Programming Assignment 3 - Convolutional Neural Network with CIFAR-10 Data
Hayleigh Sanders RIN #661195735

Purpose

Convolutional neural networks offer an advantage over multilayer neural networks in the ability to process multi-dimensional data. The following project is an implementation of a convolutional neural network to classify images from the CIFAR-10 dataset.

CNN Theory

A convolutional neural network is composed of an input, several sub-stages, fully connected layers and an output composed of class label predictions. These sub-stages are composed of the convolution, activation and pooling steps. The partially connected layer is composed of the convolution layer, activation layer and pooling layer. Multiple input channel convolution on an image is given by,

$$\sum_{l=1}^{D} \sum_{i=1}^{K} \sum_{j=1}^{K} I(r+i-1, c+j-1, l)\, W(i,j,l) + W_0$$

Where W and W_0 are the weights and their biases for the layer. Convolution causes the layer to be partially connected, meaning that each node is only connected to part of the nodes in the previous layer. This parameter sharing allows the CNN to be more computationally effective than a fully connected neural network, due to having less parameters. After convolution, the resulting tensor C(c,r) is passed through an activation function to become A(c,r). Here, ReLu is used, and is given by

$$A(c,r) = ReLu(C(c,r)) = max(0, C(c,r))$$

Next, A(c,r) is passed through the pooling layer, where its dimensions will be reduced through the pooling process. Max pooling was used in this project, given by,

$$P[r][c] = \frac{1}{dxd} \sum_{i=1}^{d} \sum_{j=1}^{d} A\big[(r-1)\cdot s+i\big]\big[(c-1)\cdot s+j\big]$$

The maximum value in a pixel neighborhood is chosen for the new pixel value in P[r][c].

The result from the pooling layer can then be fed into another convolutional layer, activation layer and pooling layer for a number of layers. Here, three convolutional filters were used, with two pooling layers.

The result from the partially connected layer is then flattened into a single vector of values and used to create the fully connected layer. Each fully connected layer applies weights to predict the final value. The final fully connected layer will be a 1XN vector, where N is the number of classes, representing the probabilities that an image belongs to each class. Backpropagation is then performed through the layers, from the output layer to the input layer, to obtain the weight gradients. The weights for each layer are trained with these gradients over a number of epochs according to a loss function. This project uses the cross-entropy loss function given by,

$$L(y, \hat{y}) = -\frac{1}{M}\sum_{i=m}^{M}(y^m \log(\hat{y}^m) + (1 - y^m)\log(1 - \hat{y}^m))$$

CNN architecture

The input layer is a (32*32*3) tensor representing a 32*32 pixel image with each pixel having three values for the RGB channel. The inputs were normalized by calculating and subtracting the mean, and scaled between 0 and 1 by dividing the data by 255. The images correspond to one of 10 class labels as a value from 0 to 9. The corresponding label for each image was one-hot encoded to a 1x10 array, with the position of the correct label equal to one and the rest zero. These arrays were then combined into a two dimensional label tensor, with the rows corresponding to the size of the number of images processed in a training batch and the columns corresponding to the number of classes.

This model was constructed with 10 layers in total, with three convolutional layers, two pooling layers, a flattened layer, three fully connected layers and an output. The first convolutional layer consists of 32 5x5 filters, corresponding to 2,400 weights.The input tensor was convolved with the first filter using the conv2d function, producing a 1x32x32x32 tensor. The stride of all convolutional filters is one, meaning the filter window moves by one pixel for every convolving step, and the padding is zero, meaning that the image stays the same size for each convolution step. The result was then activated using the ReLu activation function, producing a tensor of the same size. The pooling layer was then formed with the max pool function using this result from the activation layer, forming a 1x16x16x32 tensor. The max pooling layer shrinks the image tensor by taking the largest value in its window. The stride was 2 for all pooling layers, meaning the tensor will shrink by a factor of 2 after each pooling layer.

The second convolutional layer consists of 32 5x5 filters applied to the max pooling result with the conv2 function, resulting in a 1x16x16x32 tensor. The second convolutional layer has 2,400 weights. It was activated with the ReLu function like the previous layer. When the max pool function is applied to this result, it becomes a 1x8x8x32 tensor.

The third convolutional layer consists of 64 3x3 filters, corresponding to 1,728 weights. When the tensor from the previous pooling layer is convolved with the filters, it becomes a 1x8x8x64 tensor. This tensor is then flattened into a 1x4096 vector, and converted into a 1x128 fully connected layer. The result is then converted into a 1x256 fully connected layer, and then a 1x512 fully connected layer.

Finally, the last fully connected layer is converted to a 1x10 output corresponding to the predicted probabilities of each class.

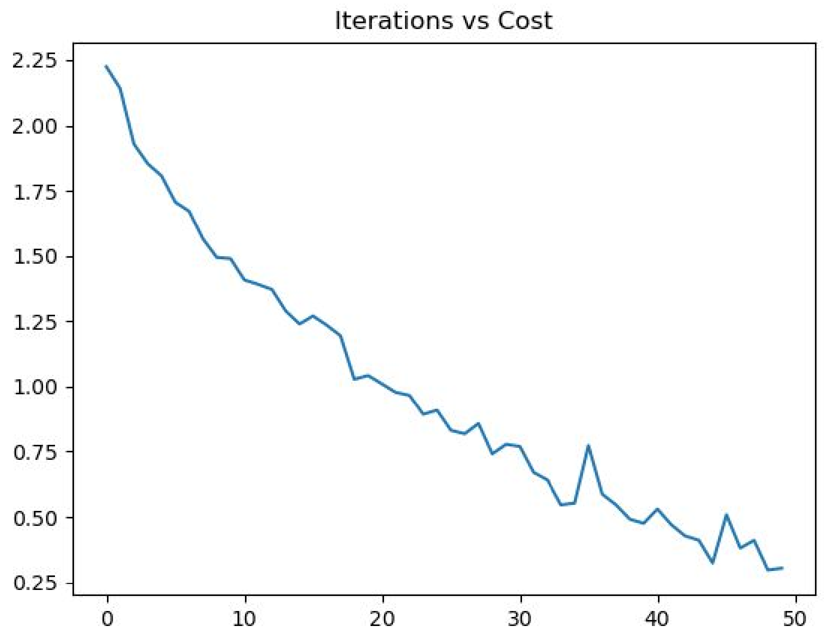
The cost was computed using the tensorflow nn.softmax_cross_entropy_with_logits function, due to loss being measured over multiple classes. AdamOptimizer was used to minimize the cost function.

Hyper-parameters

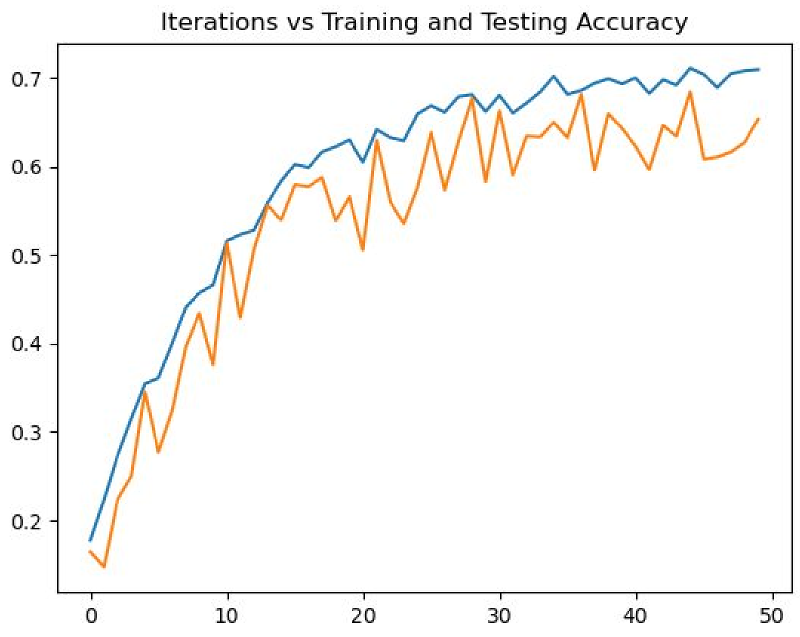
Hyper-parameters are the number of epochs, batch size, and learning rate. The epochs were set to 10 and the batch size was set to 500 due to the use of a GPU to train the CNN, so a larger batch size could be used and still train the model in a reasonable amount of time. The learning rate was chosen by trial and error to maximize the accuracy of the model, and was decreased by a small amount after each iteration. The batch size was set to 500 after trial and error.

Training and testing error over epochs

The iterations vs the loss is shown below.



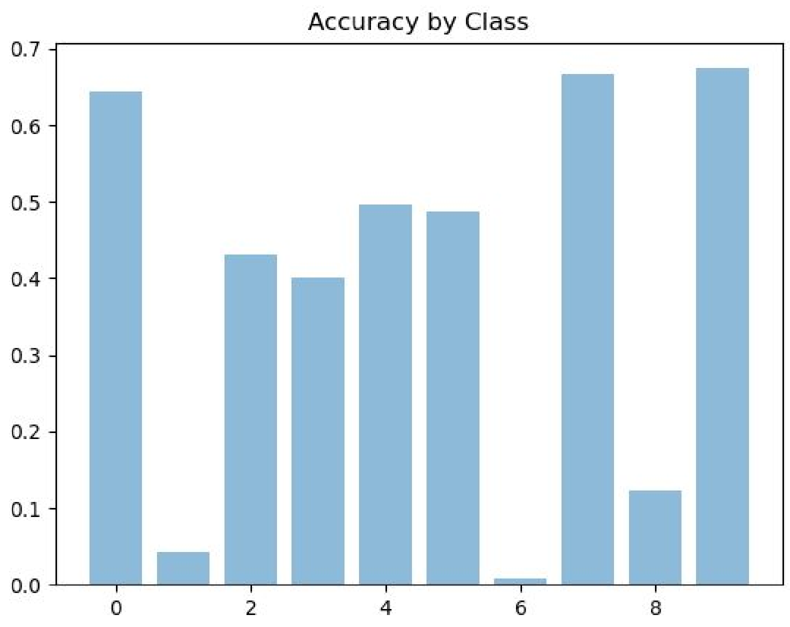
The iterations vs accuracy for the training and testing data sets are shown below.



The classification accuracy approaches about 70% as the iterations increase.

Classification performance by class

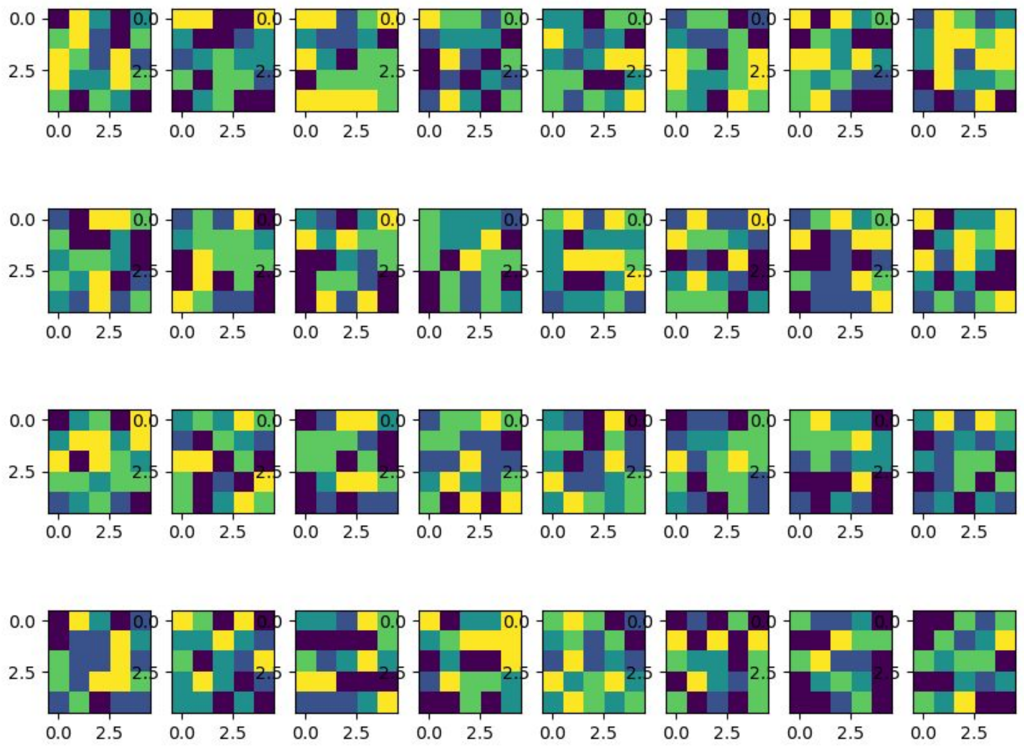
Classification performance for each class is shown below.



The model appeared to have poor performance classifying categories 1 and 6 (automobile and frog), and good performance on categories 0, 7 and 9 (airplane, horse and truck).

Filter weights

The filter for the first convolutional layer is shown below. It consists of a 5x5x3x32 tensor representing 32 5x5 filters.



Conclusion

The model was able to achieve about 70% accuracy, and seemed to successfully classify images.