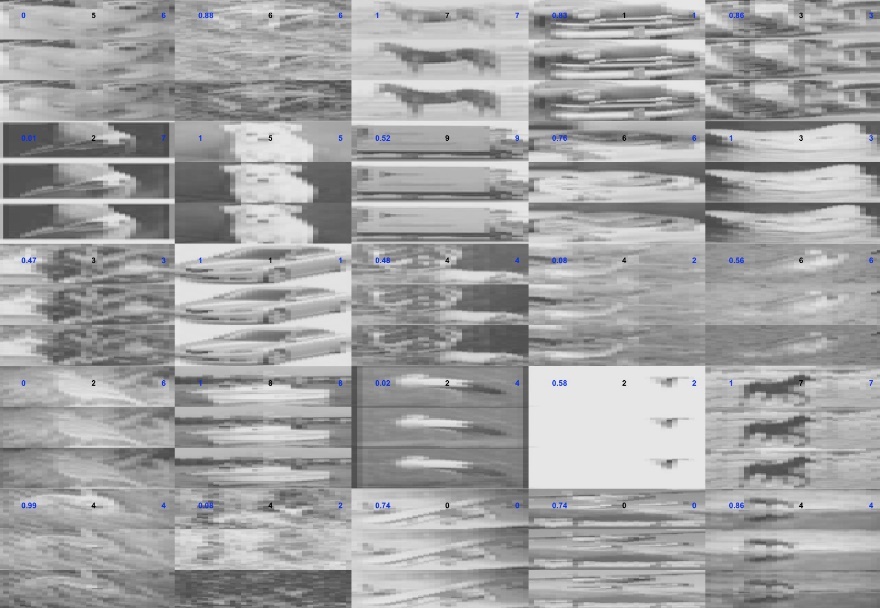
We randomly picked 25 images from the test data set and displayed in the 5 x 5 grid as instructed.

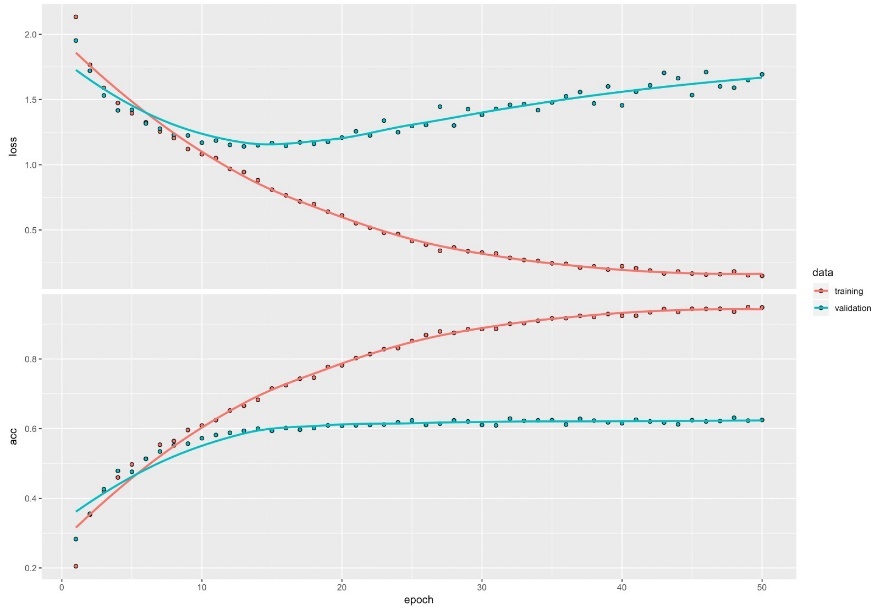
The rest of the report contains the 5 x 5 image grid and History plot for each of the above 5 iterations mentioned in the results respectively.

1. **History plots and prediction images:**

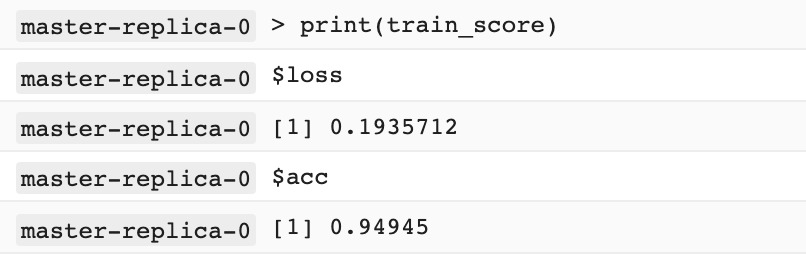
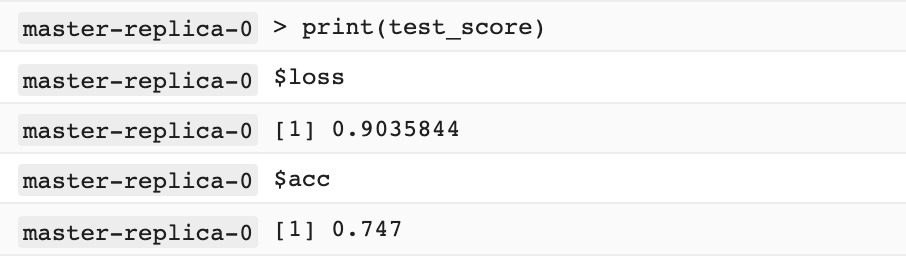
Iteration 1

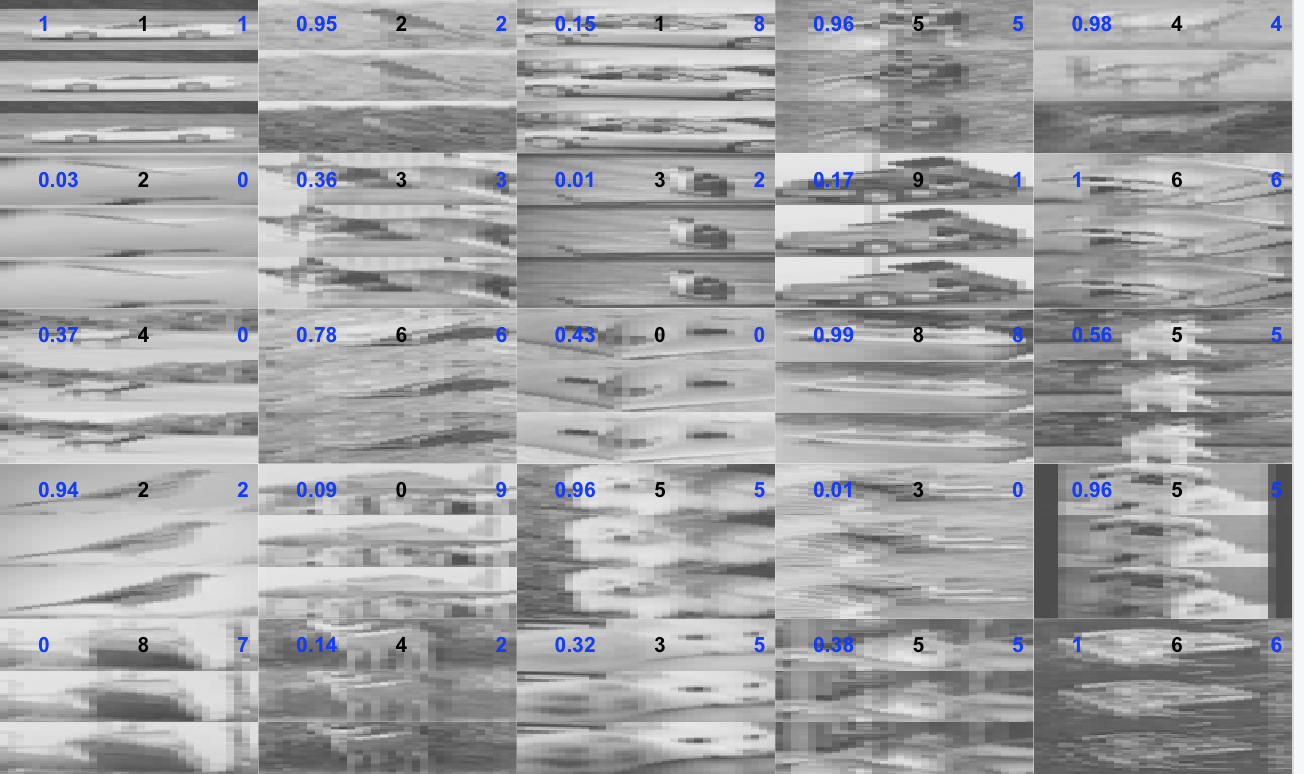


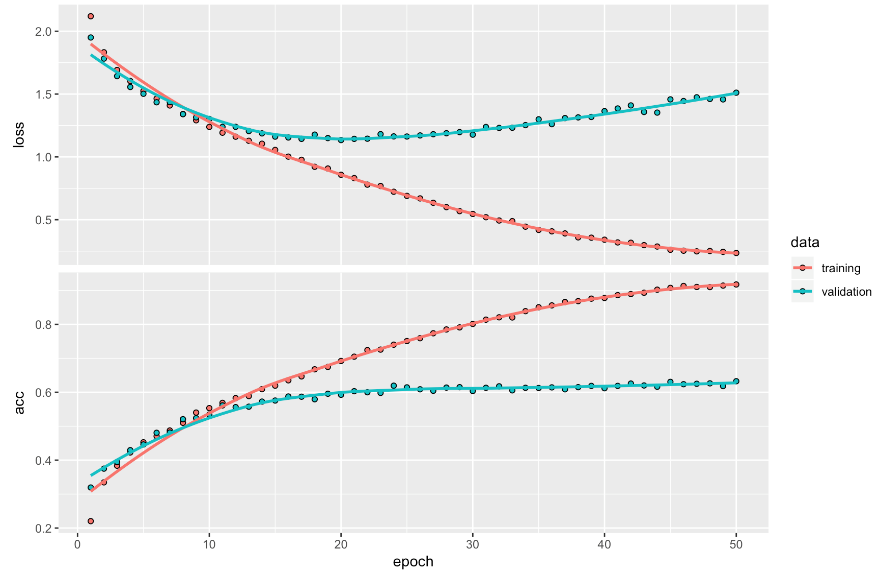




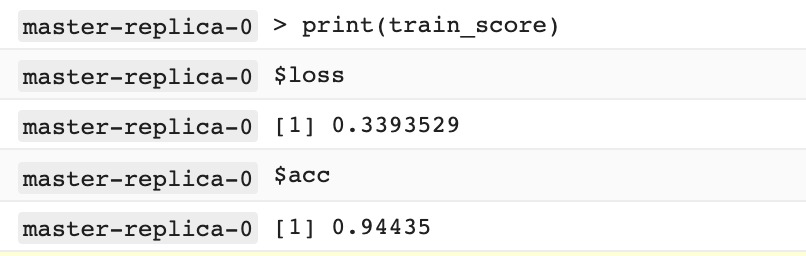
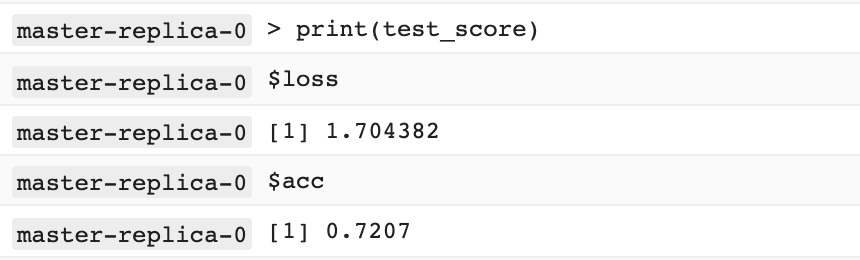
Iteration 2

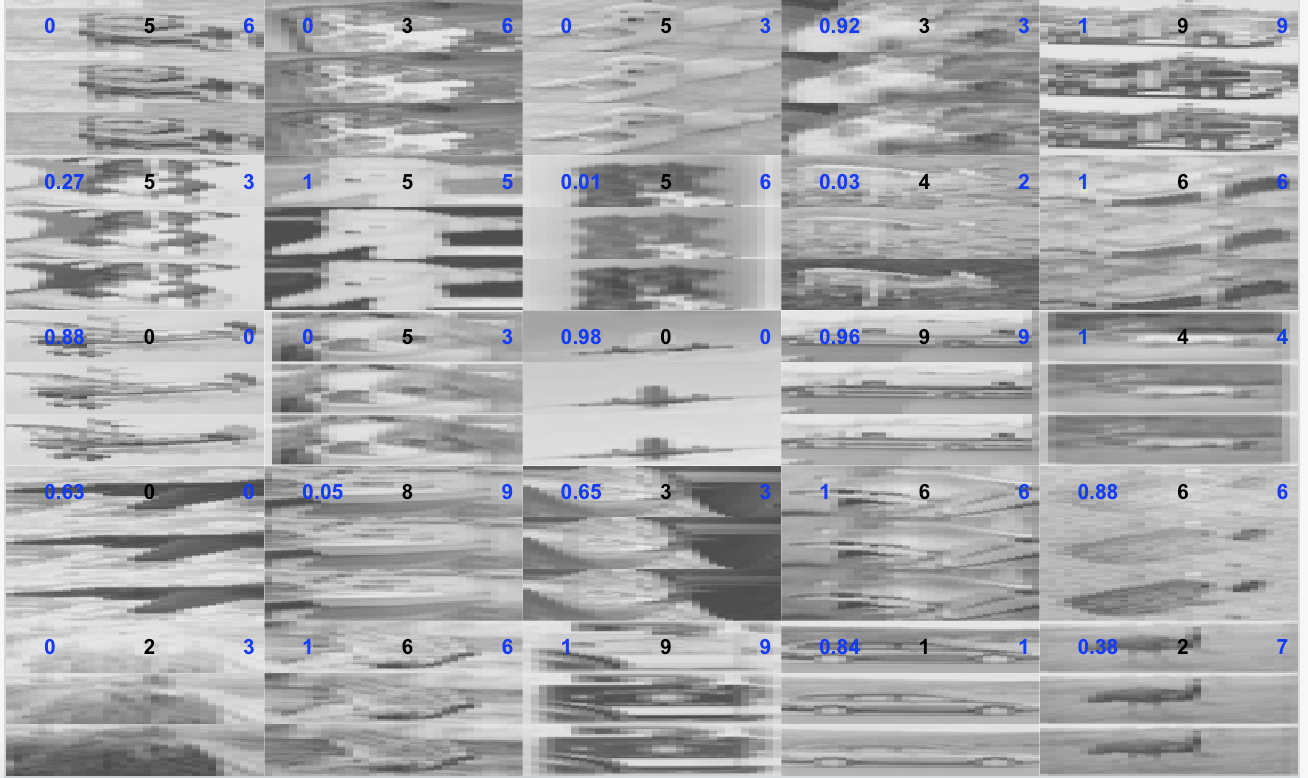


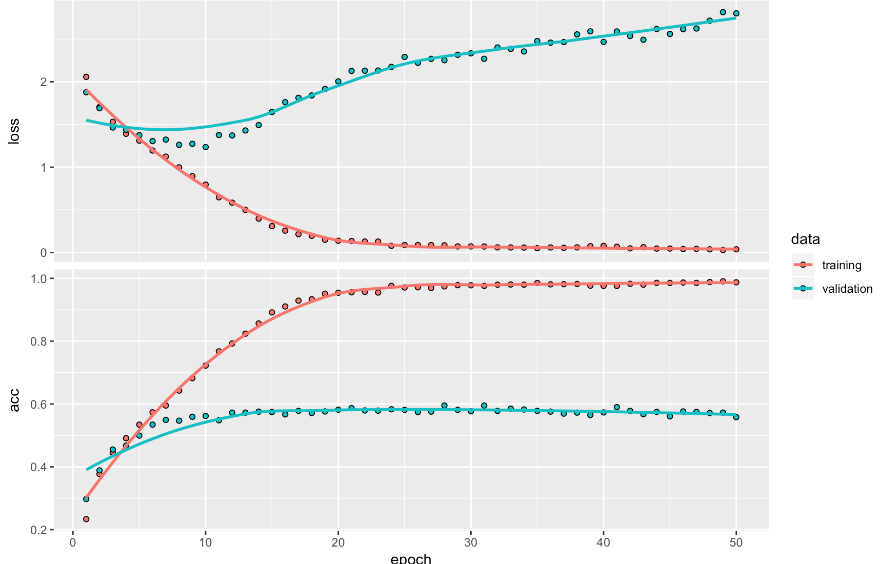




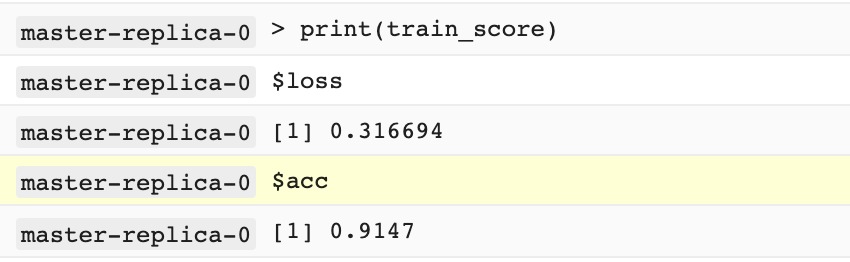
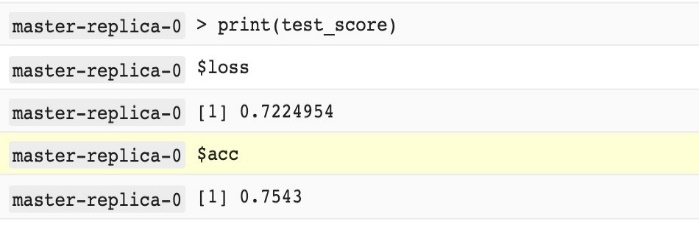
Iteration 3

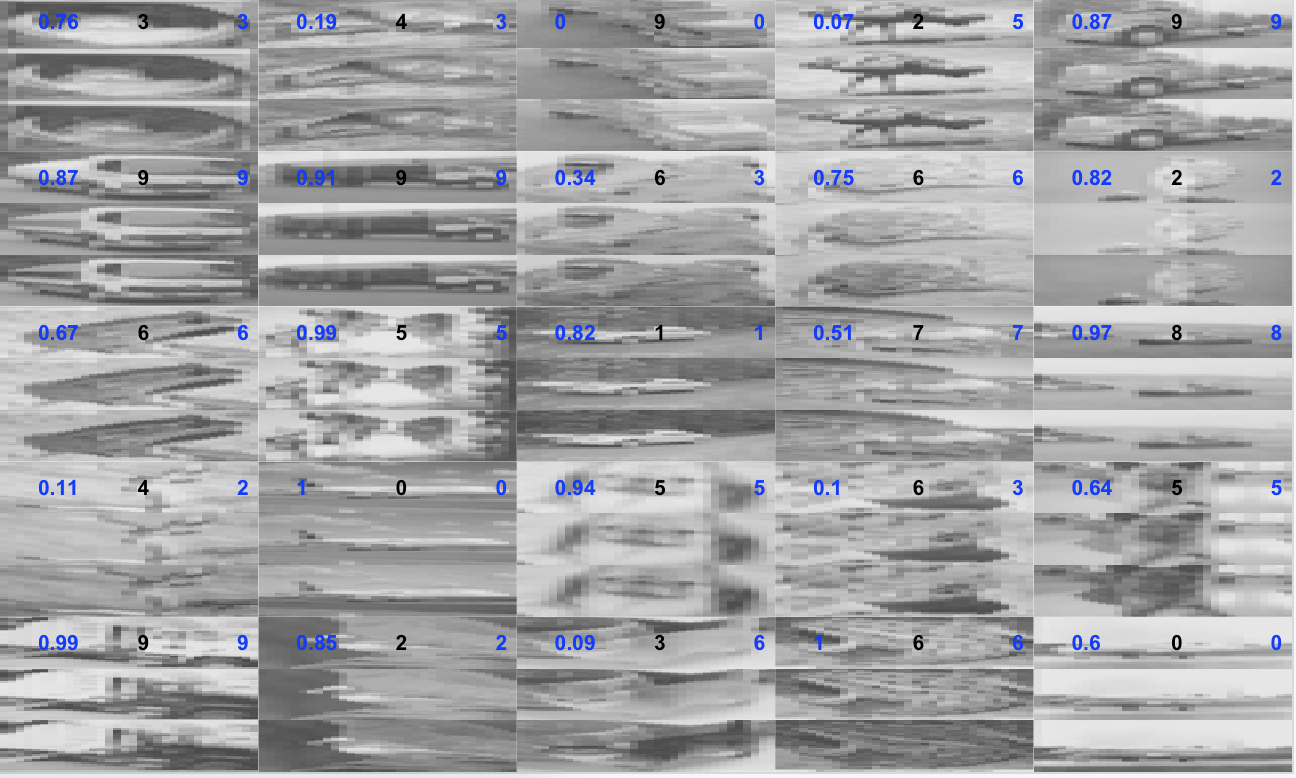


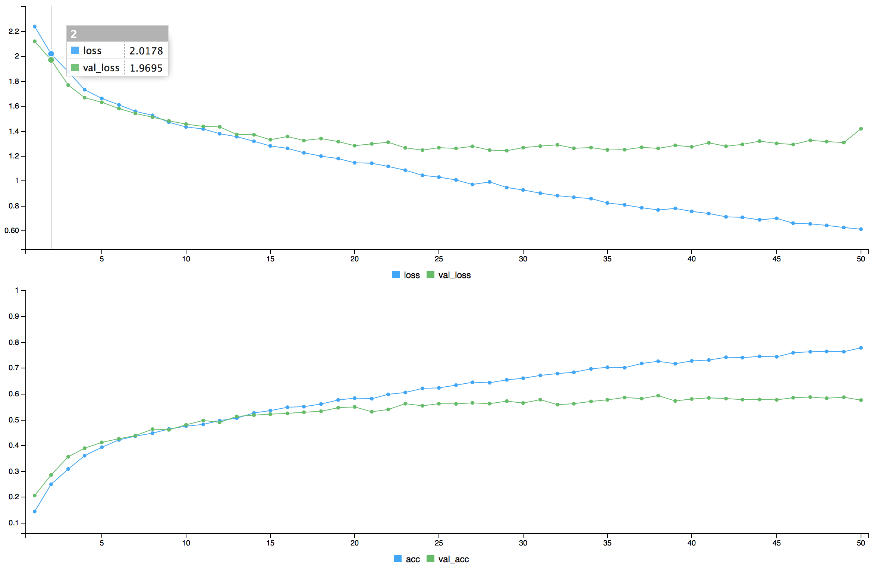




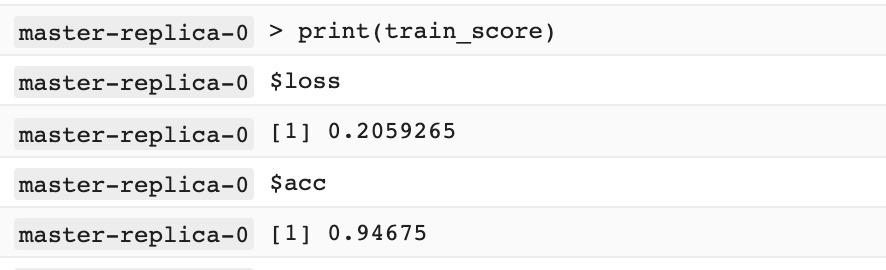
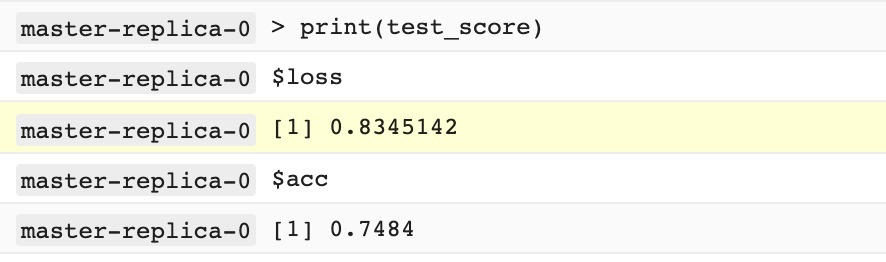
Iteration 4

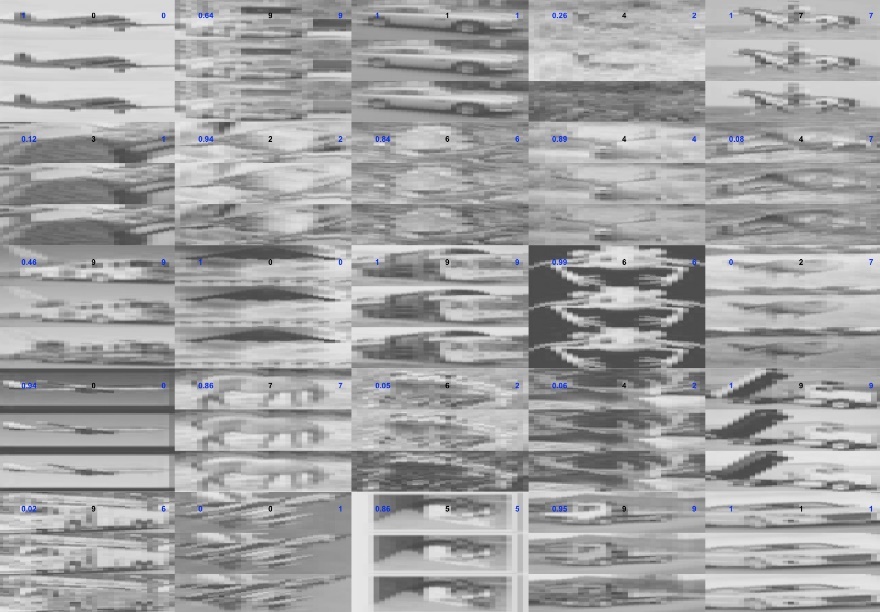


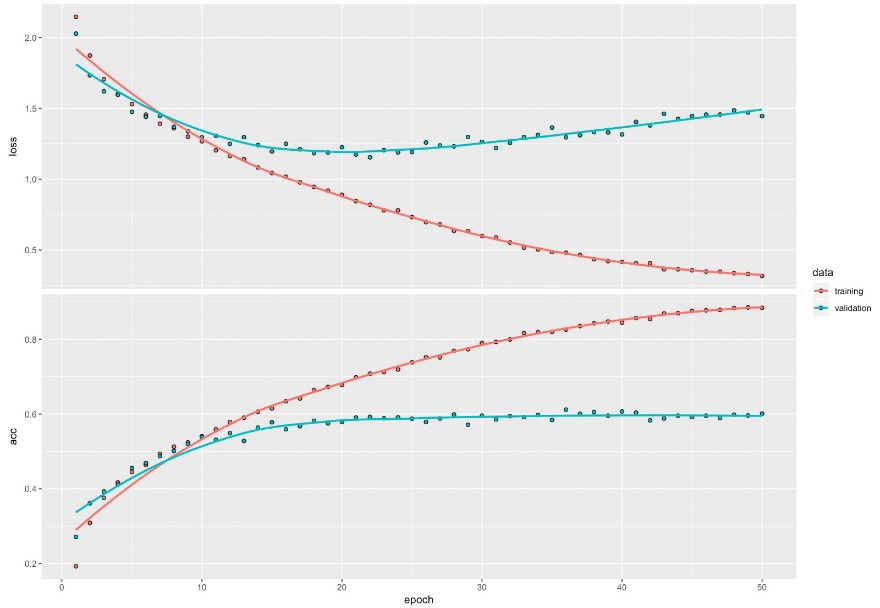




Iteration 5







**Limitations, problems faced:**

Issues with external image packages

We have faced issues running the R code in the Google Cloud environment.

One of many is issue with the **EBImage** package that we have chosen to use for image related functionalities like reading an image, converting to greyscale and resizing.

**On analysing further, we came to the following conclusions:**

GCloud uses a package dependency system called ‘**packrat’** to install required libraries in the VM instance. EBImage however is installed from the ‘**BiocManager’** package and packrat is not able to find the correct URL to download the ‘BiocManager’ package, where it halts the execution. We have even tried **sourcing** the package directly, but it is ignoring all the installation details but instead relying completely on packrat.

We’ve tried other packages as well like ‘**imager’** and ‘**magick’** and **‘openImage’**. ‘imager’ is somehow dependant to ‘EBImage’ and so it’s failing for the same reason that EBImage is failing.

lso, with magick package we encountered the issue with packages. It’s a different issue altogether, wherein it says that ‘egrep’ installation is failing.

On further exploration, we have found that **keras** have inbuilt functions that can handle basic image operations. **However, that one has a serious limitation. It only accepts file system path, it won’t read images from network URL (https path)**. It is a limitation because, in cloud we cannot download and access these images directly because we cannot get the exact directory location where it is storing these files.

**Workaround**: We have written a small script to download the zip data from remote server and unzip it in the local. When we run the script, it gets the data and submits to the cloud.