BANA 7047: Data Mining II

Traditional Machine Learning Algorithms vs Deep Learning for Image Classification

Sahoo, Sagar

Singal, Harsh

Singh, Ashutosh

MS Business Analytics

Table of Contents

Introduction

Image identification powered by innovative machine learning has already been embedded in several fields with impressive success. It is used for automated image organization of large databases, facial recognition on social networks such as Facebook. Image recognition makes image classification for stock websites easier, and even fuels marketers' creativity by enabling them to craft interactive brand campaigns.

And recently, Deep Neural Networks (DNNs) have emerged as powerful machine learning models that exhibit major differences from traditional approaches for image classification. Convolutional Neural Networks (CNNs) is a special architecture of artificial neural networks, proposed by Yann LeCun in 1988 and CNNs coupled with computational advancements has pushed the limits of what was possible in the domain of Digital Image Processing.

However, we cannot undermine the traditional techniques which had been undergoing progressive development in years prior to the rise of Deep Learning. Classification algorithms such as logistics regression, decision tree along with the ensemble and boosting techniques have performed exceptionally well at some places. *Through this, we will try to analyse the performance of various deep learning and traditional machine learning approaches for image classification and compare the results for the same*.

Here we start with creating a framework to break down images into pixels and then design a linear classifier to check its performance. Then, we will proceed with the traditional algorithms i.e. Decision Trees, Random Forests followed by Gradient Boosting and check if they can capture the non-linear relationships among pixels. And finally, we will use CNNs for classification and evaluate its performance.

Keywords: *Image Recognition, Deep Learning, Traditional Machine Learning*

About the Data

For the study, CIFAR-10 database which was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton [1] has been used. It is an established computer-vision dataset usually used for object recognition. It is a subset of the 80 million tiny images dataset and consists of 60,000 32x32 color images containing one of 10 object classes, with 6000 images per class. Since the dataset has been already formatted and pre-processed, it will help us focus more on the modelling aspect as compared to data processing aspect of the project.

Data Exploration

This dataset consists of 60,000 32x32 colour images in 10 classes, with 6000 images per class. We have 50,000 training images and 10,000 test images. The dataset file is divided into five training batches and one test batch, each with 10,000 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. Therefore, first all the batches of image were compiled together to create a single array of size 50000*3072 for training data and 10000*3072 for testing data. Similarly, a single training and testing labels arrays were created.

For each image, data were stored in an array of 3072 pixels having each pixel value ranging from 0 to 255. Therefore, to generate the image through python package, we reshaped the array into 32*32*3 shape through method depicted in the following image –

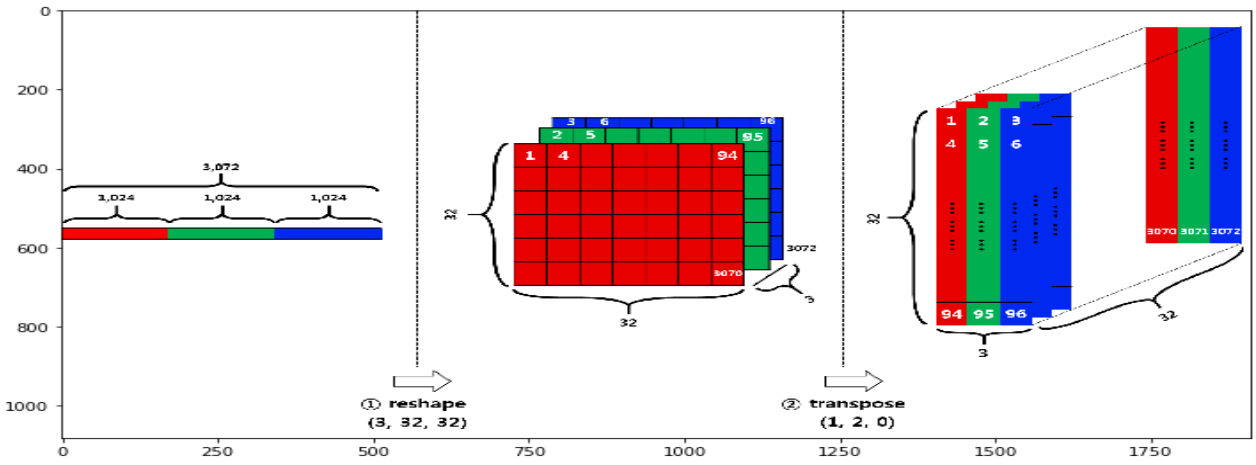
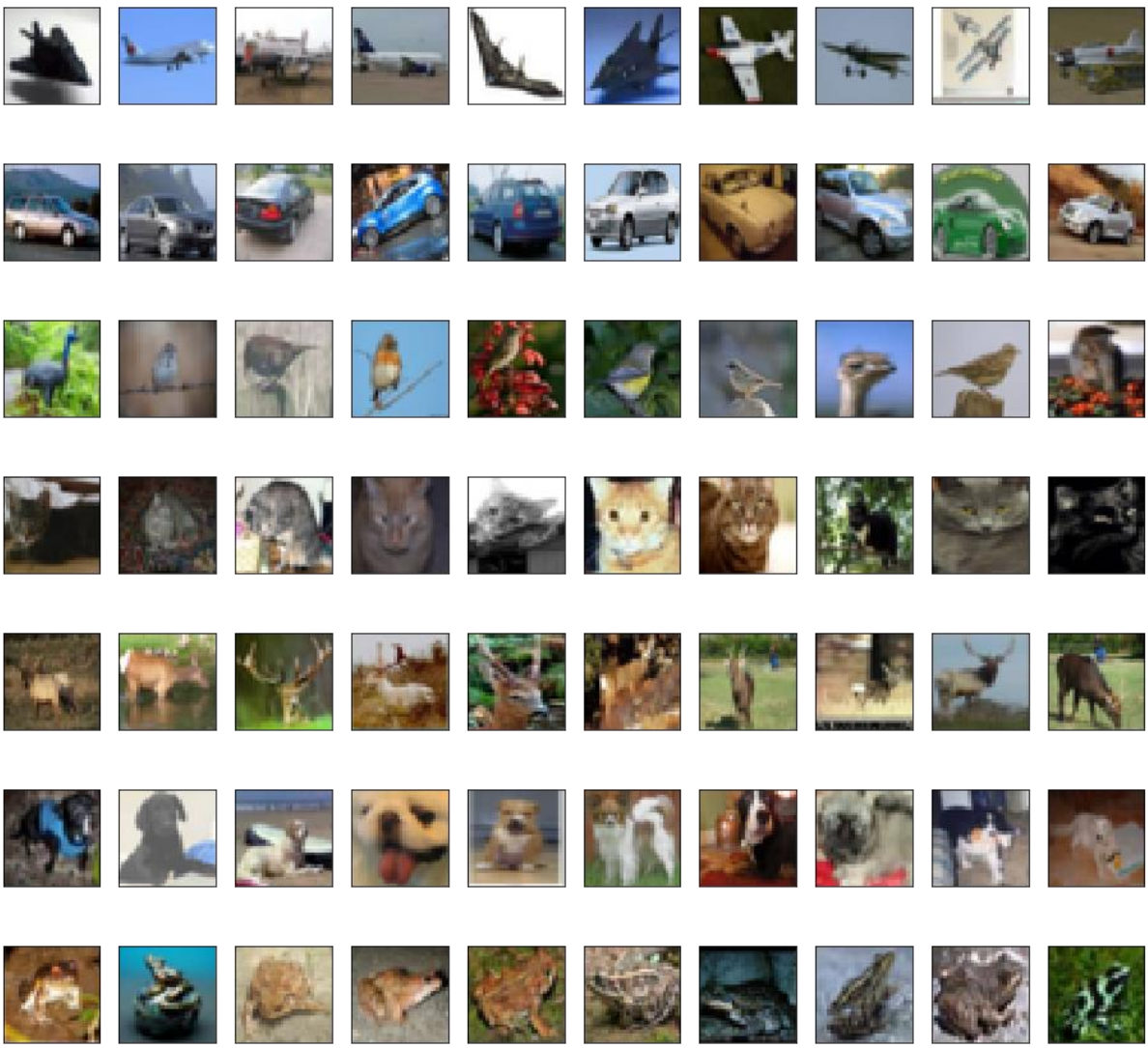


Figure 1. Pixel re-shaping using numpy

Sample images from each class in the dataset are shown below –



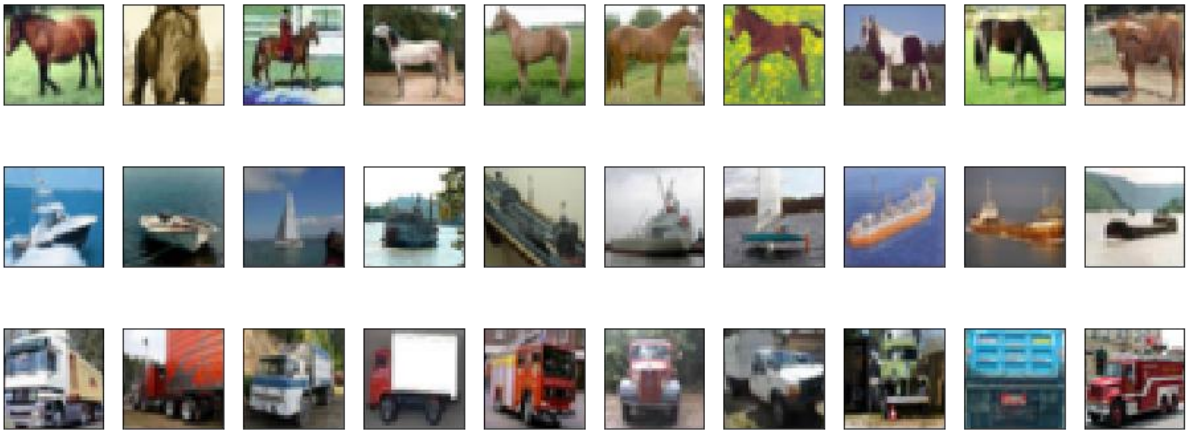


Figure 2. Sample images from the dataset

In order to understand how the images, appear under certain channels, red, blue and green channels were subsequently dropped from the pixel array.

Figure 3(a). Images with all channels intact in array

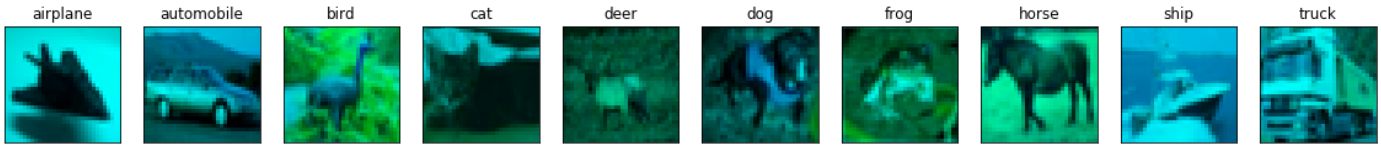


Figure 3(b). Images with red channel dropped from array

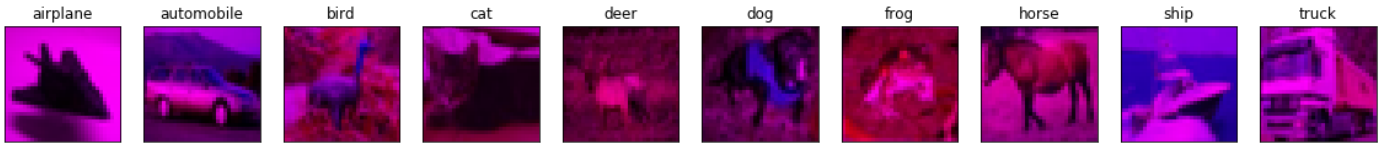


Figure 3(c). Images with green channel dropped from array

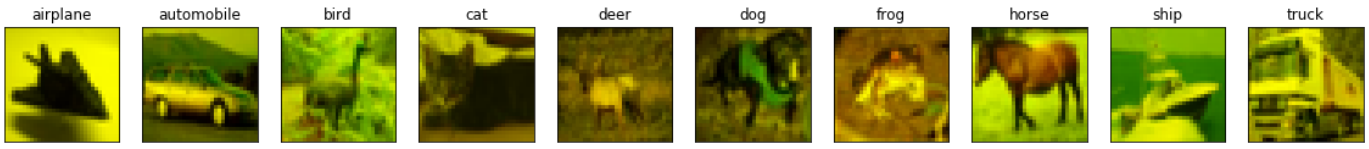


Figure 3(d). Images with blue channel dropped from array

Experiments and Evaluation

This section provides an overview of different experiments that we used to evaluate the performance of our image classification machine learning and deep learning framework. We split our training dataset of 50000 images into 40000 training and 10000 validation set. We calculated the validation accuracy using the validation set in order to determine the best model. After that we chose the model with the highest training / validation accuracy to do predictions on the unseen test data of 10000 images.

$$Accuracy = \frac{Number\ of\ \ Correct\ Classifications}{Total\ Number\ of\ Classifications}$$

Traditional Algorithms

- **Logistic Regression**

  Logistic regression is **typically** used for binary classification problems. Since, there are 10 classes, it is not feasible to apply logistic regression directly. Rather One Vs Rest logistic regression was applied on the scaled dataset with Min-Max Scaler in order to understand how logistic regression behaves in an image recognition frame. With a quad core i-7 8th generation processor and 16 GB RAM, it took approximately 45 minutes to train. Finally, we achieved training accuracy of **51%** and validation accuracy of **39%.**

- **Decision Tree Classification**

  Decision Trees inherently performs multi-class classification. The idea of a decision tree is to partition the input space into small segments, and label these small segments with one of the various output categories When the classifier was applied on the dataset with scoring criterion as “gini index”, “max depth” of 12 and “min samples at any split” value

of 80, we achieved training accuracy of **45%** and validation accuracy of **31%.** Decision Trees did well capture the non-linearity in the images.

- **Dimensionality Reduction – Principal Component Analysis**

  PCA is a simple, non-parametric method of extracting relevant in- formation from high dimensional data sets while preserving as much 'variability' (i.e. statistical information) as possible. With minimal additional effort, PCA provides a roadmap for how to reduce a complex data set to a lower dimension simplified structure that often under- lie it. So, in our frame, we have 3072 dimensions for each image. How do we know the optimal number of lower dimensions? While transforming the data into lower dimensions, we kept in mind that we need to preserve 99% of the variability in the data.

  In multivariate statistics, a scree plot is a line plot of the eigenvalues of factors or principal components in an analysis. The scree plot is used to determine the number of factors to retain in an exploratory factor analysis or principal components to keep in a principal component analysis.

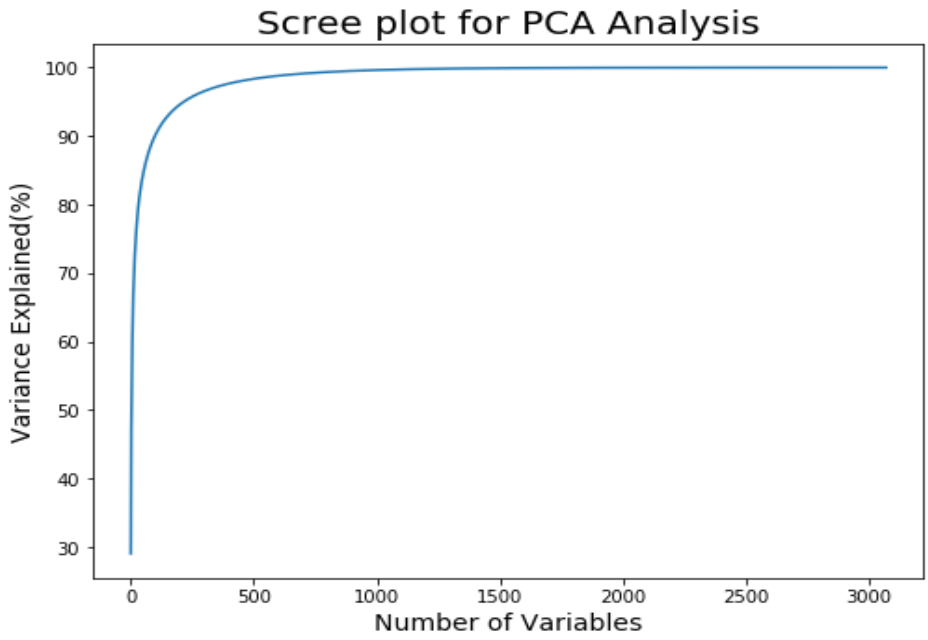


Figure 4. Scree Plot

From the plot, it was found that we need 657 components to preserve 99% of the variability in data. So now, we transformed our training/validation data from (50000, 3072) to (50000,657).

In order to check if transforming the data into lower dimensions would help us either in improved accuracy or reduced runtime, we re-applied decision trees on the transformed data. With reduced number of dimensions, we used grid search hyper-parameter tuning to find the optimum parameters for the decision trees.

Parameter	Value
criterion	gini, entropy
max_depth	4,6,8,12
min_samples_split	20,40,60,80,100

Table 1. Hyper-parameter GRID

The best parameters obtained for Decision Tree Classifier in the hyper-parameter space -

```
{'criterion': 'gini', 'max_depth': 12, 'min_samples_split': 80}
```

The corresponding training accuracy was **41%** and the validation accuracy was **31%**. Although there was not much improvement in accuracy, there was a reduction in training time.

- **Random Forest Classification**

Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. When building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split can use only one of those m predictors. A fresh sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$.

Using Random Forest Classifier, we set the n_estimators to 500, max_depth as 12, min_samples_split to 60 and used “entropy” as the split criterion.

With these parameters, we got an accuracy of **64%** for the training data. And we achieved an accuracy of **43%** for the validation data.

- **Gradient Boosting**

Random forests rely on simple averaging of models in the ensemble. The family of boosting methods is based on a constructive strategy of ensemble formation. The main idea of boosting is to add new models to the ensemble sequentially. At each iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.

Using gradient boosting classifier, we set the n_estimators to 500, max_depth to 3 and min_samples_split to 60.

Train	Loss	Remaining Time
1	89539.7828	511.02m
2	87525.0387	512.85m
3	85911.3103	512.02m
4	84545.182	513.13m
5	83329.3408	513.80m
6	82258.3022	512.76m
7	81265.4479	512.33m
70	60854.2326	242.17m
80	59355.0417	234.67m
90	57963.697	227.63m
100	56674.3639	222.63m

200	47371.1932	210.40m
300	41223.3229	160.92m
400	36450.4616	73.46m
500	32398.8828	0.00s

Table 2. Training Loss/Time across Iterations

The approximate time to train the classifier was close to **9 hours** for the mentioned hyper-parameters across 657 dimensions.

Gradient Boosting Classifier gave us a training accuracy of **75%** which is the highest among all the machine learning models. The validation accuracy that we achieved was **49%.** We could see that how Gradient Boosting outperformed the traditional algorithms. With the high accuracy of the resulting model, we can conclude that this approach could easily and efficiently be adopted in several contexts.

Now considering the training and validation accuracy achieved by all our models, we choose **Gradient Boosting Classifier** as our final traditional machine learning model.

Deep Learning Model

Deep Learning is based largely on Artificial Neural Networks (ANNs), a computing paradigm inspired by the functioning of the human brain. Like the human brain, it is composed of many computing cells or 'neurons' that each perform a simple operation and interact with each other to make a decision. Rapid progressions in Deep Learning and improvements in device capabilities including computing power, memory capacity, and power consumption have improved the performance and cost-effectiveness of vision-based applications.

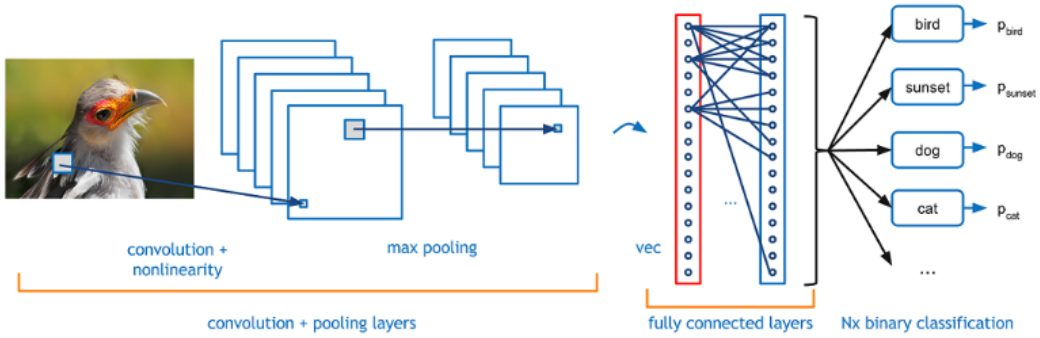


Figure 6. CNN framework in image recognition

Convolutions is one of the widely used techniques in image recognition. The process of passing a filter to the pixels array will help in detection of features. Convolutions are followed by pooling layers to reduce the size of the images/pixels and ease computations. Finally, the pixels are flattened and the reduced images are passed into multiple fully connected dense layers. The number of units in output layer depends on the number of class labels. Since we have 10 class labels, our output layer will have a soft-max layer with 10 units.

CNNs have large number of parameters and hyper-parameters, such as weights, biases, number of layers, and processing units, filter size, stride, activation function (Relu, sigmoid, tanh), learning rate etc. As convolutional operation considers the neighbourhood of input pixels, therefore different levels of correlation can be explored by using different filter sizes and strides. Different sizes encapsulate different levels of granularity; usually, small size filters extract fine- grained and large size extract coarse-grained information.

- **Multi-layered CNN**

  In our experiment, there were 2 CNN layers followed by Max Pooling and Dropout of 20%. The drop out layer is a kind of regularization in deep learning to reduce overfitting where in hidden units are randomly turned off. This was followed by another layer of

Convolution, Max Pooling and Dropout. This architecture helped us bring down the dimensions of the images from (30, 30, 32) to (6, 6, 32)

These arrays were flattened and passed into fully connected dense layer having 128 hidden units. The final output layer is a soft-max layer with 10 units which is equivalent to our number of classes. Below is a snapshot of the architecture:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_22 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 12, 12, 32)        9248
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
dropout_23 (Dropout)         (None, 6, 6, 32)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               147584
_________________________________________________________________
dropout_24 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290
=================================================================
Total params: 168,266
Trainable params: 168,266
Non-trainable params: 0
```

Figure 7. Parameters/Shape across layers

Here, we used "**Categorical_crossentropy**" as our loss function. The scoring metric was kept as "**accuracy**" while the optimizer was "**adam**". Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

The number of epochs were set to **20**. An epoch is a measure of the number of times, all the training vectors are used once to update the weights.

Based on 20 iterations, we achieved a training accuracy of **78**% and validation accuracy of **74%.** There was a huge improvement in validation accuracy as compared to Gradient Boosting algorithm. The model accuracy and model loss are plotted below:

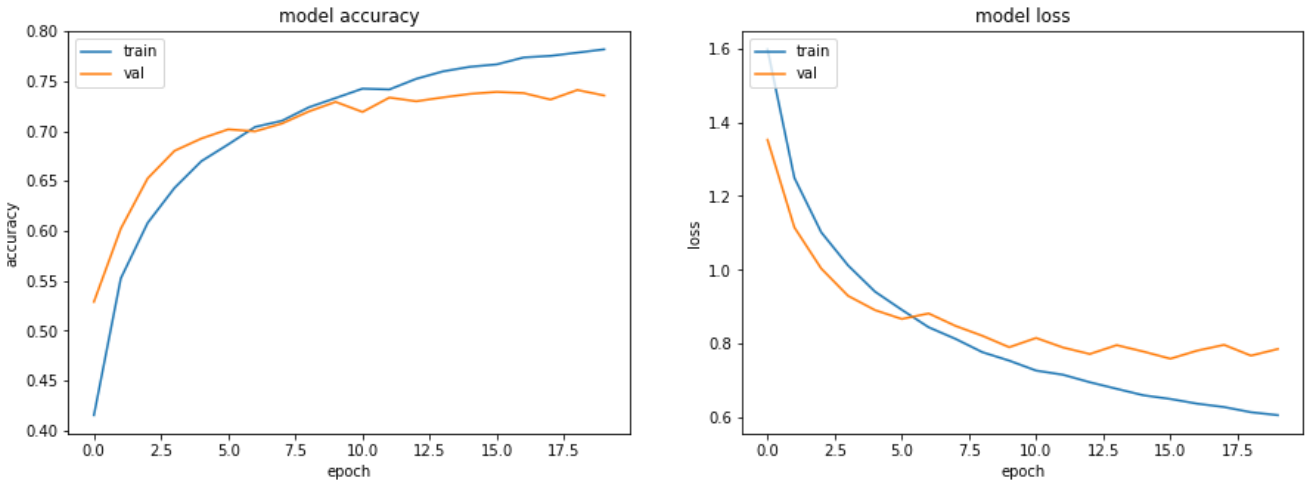


Figure 8. Model Accuracy and Loss

Different studies conducted suggested that by the adjustment of filters we can capture information at different levels of granularity. Some of the popular CNN frameworks are LeNet proposed by LeCuN in 1998, AlexNet, ZfNet and VGG.

- **VGG-16 Model**

VGG was made 19 layers deep compared to AlexNet and ZfNet to simulate the relation of depth with the representational capacity of the network. We designed the VGG framework using Keras module of Tensorflow.

With a learning rate of 0.0005, categorical_crossentropy as the loss metric, and accuracy as our scoring metric, we trained the VGG framework on our CIFAR dataset for 10 epochs. Since the framework is too deep, we need higher configuration systems which could leverage the power of GPUs/ TPUs to reduce the computation time.

After training for close to 8 hours, we could achieve training accuracy of 41% using the VGG-16 framework.

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/10
40000/40000 [==============================] - 2642s 66ms/step - loss: 2.5237 - accuracy:
0.1347 - val_loss: 2.4485 - val_accuracy: 0.1000
Epoch 2/10
40000/40000 [==============================] - 2631s 66ms/step - loss: 2.2406 - accuracy:
0.1963 - val_loss: 2.5860 - val_accuracy: 0.1000
Epoch 3/10
40000/40000 [==============================] - 2471s 62ms/step - loss: 2.0554 - accuracy:
0.2416 - val_loss: 2.3138 - val_accuracy: 0.1528
Epoch 4/10
40000/40000 [==============================] - 2733s 68ms/step - loss: 1.9352 - accuracy:
0.2836 - val_loss: 1.9122 - val_accuracy: 0.2683
Epoch 5/10
40000/40000 [==============================] - 3001s 75ms/step - loss: 1.8611 - accuracy:
0.3133 - val_loss: 1.7778 - val_accuracy: 0.3492
Epoch 6/10
40000/40000 [==============================] - 2470s 62ms/step - loss: 1.7967 - accuracy:
0.3438 - val_loss: 1.7263 - val_accuracy: 0.3802
Epoch 7/10
40000/40000 [==============================] - 2977s 74ms/step - loss: 1.7541 - accuracy:
0.3614 - val_loss: 1.6888 - val_accuracy: 0.4012
Epoch 8/10
40000/40000 [==============================] - 3356s 84ms/step - loss: 1.7119 - accuracy:
0.3827 - val_loss: 1.6550 - val_accuracy: 0.4127
Epoch 9/10
40000/40000 [==============================] - 2800s 70ms/step - loss: 1.6750 - accuracy:
0.3964 - val_loss: 1.6404 - val_accuracy: 0.4189
Epoch 10/10
40000/40000 [==============================] - 3020s 76ms/step - loss: 1.6362 - accuracy:
0.4135 - val_loss: 1.6023 - val_accuracy: 0.4369
```

Figure 9. Training/Validation Loss and Accuracy across epochs

VGG got fame due to its simplicity, homogenous topology, and increased depth. The main limitation associated with VGG was the use of huge parameters, which make it computationally expensive and difficult to deploy it on low resource systems (consistent with our findings above).

Out of Sample Evaluation

- **Traditional Algorithms**

  On the test data, using the Gradient Boosting classifier, we achieved an accuracy of **50.08%**. This is closely **matching** with our validation accuracy of **49%**. We now build the confusion matrix in order to have an overview of the predictions for each class.

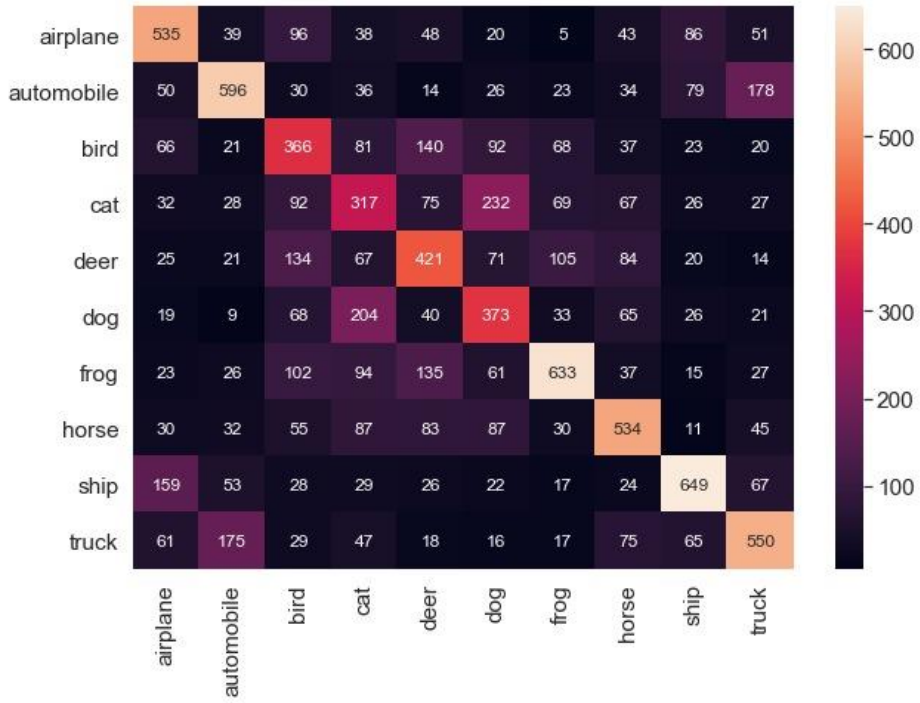


Figure 5. "Gradient Boosting" Out of Sample Confusion Matrix

The diagonal elements represent the correct classifications. The number of correct predictions was minimum for class 2, 3 and 5. For the rest of the classes, the classifier did a fair job classifying the class. Having seen the **accuracy** of these models and keeping an eye on the **huge time** (~8-9 hours) involved in training the classifiers, we will now move to Deep Learning.

- **Deep Learning**

  Now, we used our multi layered CNN architecture to do predictions on the unseen test data. The experiments gave us an accuracy of **74%** which clearly surpassed Gradient Boosting. We also build the confusion matrix for the test data to have an overview of the predictions for all the classes.

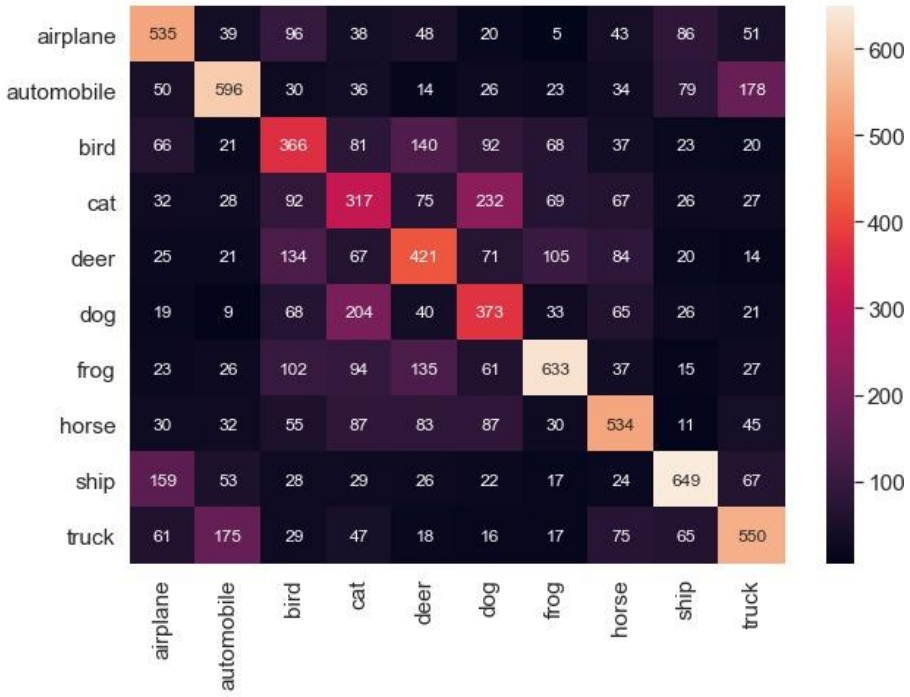


Figure 10. "Multi Layered CNN" Out of Sample Confusion Matrix

CNN did well on the unseen test data. The number of diagonal elements i.e the correct classifications were better as compared to traditional algorithms. Gradient Boosting did not do well on the class 2, 3 and 5. But CNN did classify these classes more accurately.

Summary table for in-sample and out-of sample accuracies for all the algorithms is shown below -

Classifier	**Training Accuracy**	**Validation Accuracy**	**Out of Sample Accuracy**
One vs All Logistic Regression	51%	39%	-
Decision Trees	45%	31%	-
Decision Trees (Transformed)	41%	31%	-
Random Forests (Transformed)	64%	43%	-
Gradient Boosting (Transformed)	75%	49%	50%
Multi Layered CNN	**78**%	**74**%	**74**%
VGG-16	41%	43%	-

Table 3. Model Evaluation Summary

Class by Class Comparison (Gradient Boosting and CNN)

Since we have already narrowed out the two best models in each case, we can further dig into this evaluation by comparing their accuracy scores on different image classes.

The accuracy scores were calculated using following equation:

$$Accuracy(by\ Class) = \frac{Number\ of\ \ Correct\ Classifications\ for\ particular\ object}{Total\ Number\ of\ Images\ for\ that\ object}$$

Name of Object	Gradient Boosting	Multi Layered CNN
Airplane	52.9%	74.6%
Automobile	58.3%	84.3%
Bird	36.0%	62.5%
Cat	**32.7%**	**54.4%**
Deer	42.1%	71.9%
Dog	37.6%	63.9%
Frog	63.6%	82.1%
Horse	52.5%	74.8%
Ship	**65.0%**	86.1%
Truck	56.3%	**86.4%**

Table 4. Class wise Accuracy Summary

There are couple of great insights from the above table:

- Even in the worst performance case (Cats), CNN is outperforming Gradient Boosting by more than 20%.
- There was not a single case where Gradient Boosting performed better in comparison to Multi Layered CNN model.

- Both the models tend to do well in certain object types like Ship, Frog, Automobile in contrast to others like Cat, Dog and Bird.

Conclusion

Through this work, we studied the performance of traditional algorithms and deep learning frameworks. We saw how deep learning outperformed in accuracy and computation across all frameworks on the unseen data. With careful hyper-parameter tuning (learning rates, deeper network topology, higher epochs, optimizers, data augmentation and transfer learning), we can still improve the accuracy of the CNN. CNN encourages learning in a modular fashion and thereby, making architecture simpler and more understandable. The image processing domain has undergone dramatic changes in a very short period.

This project hopefully highlighted how latest advancements helped traditional algorithms i.e Gradient Boosting capture non-linearity of pixels. However, Deep Learning innovations are driving exciting breakthroughs in the context of Image recognition and object detection.

References

[1] The CIFAR-10 dataset by by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton [https://www.cs.toronto.edu/~kriz/cifar.html]

[2] VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION by Karen Simonyan∗ & Andrew Zisserman [https://arxiv.org/pdf/1409.1556.pdf]

[3] Scree Plot [https://en.wikipedia.org/wiki/Scree_plot]

[4] Gradient boosting machines, a tutorial by Alexey Natekin and Alois C Knoll [https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021/full]

[5] Deep Learning vs. Traditional Computer Vision by Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka

Krpalkova, Daniel Riordan, Joseph Walsh @ IMaR Technology Gateway, Institute of Technology Tralee, Tralee, Ireland

[6] Usage of optimizers in Keras @ https://keras.io/optimizers/

[7] Adam: A Method for Stochastic Optimization by Diederik P. Kingma, Jimmy Ba @ 3rd International Conference for Learning Representations, San Diego, 2015

[8] A Survey of the Recent Architectures of Deep Convolutional Neural Networks by Asifullah Khan , Anabia Sohail , Umme Zahoora , and Aqsa Saeed Qureshi @ Pattern Recognition Lab, DCIS, PIEAS