IEE 577: DATA SCIENCE FOR DECISION SYSTEM ANALYTICS

CIFAR 10 IMAGE CLASSIFICATION

Prakash Sudhakar[a], Ashish Santha Kumar[b]

Abstract:

The objective of this project is to perform Image Classification on CIFAR-10, a dataset of 60,000 images with small RGB small images of size32 x 32. Image classification is performed using various machine learning models, convolutional neural networks with multiple optimizers and a pre-trained CNN – VGGNet with 16 layers. We evaluate the model based on multiple performance metrics of each of the above-mentioned models in order to determine the best suitable model.

Introduction:

Image classification is of great importance for people to interact with each other and with the natural world. We possess a tremendous capacity for visual recognition, as we can almost easily identify objects encountered in our lives such as animals, faces and food. In particular, humans can easily recognize an object even though it may vary in position, scale, pose and illumination. This ability is called core object recognition and is carried out through the ventral stream of the human visual system. In the field of computer vision, several experiments have tried to build systems capable of imitating the ability of humans to identify objects. Despite several decades of effort, machine visual recognition was far from human performance. Nevertheless, machine performance has improved significantly since the last years, due to the re-emergence of convolutional neural network (CNNs) and deep learning. Like traditional neural networks, which are inspired by biological neural systems, the architecture of CNNs for object recognition is feedforward and consists of several layers in a hierarchical manner. Here, we will try to compare the different methods for classifying images in order to highlight the contribution of Deep learning in the field of Computer Vision.

Data Description:

CIFAR-10 is a widely used image classification data set from Kaggle. With a training data of 50,000 images, it is a large data set with tiny images of 32 x 32 dimensions. The test data set consists of a similar part, but only has 10,000 images. There are several challenges in the CIFAR-10 data set - its images reflect a broad variation in size, position, pose, and illumination.

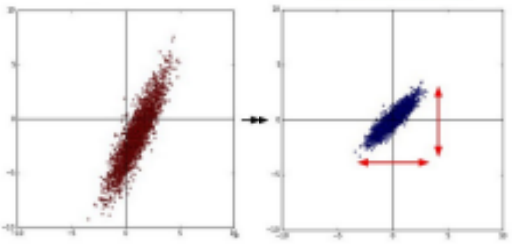
Looking at the data component first, we see that the dimensions of each batch file are 10000 x 3072. The first 1024 integers for each row in the data set are the red values for the image. The next 1024 integers are the green values for the image and the last 1024 integers are the blue values. Understanding this is crucial when pre-processing the data or building classifiers that use individual color components for classification. Finally, we can see that each integer value ranges from 0 to 255, representing an individual red, green or blue value for a single pixel. The labels function provided for each component is a 1 to 1 map for a predefined category. There are 10 categories included in the dataset and their mappings. The number of images in each of those categories is the same.

Preprocessing Data:

Normalization:

Normalization is one of the most widely used pre-processing measures to control image data. Figure 1 shows the difference between the

original and normalized data. The purpose of normalization is to scale the data points within its unit length. Throughout neural networks, imbalanced character scales will induce bias to the product of multiplication of gradient vector and learning rate. In the case of images, even though the value of each pixel per channel is fixed to be within an equivalent range, we still choose to apply normalization as the best practice and to achieve the best possible performance.



Original data *Normalized data*

Figure 1. Plot of Original vs Normalized data

One-Hot Encoding:

One-hot encoding is a collection of bits among which the legal combinations of values are only those with a single high bit and all the others low. Performing one-hot encoding converts the matrices into binary matrices of width 10. We do this because we cannot explicitly feed categorical data to a machine learning algorithm. We need to convert them to numeric values in order to obtain better predictions. We do this by using the to_categorical () function from Keras library.

Train – Validation Split:

In order to train the data, it is very important to split the entire training batch into two subgroups - training set and validation set. We shuffle the data and make sure that no bias is imposed on any portion of the dataset. For this project, the training and validation data is split in a ratio of 2:1.

Machine Learning Algorithms:

Principal Component Analysis:

PCA is a commonly used algorithm that is used to pre-process the data and reduce data dimensionality. The main objective of PCA is to center the data points to mean location and factorize the covariance matrix to SVD (Singular Vector Decomposition) Matrix. The immediate step is to visualize the dataset using non-zero eigenvectors spanning over a lower-dimensional space. Like other feature selection algorithms, PCA causes information loss, so fine-tuning the component number is needed to achieve a good balance between accuracy and time

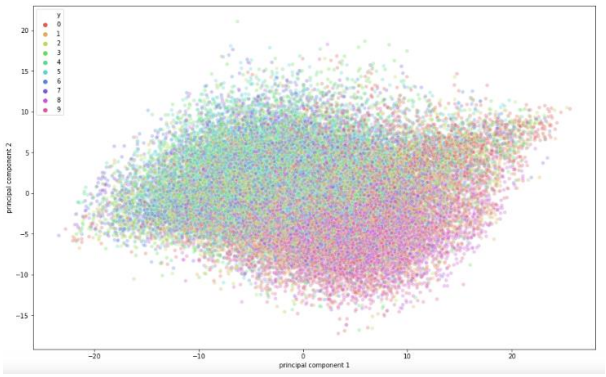


Figure 2. PCA (components = 2)

Figure 1 displays a scatter plot of the 10 classes of data when they are plotted against each of the two extracted principal components. We can see data points clustering into two data segments (i.e.) the 10 classes of data are divided into segments based on similarities in their principal components.

Support Vector Machine (SVM):

SVM is a supervised Machine learning algorithm generally used for regression analysis and

classification. The core idea of SVM is the principle of structural risk minimization, which in practice is to find a hyperplane that defines the maximum margin of class separation. SVM has impressive performance with datasets where data is of a higher dimension with many attributes. In this project, both linear SVM and Gaussian SVM (or RBF SVM) were used and the accuracy was 40.52 and 53.67 respectively. Figure 3 shows the difference throughout classification between liner and Gaussian. The confusion matrix is shown in Appendix B.

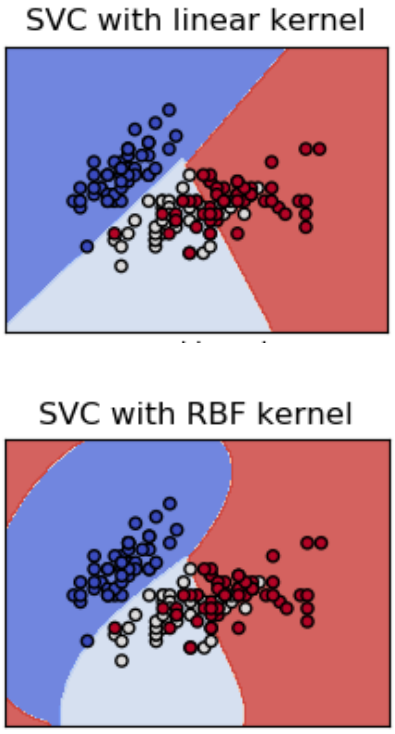


Figure 3. SVM Classifier: Linear vs RBF Kernels

Random Forest:

Random forest consists of several decision trees in a random method and they function by averaging multiple deep decision trees, trained on different parts of the same training set, with the intention of reducing the variance. There is no correlation between the decision trees in the random forest and as a result, when the test data enters the random forest, the classification of each decision tree will decide what type of the sample should belong to, and ultimately the result will be the largest number of occurrences in all the decision trees (the weight of each decision tree is to be taken into account). By performing Random Forest, we arrive at an accuracy of 51.18 % and the confusion matrix for this model is given in Appendix B.

Logistic Regression:

Logistic regression is used to characterize the data and illustrate the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. Logistic Regression follows a sigmoidal curve which is represented in Figure 4. Multiclass classification with logistic regression can be achieved either by means of a one-vs-rest scheme in which a binary classification problem of data belonging to or not to that class is performed for each class or by changing the loss function to cross-entropy loss.

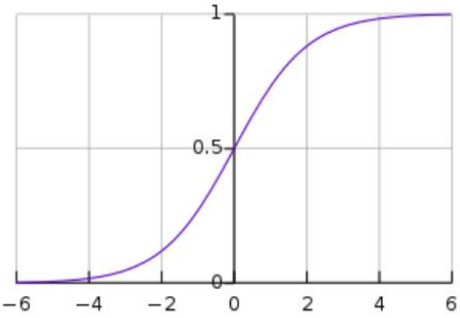


Figure 4. Sigmoid Function

We get an accuracy of 39.74 %. The confusion matrix for the model is shown in Appendix B.

k- Nearest Neighbor (k-NN):

k-nearest neighbor algorithm is a method for classifying objects based on closest examples of training in the feature space. The working of this

algorithm is based on classification of images based on the distance between the feature factors. Simply put, the k-NN algorithm classifies unknown data points by finding the most common class among the k-closest examples. The training process for this algorithm consists only of storing feature vectors and labels of the training images. In the classification process, the unlabeled query point is simply assigned to the label of its k nearest neighbors. After converting each image to a vector of fixed length with real numbers, we use the most common distance function for KNN which is Euclidean distance:

$$d(x,y) = \|x-y\| = \sqrt{(x-y)\cdot(x-y)}$$

$$= (\sum_{i=1}^{m}((x_i - y_i)^2))^{1/2}$$

We get an accuracy of 38.19 % and the confusion matrix is given in Appendix B.

Convolution Neural Network (CNN):

CNN is one of the most popular deep learning frameworks that has revolutionized image recognition and computer vision. They are most widely used to examine visual imagery and often work behind the scenes in image classification. A basic CNN model consists of convolution layers, pooling layers and dense layers (or fully connected layers). In a convolutional layer, neurons only receive input from a subarea of the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. The function of the convolution layer is to convert the input image in order to extract important features. An Activation function is used in the convolution layer to perform a nonlinear transformation of the input and allows it to learn and perform more complex tasks. Pooling layers helps to reduce the size of the image and to concentrate on features that are significant. The output is then flattened before being fed to the dense layers. This is accomplished because the fully connected layers do not require ordered input in any form and the flattening layer converts them to a one-dimensional matrix. The CNN architecture for a simple image classification model is shown below (Figure 5). The CNN model is shown in Appendix A. The CNN model is fit using six different types of optimizers. The implementation of each of these optimizers is given in Appendix B. The accuracy and loss of these six optimizers are compared and the results are displayed below in Table 1.

Optimizer	Accuracy (in %)	Loss
Adam	76.96	1.3139
RMSProp	75.39	1.7220
Adadelta	77.36	1.6423
Adagrad	76.84	1.1852
Adamax	78.08	1.3094
SGD	77.86	1.1751

In order to fit the model, we need to choose the optimizer with maximum accuracy and minimal loss and hence we choose Stochastic gradient descent (SGD) as it has the second-highest accuracy and minimum loss.

Data Augmentation:

Data Augmentation in an image classification model is the process of artificially increasing the image variations in the datasets by flipping, enlarging and rotating the original images. It is one of the regularization strategies widely used in deep learning models. Two types of augmentation are applied to the final model –

positional augmentation and color augmentation.

Two fully connected dense layers are used, in which the first layer has 128 channels and uses he_uniform initializer. The second layer is a softmax layer used to convert the output of the final layer to follow probabilistic distribution.

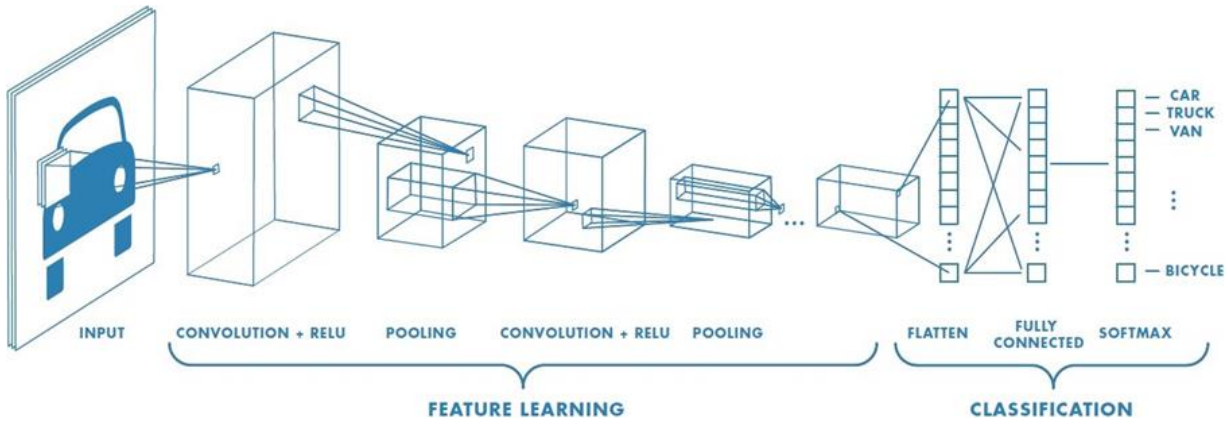


Figure 5. CNN Architecture

VGGNet-16:

The baseline CNN model with SGD optimizer provided us with an accuracy of 77.86 % which is descent but can be further improved. Finally, we are using VGG-16 which is a pre-trained CNN model proposed by K. Simonyan and A. Zisserman from the University of Oxford to develop our model.

The architecture of the VGG16 model has an input to the convolutional 1 layer of fixed size 32 x 32 RGB image. The image is passed through a stack of convolutional layers, where the filters were used with a very limited receptive field: 3×3. Max-pooling is performed over a 2×2-pixel window with stride of 1.

Batch normalization is used in this model to standardize the inputs given to the layer for each mini batch, which ultimately results in the stabilization of the learning process and drastically reduces the number of training epochs needed to train the VGGNet model.

The VGG16 model gives us an accuracy of 87.57 % which is very good for this dataset. The training accuracy vs validation accuracy graph is given in Figure 6.

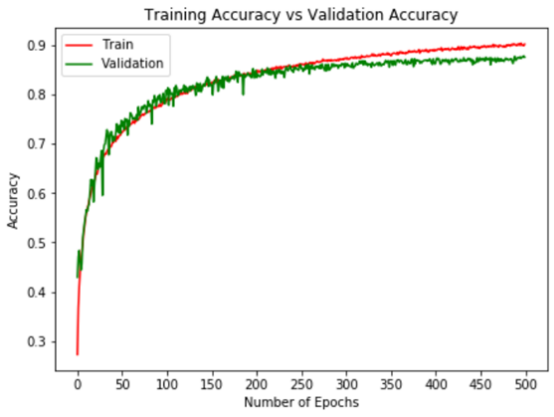


Figure 6. Accuracy Visualization for VGGNet-16

As we can see, both the training and validation accuracy is high and are very close to each other, which means a model with the right complexity has been chosen.

Conclusion:

After building all these models we can see that our accuracy ranges from 38.19 to 87.57 %. It is clear from this project that when it comes to image classification machine learning is not enough and we need to used deep learning frameworks like CNN to achieve satisfactory results. We also find that the pre-trained model VGG 16 is the best image classification model with an accuracy of 87.57 %.

The comparison of the accuracy of the different models is shown in Figure 7.

Future Work:

This model can be improved by tweaking the design of CNN architecture and learning rate of the optimizers. We could also try the other optimizers and test the accuracy that they achieve. As a Future work, we can try to train the model using other Pre-trained models like AlexNet, ResNet, Google Inception and LeNet.

Real World Applications:

- Decision making by road sign classification in self-driving cars.
- Extensively used to improve security in cars, smartphones, office, bank and home.
- Improve customer service in the e-commerce sector through visual product search.
- Classification of objects in space through images sent from satellites in NASA.

References:

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

[2] https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c

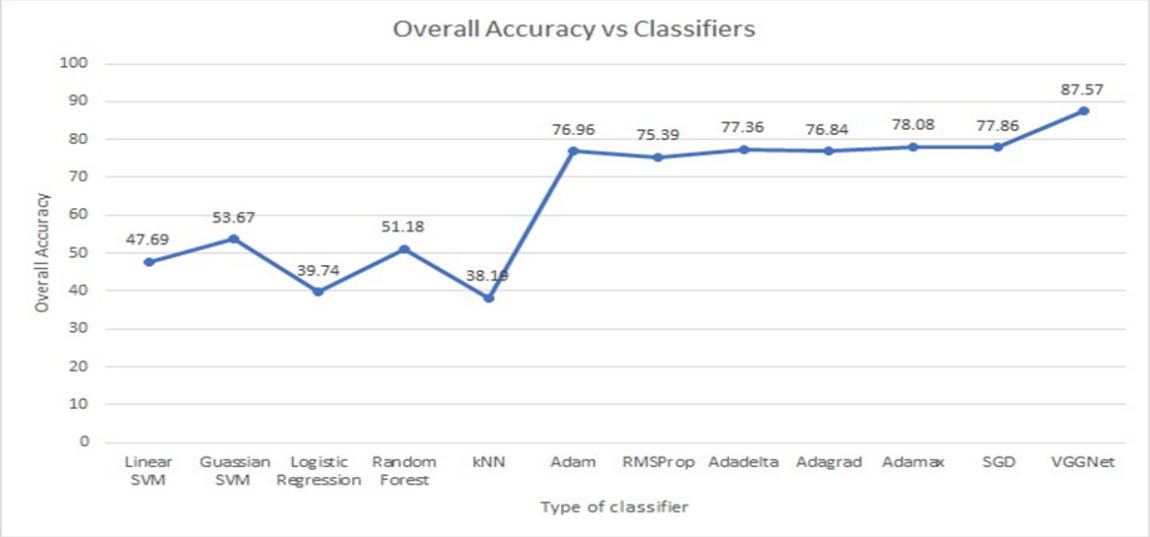


Figure 7. Comparison Chart: Accuracy for various classifiers

Appendix A: Program Code:

Baseline CNN Model:

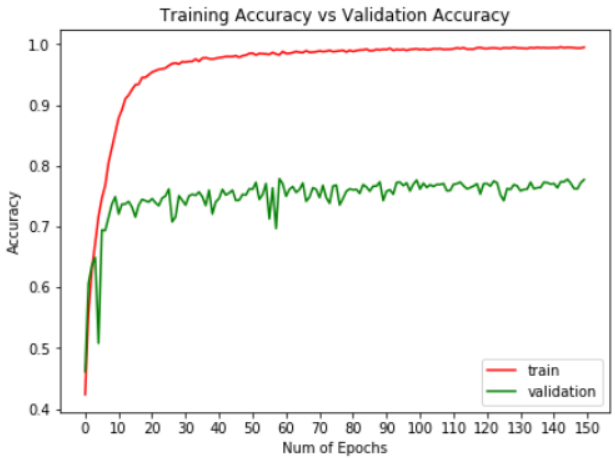
```
def get_model():
  cnn = Sequential()

  cnn.add(Conv2D(filters = 128, kernel_size = (3,3), activation = 'relu', padding = 'same',input_shape = X_train.shape[1:]))
  cnn.add(BatchNormalization())
  cnn.add(MaxPooling2D(2,2))
  cnn.add(Dropout(0.4))
  cnn.add(Conv2D(filters = 256, kernel_size = (3,3), activation = 'relu', padding = 'same'))
  cnn.add(MaxPooling2D(2,2))
  cnn.add(Conv2D(filters = 512, kernel_size = (3,3), activation = 'relu', padding = 'same'))
  cnn.add(BatchNormalization())
  cnn.add(MaxPooling2D(2,2))
  cnn.add(Dropout(0.4))

  cnn.add(Flatten())

  cnn.add(Dense(units = 256, activation = 'relu'))
  cnn.add(Dense(units = 512, activation = 'relu'))
  cnn.add(BatchNormalization())
  cnn.add(Dense(units = 10, activation = 'softmax'))

  return cnn
```

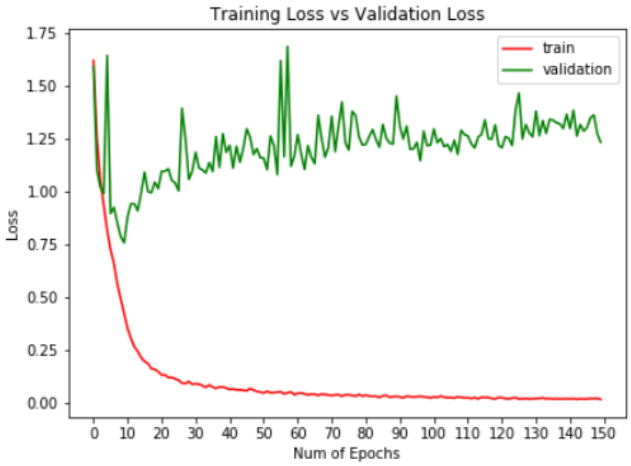
VGGNet – 16 CNN Model:

```
def define_model():
  model = Sequential()
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same', input_shape=(32, 32, 3)))
  model.add(BatchNormalization())
  model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.2))
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(BatchNormalization())
  model.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.3))
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(BatchNormalization())
  model.add(Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
  model.add(BatchNormalization())
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.4))
  model.add(Flatten())
  model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
  model.add(BatchNormalization())
  model.add(Dropout(0.5))
  model.add(Dense(10, activation='softmax'))
  # compile model
  opt = SGD(lr=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
  return model
```

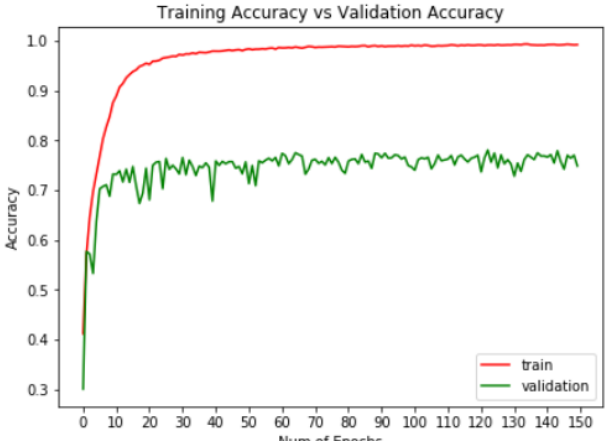
Appendix B: Visual Comparison:

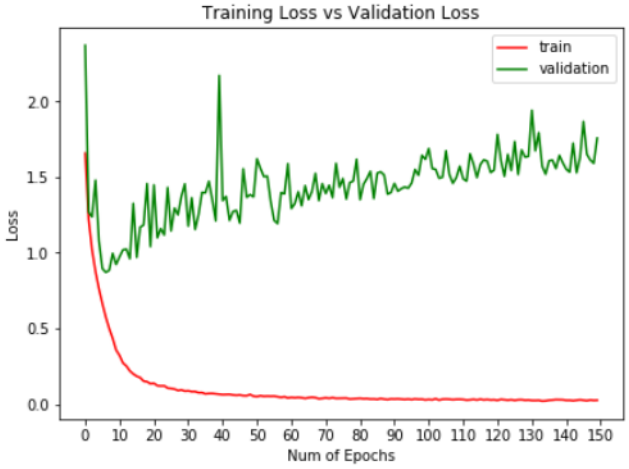
Performance Metrics Plot: Adam Optimizer



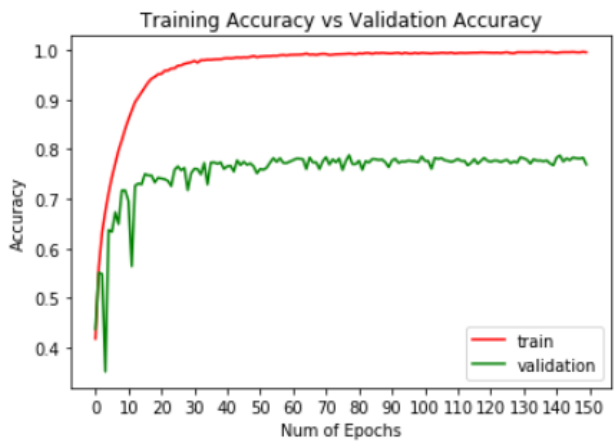


Performance Metrics Plot: RMSProp Optimizer

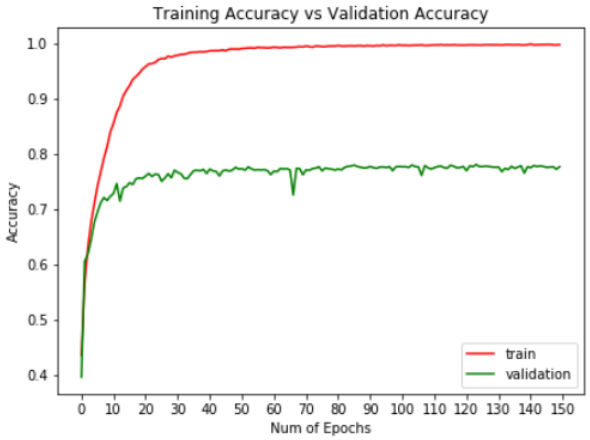


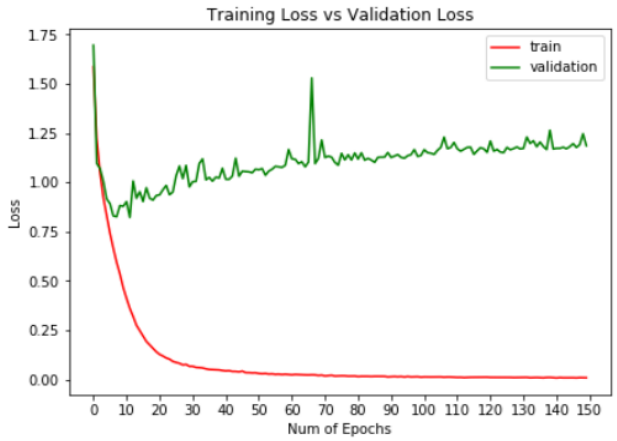


Performance Metrics Plot: Adadelta Optimizer

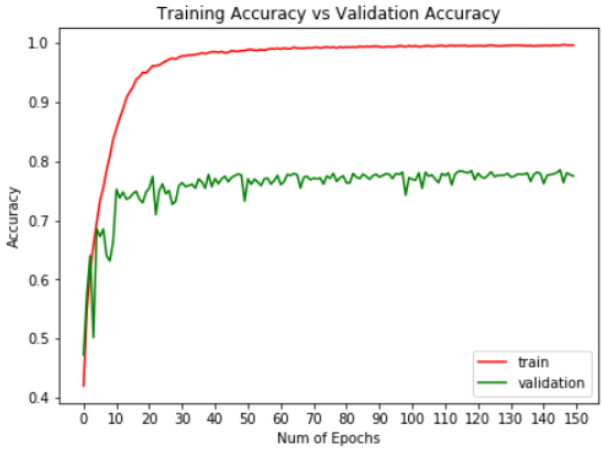


Performance Metrics Plot: Adagrad Optimizer

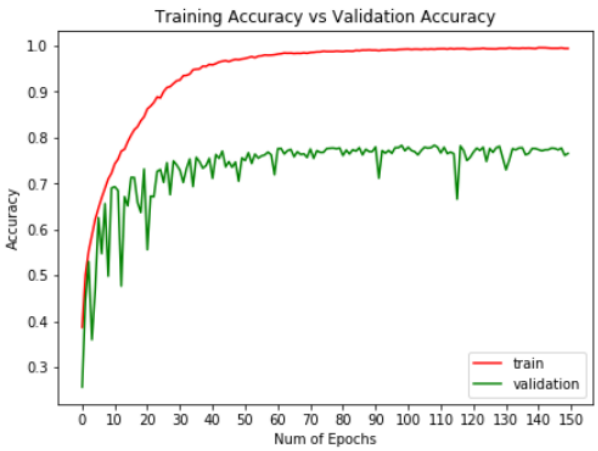


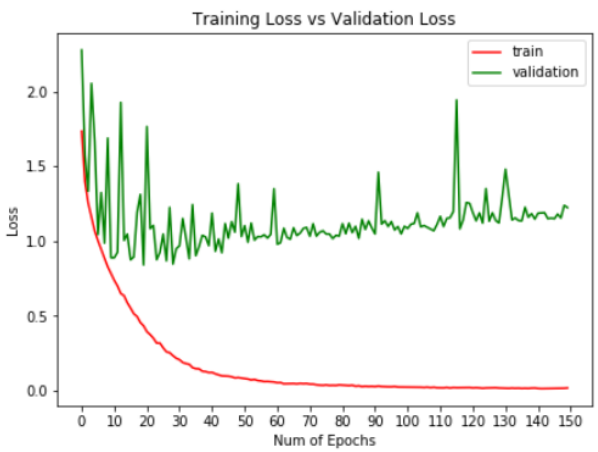


Performance Metrics Plot: Adamax Optimizer

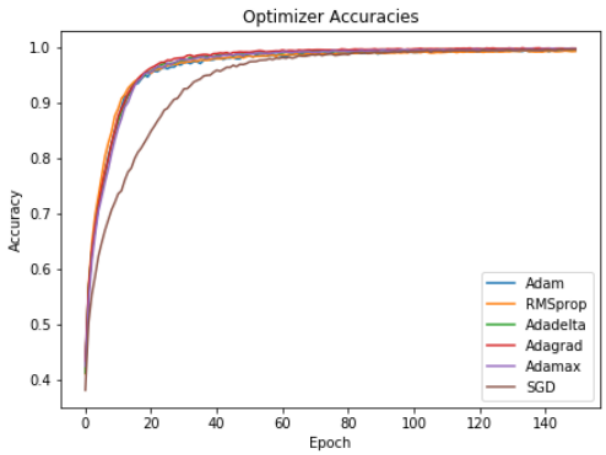


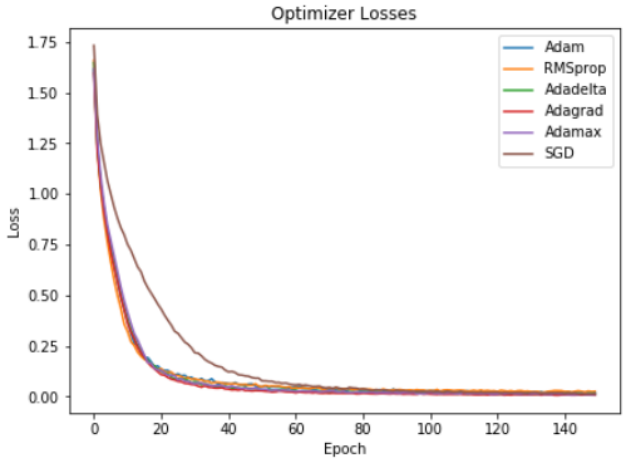
Performance Metrics Plot: SGD Optimizer



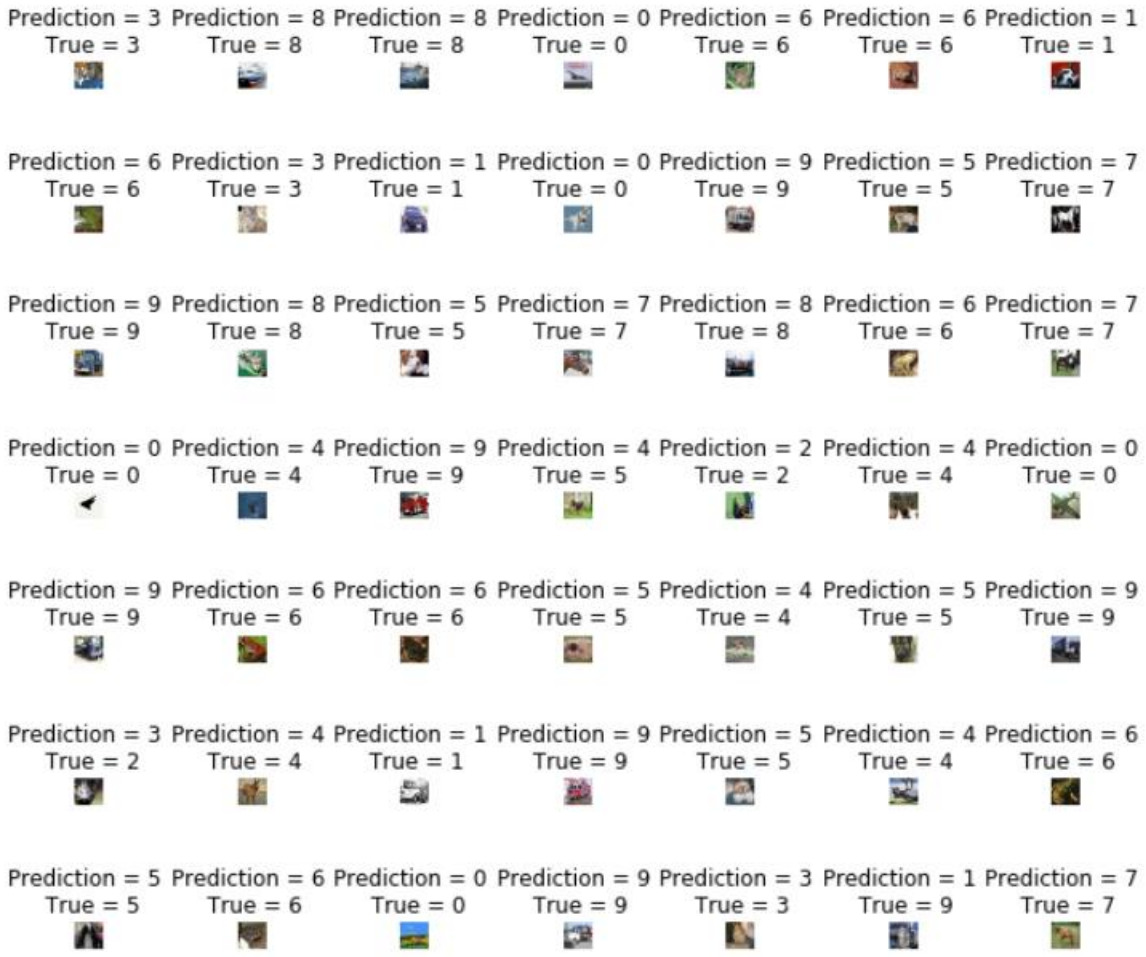


***Performance Metrics Plot: Over*all Comparison of various Optimizers**

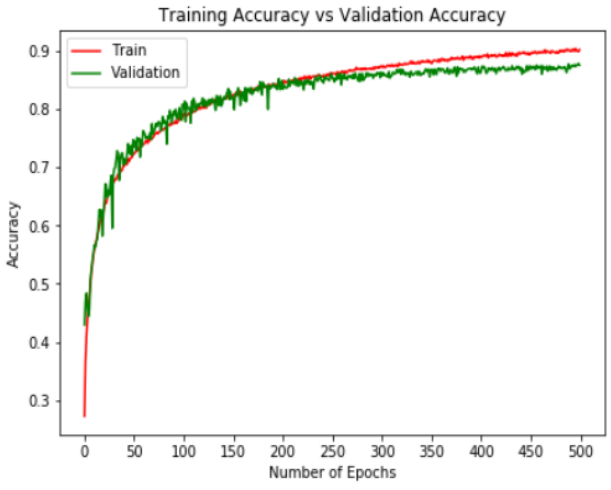




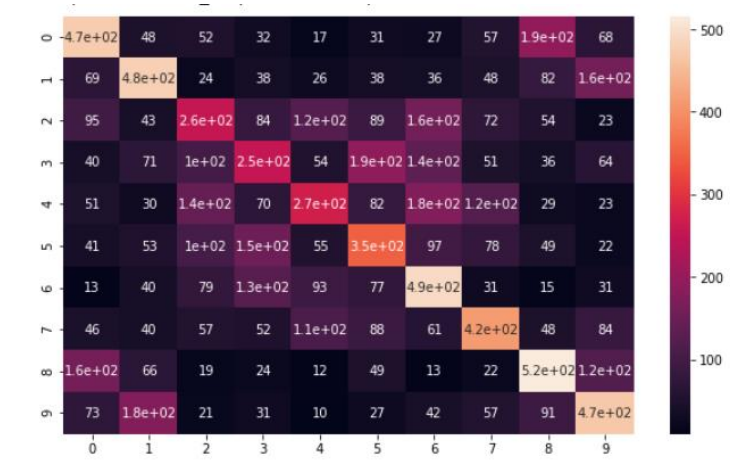
VGGNet – 16: Testing Phase – Prediction vs Ground Truth



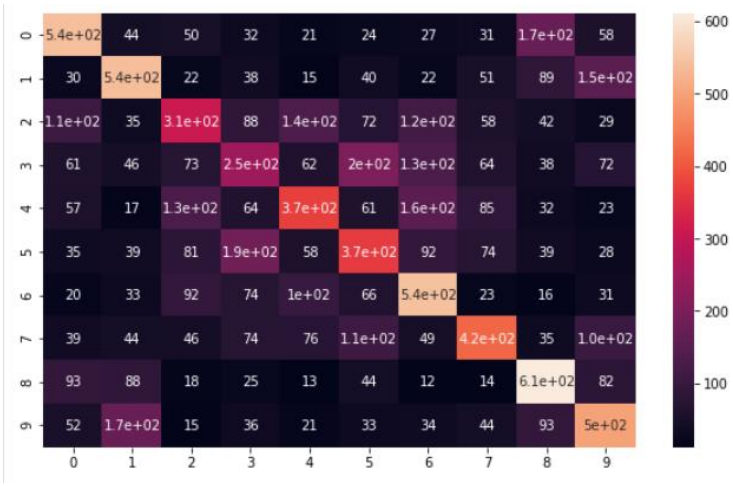
Performance Metrics Plot: VGGNet – 16



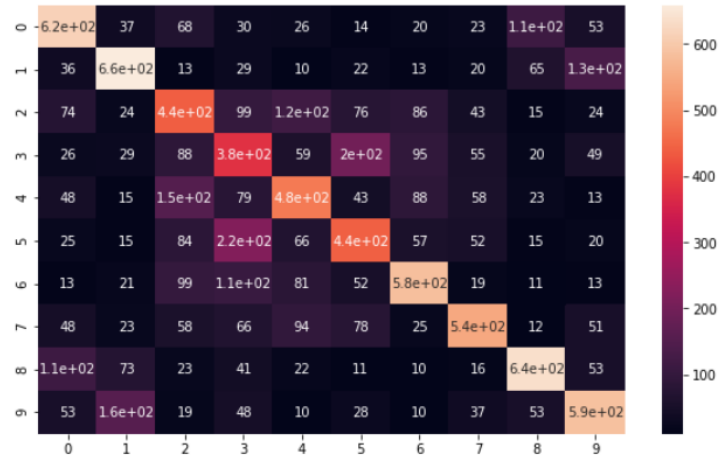
Confusion Matrix: Logistic Regression



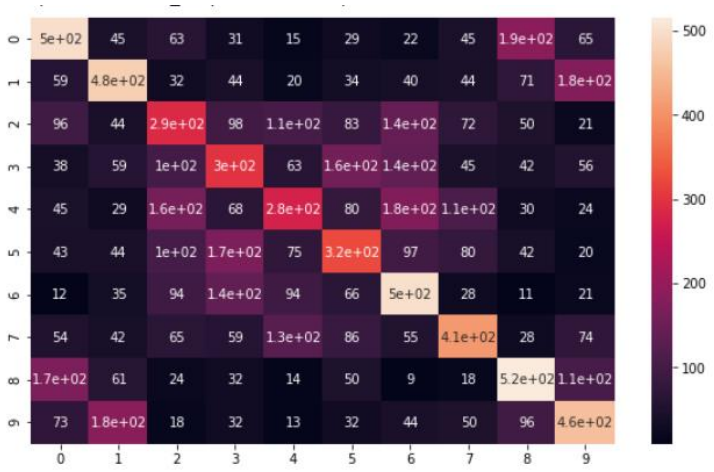
Confusion Matrix: Random Forest Classifier



Confusion Matrix: SVM – Gaussian Kernel



Confusion Matrix: SVM – Linear Kernel



Confusion Matrix: kNN