Mini Project 2

Noah Schiro
Jason Swope

November 24th, 2023

EE456: Introduction to Neural Networks

Pennsylvania State University

1 Introduction

In this project, we aim to demonstrate the capabilities of a convolutional neural networks (CNN), particularly their efficacy in classification tasks for imagery. For this, we are using the CIFAR-10 dataset. This dataset consists of 60,000 images spread across 10 classes where each image is 32x32x3 (width, height, and channels) in size. For the model, we decided to create our own architecture to see how our model performs against something that has been well optimized for this task (VGG-16, ResNet, or ImageNet for instance).

The objective of this project is to create a balance between inference speed and model accuracy. In embedded applications, computer vision still needs to be heavily optimized in order to run on the target device at reasonable frame rates. Using state of the art architectures such as VGG-16 is not feasible for certain compute platforms if one expects the platform to run at 60 frames-per-second. VGG-16 contains 138 million parameters, leading to extremely slow inference. We will create a smaller model with 150k parameters, while still getting appreciable performance.

We seek to deliver the following:

1. Plot of training loss over epochs
2. Plot of validation loss over epochs
3. Plot of accuracy on the validation set over each epoch
4. Confusion matrix for our classes (at the end of training)
5. Precision and recall for each class (at the end of training)
6. Associated code to replicate efforts

2 Implementation

2.1 Model architecture

The model consists of 3 convolutional layers, a flattening step, a feed forward network, and finally a softmax on the output.

1. Convolutional block 1
    (a) 2D convolution (3 in-channels, 64 out-channels, kernel size 3, padding 1)
    (b) Relu
    (c) 2D Maximum pooling
2. Convolutional block 2
    (a) 2D convolution (64 in-channels, 64 out-channels, kernel size 3, padding 1)
    (b) Relu
    (c) 2D Maximum pooling
3. Convolutional block 3
    (a) 2D convolution (64 in-channels, 64 out-channels, kernel size 3, padding 1)
    (b) Relu
    (c) 2D Maximum pooling
4. Flatten the tensor
5. Classification step

(a) Linear layer (64 dimensional input to 1024 output)

(b) ReLU

(c) Linear layer (1024 dimensional input to 10 classes)

6. Softmax

2.2 Data preprocessing

The data was read in using Pytorch's torchvision library, converted to a tensor, and normalized using 0.5 as the mean and standarad deviation for each channel. In the future, this could be improved by computing the actual mean and standard deviation for the library.

We did not select a train-test split as the CIFAR-10 dataset comes with a seperate training and testing set already.

2.3 Model training

The model was trained in a standard fashion, with some notable exceptions. First is the hyperparameters settings:

- Batch size was set to 64. Such a high batch size seemed to lead to quicker convergence as well as faster processing due to GPU parrallelism.
- Epochs was set to 50. We observed minimal improvements after 50 epochs.
- Intial learning rate was set to 0.1. While this seems quick high, we didn't run into any exploding gradient issues with this and it led to extremely quick convergence.

- Each epoch, we exponentially decayed the learning rate with the following rule

$$lr_{e+1} = lr_e * 0.95$$

After 50 epochs, our learning rate would be

$$0.1 * 0.95^{50} = 0.007694498$$

In the interest of time and resources, several strides were also made to decrease the training time. Firstly, a NVIDIA 3060 GPU was deployed as opposed to training with a CPU. The typical optimization that come with using a CUDA enabled card were also used, such as memory pinning and increasing CPU workers in the dataloading step of training.

On top of this, this GPU happens to use the Ampere architecture, which contains tensor cores. These cores should see a 2-3x speed up if we decide to use automatic mixed precision (AMP). This should also decrease our memory requirements and therefore we can increase our batch size (which is typically limited by the VRAM available to you). With these considerations in mind, we did take advantage of AMP optimizations.

At teh end of training, we were able to acheive a training time of 2 minutes and 56 seconds for a model that acheives 72.61% accuracy.

3 Results

The CNN performed very well. The model reached a validation accuracy of 72.61% and a validation loss of 1.73609. We found 50 epochs optimized the model without wasting extra compute

time. The plots of accuracy vs epochs and loss vs epochs is shown in Figure 1. Figure 1 shows that there is a discrepency between the training and testing dataset, as three is a clear separation between the training and testing datatsets in both accuracy and loss. This incidcates the model is slightly overfitting the training dataset, so randomly choosing training and testing datasets ourselves or using cross-validation might have been advantageous.

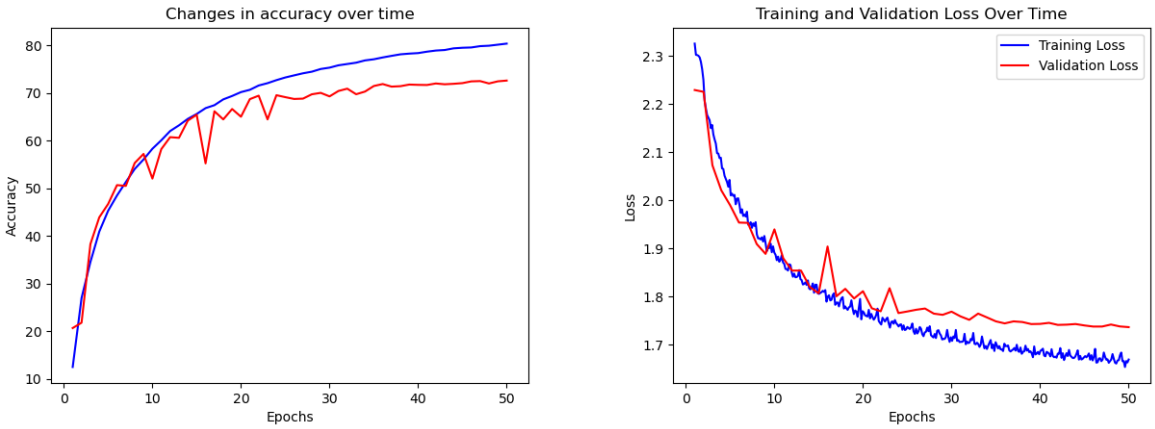

Figure 1: Plot of the accuracy vs epochs (Left) and the plot of the loss vs epochs (Right)

The confusion matrix of the model is shown in Figure 2. Correctly classified datapoints appear along the primary diagonal, and Figure 2 shows that the model performed quite well. However, there are some classes that the model struggles to separate, although these classes are logical such as cats and dogs, and ships and airplanes.

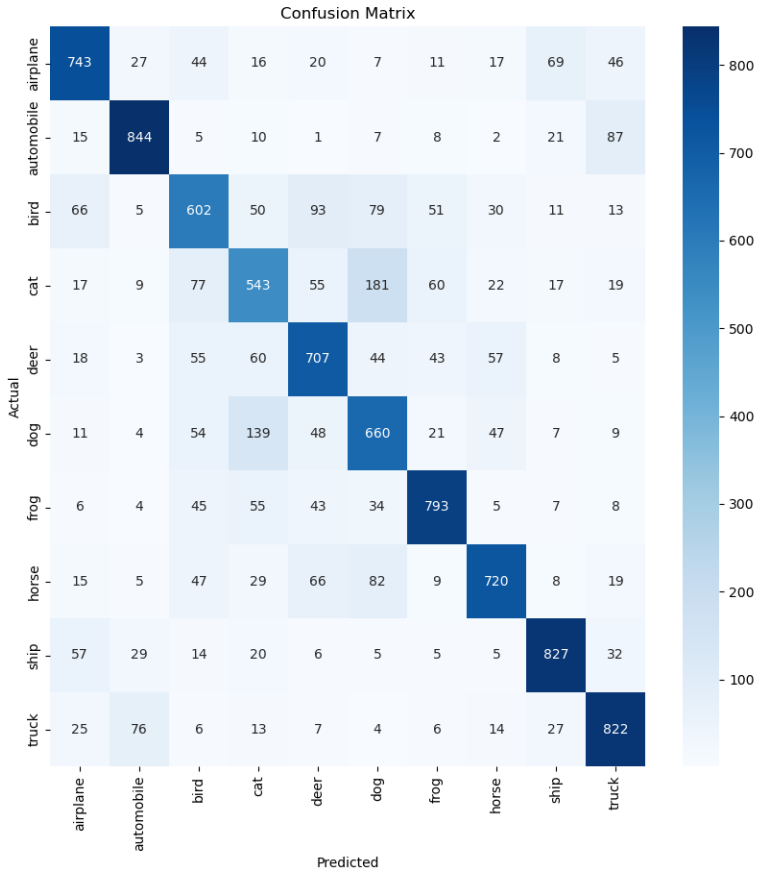

Figure 2: Plot of the confusion matrix

Looking at some example images and their classifications shown in Figure 3, it appears the model is quite accurate, achieving a confidence of over 98% on most of the images. However, these example images do show some of the inaccuracies in the model. The model incorrectly classified the frog in the top right, which the model classified as a deer with a decent confidence, which indicates it has learned that deer are brown shapes with green backgrounds. Additionally, the models struggled to accurately predict the automobile in the second row and second coluimn. It did accurately predict the class, however it only was about 50% confident in its prediction, which is considerably lower than the confidence levels of the other examples.

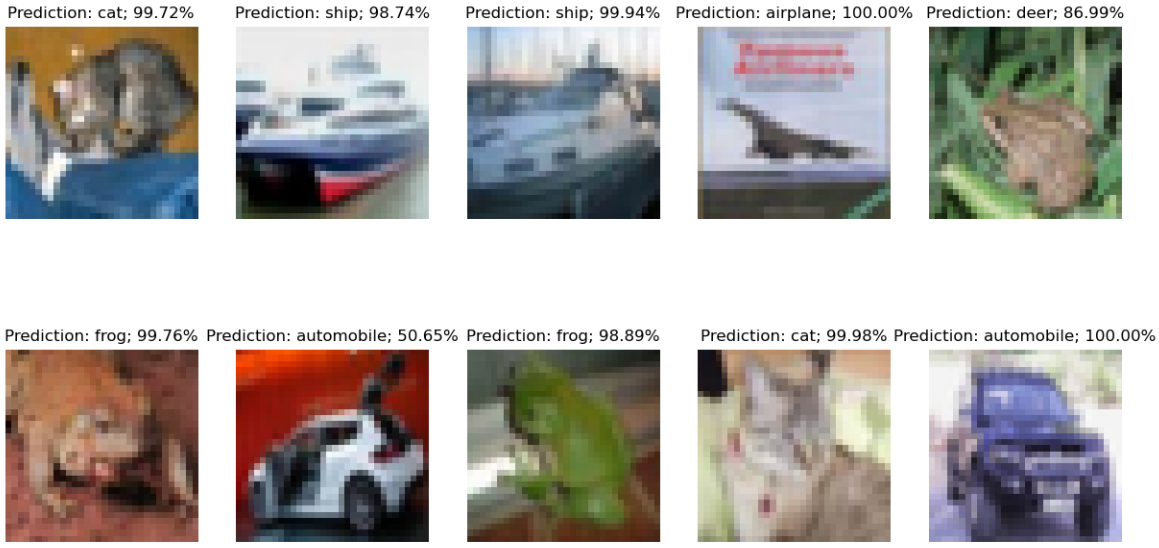

Figure 3: Image of some example images and their classifications

The precision and recall can be seen in Figure 4. The precision and recall for each class are quiet similar, which indicates the model is not prioritizing or leaving behind any of the classes. The best classes are Ships and Automobiles with both precision and recalls above 0.8, and the worst are Cats and Dogs, which have precison and recall below 0.6.

```
Class airplane: Precision=0.7636, Recall=0.7430
Class automobile: Precision=0.8390, Recall=0.8440
Class bird: Precision=0.6344, Recall=0.6020
Class cat: Precision=0.5807, Recall=0.5430
Class deer: Precision=0.6759, Recall=0.7070
Class dog: Precision=0.5984, Recall=0.6600
Class frog: Precision=0.7875, Recall=0.7930
Class horse: Precision=0.7835, Recall=0.7200
Class ship: Precision=0.8253, Recall=0.8270
Class truck: Precision=0.7755, Recall=0.8220
```

Figure 4: Image of the precision and recall for each class