Comparison of Classifiers on the CIFAR-10 Dataset

Pranav Mody
prmody@ncsu.edu

Sahil Mehta
shmehta@ncsu.edu

Aditya Govil
agovil@ncsu.edu

1 Introduction

With increasing popularity of devices capable of taking images and videos, the number of images captured are rapidly increasing. Being able to process all these images and categorize them according to their categories can be considered very useful in digital image analysis. This skill can be applied to various application such as handwriting recognition, image search, object detection, face recognition and many more.

Our project is about understanding different machine learning algorithms for image classification. The data set which we have used for this project is CIFAR-10 data set which is a collection of images that are commonly used to train machine learning and computer vision algorithms. The CIFAR-10 data set contains 60,000 32 X 32 color images in 10 different classes.The 10 different classes represent airplanes, cars, birds, cats, deer, gods, frogs, horses, ships and trucks. There are 6,000 images of each class. There are 50000 training images and 10000 test images. The steps involved in image classification are : Pre-processing, selecting different classifiers, analysing the classifiers by tuning hyper parameter, comparing different classifiers and selecting the best one.

2 Background

One of the earliest classifications done on the CIFAR-10 data set used machine learning models by considering the pixels as a feature and applying classification techniques but the result's accuracy was low. In [4], using a single hidden layer resulted in an overall accuracy of 64.84%. Cirean [4] realised that traditional computer and machine learning techniques were not capable of achieving higher scores. To accommodate the subtleties found in human perception of images, they relied on Deep Convolutional Neural Networks(DNN). For the CIFAR-10 data set, the DNN constructed consisted of 3 maps (one for each color present)and 10 layers. With this new methodology Cirean was able to achieve a result of 89.14%. Similarly, many different DNN models with different activation functions such as ReLU were used on CIFAR-10 data and similar results were achieved.

3 Proposed Method

3.1 Intuition

The idea is to try to understand and figure out if simple machine learning algorithm such as Logistic Regression and Support Vector Machine are good enough for image classification applications and if they are not, they will serve as a baseline model for the new CNN model we built. We know there are already many pre-trained model for image classification so this may not be the state of the art solution to anything but this is our attempt on trying to understand why CNN is better than other machine learning algorithms for image classification.

3.2 Classifiers

We have decided to train the CIFAR-10 on various classifiers but we were very excited about implementing a Convolutional Neural Network and hence we started with a CNN and we will be adding more classifiers to the list.

1. **Logistic Regression :** This is a supervised machine learning technique used for classification problems. It is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). For this project, we are using the *LogisticRegression* model from the sklearn library.
2. **Support Vector Machine :** SVM is a supervised machine learning technique used for multi-class classification problems.It is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. For this project, we are using the *SVM* model from sklearn library.
3. **Convolutional Neural Network :** As the name suggests, CNN is a sub-category of neural networks which is designed specifically to process input images. We can classify the architecture into two main blocks.

   The first block of the convolution neural network performs the function of feature extraction. To do this, it performs template matching by applying convolution filtering operations. The first layer filters the image with several kernels and returns feature maps, which are then normalized with an activation function and/or resized. This process can be repeated several times. Finally, the values of the last feature maps are concatenated into a vector. This vector defines the output of the first block and input of the second block.

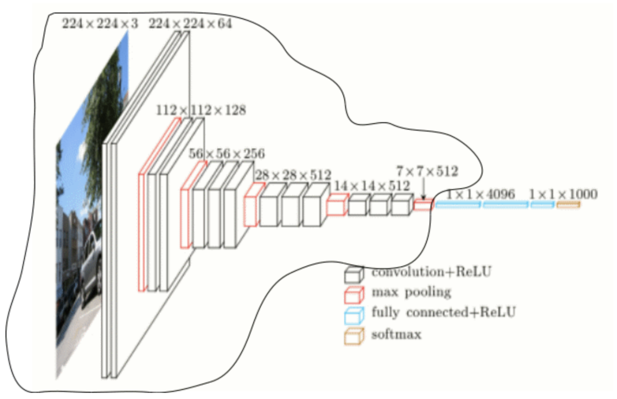


Figure 1: First Block of CNN

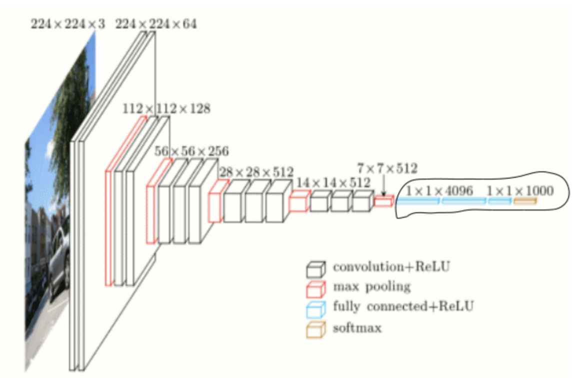


Figure 2: Last Block of CNN

The second block is what all neural networks use for classification. The input vector values are transformed to return a new vector which contains as many elements as there are classes. Each element of the vector represents the probability that the image belongs to that class and hence are between 0 and 1. These probabilities are calculated by the last layer of this block, which uses a logistic function(binary classification) and in our case, softmax function (multi-class classification) as an activation function.

A convolutional network can be formed by stacking different layers. There is no definite way to make a CNN other than the input and the output layer. There are many different types of layers that applies specific filters on the day. Each layer has multiple hyper parameters which on tuning can produce different results.

4 Experiments

4.1 Pre-processing

For this project, we are working with three machine learning classifiers: Logistic regression, Support Vector Machine and Convolution Neural Network

- **Input Shape:** The initial pre processing for Logistic Regression and Support Vector Machine will be different from the pre processing for CNNs because of the different format of the input required by the classifiers. So, the original one batch data is (10000 X 3072) matrix expressed in numpy array. The number of columns (10000) indicates the number of sample data. As stated in the CIFAR-10 data set, the row vector (3072) represents an color image of 32 X 32 pixels. As we are going to use CNN for classification, the original row vector is not appropriate. In order to feed an image data into a CNN model, the dimension of the input tensor should be either (width x height x num_channel) or (num_channel x width x height). It depends on your choice. We are going to use the first choice as that is the default choice in tensorflow's CNN operation. As for Logistic Regression and SVM, each pixel is considered as a feature and hence the data is straightened as 3072 columns (32 X 32 X 3). This is the only difference in pre processing of images for CNN v/s Logistic Regression and SVM.

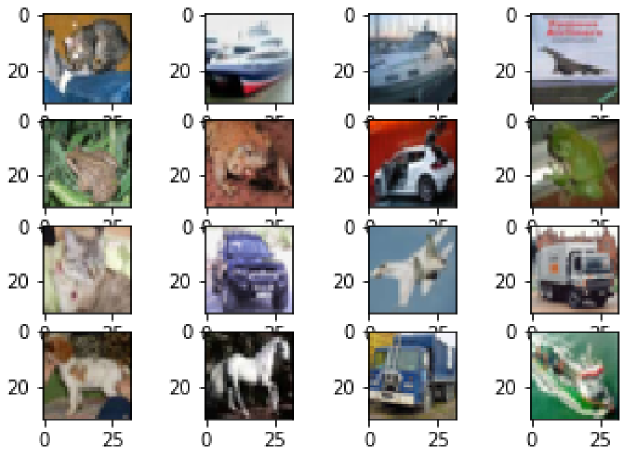


Figure 3: CIFAR-10 Data Set

- **Normalization:** This step is same for all the classifiers we are using. After changing the format of the input data, we normalize the data. Normalization is basically transforming the original image data which is in the range 0 to 255, in the range of 0 to 1. The need of normalization is to facilitate better convergence for the model. If the data is not normalized, when back-propagation process is performed to optimize the network, this could lead to exploding/vanishing gradient problems. In order to avoid the issue, it is better to let all the values be around 0 and 1. Below is the CIFAR-10 data before normalization.
- **One-hot encoding:** This step is only applicable for CNN model. The next step is one-hot encoding. In order to understand the need for one-hot encoding, we need to understand the output of the model. The output of the model is a set of probabilities of each class of image

based on the model's prediction result. In order to express those probabilities in code, a c=vector having the same number of elements as the number of classes of image is needed. CIFAR-10 provides 10 different classes of the image, so we need a vector of size 10.

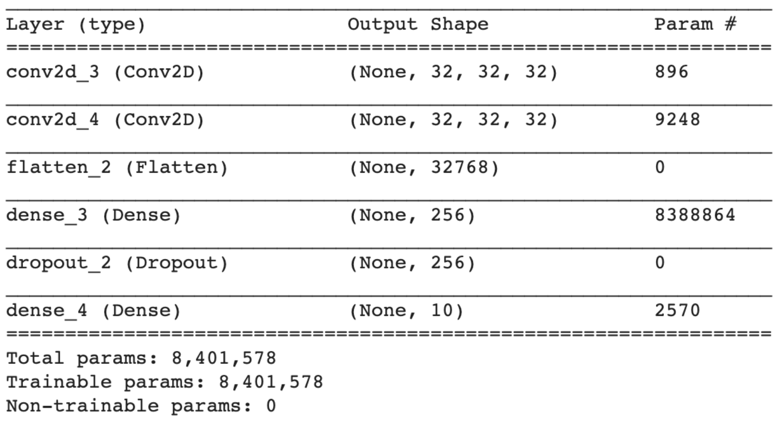
- **Training-Validation Split:** This step is only applicable for CNN model. Lastly, CIFAR-10 data is already split into training and testing set. We split the training part into training part and validation part such that 90% is training and 10% is validation. Now, the data is ready to be fed to a neural network.

4.2 Model Training

Firstly, we pre processed the data as explained above and ran the Logistic Regression using python's machine learning library sklearn and tuned various hyper parameters for the same. We performed the exact same steps for Support Vector Machine algorithm. Then we designed a simple Convolutional Neural Network using keras with tensorflow back-end, numpy to manipulate input as required by the Input layer of CNN, sklearn for splitting the training and validation set and matplotlib to plot results.

Training a neural network for Image data is a cost heavy process in terms of time and hardware requirements. We used google colab notebooks to write the as it provides free GPU which helps accelerate the process. We have tried training data on various versions of CNN by adding and removing various layers in an attempt to understand how a particular layer effects the performance of the model. There were many hyper-parameters to be optimized so for now we tried a couple of hyper parameters.

The figure below shows the CNN architecture we implemented. We have 2 convoluted layers, 1 dense layer, 1 dropout and the last softmax layer with loss set to categorical cross-entropy and activation function Relu for all layers except last.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
flatten_2 (Flatten)          (None, 32768)             0         
_________________________________________________________________
dense_3 (Dense)              (None, 256)               8388864   
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570      
=================================================================
Total params: 8,401,578
Trainable params: 8,401,578
Non-trainable params: 0
```

Figure 4: Basic CNN Architecture

4.3 Hyper Parameter Tuning

When creating a machine learning model, we are presented with design choices as to how to define our model architecture. Often times, we don't immediately know what the optimal model architecture should be for a given model, and thus we'd like to be able to explore a range of possibilities. In true machine learning fashion, we ideally asked the machine to perform this exploration and select the optimal model architecture automatically. Parameters which define the model architecture are referred to as hyper-parameters and thus this process of searching for the ideal model architecture is referred to as hyper-parameter tuning.

- **Logistic Regression :** Here, we tried different values of C and penalty. C in Logistic Regression is inverse of regularization strength. It must be a positive float. Like in support vector machines, smaller values specify stronger regularization. And penalty is used to specify the norm used in the penalization.

- C: [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- penalty: [l1, l2]

Best Parameters: C = 0.1 and penalty = *l1*

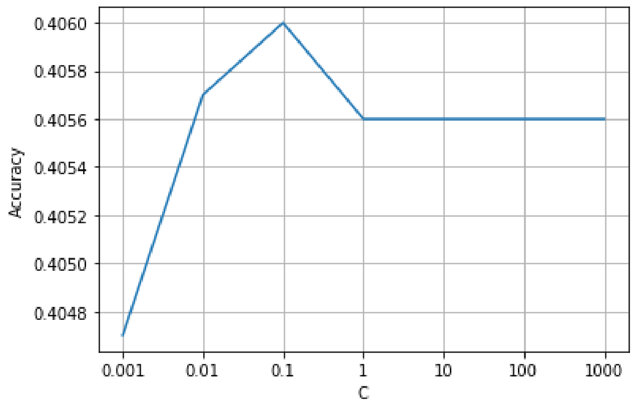


Figure 5: Hyper-parameter Tuning for Logistic Regression

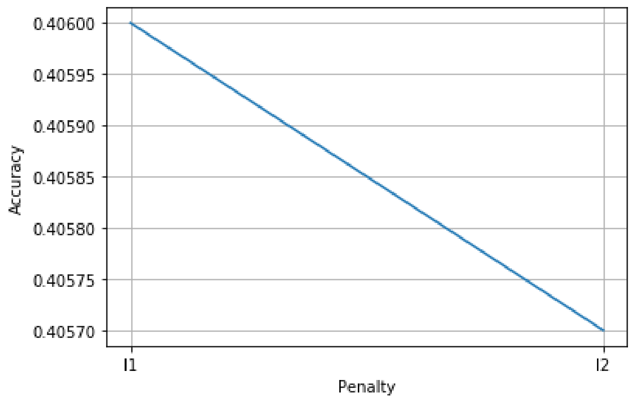


Figure 6: Hyper-parameter Tuning for Logistic Regression

- **SVM :** For SVM, we tried different values of the kernel parameters. The kernel parameter Specifies the kernel type to be used in the algorithm. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples).
  - kernel: [linear, rbf, poly]

  Best Parameter: kernel = rbf
- **CNN :** There are a lot of hyper parameters that can be played with in CNN but for this project we worked on batch size, number of epochs and optimizers. Batch size defines the number of samples to work through before updating the internal model parameters whereas Epoch defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. Optimizers update the weight parameters to minimize the loss function. Loss function acts as guides to the terrain telling optimizer if it is moving in the right direction to reach the bottom of the valley, the global minimum.
  - Batch Size: [8, 16, 32, 54, 128, 256]
  - #Epochs: [5, 10, 15, 20, 25, 30, 50]
  - Optimizers: [sgd, rmsprop, adagrad, adadelta, adam . adamax, nadam]

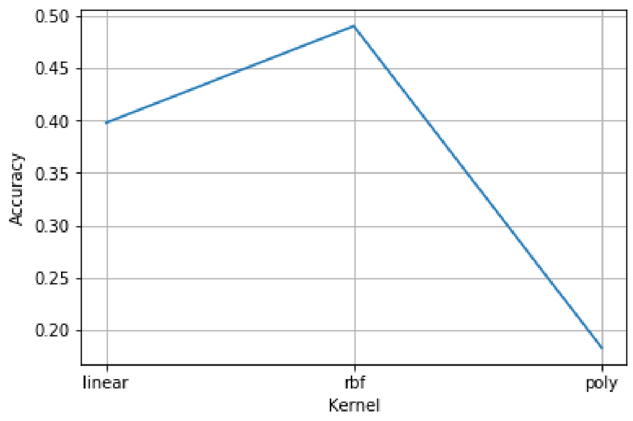


Figure 7: Hyper-parameter Tuning for SVM

Best Parameters: Batch Size = 16, #Epochs = 25, Optimizer = *adadelta*

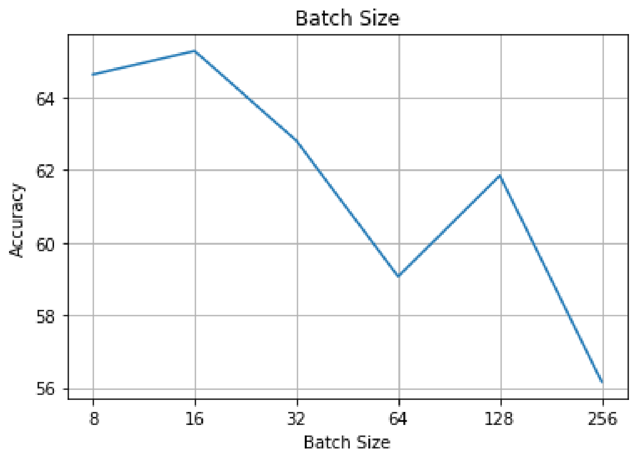


Figure 8: Hyper-parameter Tuning for CNN

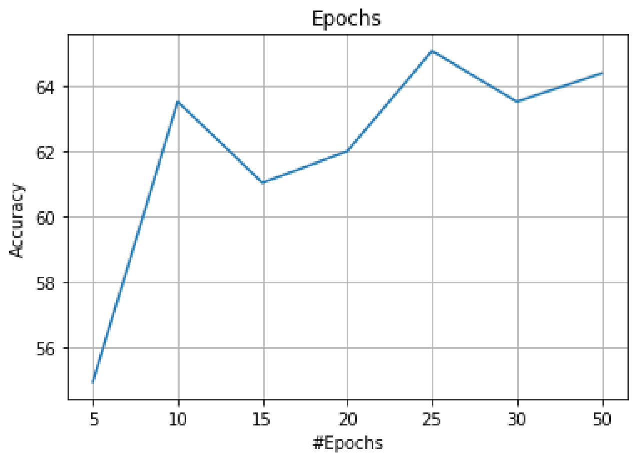


Figure 9: Hyper-parameter Tuning for CNN

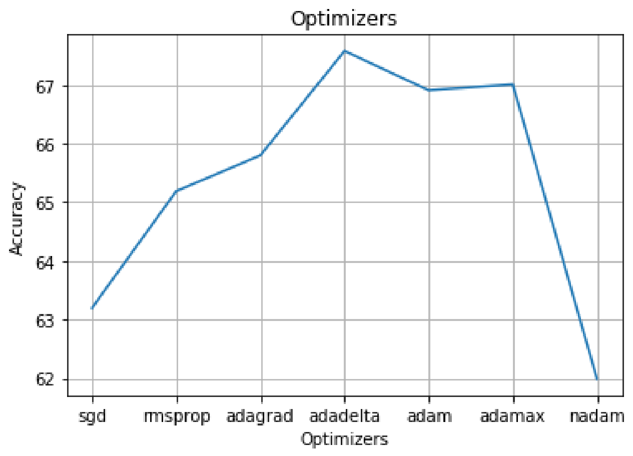


Figure 10: Hyper-parameter Tuning for CNN

5 Results

After, training the all the classification models along with their hyper parameters, we got the following accuracy.

- **Logistic Regression :** ~40%
- **Support Vector Machine :** ~49%
- **Convolutional Neural Network :** ~65%

6 Conclusion and Discussion

From this we can definitely conclude that, Convolution Neural Network is the best for image classification. By manipulating the architecture of CNN like adding more layers, dropouts, pooling and applying early stopping and tuning various other parameters like learning rate, loss functions, kernel size and many other, we can get better accuracy.

But also during running various models we realized that when it comes to training image data for image classification, accuracy may not only be the factor that will dominate model selection. In the currents scenario, the resource required in terms of time and computational power for complex CNNs to run which will give good accuracy are very much. There are free online resources that give you enough memory to run simple models but for complex model we'll have to pay which is difficult as a student for academic purposes. So here are our two cents on how we can possibly deal with such issues. Firstly we can try transfer learning, in transfer learning we can get weights form state of the art already trained models available which took days to train. This way we save most of the training time but the drawback here is the type of data. If the data type you have is different, it becomes useless as the features won't be useful for classification. One another idea is to use autoencoders, we can input the image data to the encoder and remove the decoder part and attach a CNN to the code part. This way only the latent features select by the encoder goes for training. Now this might reduce the accuracy a bit but will also reduce the training time and trainable parameters by a huge factor. This option might not work for CIFAR-10 as the image size is very small (32 X 32) and hence there are not very much latent features to be captured but it is worth exploring in case of datasets like ImageNet which has each image of size (256 X 256) or any other dataset which has high quality images

7 References

[1] Alex Krizhevsky , Ilya Sutskever, & Geoffrey E Hinton. (2012) *ImageNet Classification with Deep Convolutional Neural Networks*

[2] Alex Krizhevsky. (2009) *Learning Multiple Layers of Features from Tiny Images*

[3] Di Lu, Yaru Zhang, & Nealan Vettivelu. *A Comparison of Classifiers on the CIFAR-10 Dataset*

[4] Dan Cirean, Ueli Meier, & (2012) *Jurgen Schmidhuber. Multi-column Deep Neural Networks for Image Classification.*

[6] Karen Simonyan, Andrew Zisserman. *Very Deep convolution networks for Large-scale image recognition*

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition*

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens. *Rethinking the Inception Architecture for Computer Vision*

Github link for the project: `https://github.com/mehtasahil31/CSC522-Final-Project/tree/master`