ECSE 4850 - Programming Assignment 3

Philip Paterson

March 2024

1 Introduction - Problem Statement

In Programming Assignment 3, we program a convolutional neural network (CNN) with three convolutional layers, two max-pooling layers, and two fully-connected layers, to perform multi-class classification to classify images into 10 classes in the CIFAR-10 dataset.

The 10 classes in the CIFAR-10 dataset are as follows: "truck," "ship," "horse," "frog," "dog," "deer," "cat," "bird," "automobile," and "airplane." In the dataset there are 60,000 images, with 50,000 being part of the training set and 10,000 being part of the testing set. Each image is a 3-channel colored 32x32x3 image.

In this report, I will describe the structure of the model used, how the experiment was set up, the results of my experiments, analysis on my results, and make concluding remarks.

2 Model Description

2.1 Data Augmentation

I first transformed my data by performing horizontal and vertical flips on the images with the probability of being flipped being $p = 0.5$.

2.2 Convolutional Neural Network Architecture

My model is a Convolutional Neural network with the following layers: three convolutional layers, two max-pooling layers, and two fully-connected layers. The ReLU activation function is used between the convolution layers and max-pool layers, along as being used between the last convolutional layer and first fully-connected layer and between the first and second fully-connected layers. In addition, to avoid overfitting, batch normalization is used between the convolution and activation layers, as well as dropout being utilized between the first and second fully connected layers.

Please note that when I describe my dimensions, I will be using the convention of (height x width x depth) for a three-dimensional tensor. Thus, a 32x32x3 image has a depth of 3, or 3 channels, and a height and width of 32.

2.2.1 Mathematical Description

We first will cover the convolutional layer. Let $\mathbf{X}^{M\times N\times D}$ be the input image with D channels. Thus, for the CIFAR-10 dataset, $M = N = 32$ and $D = 3$. Additionally, define $\mathbf{W}^{c\ K\times K\times D}$ as the filter and $\mathbf{W}_0^c$ as the bias. Also, let the convolutional layer be $\mathbf{C}^{N_r^c\times N_c^c}$ with stride s (assuming zero-padding), where $N_r^c = \frac{M-k}{s} + 1$ and $N_c^c = \frac{N-k}{s} + 1$ are the height and width respectively. Thus, $\mathbf{C}$ can be calculated as follows:
For r = 1 to N_r^c
For c = 1 to N_c^c

$$\mathbf{C}[r][c] = \sum_{l=1}^{D}\sum_{i=1}^{K}\sum_{j=1}^{K}\mathbf{X}[(r-1)\times s+i][(c-1)\times s+j][l]\mathbf{W}^c[i][j][l] + \mathbf{W}_0^c$$

Next, to cover the convolution to activation layer $\mathbf{A}^{N_r^a \times N_c^a}$, where $N_r^a = N_r^c$ and $N_c^a = N_c^c$.
For r = 1 to N_r^a
For c = 1 to N_c^a

$$\mathbf{A}[r][c] = \text{ReLU}(\mathbf{C}[r][c])$$

Afterwards, from the activation layer to the pooling layer. If we define the pooling neighborhood size to be $d \times d$ and the pooling layer as $\mathbf{P}^{N_r^p \times N_c^p}$, where $N_r^p = \frac{N_r^a - d}{s} + 1$ and $N_c^p = \frac{N_c^a - d}{s} + 1$ and s is the pooling stride, the max-pooling layer can be calcualted as follows:
For r = 1 to N_r^c
For c = 1 to N_c^c

$$\mathbf{P}[r][c] = \max_{1 \le i \le d, 1 \le j \le d} \mathbf{A}[(r-1) \times s + i][(c-1) \times s + j]$$

From the pooling layer to the fully connected layer, we first need to vectorize $\mathbf{P}$ to $\vec{P}$. This can be done as follows:

for $r = 1$ to N_r^p **do**
  for $c = 1$ to N_c^p **do**
    $\vec{P}[(r-1)N_c^p + c] = P[r][c]$
  end for
end for

Afterwrds, the fully connected layer $\mathbf{F} \in \mathbb{R}^{N^F \times 1}$ can be found as follows:

$$\mathbf{F} = \phi((\mathbf{W}^F)^T \vec{P} + \mathbf{W}_0^F),$$

where ϕ is the activation function, $\mathbf{W}^F$ are the weights, and $\mathbf{W}_0^F$ is the bias.

After propagating forward throughout all the layers, the output layer can be calculated as follows:

$$\hat{y} = g\left((\mathbf{W}^o)^T \mathbf{F} + \mathbf{W}_0^o\right)$$

where g is the output function (which is the softmax function in this model), $\mathbf{W}^o$ are the output layer weights and $\mathbf{W}_0^o$ is the bias.

2.2.2 CNN Architecture Overview

The following layers and dimensions of layers in Table 1 can be calculated using the equations in the previous section, section 2.2.1.

2.2.3 CNN Model Visualization

The CNN model visualization can be seen in Figure 1.

3 Experimental Setup

Here I will describe the setup of the experiments.

3.1 The CIFAR-10 Dataset Split

The 10 classes in the CIFAR-10 dataset are as follows: "truck," "ship," "horse," "frog," "dog," "deer," "cat," "bird," "automobile," and "airplane."

In the dataset there are 60,000 images, with 50,000 being part of the training set and 10,000 being part of the testing set. Each image is a 3-channel colored 32x32x3 image.

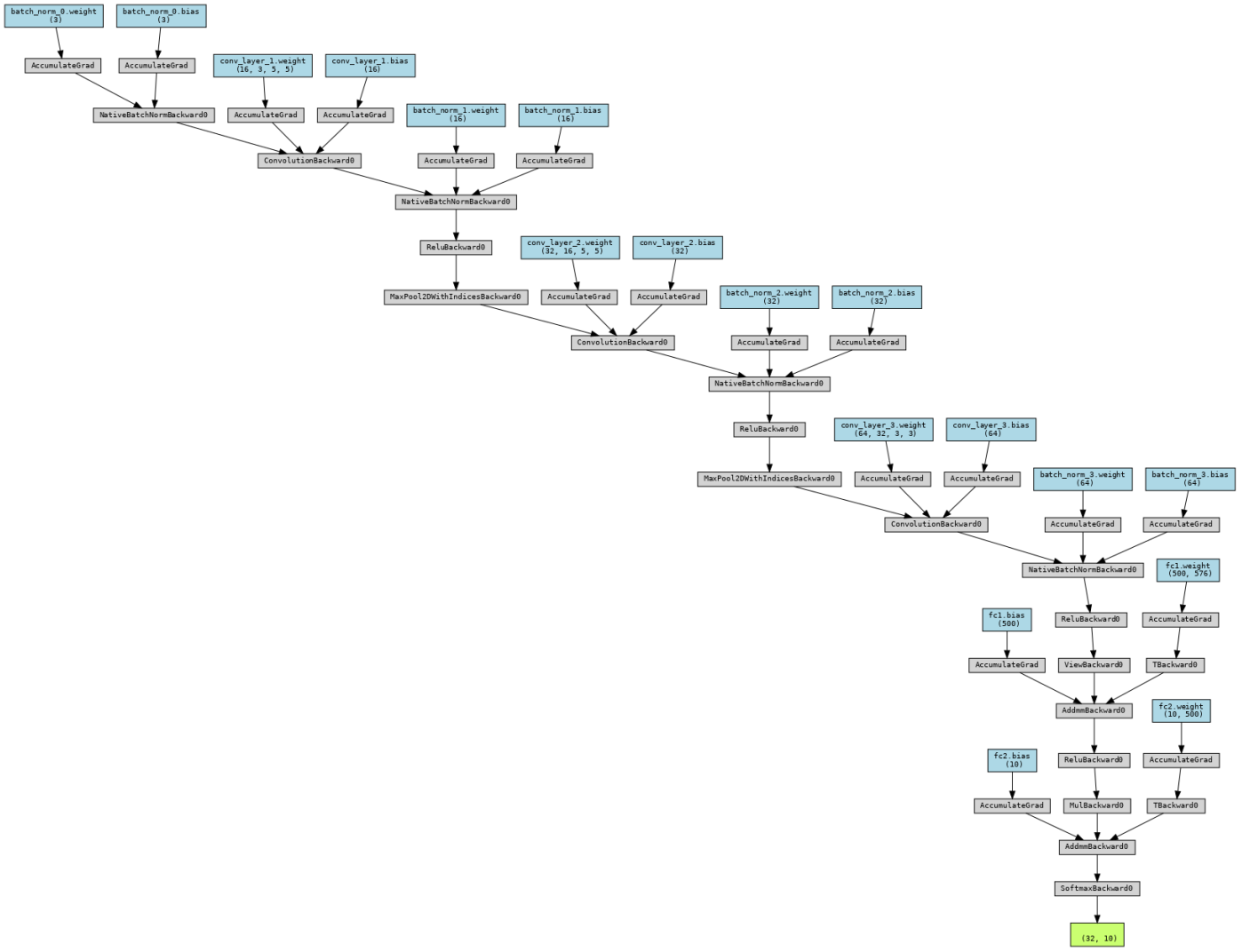

Figure 1: torchviz visualization

Layer	**Dimensions** ($H \times W \times D$)
Input	$32 \times 32 \times 3$
Batch Normalization (2D)	-
Convolutional Layer 1 (16 5×5 filters)	$28 \times 28 \times 16$
Batch Normalization (2D)	-
ReLU Activation	-
After Max Pooling 1 (Pooling Area: 2×2; Stride: 2)	$14 \times 14 \times 16$
Convolutional Layer 2 (32 5×5 filters)	$10 \times 10 \times 32$
Batch Normalization (2D)	-
ReLU Activation	-
After Max Pooling 2 (Pooling Area: 2×2; Stride: 2)	$5 \times 5 \times 32$
Convolutional Layer 3 (64 3×3 filters)	$3 \times 3 \times 64$
Batch Normalization (2D)	-
ReLU Activation	-
Flatten	576
Fully Connected Layer 1	500
ReLU Activation	-
Dropout (p = 0.1)	-
Fully Connected Layer 2	10
Sigmoid Activation	-
Output	-

Table 1: Convolutional Neural Network Layers and Dimensions

3.2 Hyper parameters

- Batch size: 64
- Epochs: 100
- Optimizer: Stochastic Gradient Descent (`torch.optim.SGD`)
  - Learning Rate = 0.006
  - Momentum = 0.9

3.3 Weight Initialization

The weight initialization was performed using the default PyTorch weight initialization when an instance of the model is created. For example, for linear layers, PyTorch initializes weights using following formula (from `https://github.com/pytorch/pytorch/blob/main/torch/nn/modules/linear.py`):

$$\mathcal{U}(-\sqrt{k}, \sqrt{k})$$

where

$$k = \frac{1}{\text{in feature number}}$$

And initializes the convolutional layer weights using a uniform distribution between a positive and negative standard deviation, derived from the input channels (source: `https://github.com/pytorch/pytorch/blob/08891b0a4e08e2c642deac2042a02238a4d34c67/torch/nn/modules/conv.py#L40-L47`).

4 Experimental Results

Average Classification Error on the Test Dataset: 27.2%
Average Classification Accuracy on the Test Dataset: 72.8%

The following figures describe my experimental results:

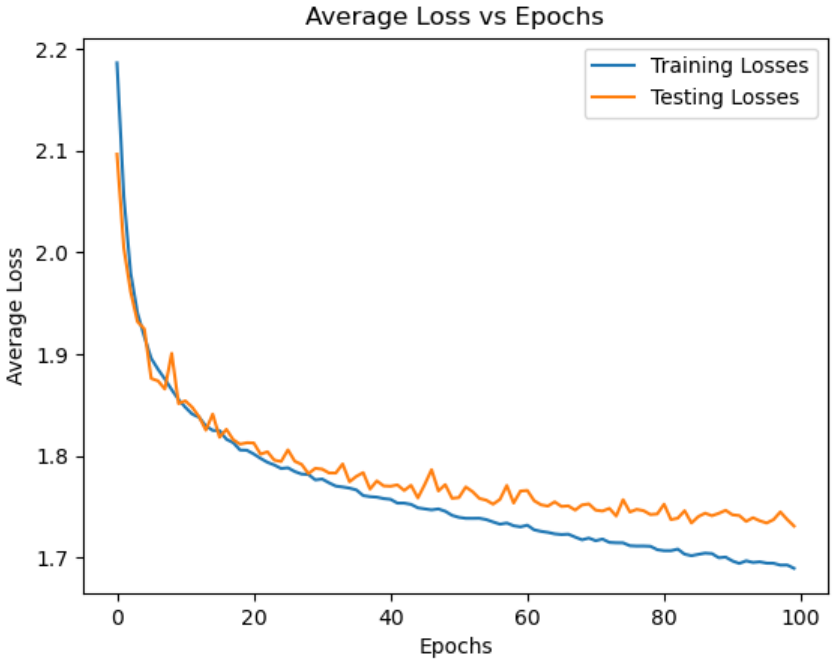

Figure 2: Average Loss vs Epochs

- Average loss versus epochs: Figure 2
- Average accuracy vs epochs: Figure 3
- Confusion Matrix for the 10 classes: Figure 4

5 Analysis

The visualization of my filters can be seen in Figure 5.

Based on my setup and results, I was able to achieve an average test accuracy of 72.8%, which was about $4-5\%$ below my train accuracy.

In order to achieve this accuracy, the biggest factor that helped was batch regularization, as it allowed for faster convergence without as much overfitting. I believe this was because there was much variation in the input dataset, which regularization helps with by not being as affected by outliers.

I also further reduced overfitting using dropout and data augmentation for the inputs, where I transformed the images by flipping them horizontally and vertically with equal probabilites ($p = 0.5$).

As for the confusion matrix, it seems that dogs and cats are often confused, which I speculate to be because dogs and cats are both animals with similar features. In general, it seems as though vehicles were confused with other vehicles and animals were confused with other animals more often (although airplanes and birds were also often confused, which follows as airplane designs are inspired by birds and both involve flight).

6 Conclusion

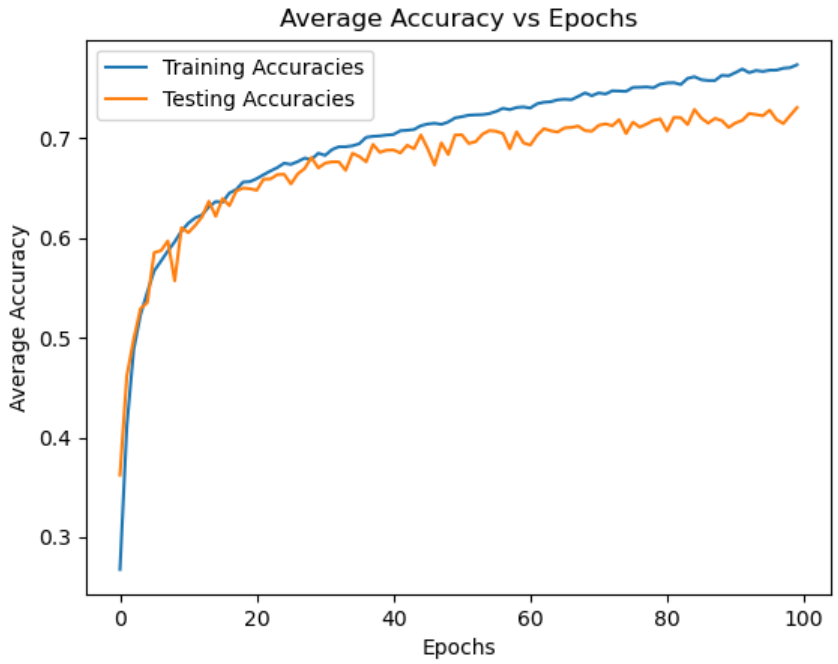

Figure 3: Average Accuracy vs Epochs

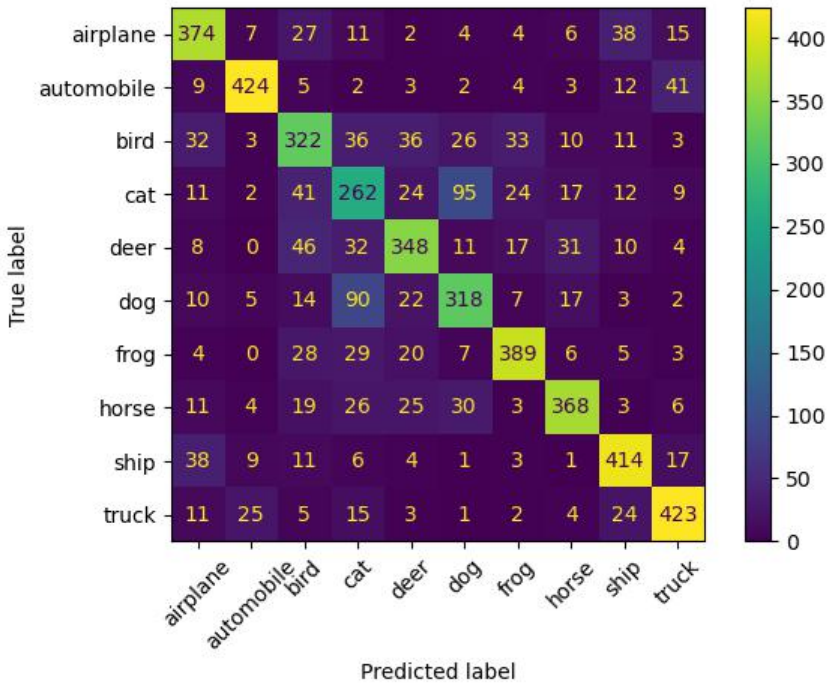

Figure 4: Confusion Matrix for 10 classes in CIFAR-10 Dataset

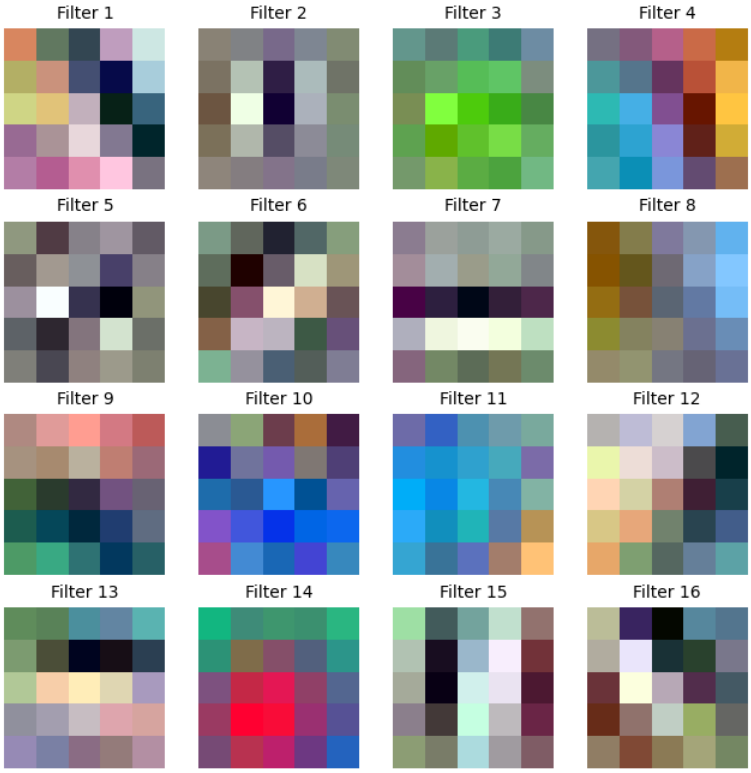

Figure 5: 16 Filters for the first Convolutional Layer