# Deep Learning Homework 2: Convolutional Neural Network

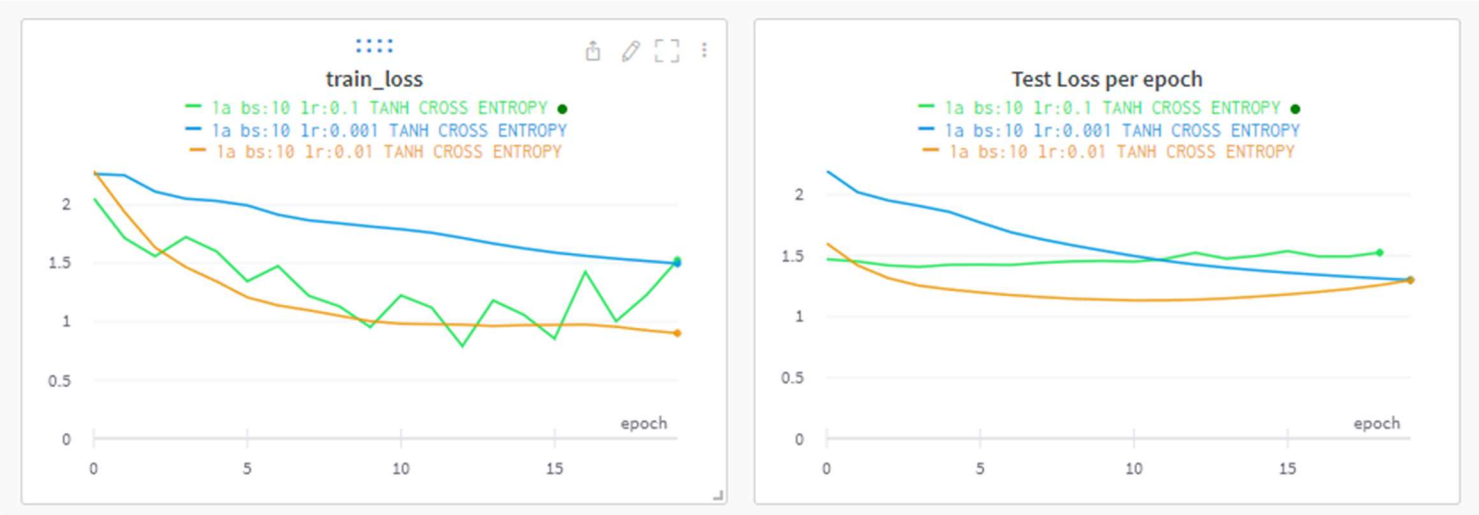
The goal of this assignment is to gain some experience with CNNs. You will use the CIFAR-10 dataset for all the experiments.

Construct a simple CNN that follows the architecture of LeNet. However, the images have 3 channels. The first kernel will be 6 5x5x3 kernels that takes us to 28x28x6. The remainder of the network is the same. Stride is 1 for convolution and 2 for pooling layers. You need to train the network for different choices of parameters: learning rate 0.1, 0.01, 0.001; activation function = sigmoid, Tanh; loss function = MSE, cross-entropy (3x2x2 = 12 combinations). Train for a sufficient number of epochs to see a low train loss.

# 1a.

*Submit plots of training error vs epoch and test error vs epoch for each combination of parameters. Describe the reasons for what you observe*

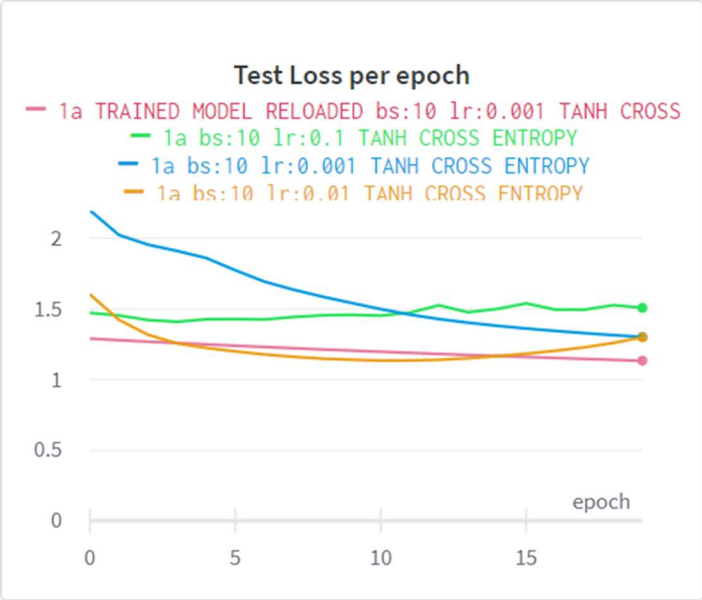
For convenience of comparing runs, I've grouped these into groups of three runs. Each group contains a combination of an activation function and loss function, with all three learning rates in each combination in each group.

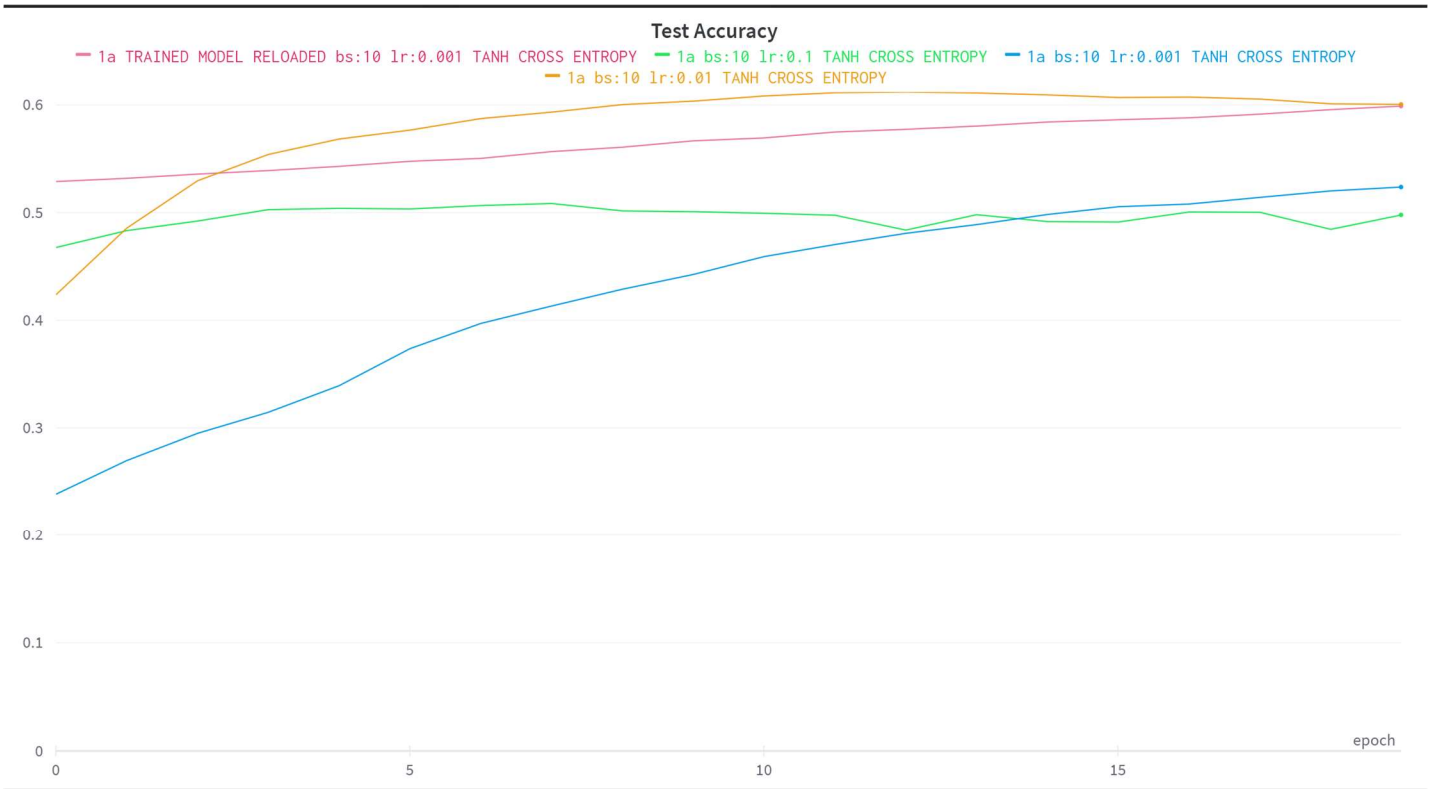



For the above, it looks like the orange run (with learning rate of 0.01) is overfitting since the test loss starts to increase around epoch 10 but the training loss still decreases. This can also be seen in the test accuracy (number of correct predictions / total number in test set) which seems to decrease after epoch 10.

The green run, with a learning rate of 0.1, looks like it keeps overshooting the global minima due to an exploding gradient. The learning rate is too high.

Upon looking at these results I decided to train the blue line for 20 more epochs for a total of 40 epochs. I had saved the model parameters and simply trained it with the same hyperparameters including the learning rate of 0.001 which resulted in the pink line in the following chart, so the pink link carries over where the blue line left off:

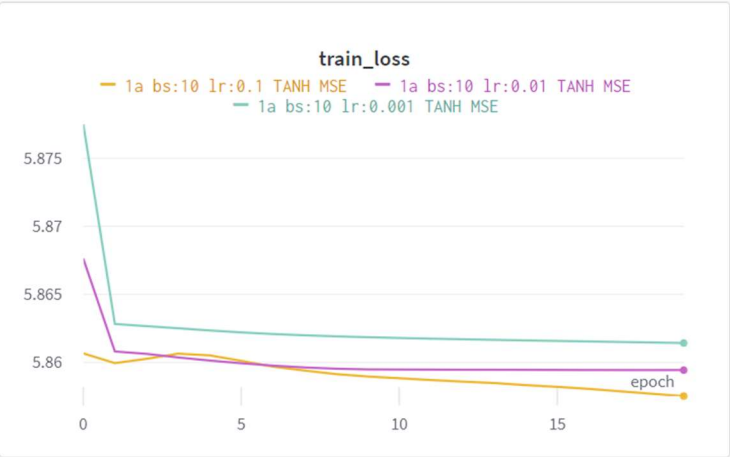




I also computed the Test Accuracy at every epoch. Around epoch 10 the orange run starts to overfit. At that point it has the highest accuracy, but the pink run (which as stated is actually at 40 epochs) overtakes the orange run for best accuracy since the pink run is slower to learn but is not overfitting.

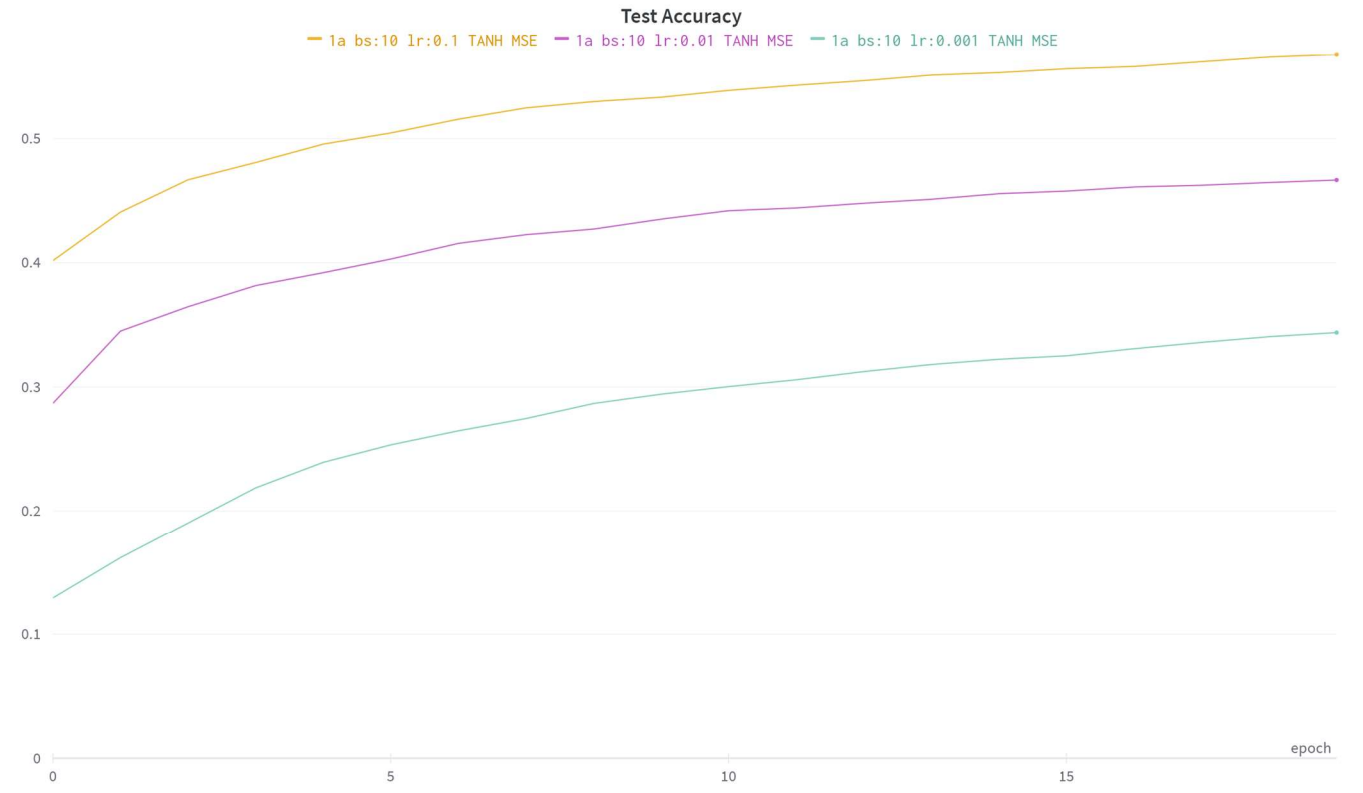
To calculate Mean Squared Error loss I converted the labels which were integer values to one-hot vectors.

The following is each of three learning rates using the Tanh activation and MSE loss function:





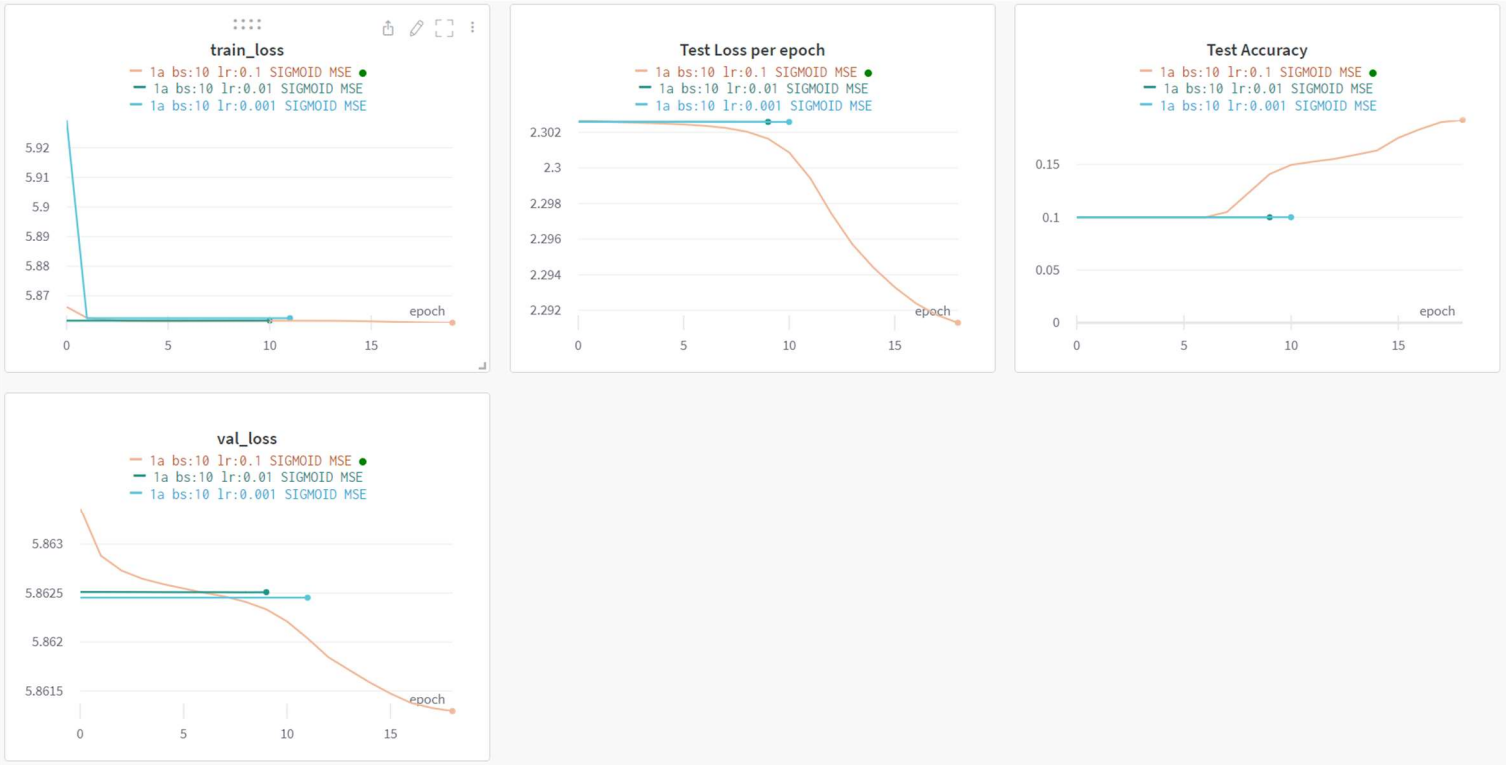
And here is their testing accuracy:




These runs appear very similar to each other. They all train slowly but consistently without appearing to overfit or succumb to exploding or vanishing gradients. However, their overall performance is inferior to Cross Entropy Loss when the correct hyperparameters in CEL are used.

The remaining runs in this section utilize the sigmoid activation. None of them developed appreciable performance presumably due to vanishing gradients that tend to plague sigmoid activations.

The following is each of three learning rates with sigmoid activations and MSE loss functions. Note that the y axis scales to the difference between the min and max values for each run, so although some might appear to train, the actual progress is insignificant compared to tanh activations.

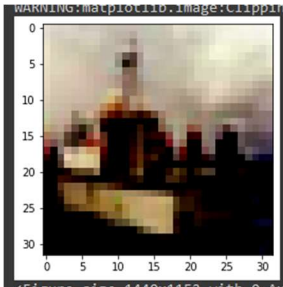



# 1.b

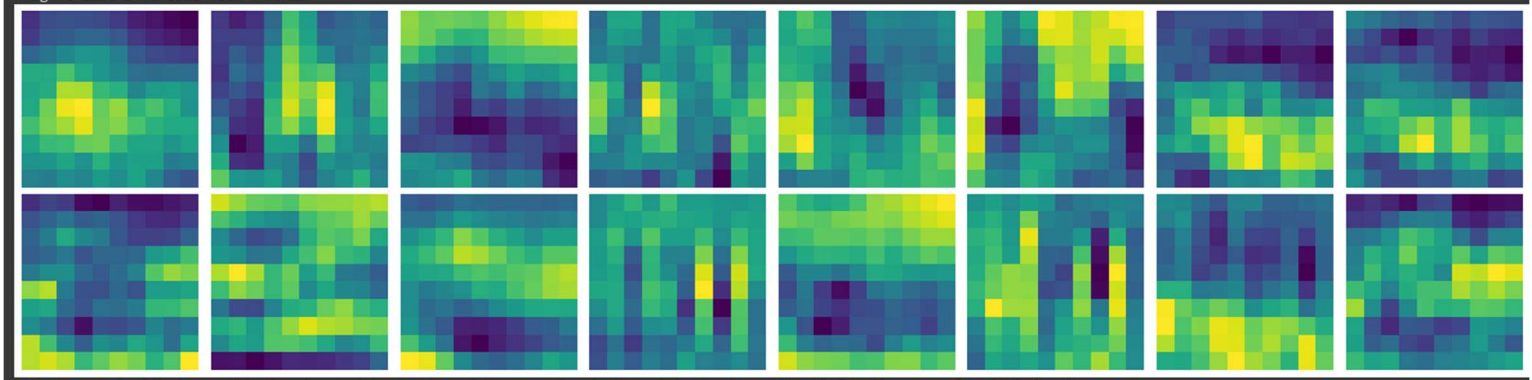
*For any one trained network and parameter choice, display feature maps of 10 images at the last convolution layer. What has your network learned? Here is one helpful tutorial: https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-networks/*

These are visualizations of the feature maps from the second convolution layer for the model that was trained with learning rate of 0.001, Tanh activations, Cross Entropy Loss, and a batch size of 10.

Note that under the feature map visualizations there I have also printed the model's prediction on the image. For some of the images the network outperformed me due to the low resolution of the images.

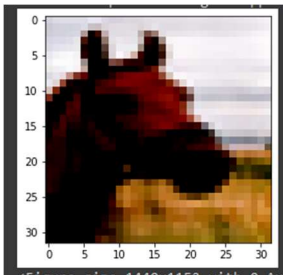


<Figure size 1440x1152 with 0 Axes>

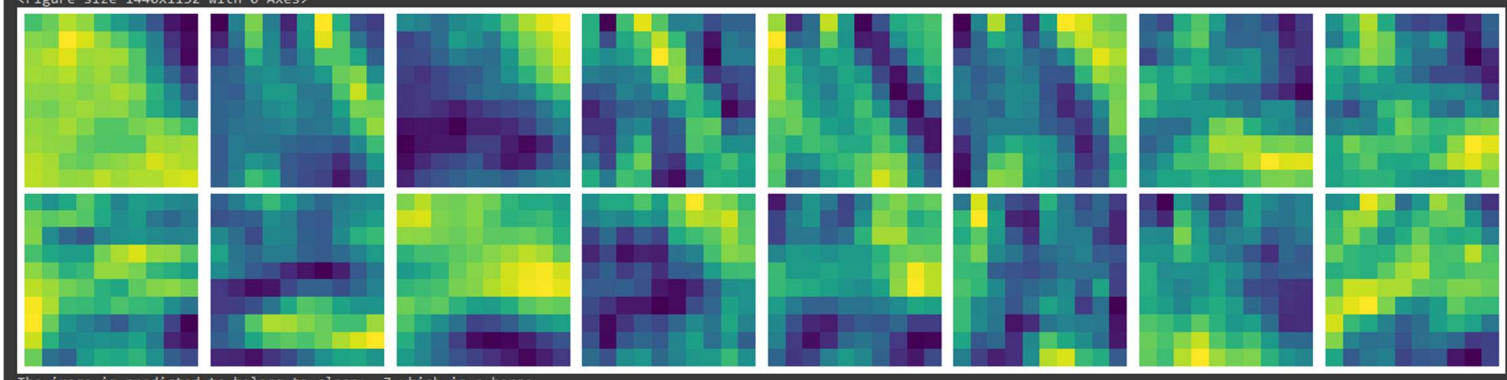


WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

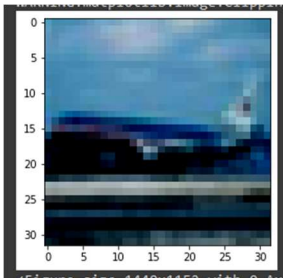
The image is predicted to belong to class:  9 which is a truck



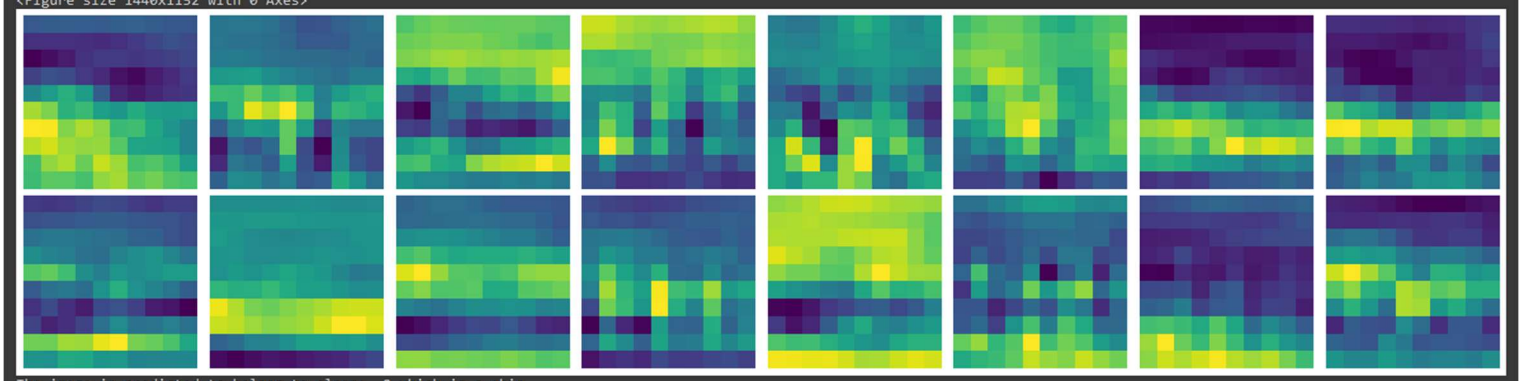
<Figure size 1440x1152 with 0 Axes>



The image is predicted to belong to class:  7 which is a horse

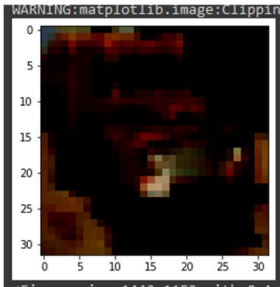


<Figure size 1440x1152 with 0 Axes>

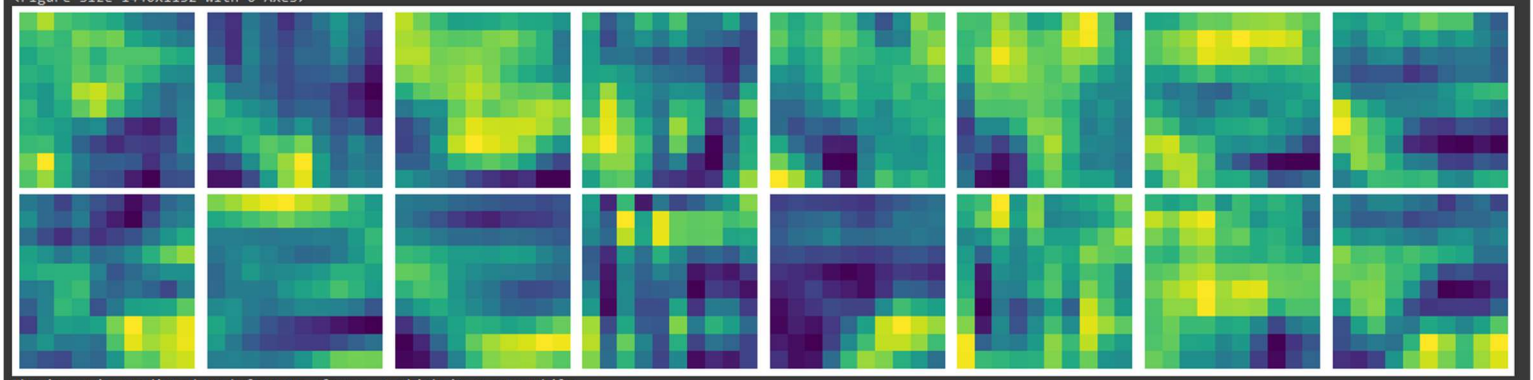


The image is predicted to belong to class:  8 which is a ship

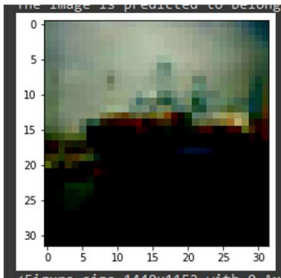
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



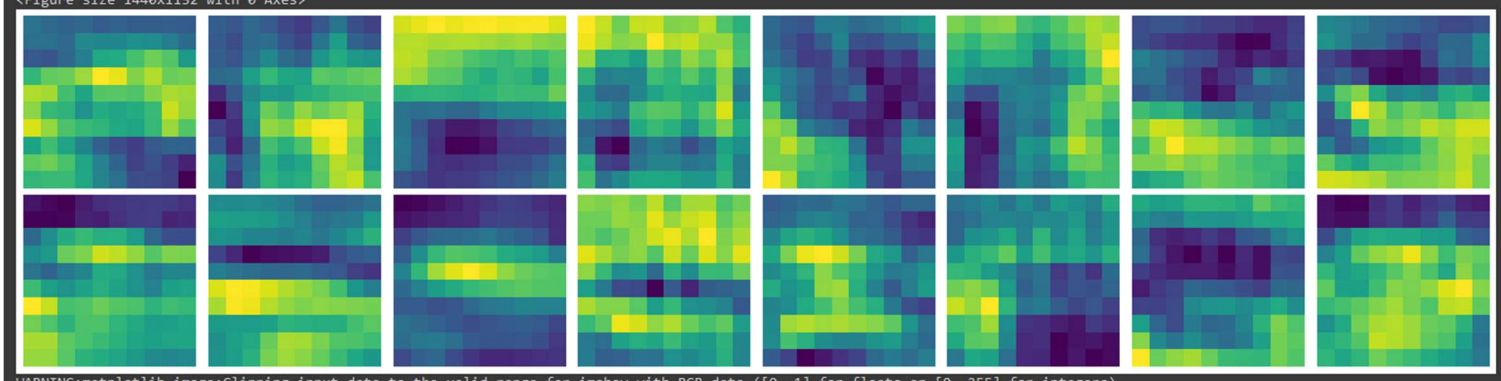
<Figure size 1440x1152 with 0 Axes>



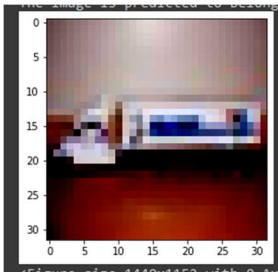
The image is predicted to belong to class:  1 which is a automobile



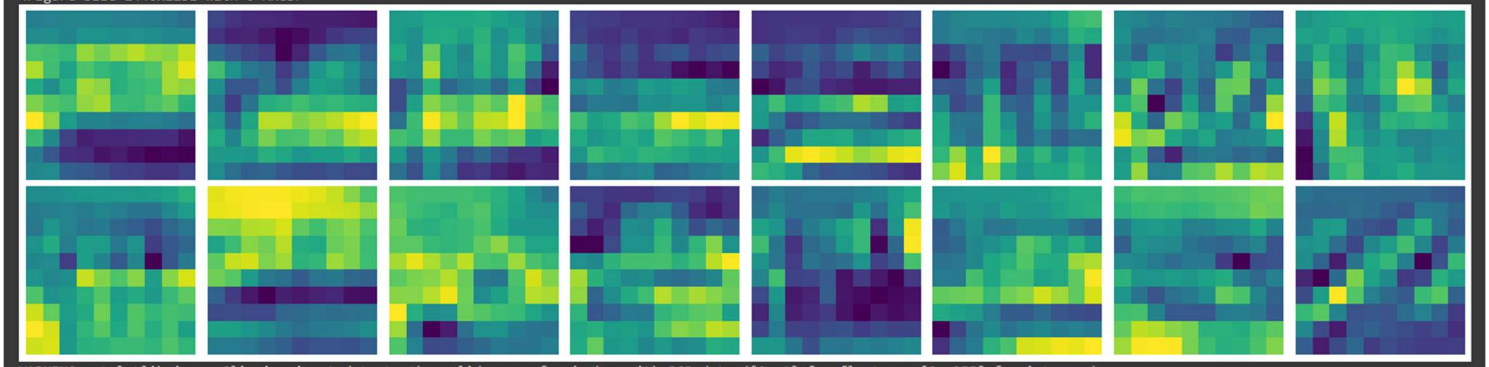
```
<Figure size 1440x1152 with 0 Axes>
```



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
The image is predicted to belong to class:  8 which is a ship
```

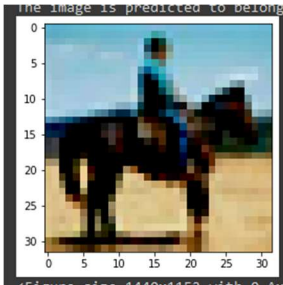


```
<Figure size 1440x1152 with 0 Axes>
```

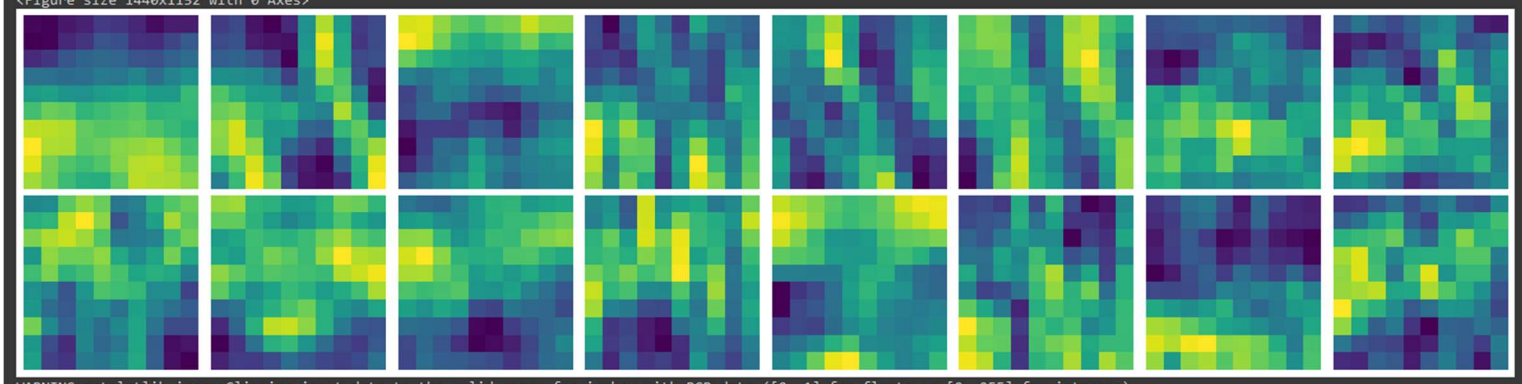


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
The image is predicted to belong to class:  8 which is a ship
```

The image is predicted to belong to class:  3 which is a cat



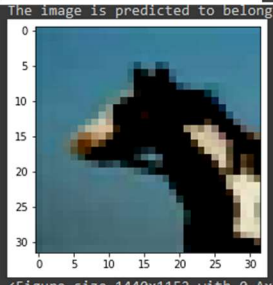
<Figure size 1440x1152 with 0 Axes>



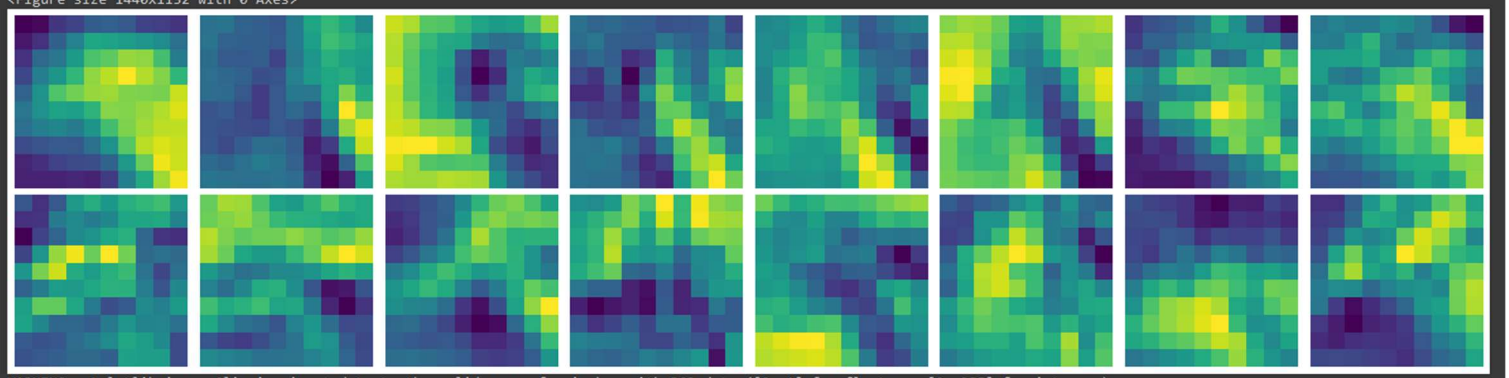
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

The image is predicted to belong to class:  7 which is a horse

The image is predicted to belong to class:  8 which is a ship

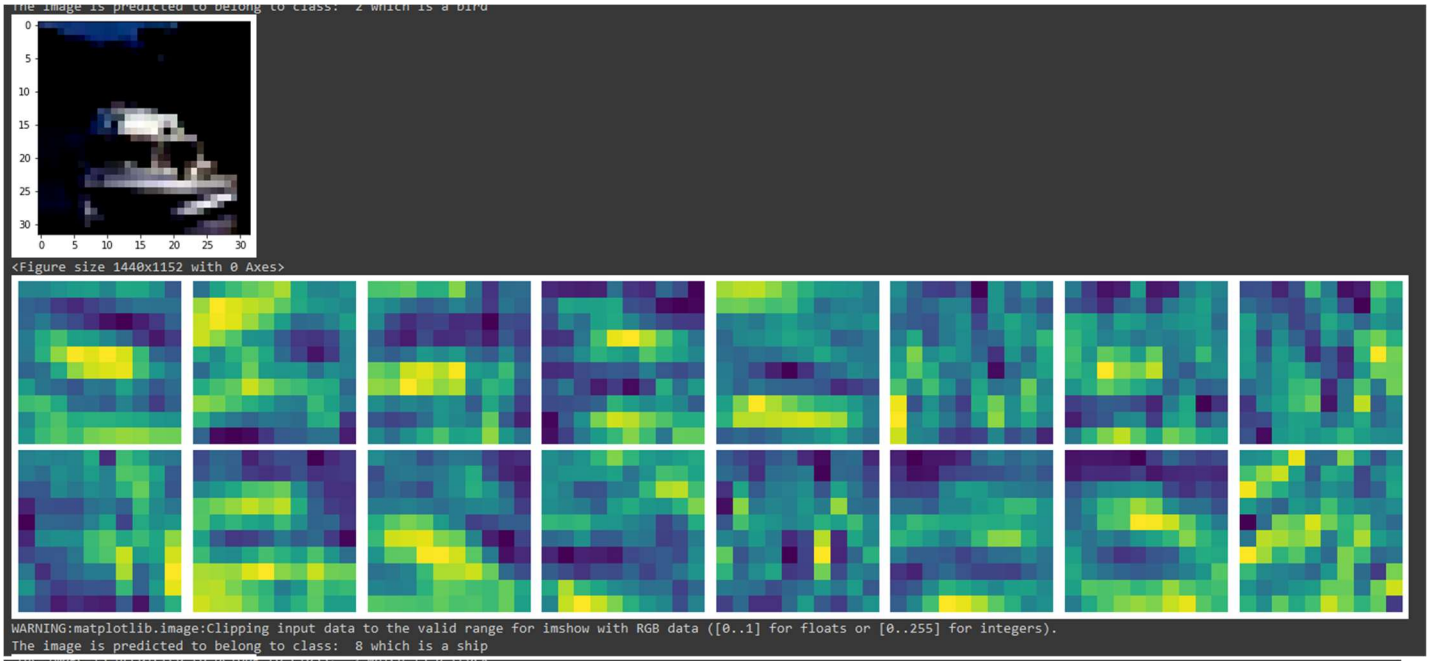


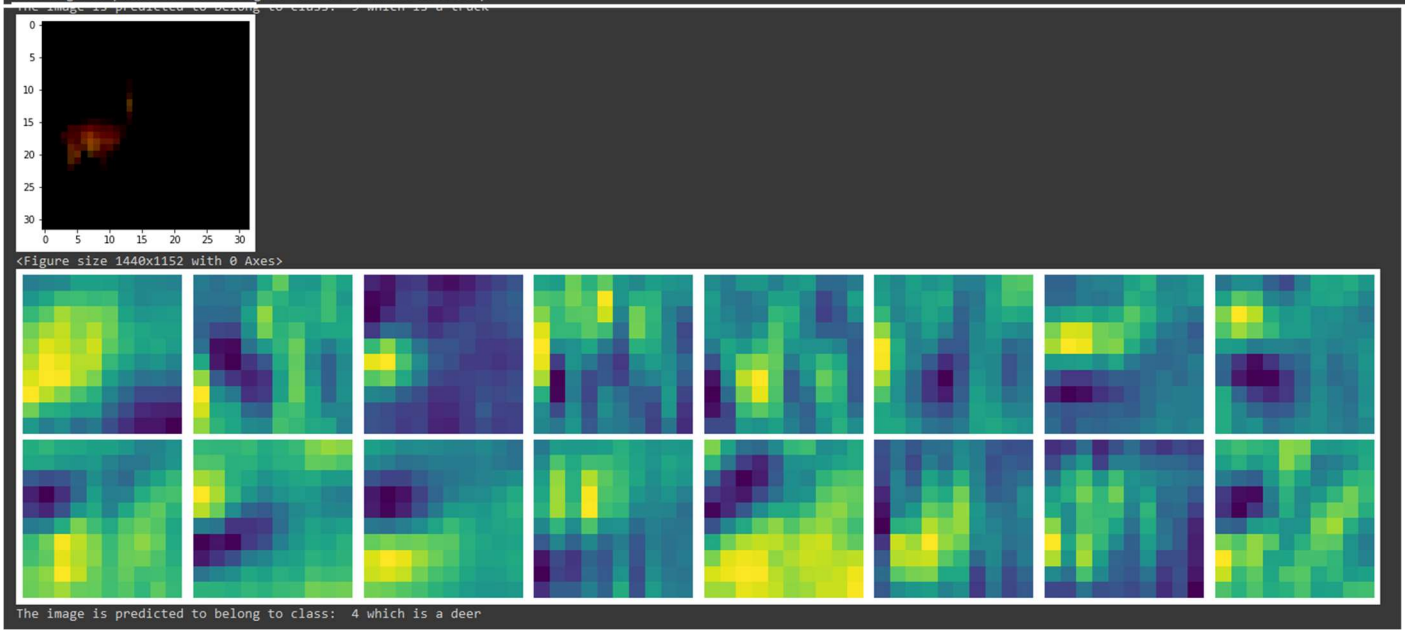
<Figure size 1440x1152 with 0 Axes>



WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

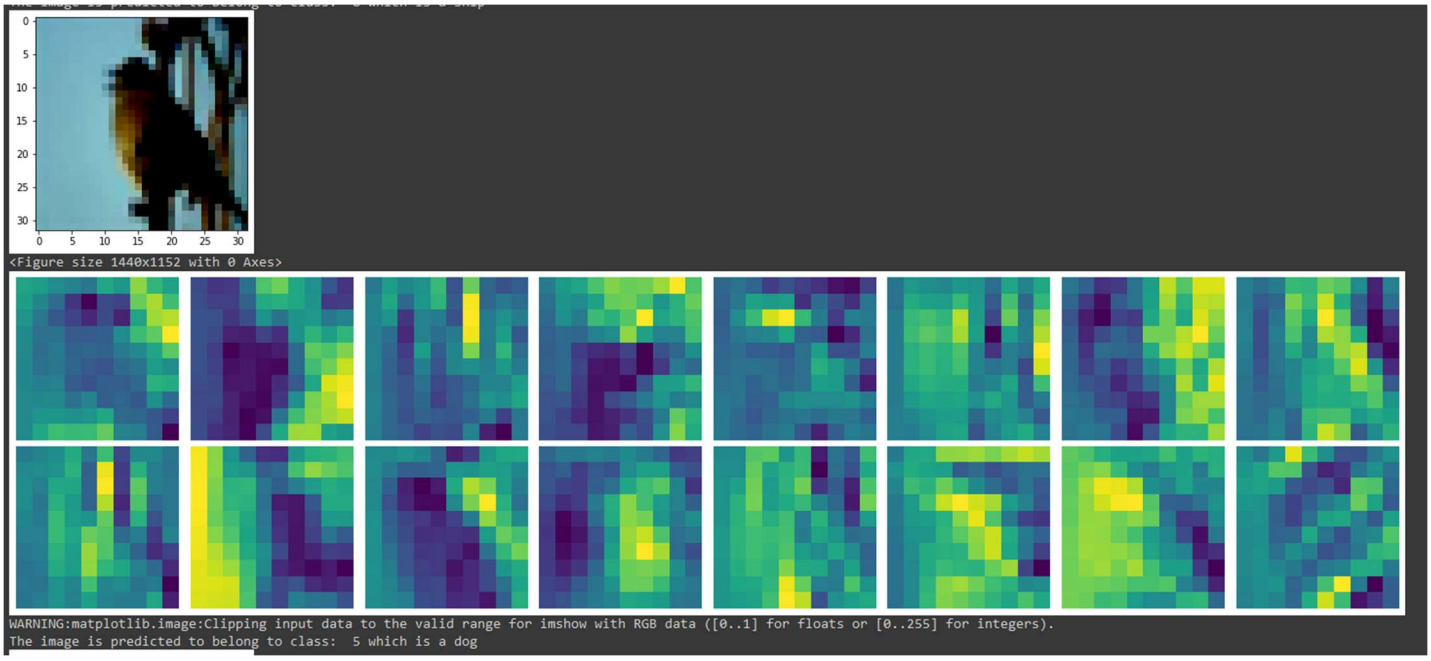
The image is predicted to belong to class:  0 which is a airplane

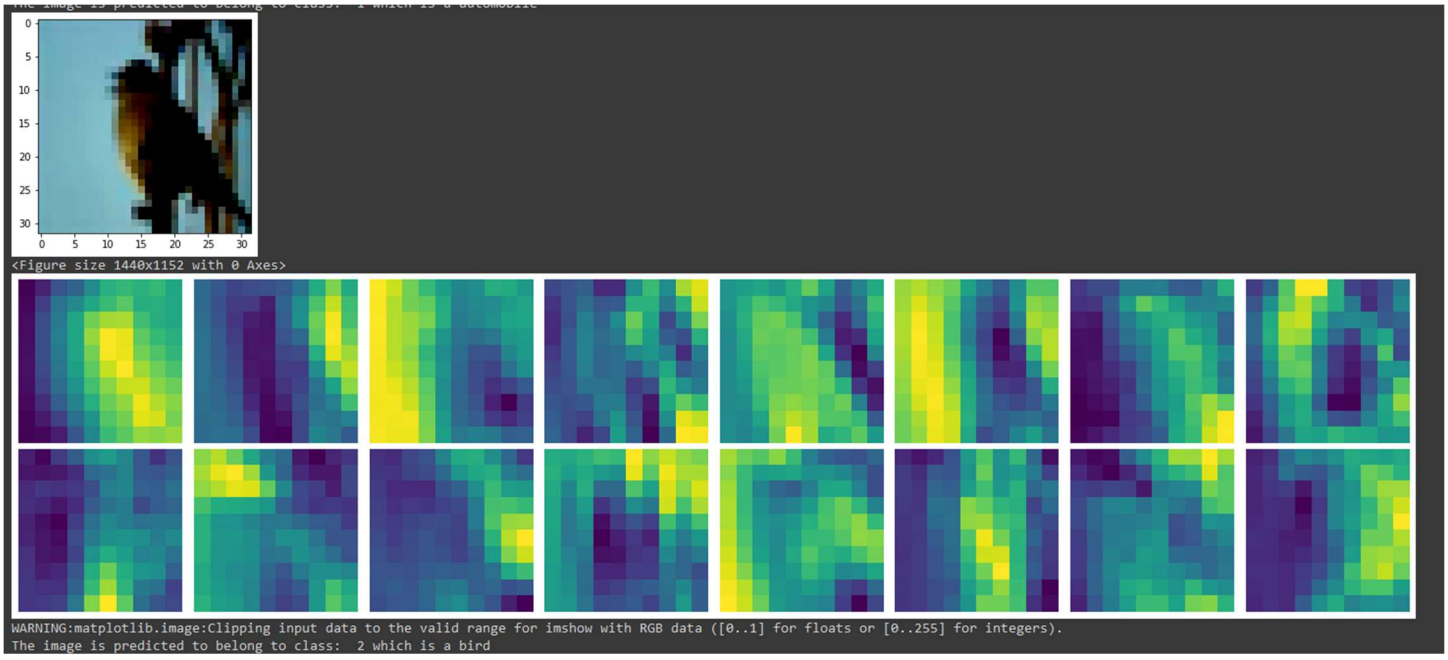

The image is predicted to belong to class:  2 which is a bird
<Figure size 1440x1152 with 0 Axes>
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
The image is predicted to belong to class:  8 which is a ship


<Figure size 1440x1152 with 0 Axes>
The image is predicted to belong to class:  4 which is a deer

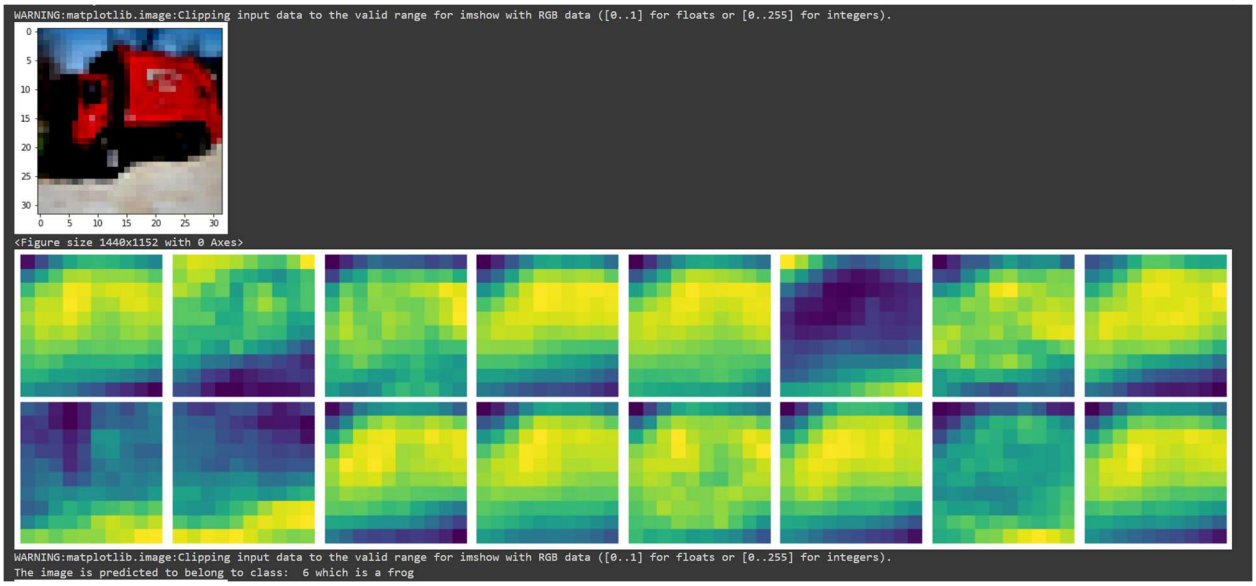
d

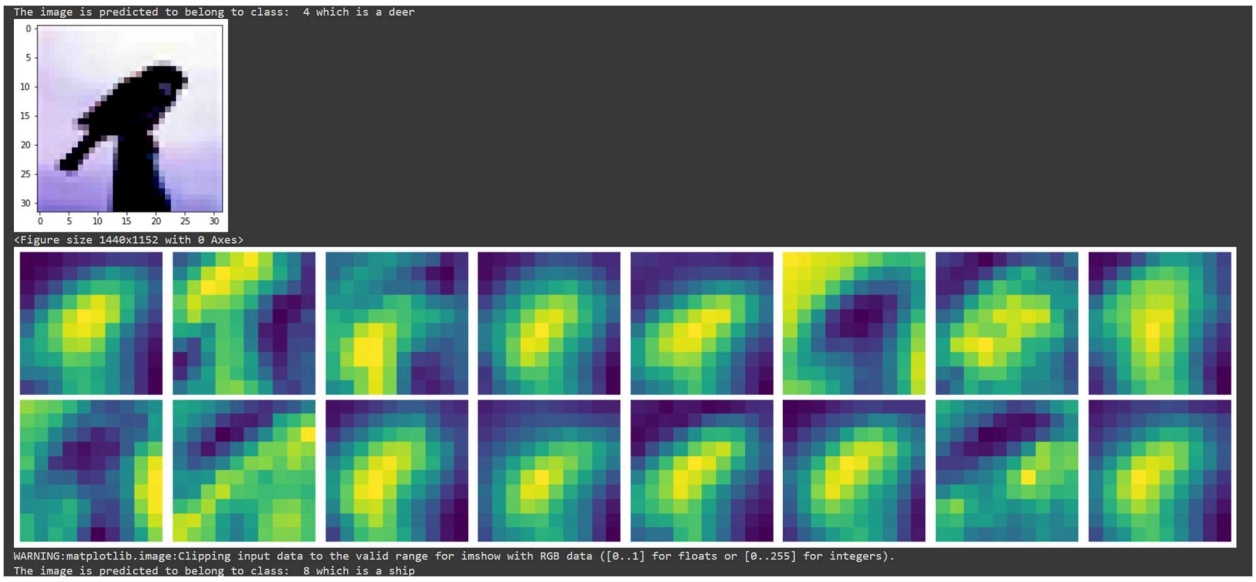
**Interestingly, the features maps for a model that performed poorly, such as those that use sigmoid activations, have features maps that are less defined and appear to have less variation in general. Here's a comparison of the same image with a well-performing network and a poorly performing sigmoid-based model:**

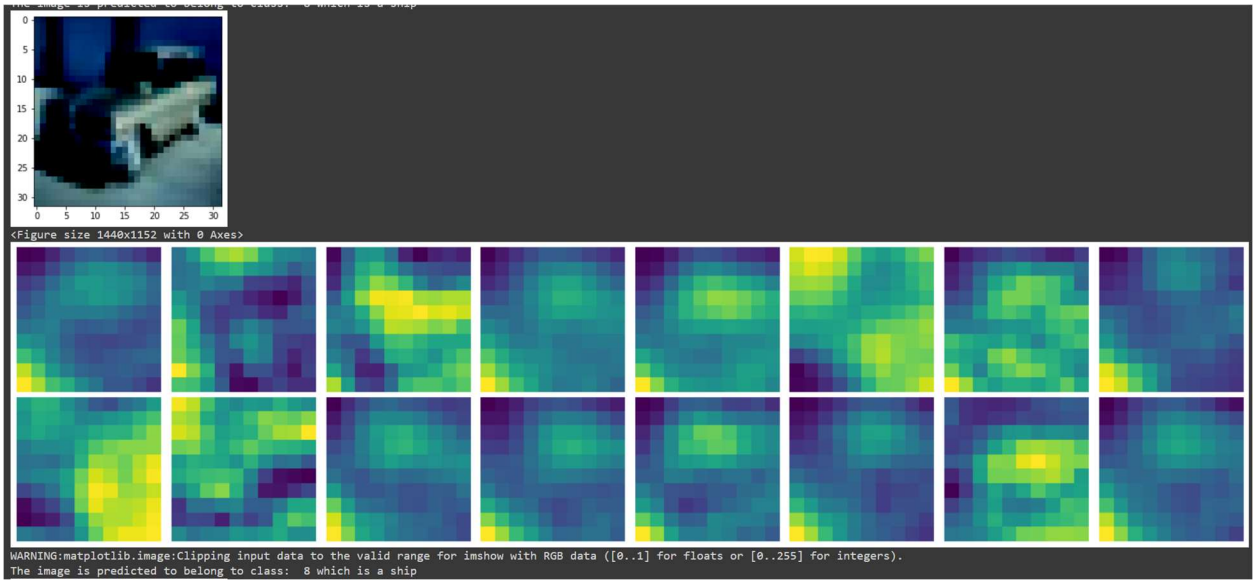


**Interestingly, the features maps for a model that performed poorly, such as those that use sigmoid activations, have features maps that are less defined and appear to have less variation in general, such as the following:**



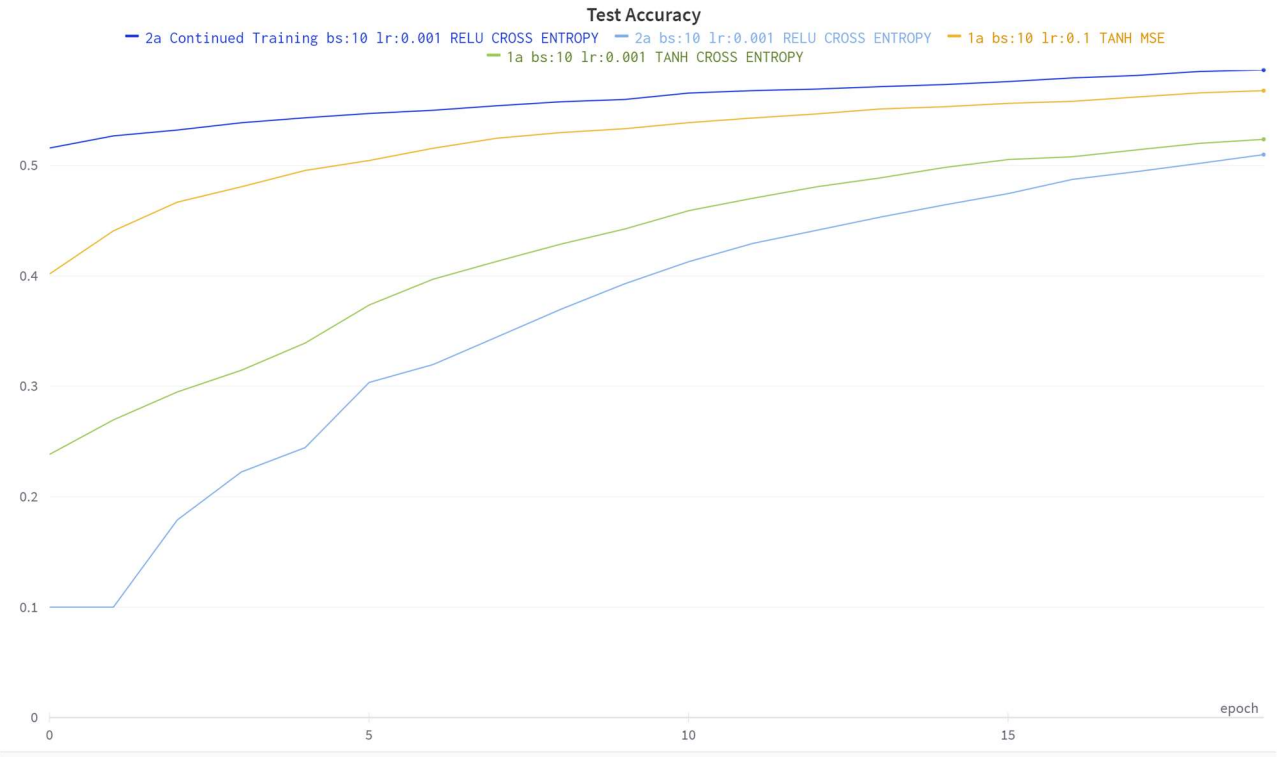
# 2a.

Select Relu, learning rate of 0.001, cross-entropy loss. Change the network to use 3x3 kernels for both convolutions rather than 5x5. Same pooling. Train for a sufficient number of epochs to see a low enough train loss.

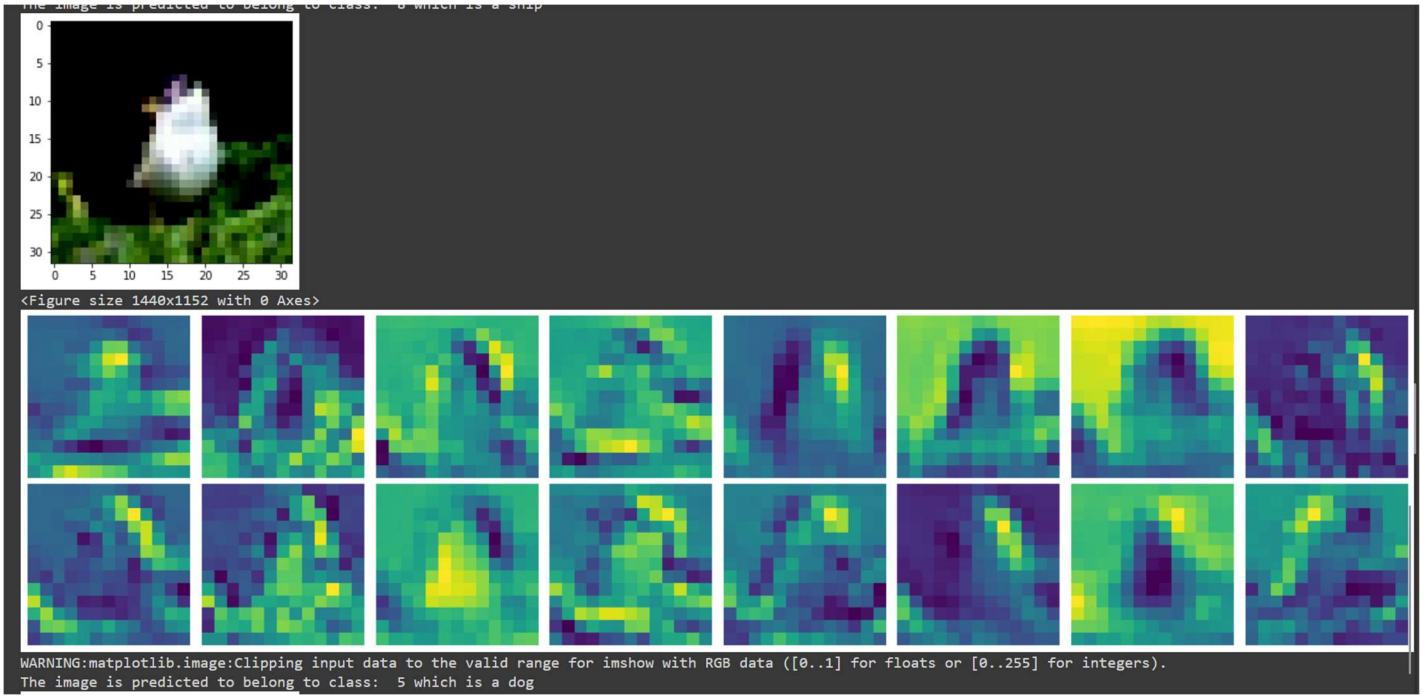
a. *Plot the error (test, train) vs epoch and discuss how your results differ from the previous network*

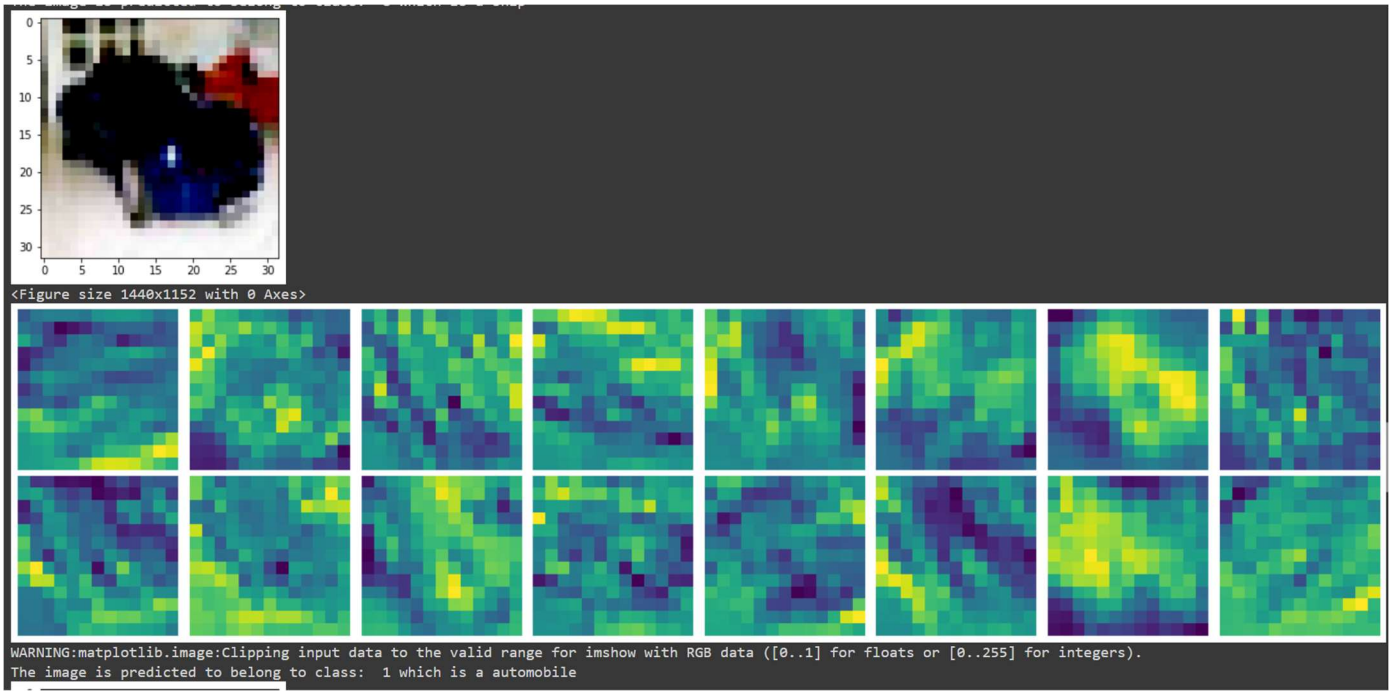

My initial setup was to train each model for 20 epochs. For some I reloaded the model to train for an addition 20 to see what happened. I never fixed the logging in my code, so in this case the light blue line is the 3x3 ReLU model for the first 20 epochs and the dark blue is for the second 20 epochs. Apologies for the confusion.

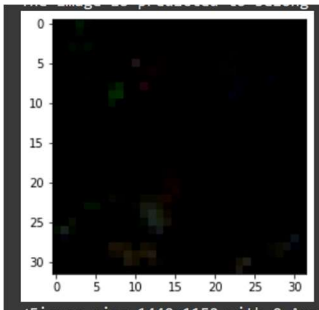
It looks like this network took a little longer to train than prior successful networks, but it looks more resilient to overfitting. The test training loss reaches an mostly flat line before the test accuracy does.



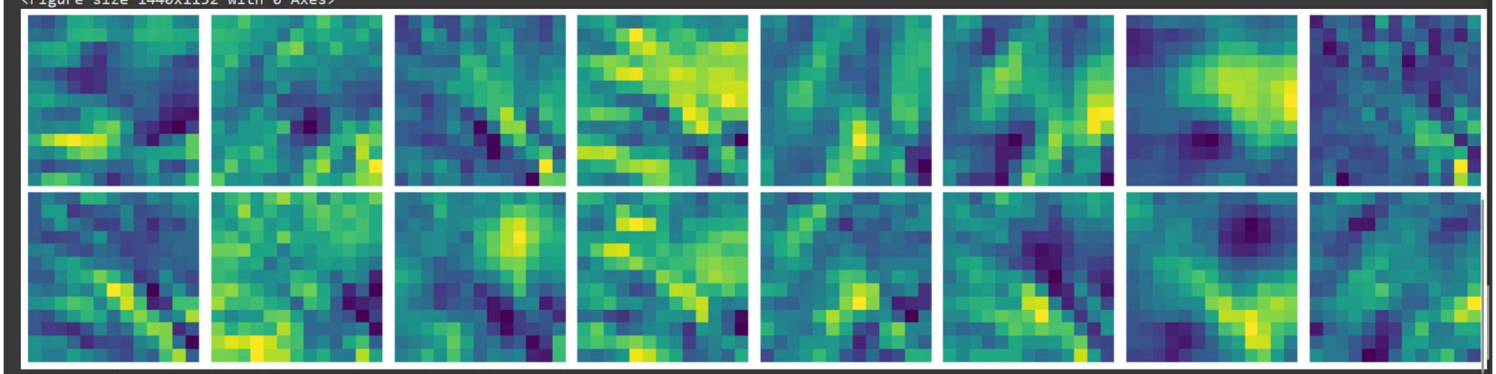
**As a bonus, the feature maps in the 3x3 kernal ReLU model look more detailed than the ones with the tanh activations at the second convolution:**



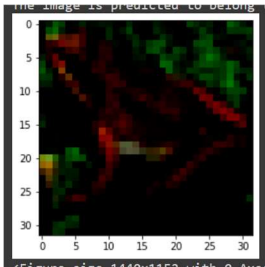





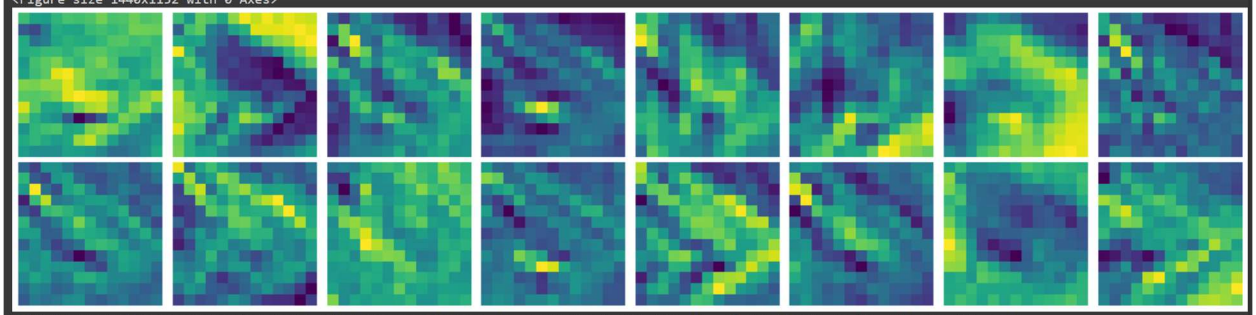

```
<Figure size 1440x1152 with 0 Axes>
```



```
The image is predicted to belong to class:  2 which is a bird
```




```
<Figure size 1440x1152 with 0 Axes>
```



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
The image is predicted to belong to class:  6 which is a frog
```

# 3a.

Build a CNN with 5 convolution layers with 3x3 kernels and corresponding 2x2 average pooling with stride 1. Use *padding* at each convolution layer so that the size of the output matches the input after each convolution layer. Train the network for any choice of model parameters.

a. *How long did it take to train? Why?*

For 20 epochs it took 25 minutes and 14 seconds.

Parameter size of network from part 2a:

```
INFO:pytorch_lightning.callbacks.model_summary:
  | Name | Type      | Params
---------------------------------
0 | c1   | Conv2d    | 168
1 | s2   | MaxPool2d | 0
2 | c3   | Conv2d    | 880
3 | s4   | MaxPool2d | 0
4 | fc5  | Linear    | 69.2 K
5 | fc6  | Linear    | 10.2 K
6 | fc7  | Linear    | 850
---------------------------------
81.3 K    Trainable params
0         Non-trainable params
81.3 K    Total params
0.325     Total estimated model params size (MB)
```

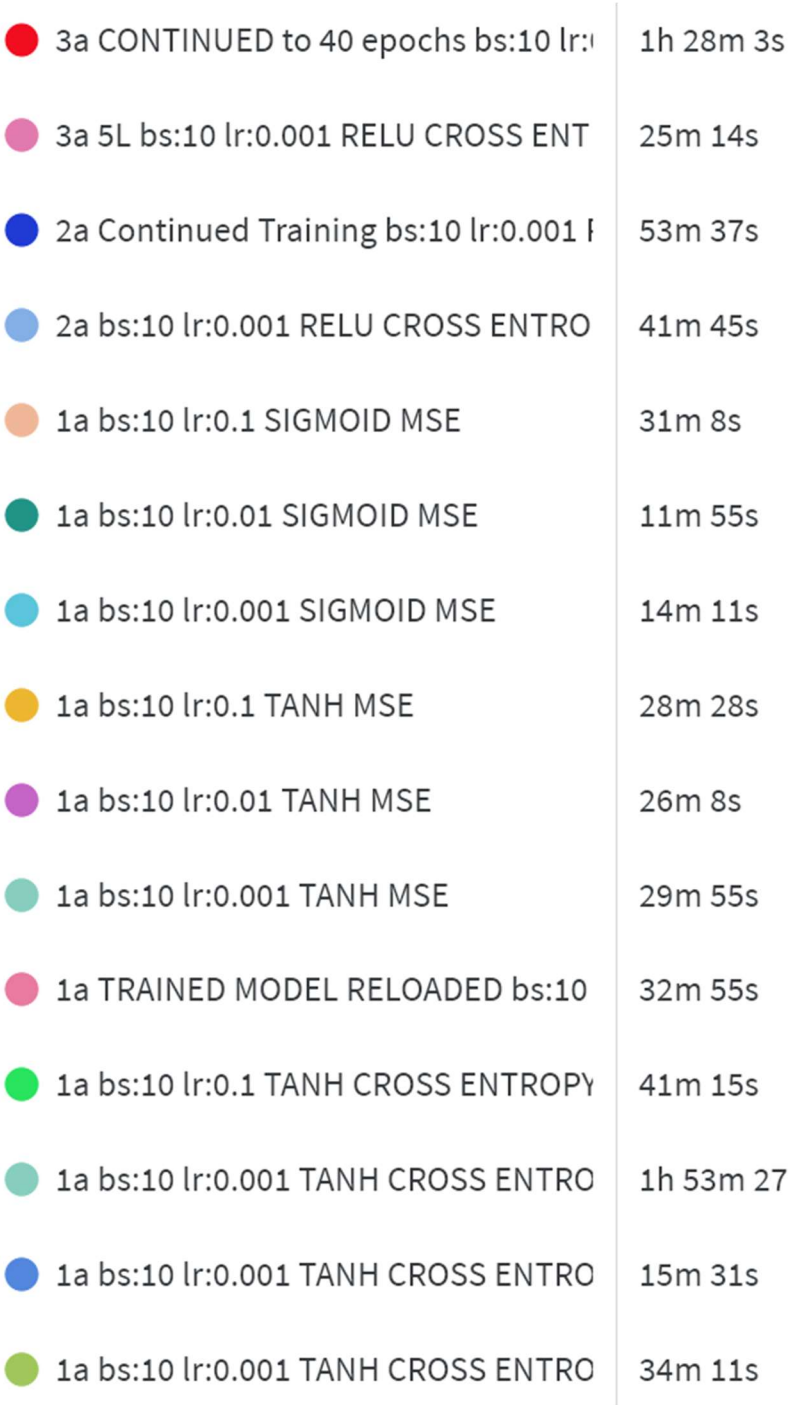
Parameter size for the current model with 5 convolution layers:

```
   | Name | Type      | Params
-----------------------------------
0  | c1   | Conv2d    | 168
1  | s2   | MaxPool2d | 0
2  | c3   | Conv2d    | 660
3  | s4   | MaxPool2d | 0
4  | c5   | Conv2d    | 2.6 K
5  | s6   | MaxPool2d | 0
6  | c7   | Conv2d    | 10.4 K
7  | s8   | MaxPool2d | 0
8  | c9   | Conv2d    | 41.6 K
9  | s10  | MaxPool2d | 0
10 | fc5  | Linear    | 1.5 M
11 | fc6  | Linear    | 10.2 K
12 | fc7  | Linear    | 850
-----------------------------------
1.5 M     Trainable params
0         Non-trainable params
1.5 M     Total params
6.164     Total estimated model params size (MB)
```

The model from 2.a had 81.3k parameters, and the current model has 1.5 M parameters, so the current network is over 18 times larger.

I was expecting this model to take significantly longer to train. However, the runtime for the current model for 20 epochs was 25 minutes. Prior models, such as the ones that used the tanh activation and cross entropy loss with varying learning rates took around the same amount of time.

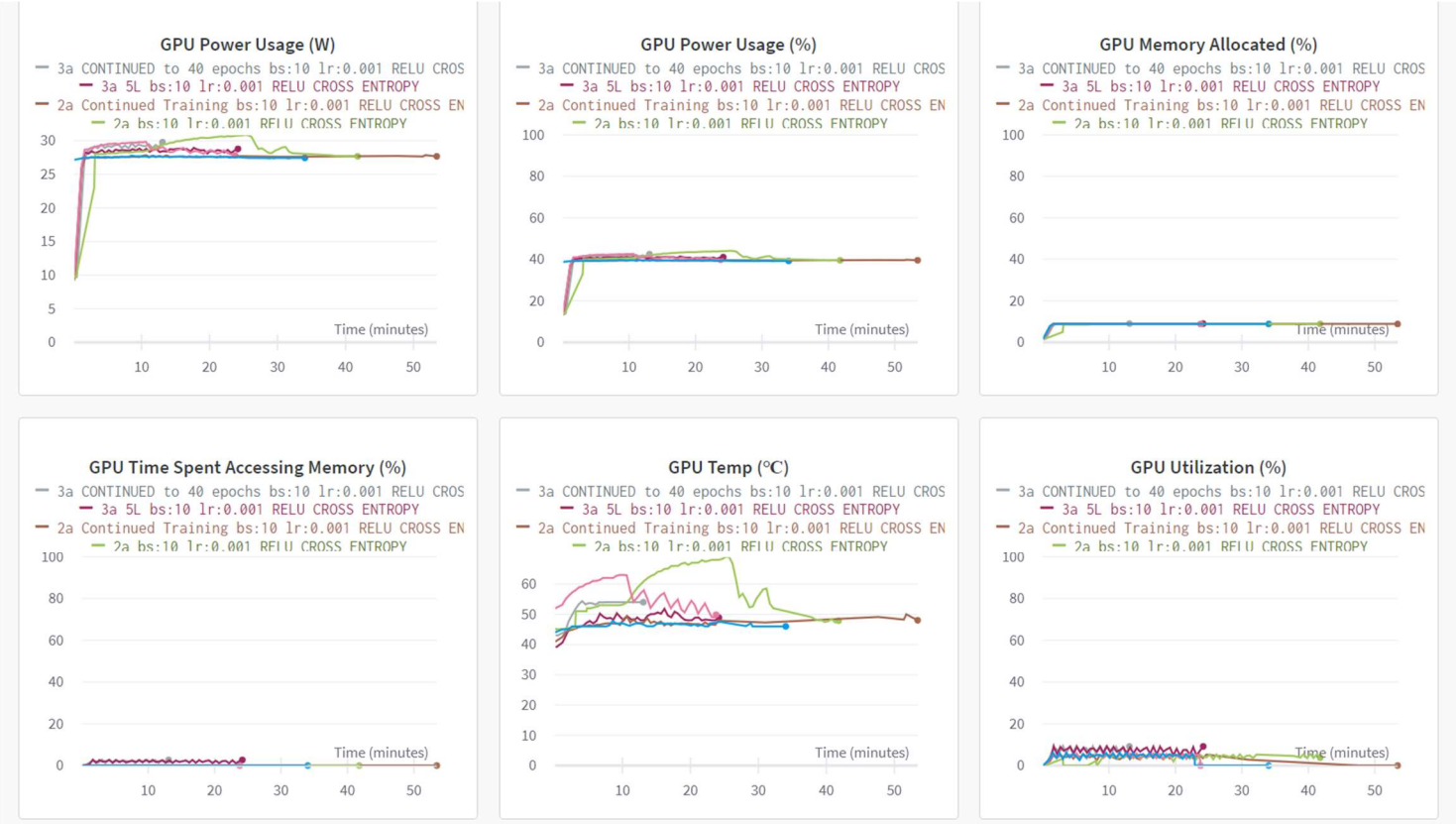
The current model is the two top runs labeled 3a. Note that the runs where “Continued” or “Reloaded” models are indicated in the run name I had taken a model that was already trained for 20 epochs and trained for another 20. It looks like the logging system made cumulative tracking of the time so those runs look much longer than they actually were.



I’m not sure why I got the rather counterintuitive result of the larger models taking less time to train than the smaller ones.. I am using Google Colab with their Tesla T4 GPU which is their basic GPU.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 510.47.03    Driver Version: 510.47.03    CUDA Version: 11.6     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   43C    P8    10W /  70W |      3MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

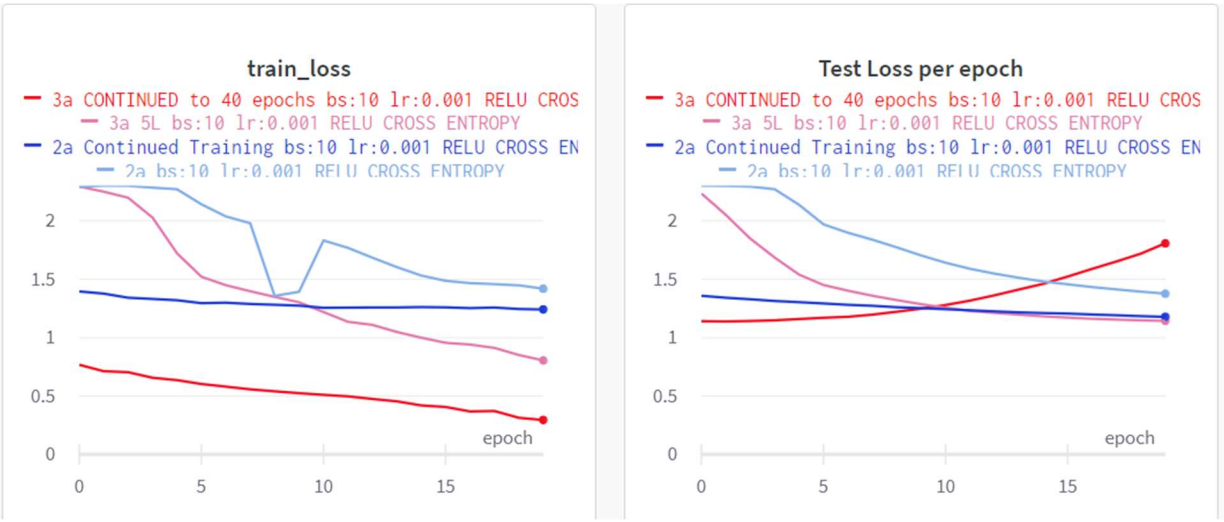
GPU Usage looks similar across runs, with the exception of the green run here, the ReLU run with LR of 0.001 and Cross Entropy Loss. I’m not sure what caused that spike in GPU usage or temperature.



# 3b.

b. *How does the accuracy of this network compare against the previous two? Explain*

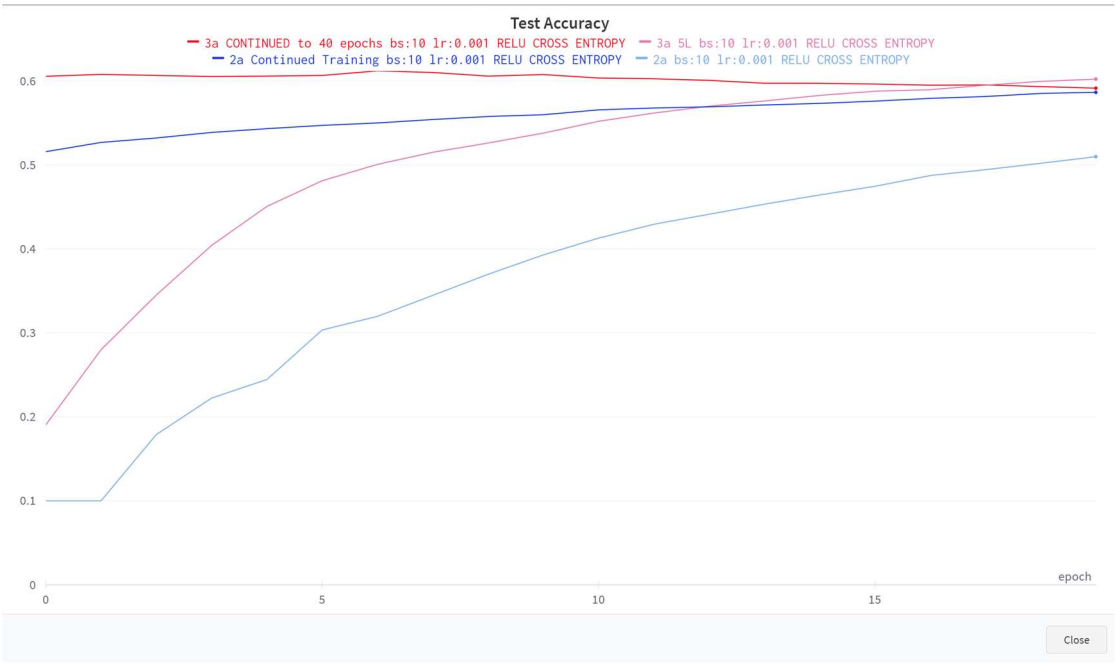
Below is a plot of the 5 convolution model and the previous model form 2




Once again I am using a poorly devised visualization in that the light blue and dark blue lines represent the model from 2a. The light blue line is the first 20 epochs and the dark blue is for epochs 20-40. The same goes for the light red and dark red lines representing the model from 3a. I apologize for this confusion but I wanted to reload each model and train them another 20 epochs and I didn't have time to reconfigure the logging.

The new model with 5 convolution layers did extremely well in the first 20 epochs, outperforming all the other models. But then after 20 epochs it succumbed to overfitting. If we look at the dark red line, the training loss continues to decline but the test loss starts increasing after the 20$^{th}$ epoch.

This is also evident in examining the testing accuracy, which declines as well, but not as severely as the test loss. I'm not sure why it declines less severely.



And here is the 5-layer network (once again in light and dark red) compared to two well performing runs from 1a.