Methods for CIFAR-10 Image Classification

Yulun Lu | Derek Mitchler | Andre Tom

1 Introduction

Image classification is a longstanding problem in the computer vision field of statistical machine learning. Its applications are numerous, but one of the more recent applications for image classification lies in autonomous vehicles. Self-driving cars rely on their cameras and other sensors for input data, which then need to be processed before the car can make a decision. For example, autonomous cars need to be able to recognize and differentiate between road hazards, other vehicles, pedestrians, stop signs and traffic lights, and many other things we encounter when driving. By advancing our collective techniques for image classification, we can improve the accuracy and speed of object recognition in autonomous cars to help make them safer.

One of the most well-known image datasets is CIFAR-10 [1], which consists of 60,000 32×32-pixel color images divided equally into 10 disjoint categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. This dataset is split into a training set of 50,000 images and a test set of 10,000 images. The training set is further split into 5 batches, each with 10,000 images. The 10 categories are equally represented in each subset in order to prevent bias when training models.

In this project, we decided to focus most of our efforts on Neural Networks since our research has led us to believe this to be the best technique for classifying images of this format. In particular, we chose to implement a Convolutional Neural Network (CNN) as well as a Feedforward Neural Network (FNN) so we could compare their performance. We also decided to include Logistic Regression as one of our methods as recommended by the professor. Based on our preliminary research, we believed that of the methods we chose to investigate, CNN would perform the best on this dataset. This hypothesis turned out to be correct in our experiments.

2 Methodology

We implemented several different machine learning classification techniques in order to build models from the CIFAR-10 dataset; specifically Logistic Regression (via Sci-Kit Learn as well as Pytorch), Convolutional Neural Networks (via Keras and Tensorflow), and Feedforward Neural Networks.

Following the instructions on Alex Krizhevsky's website [1] for unpacking the CIFAR-10 dataset, we first "unpickled" the compressed files to extract the data and label names into Python. We then imported the dataset into a Pandas DataFrame for easier management. We stored each images' RGB pixel information as an array in our DataFrame's `'data'` column and its corresponding label in a separate `'label'` column. The text labels for each image were originally encoded as integers 0-9, so for visualization and data verification purposes we used the provided labels to add a third column (`'name'`) to our DataFrame containing the text label (or image class) for each row.

For the parts involving Pytorch, we loaded the dataset from "`TORCHVISION.DATASETS`" in order to transform it to the form can be loaded by the data loader of Pytorch. The image data is transformed to tensor when the dimension of each image is $3 \times 32 \times 32$. The labels are stored as integers.

Figure 1: Ten random images from each class of the CIFAR-10 dataset.

3 Implementation

3.1 Logistic Regression via Sci-Kit Learn

We first tried Logistic Regression using Sci-Kit Learn but were seeing lower than expected accuracy of about 35%. We attempted to use a GridSearch in order to automatically find the best hyper-parameters for this method, but after running for many hours we failed to see improved results with any combination of parameters. After more research, we decided to try a multiclass classifier in Sci-Kit Learn called One-vs-the-rest (OvR). We used Logistic Regression as the estimator for this method and finally saw some improved accuracy in our results. After manually tweaking the parameters, we found a combination that provided better accuracy than we had seen with plain Logistic Regression with much quicker runtimes. Using an '`L2`' penalty and '`sag`' solver, we were able to get around 40% accuracy for classifying the test set of images using this model.

3.2 Logistic Regression via Pytorch

For Logistic Regression with Pytorch, since the dataset can be transformed to tensor by Pytorch's library, we skipped the step of formatting the data. We began with making the dataset iterable by load it with batch size of 100. Then, we create a model class that defines the architecture of Logistic Regression. Then, we train the model started from 3000 iterations to 20000 iterations to see how the accuracy and run time changes.

3.3 Feedforward Neural Networks

After trying Logistic Regression with Pytorch, we then move on to a 2-layer neural network to classify the images. Like doing logistic regression, we began with transforming the data and make

it iterable, and we create the neural network class with the ReLU activation function. Then we train the model with 8 and 30 epochs and record the accuracy.

3.4 Convolutional Neural Network

Convolutional neural networks were designed to map image data to an output variable. Their advantage over recurrent or feedfoward neural nets lies in their ability to develop an internal representation of a two-dimensional image which allows the model to learn position and scale in variant structures in the data for a deeper understanding and more accurate results. It does this by learning features at various levels of abstraction but more layers also increases the possibility of over-fitting which we addressed with various regularization techniques.

Luckily, the images are available in a ready-to-use format - the images are all singular, they are 32×32, and in color (RGB). The classes are known as well so we used a one-hot encoding on each sample, transforming each label into a 10-element binary vector with each class index being set to '1'. In addition, we normalized the data by converting to float and calculating the z-score for a smoother range of weights. We also augmented the data by making copies with small, random modifications such as rotation or flipping the image.

For our model, we utilized an architecture similar to the VGG models which entails stacking convolutional layers with 3×3 filters followed by a max pooling layer. These layers form a block which are repeated - the number of filters are increased as well as the depth of the network. Each layer uses a ReLU activation function which outputs the input directly if it is positive, otherwise, zero. This is important because to use stochastic gradient descent with backpropagation of errors, we need an activation function that acts linear but is actually non-linear to bridge together complex relationships.

Our model consists of 6 convolutional layers followed by a flatten layer; the output layer is a dense layer of 10 nodes corresponding to the 10 classes with a softmax activation function. Our regularization techniques consisted of Dropout layers, weight decay, and batch normalization. In each block we increased the amount of dropped units from 20, to 30, to 40%. Weight decay update the loss function to penalize the model in proportion to the size of the model weights as larger weights may potentially lead to a more complex and less stable model. Finally, batch normalization normalizes the activation of the previous layer at each batch which acts as a stabilizer.

4 Results

4.1 Logistic Regression

Using the OvR classifier in Sci-Kit Learn, we saw a maximum accuracy of about 40% when classifying the test image set. As can be seen from the confusion matrix in Figure 2, many images are misclassified using this model. For example, many dog images are predicted to be cats and vice versa. Similarly, many automobile images are predicted to be large work trucks and vice versa.
Similar to the accuracy using Sci-Kit Learn, the accuracy using Pytorch is also below 40%. The accuracy was about 34% with 3000 iterations and 38% with 20000 iterations, and the run time was about 10 seconds for every epochs which is 500 iterations.

4.2 Feedforward Neural Networks

The Feedforward Neural Networks made an improvement on the accuracy which exceed 50%. When there are 8 epochs, the accuracy of FNN was about 50%, and the accuracy increased to about 53%

after increasing the number of epochs to 30.

4.3 Convolutional Neural Network

We went through a few iterations, finally settling with 85.7% accuracy on the test data. This is a marked improvement of when we first started; without utilizing regularization techniques we ended up with a 74.7% score. Additionally, we were massively underfitting the data. We also attempted to use k-fold cross validation but this was very computationally intensive and quintupled the training time, resulting in a negligible improvement with a score of 75.1%. With our final model we first ran it with 65 epochs as a trial run to a score of 83.9%. Consulting the plotted results, we realized the accuracy improved with the number of epochs so we increased the parameter to 125. It should be noted that the number of epochs we trained on increased from 50 in the latter two tests, to 65, to 125 in the final training run.

4.4 Summary

By doing this project, we have proven our hypothesis that CNN will have the best performance on this dataset. We would have liked to run more tests with the CNN but the server began running incredibly slowly in the final days before the due date, taking an excess of 9 hours per training period. We attempted to counteract this by using AWS Sagemaker but realized the notebook was running on a CPU, not the GPU. In order to fix this we were directed to contact Amazon but decided to just run the notebook locally as time was limited. In a future test perhaps we could have added more epochs as the accuracy seemed to have a positive linear correlation with the number of epochs. In addition, we could have re-introduced k-fold cross validation now that we have a more accurate model, or added another couple layers.

References

[1] Alex Krizhevsky, *CIFAR-10 and CIFAR-100 datasets*. 2009.
`http://www.cs.toronto.edu/~kriz/cifar.html`

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

Appendix

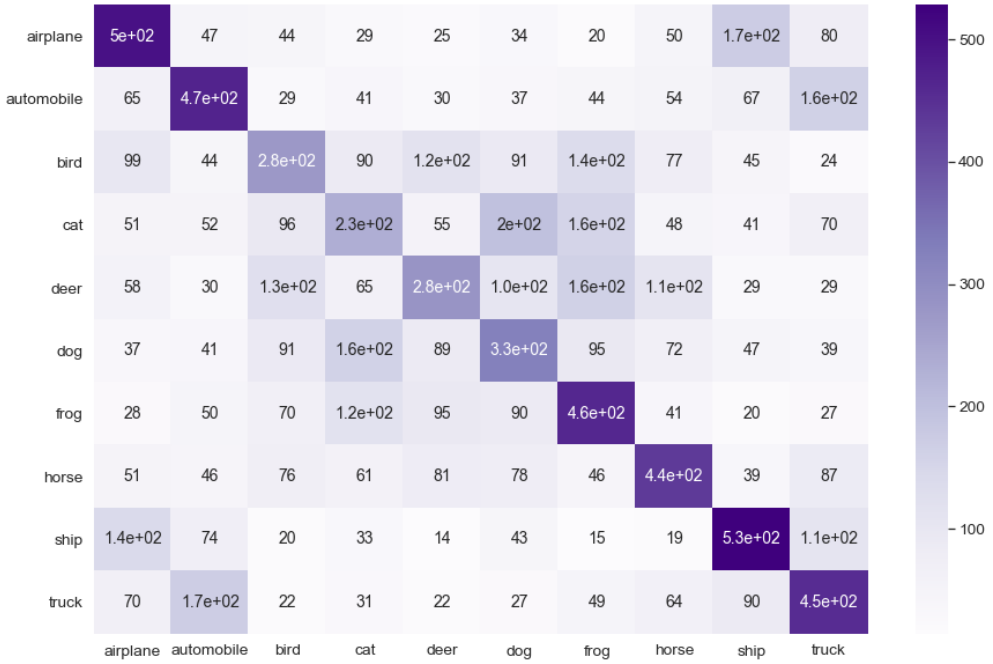

Figure 2: One-vs-the-rest Confusion Matrix.

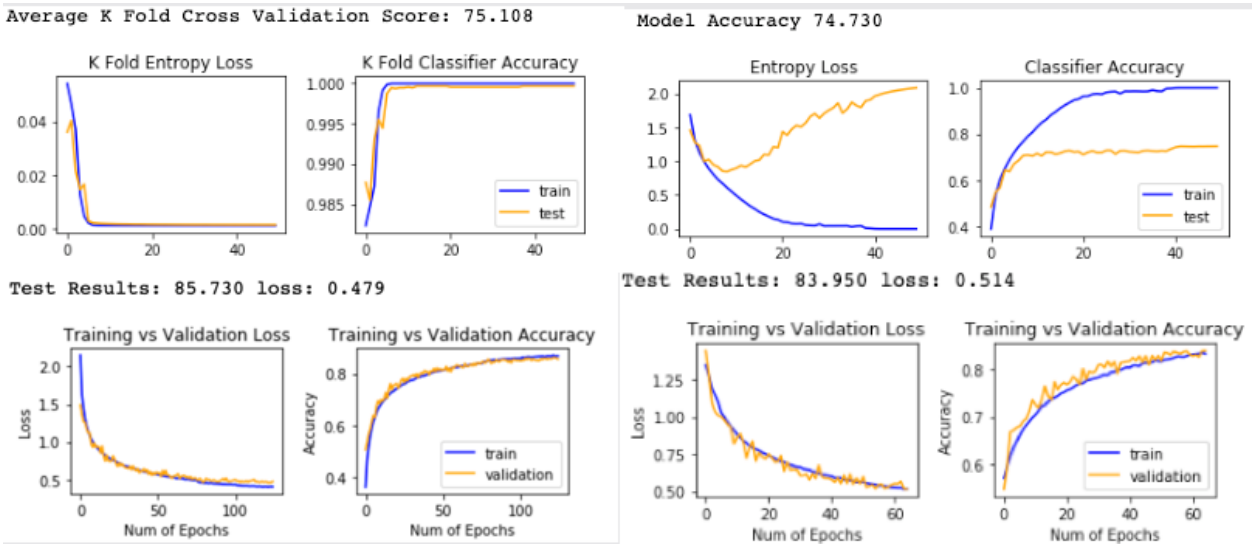

Figure 3: Convolutional Neural Network Accuracy Scores. **Top Left:** 5-Fold Cross Validation with 50 epochs. **Top Right:** Baseline model with 50 epochs. **Bottom Left:** Baseline model with dropouts, weight decay, batch normalization, and 65 epochs. **Bottom Right:** Baseline model with dropouts, weight decay, batch normalization, and 125 epochs.