Project 3: Machine Learning & Artifical Neural Networks

Sophia Strano, Luke Foley, Ilana Whittaker

Written Report

Model & Training Procedure

This project dives into building and training Deep Neural Networks (DNNs) for image classification: data preprocessing, model architecture and hyperparameter selection. We are using a multi-layered convolutional neural network (CNN). When determining our hyperparameters, we restrict the number of epochs to 30, as our model's testing accuracy plateaus and then decreases after this number. The hidden size is 300, as a higher hidden size continuously increases our testing accuracy, however this number greatly increases the time cost of running the algorithm the higher it is set at
Our scale factor is set at 100 because numbers higher or lower than this value decrease our testing accuracy.
The optimizer that we are using is Adam because it achieves a high initial accuracy, despite its performance gap between other optimizers.

In our CNN, we are using 40 filters to find patterns in our images, as a relatively high filter value increases our training accuracy. We experimented with various Our filter size is 3, because size these images are only 2x2, a size larger than this is too much to analyze to find a pattern, and a size smaller does not have enough information.Our pooling size is 5, which is relatively large, and is done because this helps us to find averages of values along long stretches of the image. Our model also has three layers, tanh, tanh, and relu activation functions respectively,with a high number of analyzed values to achieve the best performance. Smaller and larger networks tended to be less accurate for this data set, and analyzing small proportions of the data set would not affect the testing accuracy by much.

Model Performance & Confusion Matrix

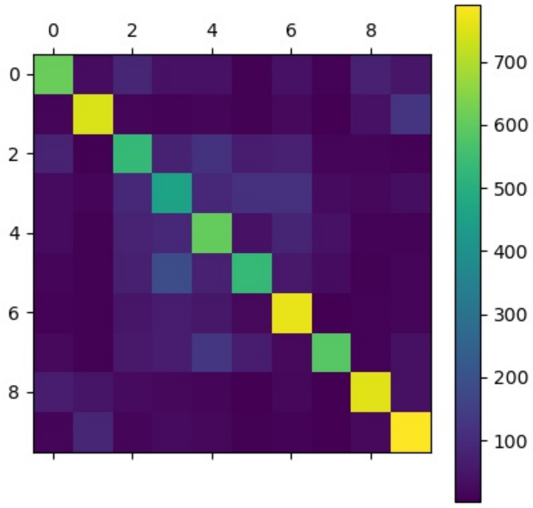
The aspects of our network that impact the testing accuracy the most are number of epochs, network architecture, and pooling size value. In our best performing model, we obtained accuracies ranging between 31%-63%, with our best accuracy being 63.48%. The output of our best performing model is located in the Model folder.

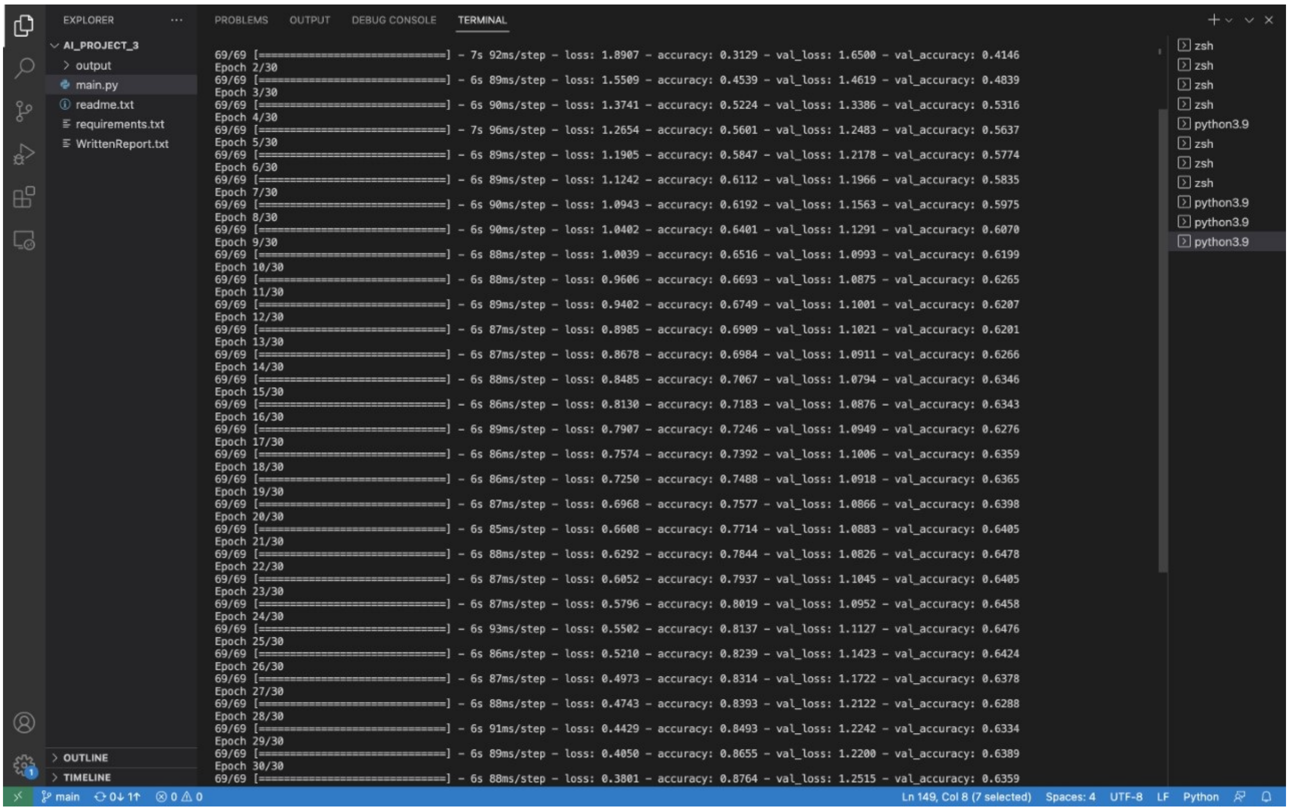
In our best performing model, our precision and accuracy were

Precision: [0.67630701, 0.78548559, 0.50965961, 0.43686354, 0.52463504, 0.59572072,
0.61793215, 0.78580815, 0.76869392, 0.70694319]

Recall: [0.608, 0.736, 0.554, 0.429, 0.575, 0.529, 0.765, 0.598, 0.771, 0.784]

Here is a confusion matrix for the results of our best performing model:





```
69/69 [==============================] - 7s 92ms/step - loss: 1.8907 - accuracy: 0.3129 - val_loss: 1.6500 - val_accuracy: 0.4146
Epoch 2/30
69/69 [==============================] - 6s 89ms/step - loss: 1.5509 - accuracy: 0.4539 - val_loss: 1.4619 - val_accuracy: 0.4839
Epoch 3/30
69/69 [==============================] - 6s 90ms/step - loss: 1.3741 - accuracy: 0.5224 - val_loss: 1.3386 - val_accuracy: 0.5316
Epoch 4/30
69/69 [==============================] - 7s 96ms/step - loss: 1.2654 - accuracy: 0.5601 - val_loss: 1.2483 - val_accuracy: 0.5637
Epoch 5/30
69/69 [==============================] - 6s 89ms/step - loss: 1.1905 - accuracy: 0.5847 - val_loss: 1.2178 - val_accuracy: 0.5774
Epoch 6/30
69/69 [==============================] - 6s 89ms/step - loss: 1.1242 - accuracy: 0.6112 - val_loss: 1.1966 - val_accuracy: 0.5835
Epoch 7/30
69/69 [==============================] - 6s 90ms/step - loss: 1.0943 - accuracy: 0.6192 - val_loss: 1.1563 - val_accuracy: 0.5975
Epoch 8/30
69/69 [==============================] - 6s 90ms/step - loss: 1.0402 - accuracy: 0.6401 - val_loss: 1.1291 - val_accuracy: 0.6070
Epoch 9/30
69/69 [==============================] - 6s 88ms/step - loss: 1.0039 - accuracy: 0.6516 - val_loss: 1.0993 - val_accuracy: 0.6199
Epoch 10/30
69/69 [==============================] - 6s 88ms/step - loss: 0.9606 - accuracy: 0.6693 - val_loss: 1.0875 - val_accuracy: 0.6265
Epoch 11/30
69/69 [==============================] - 6s 89ms/step - loss: 0.9402 - accuracy: 0.6749 - val_loss: 1.1001 - val_accuracy: 0.6207
Epoch 12/30
69/69 [==============================] - 6s 87ms/step - loss: 0.8985 - accuracy: 0.6909 - val_loss: 1.1021 - val_accuracy: 0.6201
Epoch 13/30
69/69 [==============================] - 6s 87ms/step - loss: 0.8678 - accuracy: 0.6984 - val_loss: 1.0911 - val_accuracy: 0.6266
Epoch 14/30
69/69 [==============================] - 6s 88ms/step - loss: 0.8485 - accuracy: 0.7067 - val_loss: 1.0794 - val_accuracy: 0.6346
Epoch 15/30
69/69 [==============================] - 6s 86ms/step - loss: 0.8130 - accuracy: 0.7183 - val_loss: 1.0876 - val_accuracy: 0.6343
Epoch 16/30
69/69 [==============================] - 6s 89ms/step - loss: 0.7907 - accuracy: 0.7246 - val_loss: 1.0949 - val_accuracy: 0.6276
Epoch 17/30
69/69 [==============================] - 6s 86ms/step - loss: 0.7574 - accuracy: 0.7392 - val_loss: 1.1006 - val_accuracy: 0.6359
Epoch 18/30
69/69 [==============================] - 6s 86ms/step - loss: 0.7250 - accuracy: 0.7488 - val_loss: 1.0918 - val_accuracy: 0.6365
Epoch 19/30
69/69 [==============================] - 6s 87ms/step - loss: 0.6968 - accuracy: 0.7577 - val_loss: 1.0866 - val_accuracy: 0.6398
Epoch 20/30
69/69 [==============================] - 6s 85ms/step - loss: 0.6608 - accuracy: 0.7714 - val_loss: 1.0883 - val_accuracy: 0.6405
Epoch 21/30
69/69 [==============================] - 6s 88ms/step - loss: 0.6292 - accuracy: 0.7844 - val_loss: 1.0826 - val_accuracy: 0.6478
Epoch 22/30
69/69 [==============================] - 6s 87ms/step - loss: 0.6052 - accuracy: 0.7937 - val_loss: 1.1045 - val_accuracy: 0.6405
Epoch 23/30
69/69 [==============================] - 6s 87ms/step - loss: 0.5796 - accuracy: 0.8019 - val_loss: 1.0952 - val_accuracy: 0.6458
Epoch 24/30
69/69 [==============================] - 6s 93ms/step - loss: 0.5502 - accuracy: 0.8137 - val_loss: 1.1127 - val_accuracy: 0.6476
Epoch 25/30
69/69 [==============================] - 6s 86ms/step - loss: 0.5210 - accuracy: 0.8239 - val_loss: 1.1423 - val_accuracy: 0.6424
Epoch 26/30
69/69 [==============================] - 6s 87ms/step - loss: 0.4973 - accuracy: 0.8314 - val_loss: 1.1722 - val_accuracy: 0.6378
Epoch 27/30
69/69 [==============================] - 6s 88ms/step - loss: 0.4743 - accuracy: 0.8393 - val_loss: 1.2122 - val_accuracy: 0.6288
Epoch 28/30
69/69 [==============================] - 6s 91ms/step - loss: 0.4429 - accuracy: 0.8493 - val_loss: 1.2242 - val_accuracy: 0.6334
Epoch 29/30
69/69 [==============================] - 6s 89ms/step - loss: 0.4050 - accuracy: 0.8655 - val_loss: 1.2200 - val_accuracy: 0.6389
Epoch 30/30
69/69 [==============================] - 6s 88ms/step - loss: 0.3801 - accuracy: 0.8764 - val_loss: 1.2515 - val_accuracy: 0.6359
```

To design our best performing model, we used the following activation functions in our training procedure:

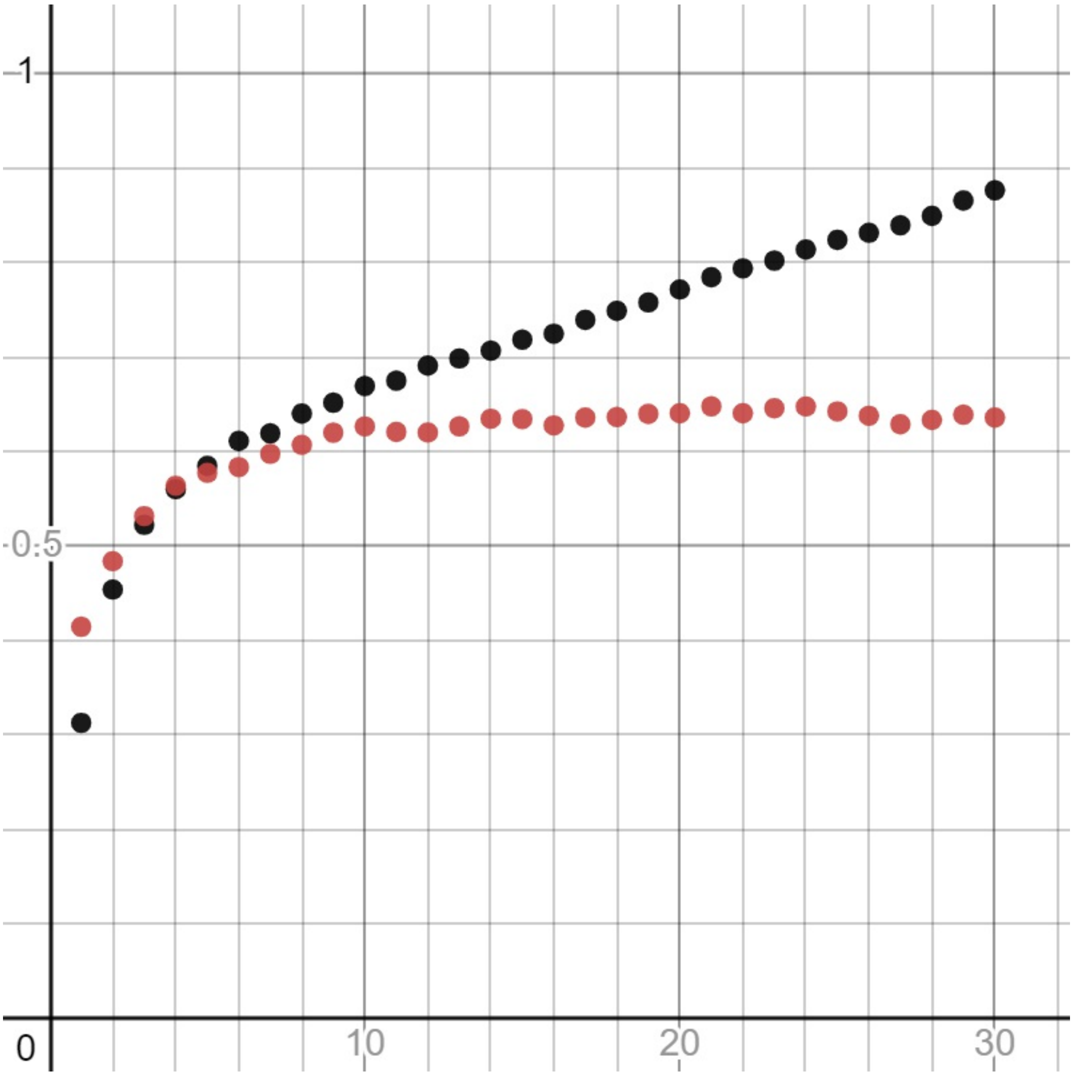
```
model.add(Flatten())
model.add(Dense((args.hidden_size)*7/8, activation="relu"))  # first layer
model.add(Dense((args.hidden_size)*5/6, activation='relu'))  # second layer
model.add(Dense((args.hidden_size)*1/5, activation='selu'))  # third layer
```

Below is an example of our model successfully classifying an image of a ship!

ship ship

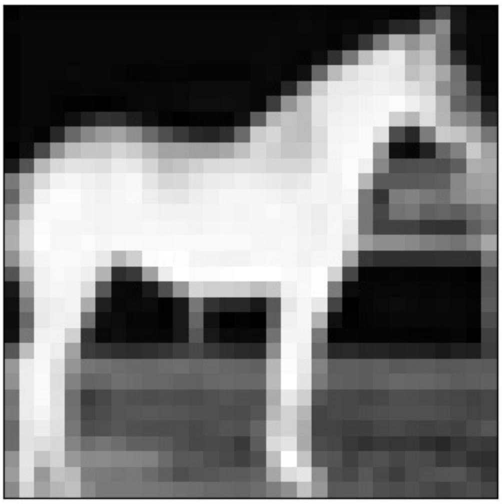
Training performance plot

The following plot represents our model's training accuracy and validation accuracy with respect to the number of training epochs (x axis) and accuracy (y axis).



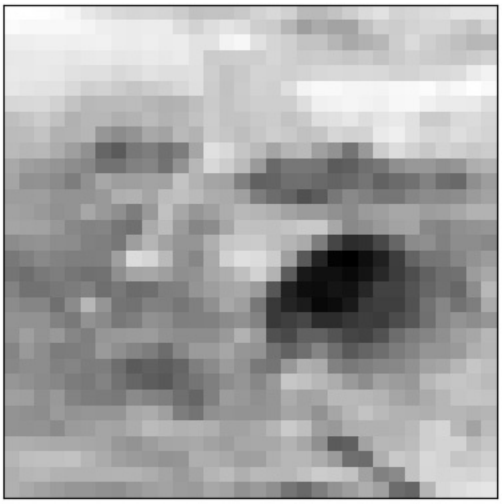
Misclassified Visualizations

Despite the high accuracy of our model, there were still a few misclassifications- here are three examples.



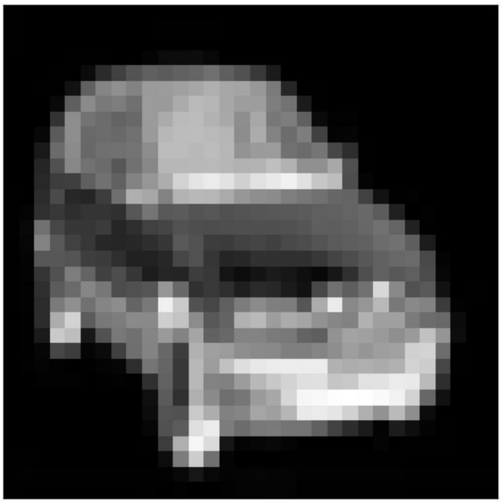
automobile horse

In this case, our model thought this difficult to discern horse was an automobile.



frog deer

Given that this image is difficult for a human to percieve, it is understandable that it was misclassified.



truck automobile

This was a close misclassification that can potentially be explained by the close resemblance of trucks and automobiles.