CS 4372 Assignment 3

Keerthi Srilakshmidaran and Rameen Usman
KXS180064 and RFU180000

Google Colab:
https://colab.research.google.com/drive/1U6RDgv4RFeoxxPya4bbKtqaPzAbgtrE1?authuser=0#scrollTo=BQYKR38SWvfD

Dataset:
https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset?select=readme%5B1%5D.txt

Our dataset is a cats and dogs image dataset that we found on kaggle through Google Research Datasets. There were around 25k images in the data originally, but we had to delete around 2000 images because they were corrupted. We used code to delete the corrupted images and uploaded the updated zipped dataset to utd box and accessed it from there for the project. The updated dataset used for our project has around 22k images with two classes- cat and dog.

Methodology:

We began with preprocessing our data. As stated above, we deleted any corrupted images in our dataset. We split our data into a train and validation set that had class labels attached. The test set was then created off the validation set. Further, we augmented our data so that we can introduce sample diversity by transforming and rotating our images. This will help our train data with overfitting. Then we created our base MobileNetV2 model. We do not include the top layer in our base model as it is ideal for feature extraction. The filter size and activation function were changed in the base model for fine tuning. Then we froze our convolution base as it prevents weights in other layers to be updated in training. After this, we created a dense layer to convert the features into a single prediction for each image, and created our model. In our model compilation, we were also able to change the learning rate for parameter tuning. Also, we trained our model on different numbers of epochs and checked our validation sets and test accuracy. After all of this we printed our images with the predicted and true labels which can be seen in this report.

Parameter Testing and Tuning:

For our parameter tuning, we tested 15 models that have different combinations of these 5 parameters: Activation Function, Learning Rate, Number of Layers, Alpha, and Epochs. Here, the alpha parameter controls the width of the network. If alpha < 1, it proportionally decreases the number of filters in each layer and the opposite if alpha > 1. The three activation functions we used were relu, softmax, and sigmoid. The number of epochs we tried were 5, 10, 15, and 20. Though we would have preferred to try more numbers, gpu and time constraints restricted us to a smaller number of epochs. For learning rate we tried rates of 0.1, 0.01, 0.001, and 0.0001. Finally, we also changed the number of layers of our model by using 8-12 layers inclusive. The architecture we used for each number of layers are as shown:

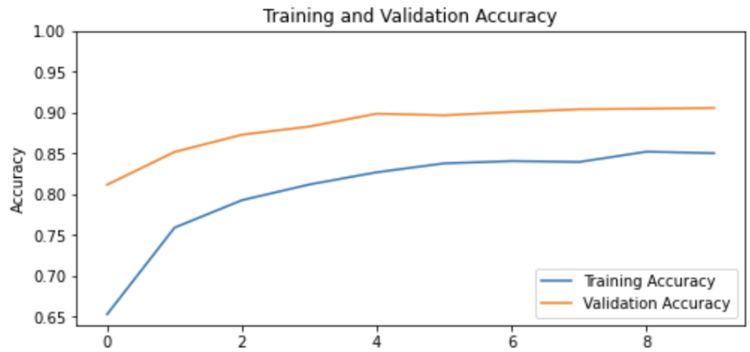
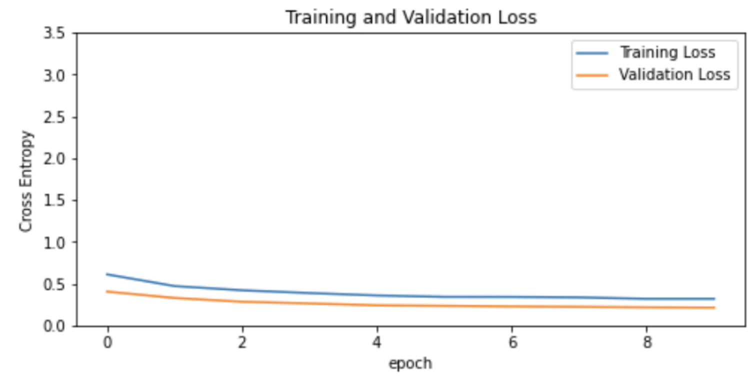
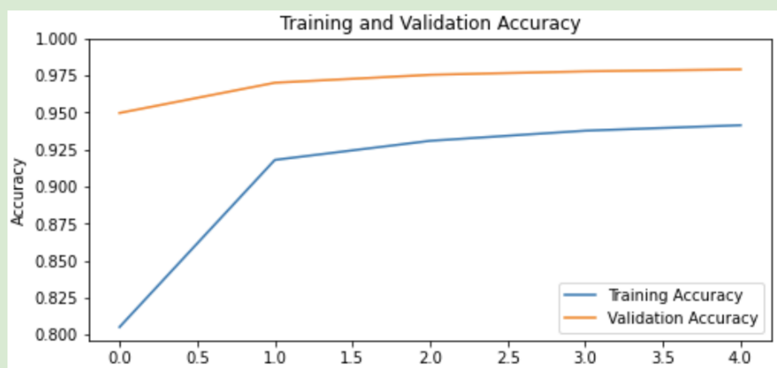
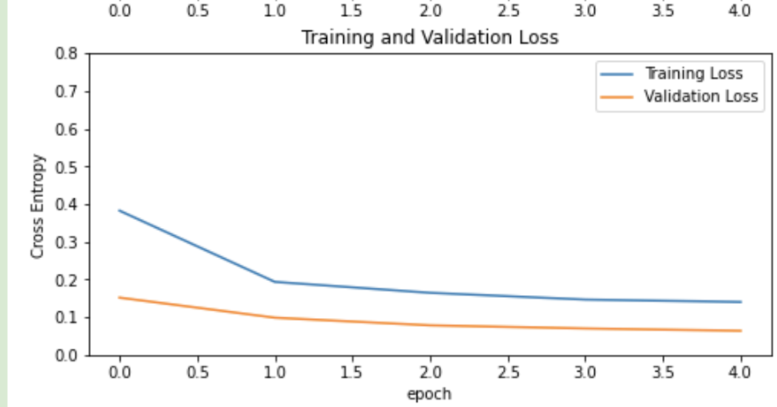
LAYER = 8 inputs = tf.keras.Input(shape=(160, 160, 3)) x = data_augmentation(inputs) x = preprocess_input(x) x = base_model(x, training=False) x = global_average_layer(x) x = tf.keras.layers.Dropout(0.2)(x) outputs = prediction_layer(x) model = tf.keras.Model(inputs, outputs)	**LAYER = 9** inputs = tf.keras.Input(shape=(160, 160, 3)) x = data_augmentation(inputs) x = preprocess_input(x) x = Dense(units=3)(x) x = base_model(x, training=False) x = global_average_layer(x) x = tf.keras.layers.Dropout(0.2)(x) outputs = prediction_layer(x) model = tf.keras.Model(inputs, outputs)	**LAYER = 10** inputs = tf.keras.Input(shape=(160, 160, 3)) x = data_augmentation(inputs) x = preprocess_input(x) x = Dense(units=3)(x) x = base_model(x, training=False) x = Dense(units=1280)(x) x = global_average_layer(x) x = tf.keras.layers.Dropout(0.2)(x) outputs = prediction_layer(x) model = tf.keras.Model(inputs, outputs)
LAYER = 11 inputs = tf.keras.Input(shape=(160, 160, 3)) x = data_augmentation(inputs) x = preprocess_input(x) x = Dense(units=3)(x) x = base_model(x, training=False) x = Dense(units=1280)(x) x = tf.keras.layers.Dropout(0.2)(x) x = global_average_layer(x) x = tf.keras.layers.Dropout(0.2)(x) outputs = prediction_layer(x) model = tf.keras.Model(inputs, outputs)	**LAYER = 12** inputs = tf.keras.Input(shape=(160, 160, 3)) x = data_augmentation(inputs) x = preprocess_input(x) x = Dense(units=3)(x) x = base_model(x, training=False) x = Dense(units=3)(x) x = tf.keras.layers.Dropout(0.2)(x) x = global_average_layer(x) x = Dense(units=1280)(x) x = tf.keras.layers.Dropout(0.2)(x) outputs = prediction_layer(x) model = tf.keras.Model(inputs, outputs)	

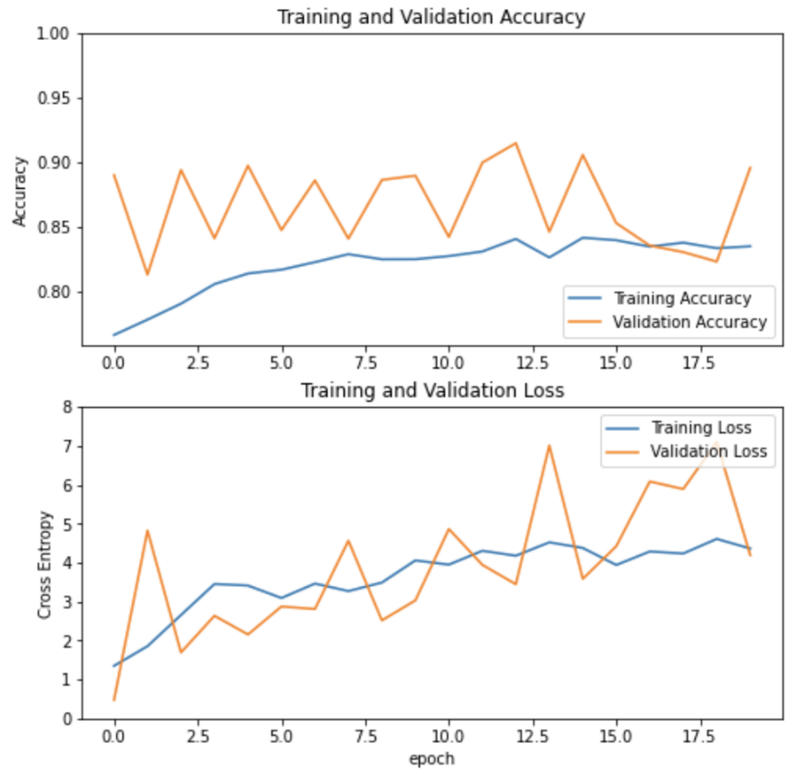
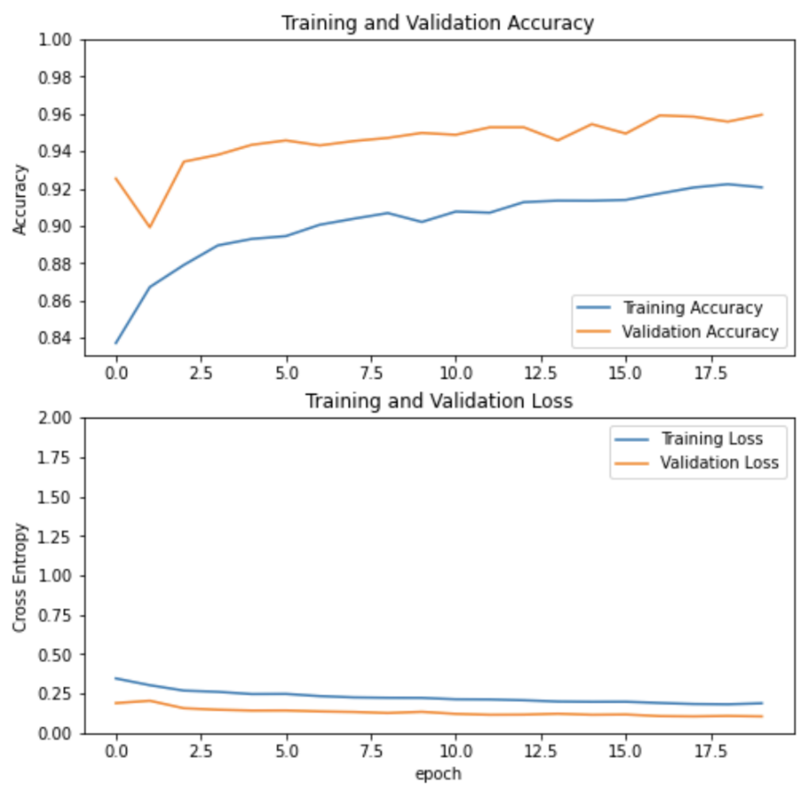
Parameter Tuning Accuracy & Loss Values:

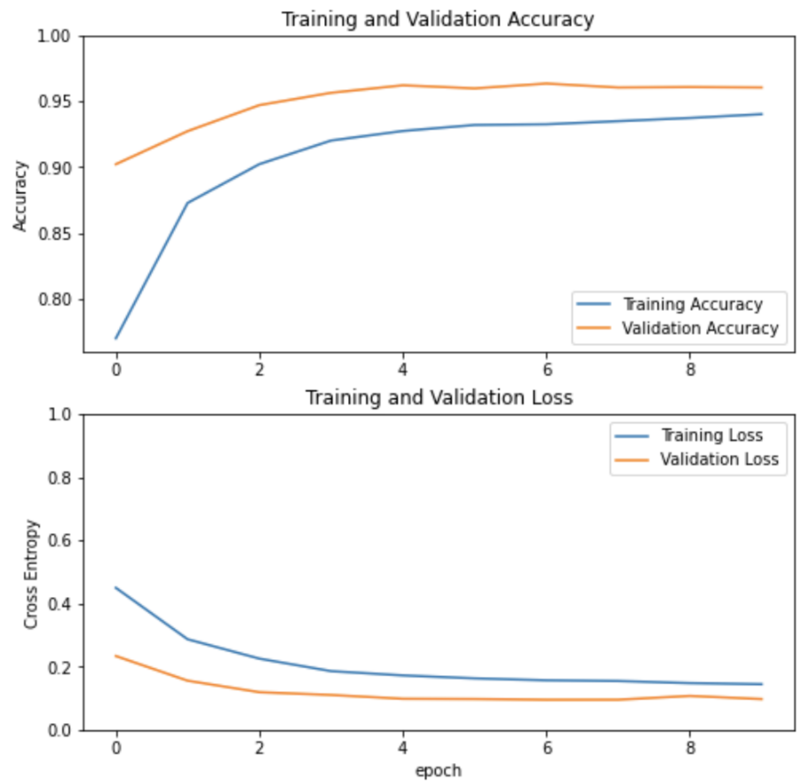
Number	Fine Tuning Parameters	Accuracy	Loss
1	Activation Function: softmax Learning Rate: 0.0001 Number of Layers: 9 Alpha: 0.5 Epoch: 5	Train: 0.8759 Validation: 0.9273 Test: 0.9402	Train: 0.2776 Test: 0.1492
2	Activation Function: relu Learning Rate: 0.001 Number of Layers: 12 Alpha: 0.5 Epoch: 10	Train: 0.9402 Validation: 0.9605 Test: 0.9565	Train: 0.1443 Test: 0.1065
3	Activation Function: sigmoid Learning Rate: 0.001 Number of Layers: 9 Alpha: 0.5 Epoch: 10	Train: 0.9158 Validation: 0.9488 Test: 0.9497	Train: 0.2001 Test: 0.1246
4	Activation Function: softmax Learning Rate: 0.0001 Number of Layers: 9 Alpha: 0.5 Epoch: 10	Train: 0.8504 Validation: 0.9059 Test: 0.8832	Train: 0.3182 Test: 0. 2387
5	Activation Function: relu Learning Rate: 0.001 Number of Layers: 12 Alpha: 0.5 Epoch: 15	Train: 0.8980 Validation: 0.9166 Test: 0.9226	Train: 0.2328 Test: 0.1713
6	Activation Function: sigmoid Learning Rate: 0.0001 Number of Layers: 9 Alpha: 0.5 Epoch: 5	Train: 0.8725 Validation: 0.9297 Test: 0.9457	Train: 0.2775 Test: 0.1484
7	Activation Function: sigmoid Learning Rate: 0.001 Number of Layers: 9 Alpha: 0.5 Epoch: 15	Train: 0.8749 Validation: 0.9119 Test: 0.9334	Train: 0.2717 Test: 0.1665
8	Activation Function: softmax Learning Rate: 0.01 Number of Layers: 11 Alpha: .35 Epoch: 10	Train: 0.9325 Validation: 0.9504 Test: 0.944	Train: 1.2579 Test: 2.5170

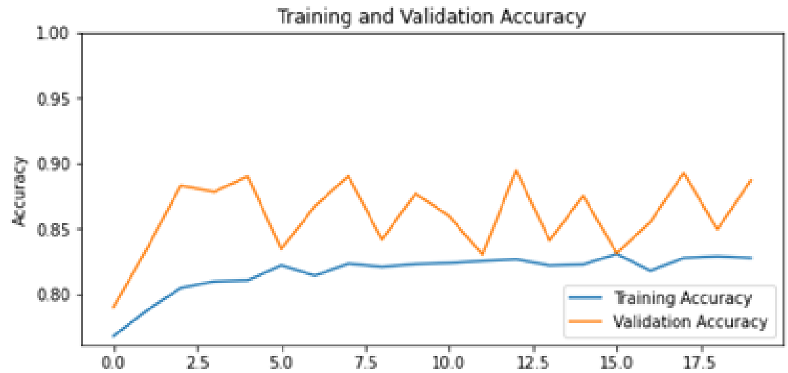
9	Activation Function: relu Learning Rate: 0.01 Number of Layers: 10 Alpha: .75 Epoch: 10	Train: 0.9325 Validation: 0.9504 Test: 0.944	Train: 4.4500 Test: 4.1976
10	Activation Function: sigmoid Learning Rate: 0.01 Number of Layers: 11 Alpha: 1 Epoch: 10	Train: 0.8314 Validation: 0.8798 Test: 0.8858	Train: 6.7142 Test: 7.8790
11	Activation Function: softmax Learning Rate: 0.01 Number of Layers: 11 Alpha: .35 Epoch: 20	Train: 0.8278 Validation: 0.8871 Test: 0.8845	Train: 3.3617 Test: 3.1095
12	Activation Function: relu Learning Rate: 0.01 Number of Layers: 10 Alpha: .75 Epoch: 20	Train: 0.8347 Validation: 0.8955 Test: 0.9049	Train: 4.3626 Test: 3.2268
13	Activation Function: sigmoid Learning Rate: 0.01 Number of Layers: 8 Alpha: 1 Epoch: 20	Train: 0.9419 Validation: 0.9732 Test: 0.9823	Train: 4.0112 Test: 3.2671
14	Activation Function: relu Learning Rate: 0.0001 Number of Layers: 10 Alpha: .75 Epoch: 20	Train: 0.9206 Validation: 0.9595 Test: 0.9633	Train: 0.1885 Test: 0.0761
15	Activation Function: sigmoid Learning Rate: 0.0001 Number of Layers: 8 Alpha: 1 Epoch: 5	Train: 0.9415 Validation: 0.9792 Test: 0.9824	Train: 0.1403 Test: 0.0502

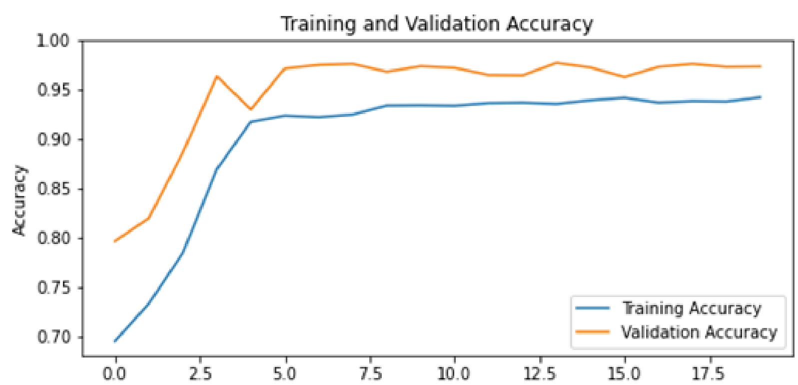
History Plots for 8 Models:

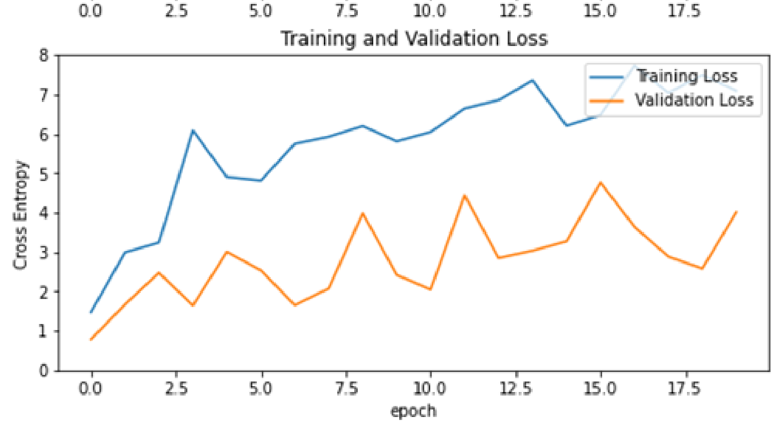
Number	Parameters	Accuracy and Cross Entropy Plots
4	Activation Function: softmax Learning Rate: 0.0001 Number of Layers: 9 Alpha: 0.5 Epoch: 10	 
15	Activation Function: sigmoid Learning Rate: 0.0001 Number of Layers: 8 Alpha: 1 Epoch: 5	 

12	Activation Function: relu Learning Rate: 0.01 Number of Layers: 10 Alpha: .75 Epoch: 20	
14	Activation Function: relu Learning Rate: 0.0001 Number of Layers: 10 Alpha: .75 Epoch: 20	

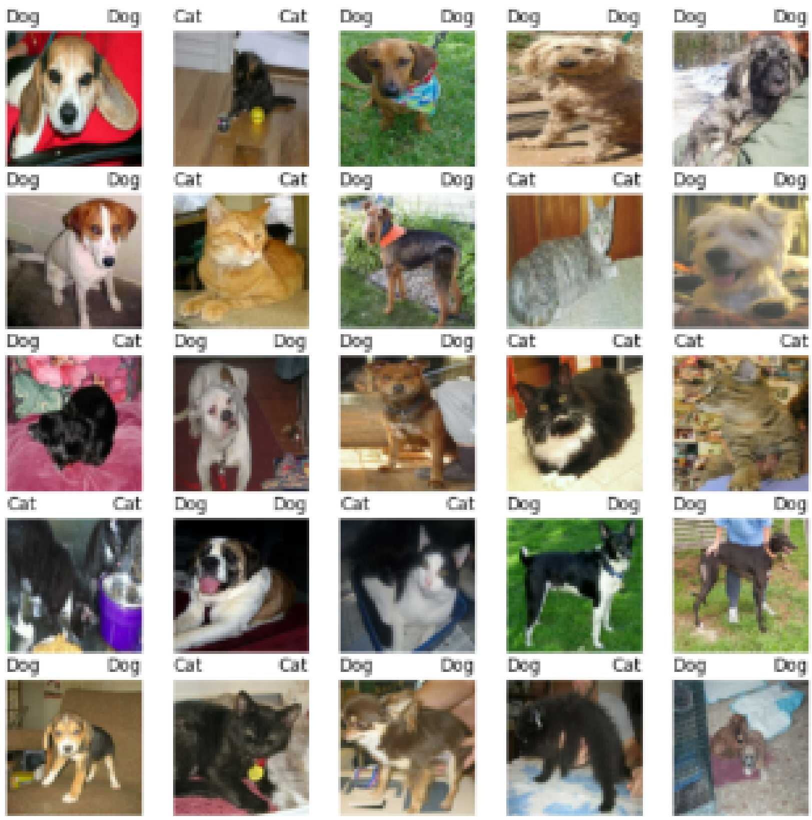
2	Activation Function: relu Learning Rate: 0.001 Number of Layers: 12 Alpha: 0.5 Epoch: 10	
9	Activation Function: relu Learning Rate: 0.01 Number of Layers: 10 Alpha: .75 Epoch: 10	

<table>
<tr>
<td>11</td>
<td>Activation Function:
softmax
Learning Rate: 0.01
Number of Layers: 11
Alpha: .35
Epoch: 20</td>
<td>


</td>
</tr>
<tr>
<td>13</td>
<td>Activation Function:
sigmoid
Learning Rate: 0.01
Number of Layers: 8
Alpha: 1
Epoch: 20</td>
<td>


</td>
</tr>
</table>

25 Test Images with Labels:
Right label: true label
Left label: predicted label



Since our model performed quite well, we can see that most of our images were classified correctly. However, in our 25 example test images, we can see that we have 2 images classified incorrectly. The first is in the 3rd row and 1st column where the true label was a dog which our model classified as a cat. The other misclassified point is located in the 5th row and 4th column where the true label was a dog and our model classified a cat.

Analysis:

After parameter tuning, we were able to notice patterns and draw inferences from our results. One main thing we noticed was the effect of learning rate on our models. One of the first models we tried used a sigmoid activation function and had a learning rate of 0.01 (model 13). We noticed from the graph for this model that as accuracy increased, so did loss. This was unusual as we expected that if accuracy increases, loss should decrease. Furthermore we noticed that after around 5 epochs training and validation accuracy were relatively stable and stopped increasing significantly. We originally thought this was because of overfitting but our high test accuracy proved otherwise. After doing some research, we found out that this may be due to a high learning rate. We lowered our learning rate to 0.0001 and reduced the number of epochs to 5, while keeping the other parameters the same, and saw that the relationship for loss and accuracy was restored- accuracy increased while loss decreased. This model (model 15) also proved to be the best model in our parameter tuning with a test accuracy of 0.9824 and train of 0.9415. This model is highlighted in a light green in the previous sections.

Another thing we noticed is that models with a smaller number of layers performed better or had a higher accuracy score than models with a larger number of models. For example, the model with a sigmoid activation, 0.01 learning rate, 1 alpha, and 8 layers (model 13) had a test accuracy of 0.9823. Conversely, when we parameter tuned with the same model listed above but changed the number of layers to 11 (model 11), the test accuracy was 0.8845. Looking at these two models specifically and others as well in general, we saw that typically when our models had a fewer number of layers, our accuracy scores were better.

We also wanted to see if training for a larger number of epochs would result in a better accuracy. Model 8 for example which has the same parameter values as model 11, but only has 10 epochs compared to model 11's 20 epochs, had a test accuracy of 0.944 while model 11 had a test accuracy of 0.8845. Similarly, model 9 which is the same as model 12 but has 10 epochs compared to model 12's 20 epochs, had a test accuracy of 0.944 while model 12's accuracy was 0.9049. We see that for our dataset and our model, lower valued epochs do better. This may be because as more epochs pass, the model starts to overfit which can lead to fluctuation in values.