Montana Pawek

Assignment 2

Model Architecture and Training Setup

To start, when describing the architecture of a CNN, it makes sense to discuss the layers. I started with three convolutional layers, as I'm aware that these layers are the truly important ones when trying to get strong and consistent image classification results. After running into some trouble with the kernel on the initial layer, I switched the kernel size to 5x5 with a padding of two and stride of one which seemed to solve the issue. The second and third convolutional layers seemed to function fine with a 3x3 kernel and padding/stride of one. I also wanted to make sure I got good features from each layer, so I set the first layer to have 16 filters, the second to have 32, and the third to have 64. In the model, these individual convolutional layers were separated by a ReLU activation function and then a pooling layer, resulting in 3 separate pooling layers. The pooling layer used MaxPool2d, and a kernel size of 2x2 and a stride of two. After this, the fully connected linear layers were utilized, and after some testing with various amounts I stuck with two; when I switched to three and then to four layers, overfitting became a major problem even with dropout layers applied in between. The first layer took 1024 dimensions, and reduced them to 256, while the second layer took that many and reduced it to 10 – one for each class.

When initially designing and training the CNN for image classification on the CIFAR-10 dataset of 32x32 RGB images, I found that most of the initial hyperparameters and settings were acceptable. The data was prepared by the function shown in the example code, and I made no changes to it. As such, I do not believe there was any data augmentation, and the data normalization was simply done by using the transforms.Normalize() function to normalize the pixel values. This data loader function also did an 80-20 split for training and validation images, which I did not change. Additionally, after some research, I found that the Cross Entropy Loss function is the standard loss function for multiclass image classification, and decided not to change it either. The same was true for the batch size when I learned I would have to continuously tweak the learning rate if I changed it to find the right balance. As for what I did change, I tested both the Adam and SGD optimizers – I used a learning rate of 0.001 for Adam and 0.01 for SGD. While I found that the Adam optimizer yielded between 65-69% accuracy during testing, the initial SGD optimizer outperformed it with an accuracy that stayed between 72-74%. After some experimentation, the optimal learning rate for SGD turned out to be around 0.005 – this learning rate produced an accuracy of 75%. I decided not to experiment with momentum for the SGD optimizer as I had already reached a respectable accuracy, and needed to move on. The number of epochs used during training was increased to 20 to see if accuracy could be improved, but there was only a small increase to accuracy and the model instead showed a degree of overfitting to the training data. Implementing early stopping would likely be a good idea if I were to return to this model in the future. The only regularization used was dropout between the linear layers, with the inclusion of another dropout layer before the first linear layer greatly helping with the overfitting problem. The dropout parameter remained at 0.25 as the results obtained during training and validation were good. Performance was evaluated by the existing code, tracking training and validation loss, as well as test image accuracy.

Training Results

Due to the easily read performance metrics, the results are clear. As can be seen below in Figure 1, while there was a certain amount of overfitting that occurred, the validation loss was still continuously decreasing until the later epochs where it plateaued. If additional methods were applied to increase generalization, such as early stopping or data augmentation, better results could likely be achieved. The training loss also shows a much smoother curve, while the validation line is more jagged and unstable. After some research, this seems to indicate that the learning itself is unstable, perhaps due to a learning rate that's still too high or just the same overfitting mentioned before. The results depicted in Figure 2 show the final test loss was calculated to be ~0.72 - less than the final validation loss - and the accuracy was measured at 75%. This is a fairly good performance, especially with the worrying amount of indicators pointing to overfitting, and can likely be improved even more with more hyperparameter tuning. Figure 2 also shows which classes gave the model the most trouble – pictures of dogs were far and away the least accurate at only 61%, followed by cats (65%), birds (66%), and deer (67%). My speculation is that the identifiers that indicate a dog can also indicate a cat as the animals have a similar profile. Horses and deer might also suffer from the same problem, however birds do not seem to have a similar looking image aside from airplanes which had a much higher accuracy rate.

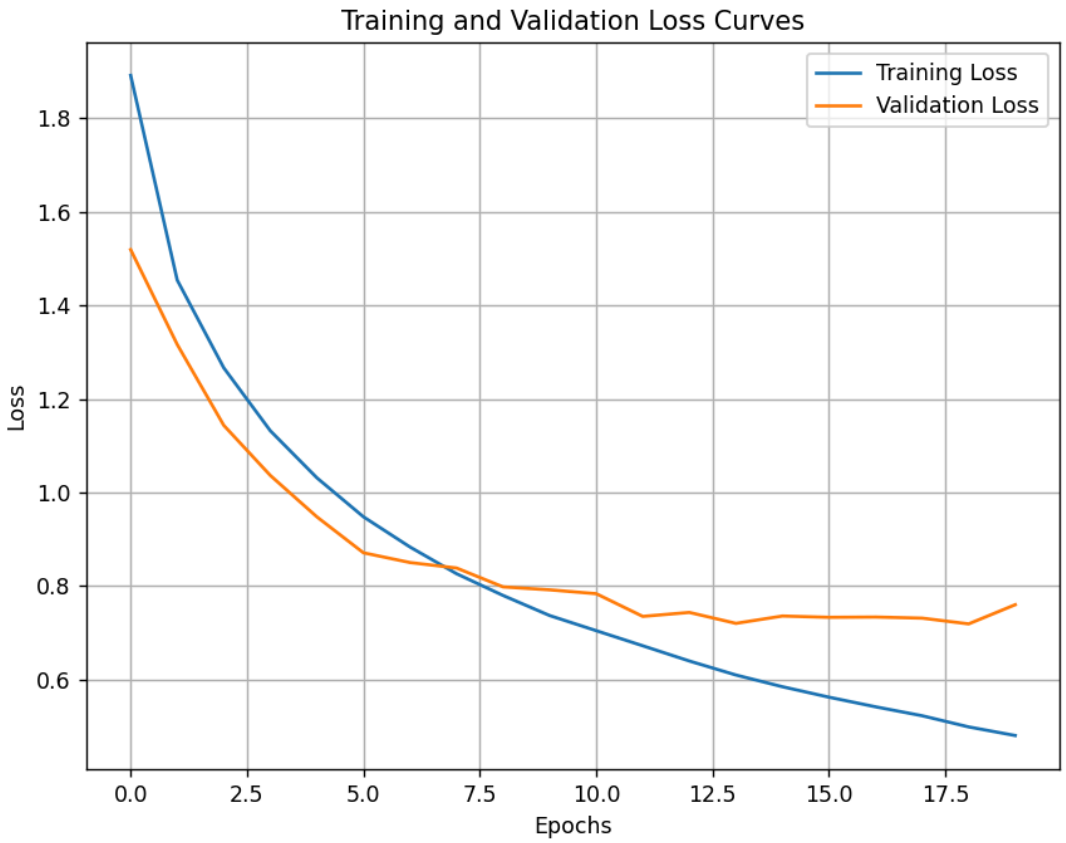

Figure 1: Training and Validation Loss Curves for final model architecture

```
Test Loss: 0.716279

Test Accuracy of airplane: 79% (793/1000)
Test Accuracy of automobile: 82% (828/1000)
Test Accuracy of  bird: 66% (662/1000)
Test Accuracy of   cat: 65% (656/1000)
Test Accuracy of  deer: 67% (671/1000)
Test Accuracy of   dog: 61% (613/1000)
Test Accuracy of  frog: 84% (840/1000)
Test Accuracy of horse: 75% (752/1000)
Test Accuracy of  ship: 87% (876/1000)
Test Accuracy of truck: 85% (859/1000)

Test Accuracy (Overall): 75% (7550/10000)
```

Figure 2: Test Loss and Accuracy Measurements

Feature Map Visualization

The three figures below show the first eight feature maps from the first convolutional layer applied to each image. It appears that several filters are common, as there are several channels in each where horizontal and vertical edges are emphasized. Additionally, some of the filters appear to detect brightness/color contrast, which is helpful in separating the image from the background. It's quite interesting however that these channels don't seem to necessarily line up the same with each image. For example, in Figure 3, the channel 2 image appears to clearly be focused on horizontal edges, but in Figure 4 this same channel doesn't appear to emphasize horizontal lines in the same way, even though they are present. Finally, in Figure 5, due to the bird's orientation most of the edges are diagonal, which means it heavily pixelates these edges to emphasize the horizontal nature of them. Channel 5 and 6 appear to be the ones that differentiate in color the most for Figure 3, but show very confusing images for Figure 4 and Figure 5. If I had to guess, based on Figure 5, these channels are more geared towards "dark" and "light" with very little in between. Finally, channels 4 and 7 almost appear to be negative outputs of the given image, which is especially apparent in Figure 5, but is also visible in Figure 4 due to the dark hull of the ship turning white, and Figure 3 where the darkest part of the sky turns white. There are even several channels which appear dedicated to mapping areas with lots of visual noise or details, such as channels 0 and 1. I'm surprised that each channel doesn't have an easy answer to what features it's finding, and that each image responds so differently.

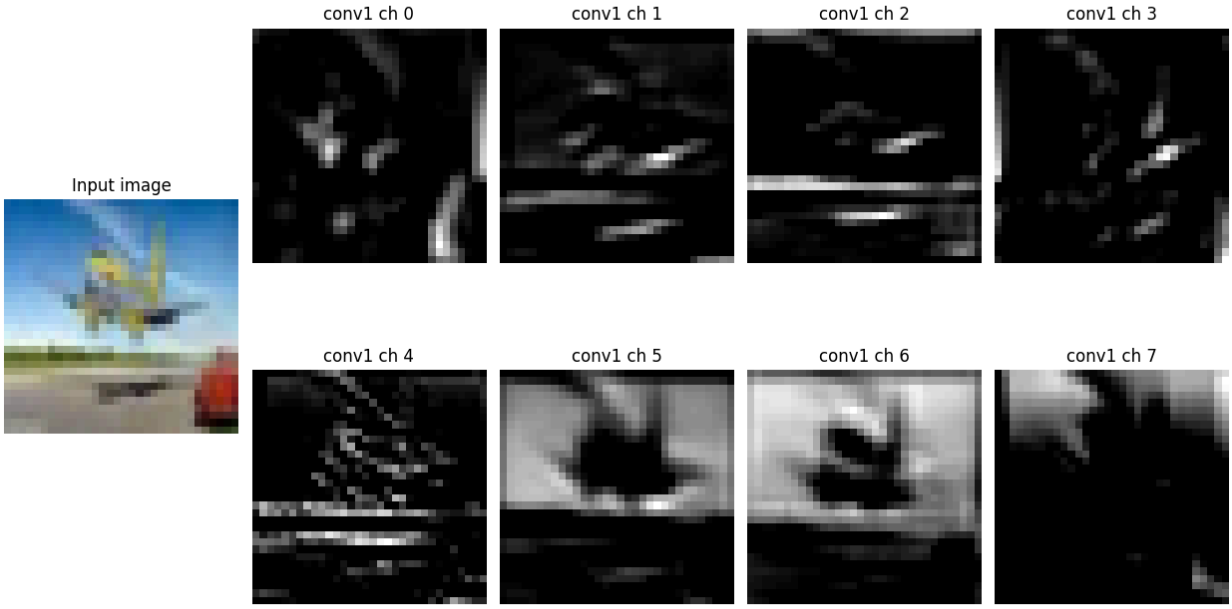

Figure 3: Airplane Image Feature Map

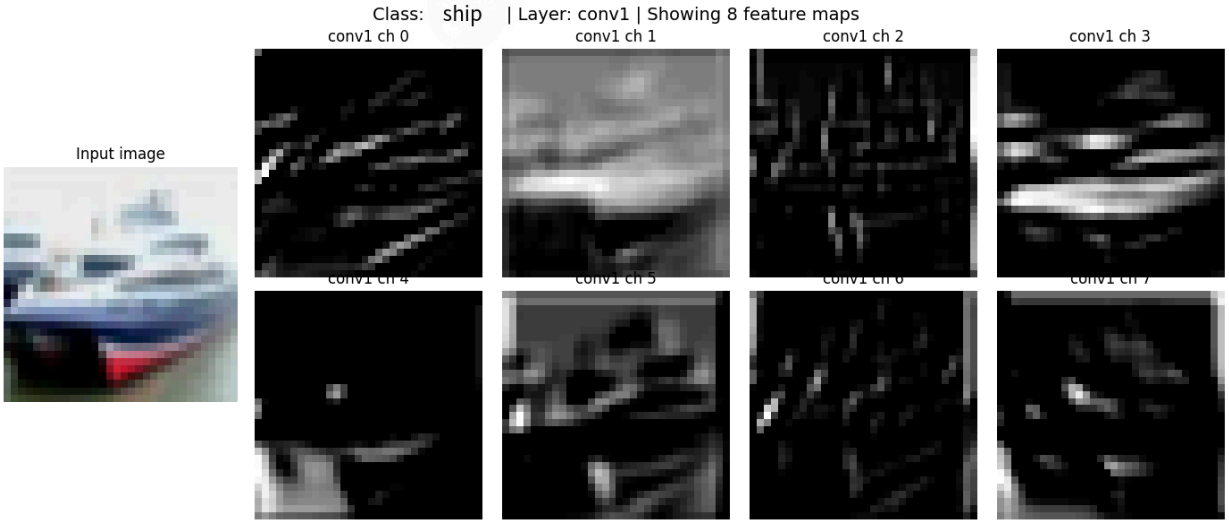

Figure 4: Ship Image Feature Map

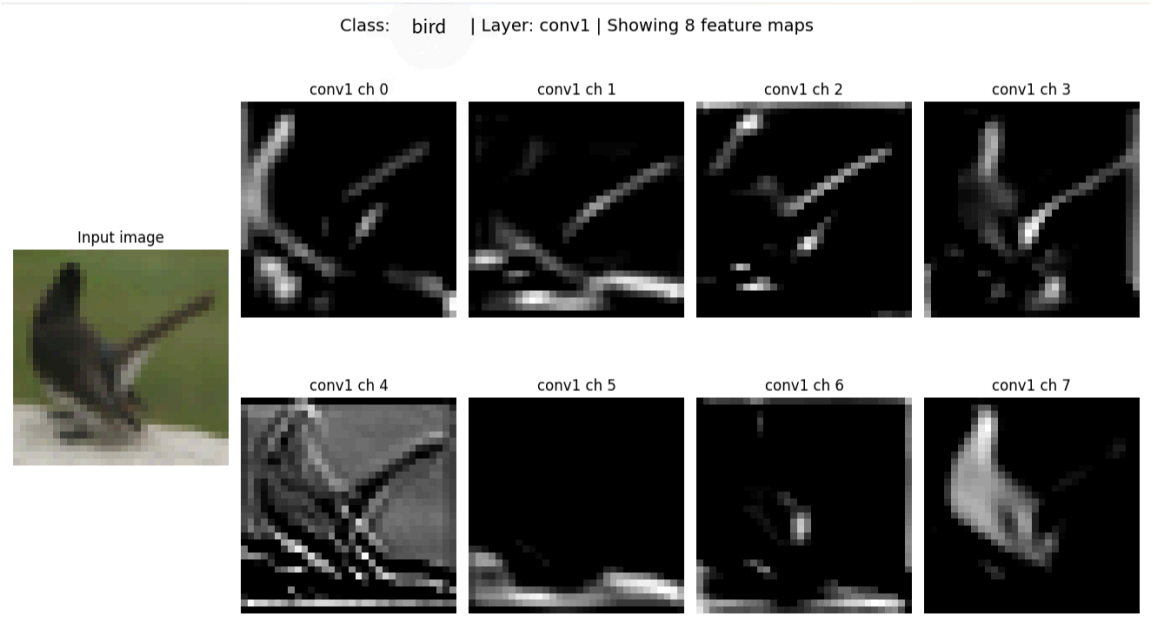

Figure 5: Bird Image Feature Map

Maximally Activating Images

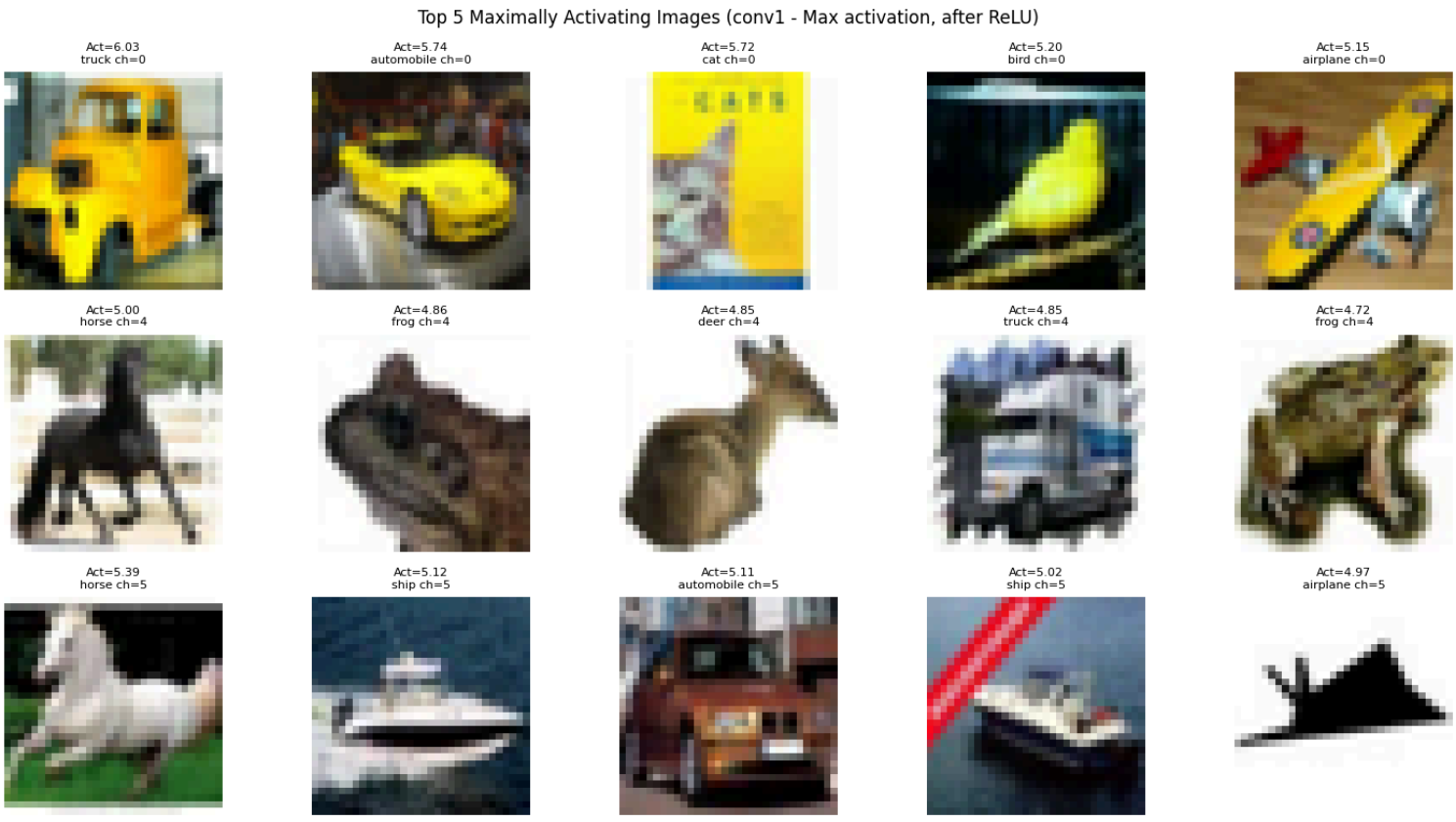

Figure 6: Maximally Activating Images for Channels 0, 4, and 5

The first obvious and interesting pattern in Figure 6 is that all the images chosen for channel 0 show yellow, indicating that perhaps that channel is focused on colors, or at least colors contrasting with their environment. The yellows are vibrant as well, and take up a significant portion of the image. Every image is different as well, which means this filter doesn't seem to respond to specific class patterns. Channel 4 is different as there are two images of frogs present, but this does not necessarily indicate a pattern. Before, I said I believed that channel 4 did an inversion of light and dark, and this makes some sense based on the images chosen. Three of the images have no background, and the two that do stand out starkly – a dark horse in front of a white fence, and a white truck in front of a dark background. Perhaps channel 4's pattern involves finding clear edges between the object of focus and the background, or inverting brightness to detect patterns that aren't clear with normal brightness. Lastly, channel 5 also does not appear to play class favorites; there are two ships, but everything else is a random class. Channel 5 might be similar to channel 4 in that most of the images portray an object clearly differentiated from its background, except for the automobile. It's not entirely clear what ties these images together, but if I had to guess based on these results and Figures 3, 4, and 5, I'd say this channel is also focused on brightness contrast in the image, but more binary – anything light is corrected to gray/white, and anything dark is corrected to black. In other words, images that have very light and very dark portions activate ReLU the most. All in all, these channels appear to focus on general features rather than class-specific patterns.

Brief Discussion and Reflection

With a final test accuracy of 75%, I'm satisfied with this version of the CNN as a rough draft. Looking at the loss curves in Figure 1, convergence happens early, but then the training loss goes down while validation stays the same indicating overfitting. The most effective way for me to improve the model would likely be to improve generalization by adjusting the dropout rate, and implementing early stopping, but perhaps augmenting the data by flipping, rotating, shrinking, or translating the images would work as well. It's likely that more convolutional layers would improve the accuracy somewhat, but it's equally likely that adding too many would confuse the results. From my 20-epoch tests, it appears that around 7 epochs is where the model converged, but validation loss plateaued around 13. The convolutional layer filters were interesting as well – I expected the first ones to be much easier to determine the focus of. At the very least, I imagined I'd be able to see which ones favored horizontal patterns compared to vertical. It appears that the early layers are much more focused on color gradients than I expected, and the edges found aren't just one cardinal direction. It was also extremely interesting how even the spread of classes was for the top 5 activating images: 2 airplanes, 2 trucks, 2 automobiles, 2 horses, and 2 ships. Only dogs were entirely missing from these 5 filters, which is interesting as the dog class was regularly the least accurate during testing.