Identification of Coffee Cultivation Areas in Brazil

Tova Perlman

MUSA 650 Final Project

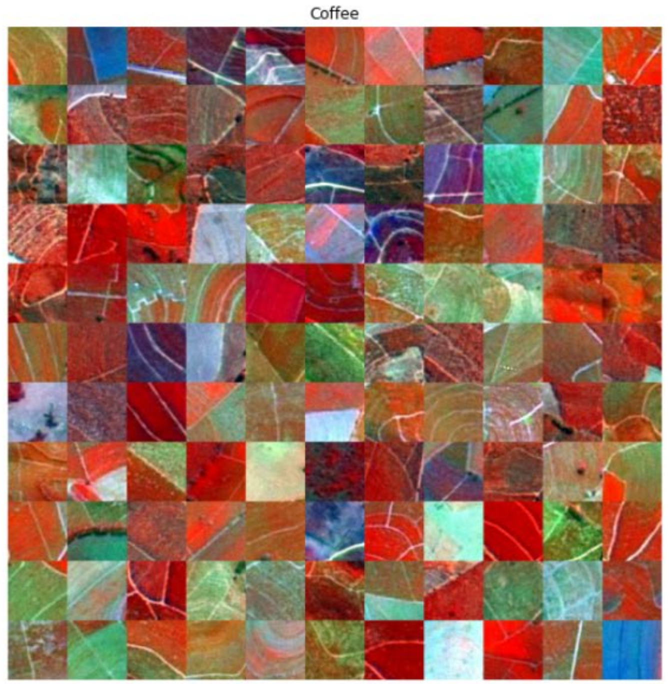
Introduction

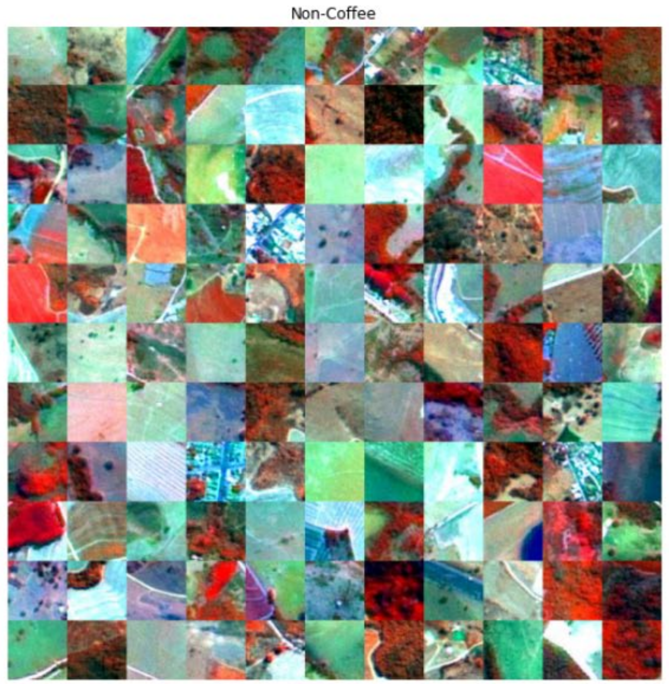
The Problem:

This dataset features satellite images of four different states in Brazil. It is used to help manage and understand coffee cultivation management and techniques. Thus, we are trying to identify images that are good for growing coffee or not good. In other words, we are solving a binary classification problem.

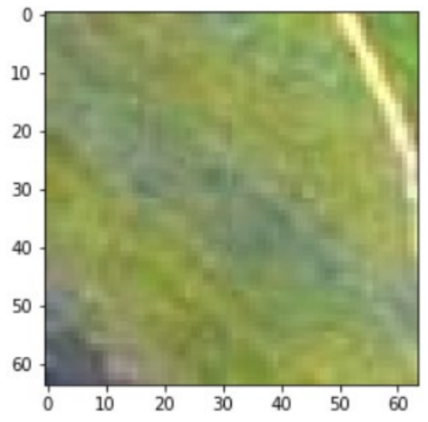
Dataset Description	Value
Number of Images	2876
Image Size	64 x 64 pixels
Number of classes	2
Balance	50/50

Here are what the images look like:

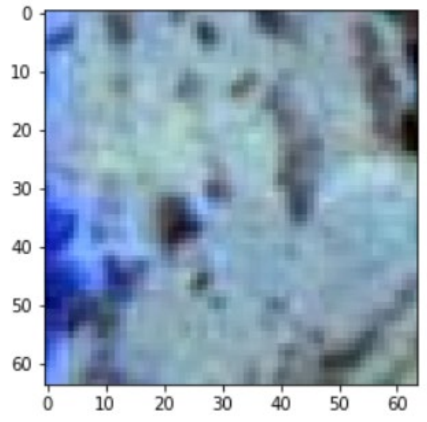




True Color:



Label Coffee



Label Non-Coffee

Classifiers

Experiment 1:
SVM:

We ran Gridsearch CV with 3 fold cross validation and found that the optimal hyperparameters for SVM were C: .01, 2 degrees and sigmoid Kernel.

Accuracy for SVM with optimal parameters: 57%

Experiment 2:
Random Forest:

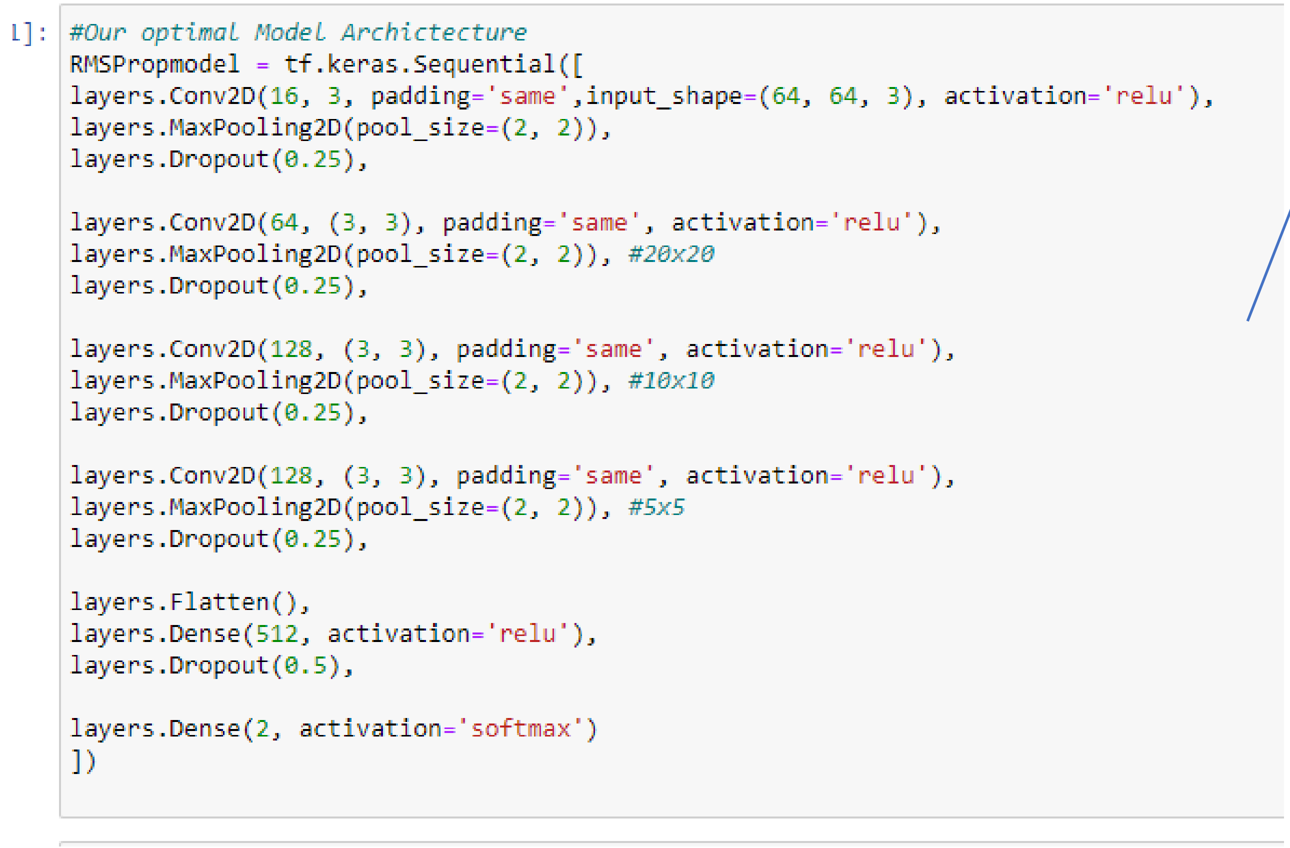
We ran Gridsearch CV with 3 fold cross validation and found that the optimal hyperparameters for Random Forest was a max depth of 5 and the number of estimators at 200.

Accuracy for Random Forest with optimal parameters: 59%

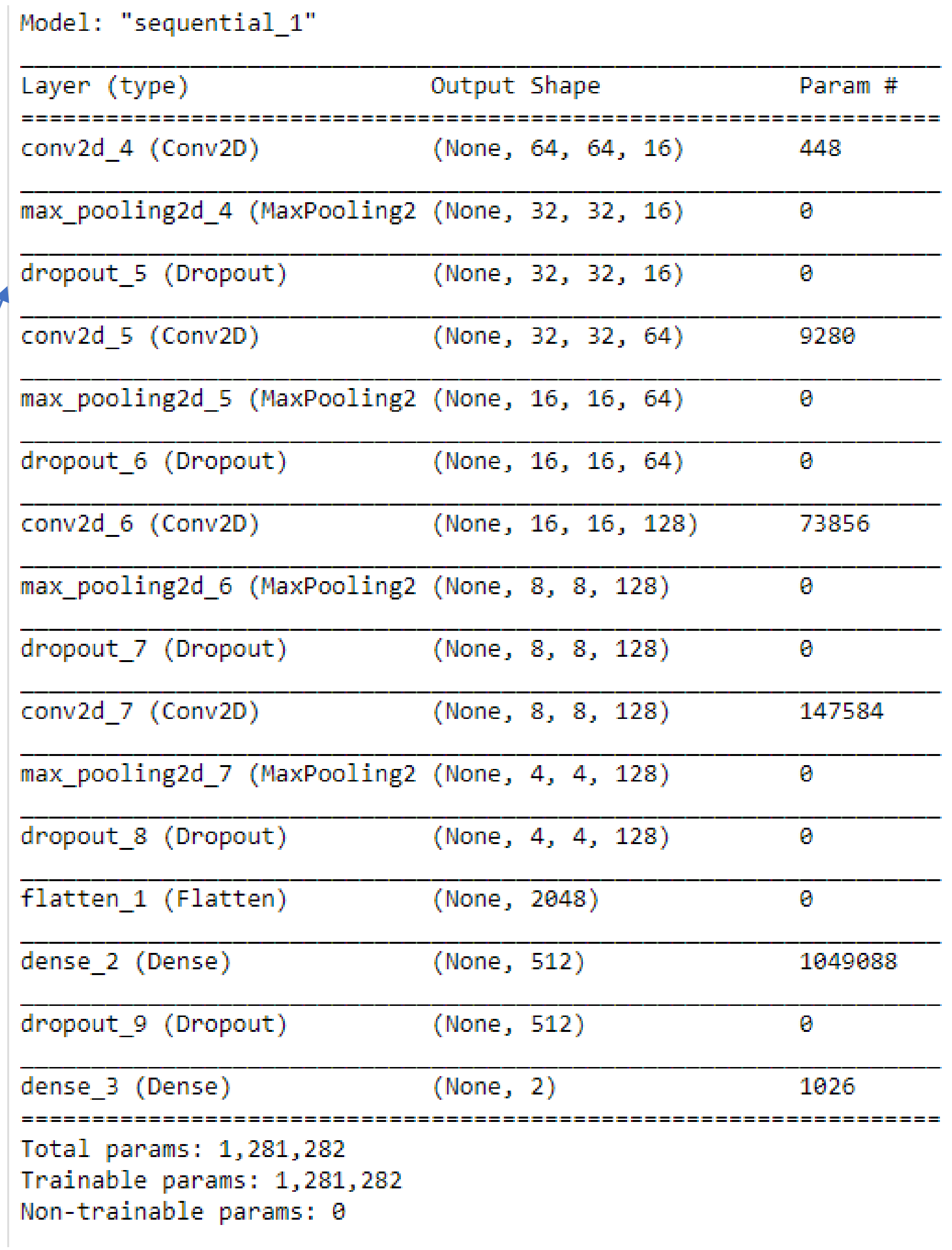
This accuracy isn’t great…. Let’s see if we can do better with a convolutional neural network

CNN Model Architecture Set Up

We set up a model architecture to use as our baseline and then we tweaked it for various optimizers. In this set up, we create four convolutional layers each with a max pooling and dropout layers. Additionally, our filters are 16, 64, 128 and 128.



```
1]: #Our optimal Model Archictecture
    RMSPropmodel = tf.keras.Sequential([
    layers.Conv2D(16, 3, padding='same',input_shape=(64, 64, 3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.25),

    layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)), #20x20
    layers.Dropout(0.25),

    layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)), #10x10
    layers.Dropout(0.25),

    layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 2)), #5x5
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),

    layers.Dense(2, activation='softmax')
    ])
```



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 64, 64, 16)        448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 64)        9280
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_8 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 1026
=================================================================
Total params: 1,281,282
Trainable params: 1,281,282
Non-trainable params: 0
```

Hyperparameter Tuning

We spent a long time tuning our model to try to increase the accuracy. The highest training accuracy we got was with the RMSProp optimizer at 91%, however this overfit to our model as the actual accuracy was at 68%.
Our major focus for parameters was on trying out different optimizers. To the right, one can see all the optimizers we tested.

In addition to that, we tested learning rates and momentums for various optimizers as well as the amount of layers in our model architecture.

For the most part, we kept our epoch size at 100 and our batch size at 70. If we were to do this again, we would have increased our batch size to 120 or more. We think this would have been better as the images were small and of low data quality and higher amount in each batch size would have helped.

Various Parameters we tested:
Optimizers:

- RMSProp
    - Learning Rates: .01,.001,.0001
- AdaDelta
- Adam
- SGD
- Nadam

Batch Size: 70, 100, 120
Epoch Size: 50, 100
Number of filters in model architecture: (16, 64, 128, 128) (32, 64, 128, 128)
The filter size: (3,3) or (5,5)
Number of layers in model architecture: 4, 5

Things that stayed consistent:

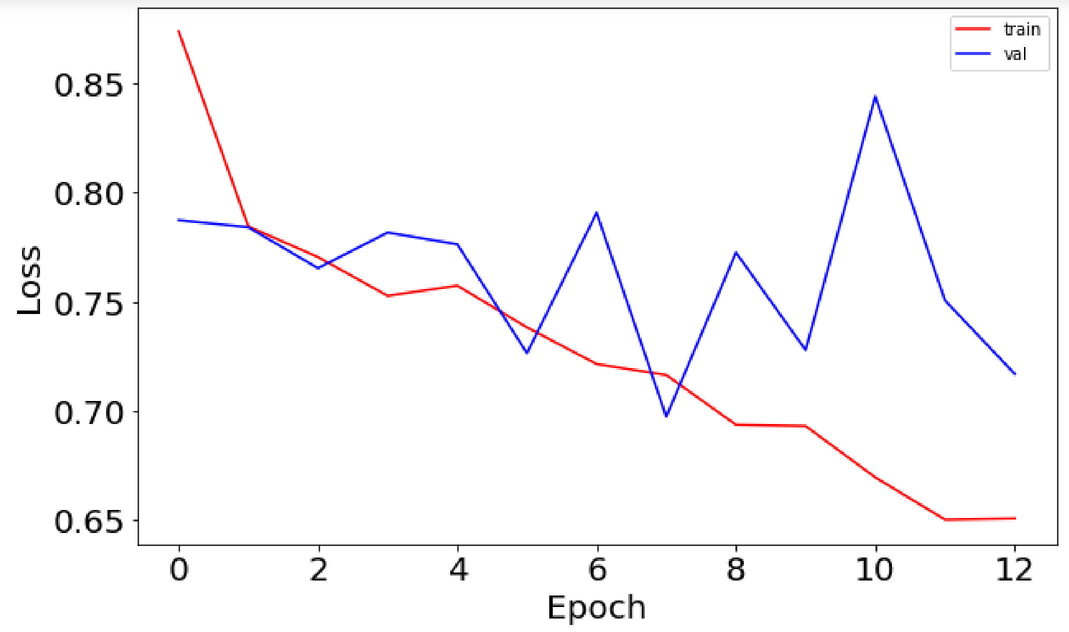
- Activation type
- Loss was always measured with binary cross entropy
- Metrics was always measured as accuracy

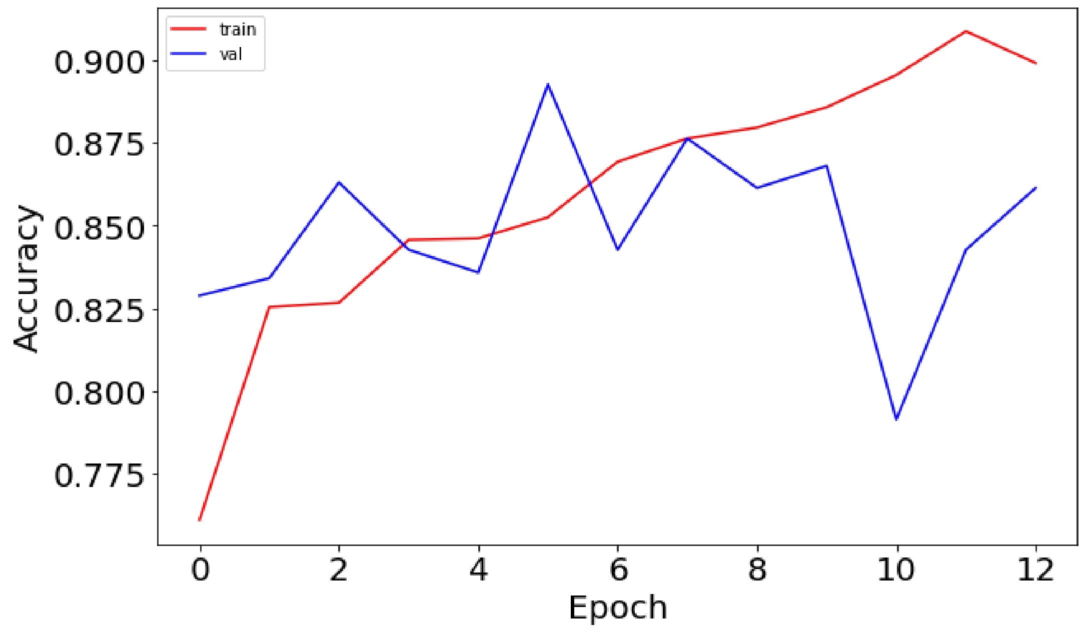
Experiment 3:

Parameters:

- RMS Prop with default parameters
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 34 Epochs

- Highest Training Accuracy: 91%
- Highest Validation Accuracy: 87 %
- Test Accuracy: 68%





Conclusion: This was our highest training accuracy but we can see that it overfit as our test accuracy is much lower.

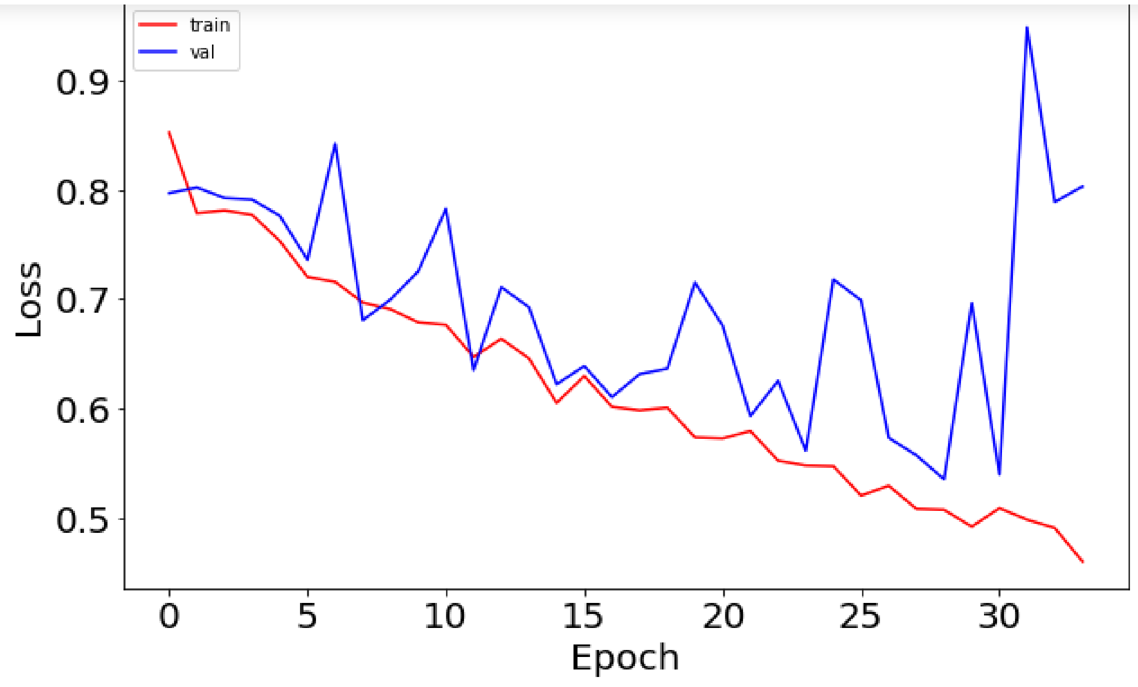
Experiment 4:

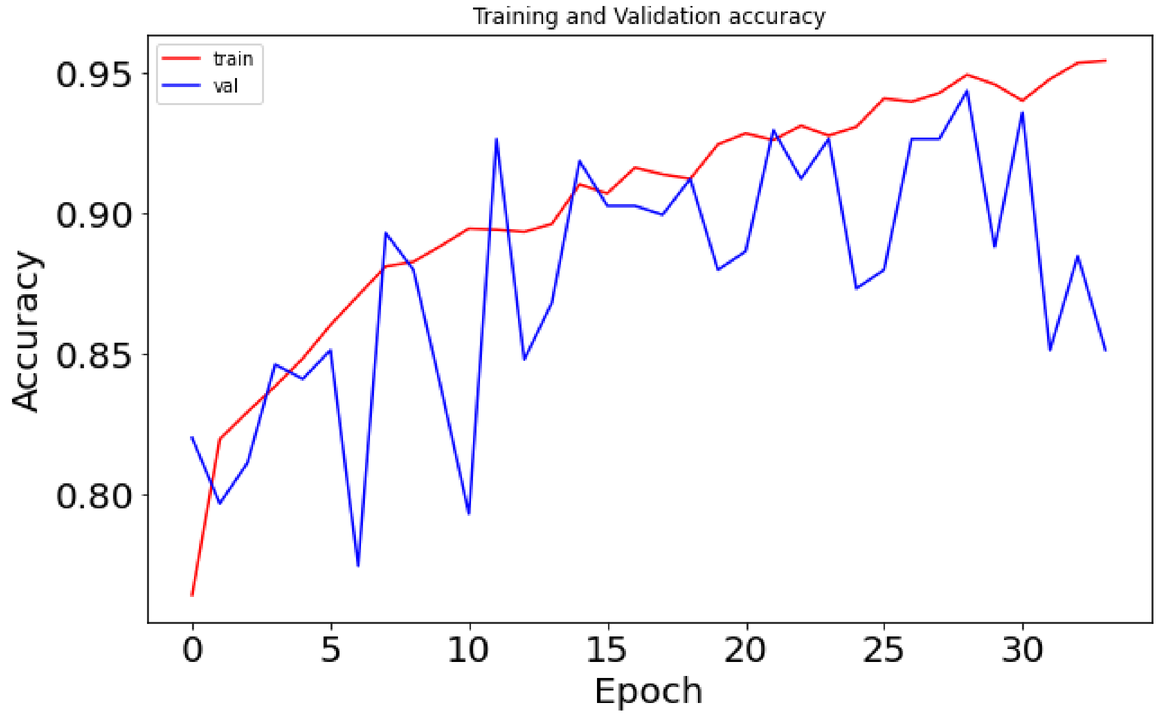
Parameters:

- RMS Prop
    - Learning rate at .001, rho: 0.9
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 13 Epochs

- Highest Training Accuracy: 80%
- Val Accuracy: 74 %
- Test Accuracy: 72%

Conclusion: Our test accuracy is better here, a smaller learning rate is good to use with RMS Prop



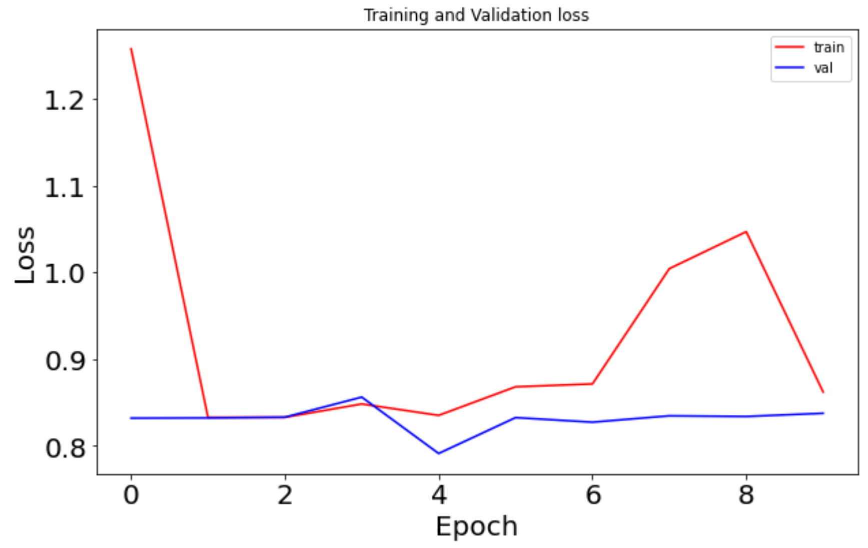


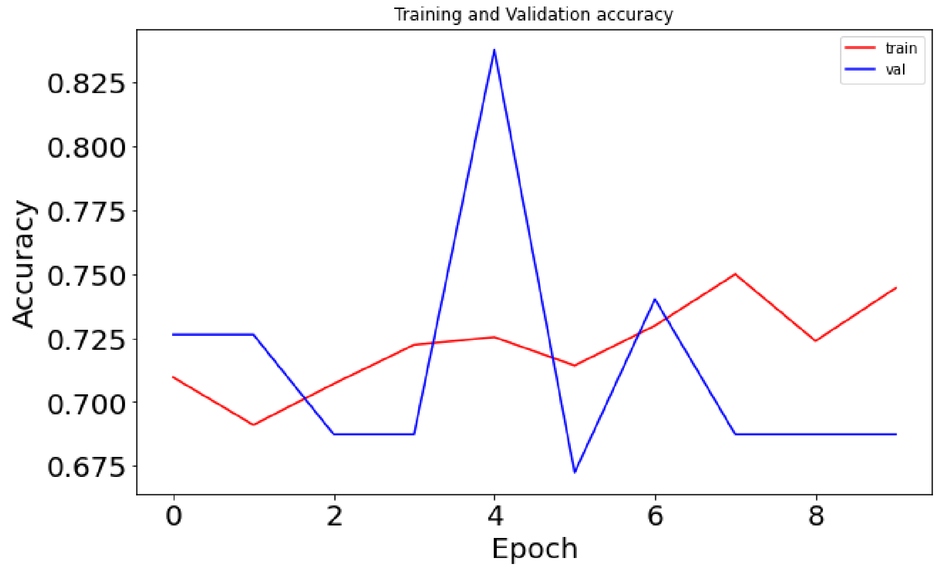
Experiment 5:

Parameters:

- Changed filters in Model architecture (16, 32, 32, 64, 128)
- RMS Prop
    - Learning rate at .01, rho: 0.9
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 10 Epochs

- Highest Training Accuracy: 55%
- Val Accuracy: 70 %
- Test Accuracy: 49%





Conclusion: Our accuracy overall was worse here. It does not appear that changing the filters or adding another layer helped. In general, adding layers does not help.

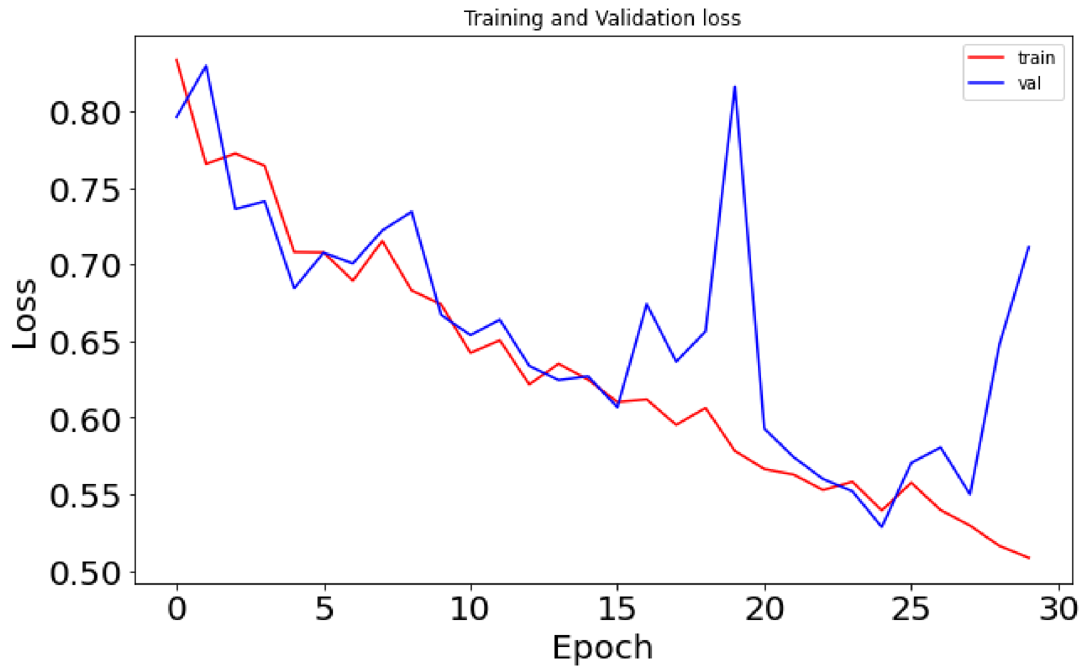
Experiment 6:

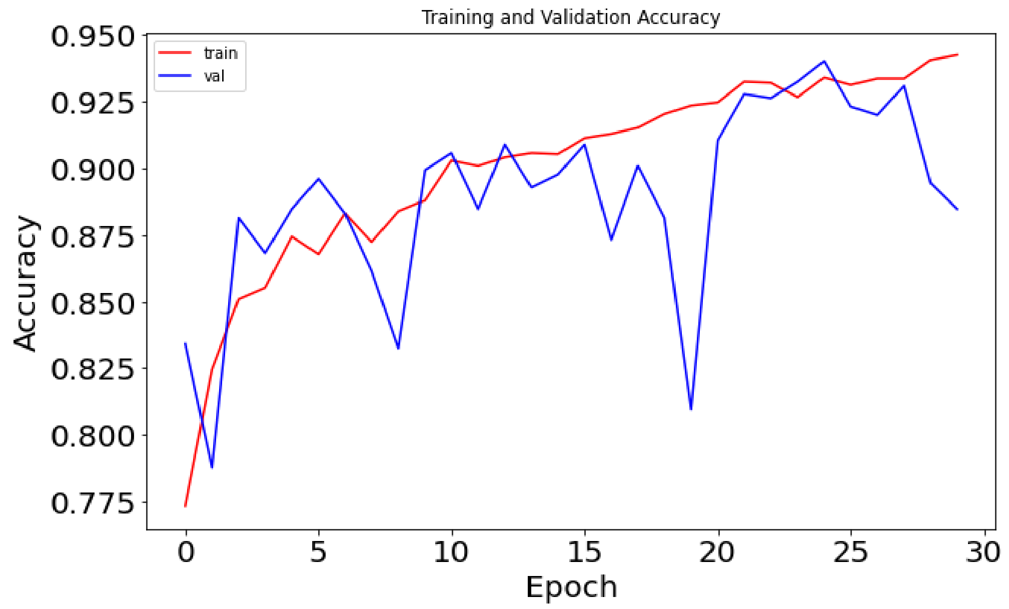
Parameters:

- Changed number of filters in Model architecture (16, 32, 32, 64, 128)
- Change filter shape to (5, 5)
- RMS Prop
  - Learning rate at .001
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 30 Epochs

- Highest Training Accuracy: 88%
- Val Accuracy: 86 %
- Test Accuracy: 75%

Conclusion: Our test accuracy is good here! Perhaps having a small learning rate and large filter size helps.





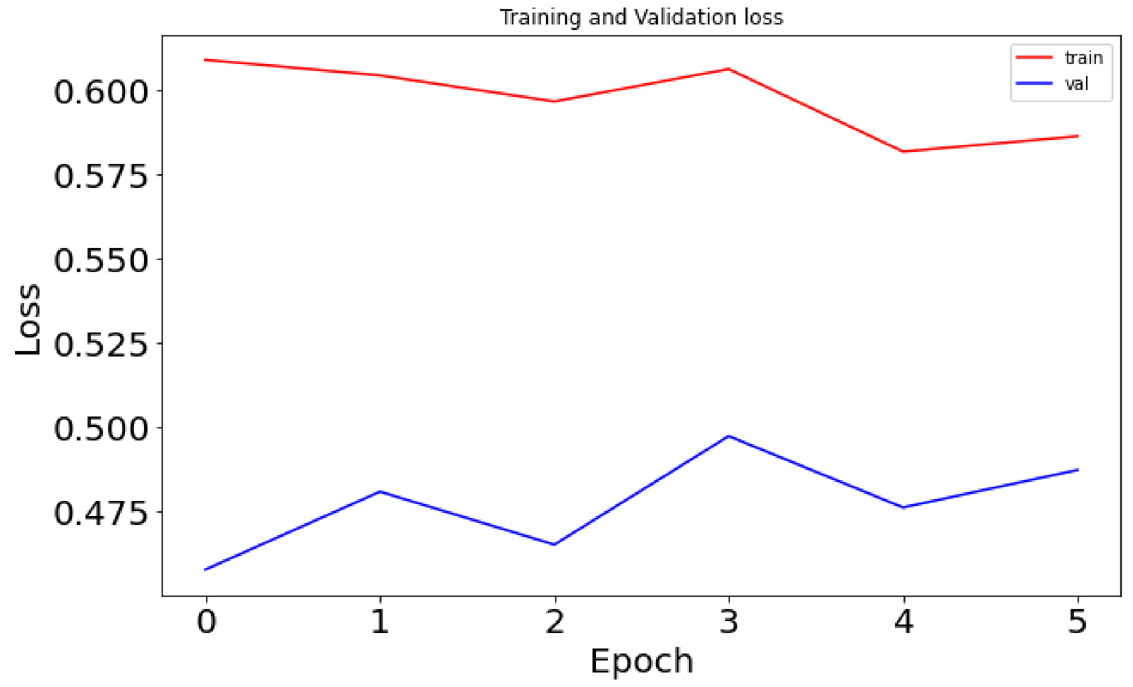
Experiment 7:

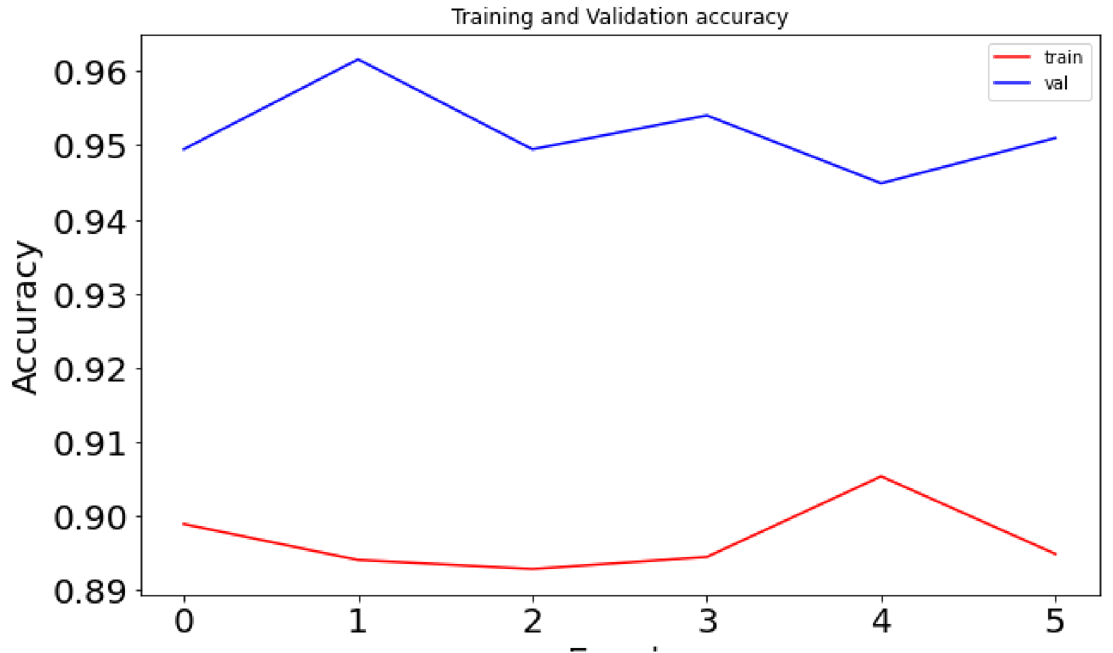
Parameters:

- ADAM with default parameters
- Batch Size: 128
- Epoch Size: 100
- Callbacks using Early Stopping after 6 Epochs

- Highest Training Accuracy: 80%
- Val Accuracy: 90%
- Test Accuracy: 79%

Conclusion: Our test accuracy is great here! This is the optimizer combined with an increase in batch size. But the graphs look a little off and I'm not sure why.



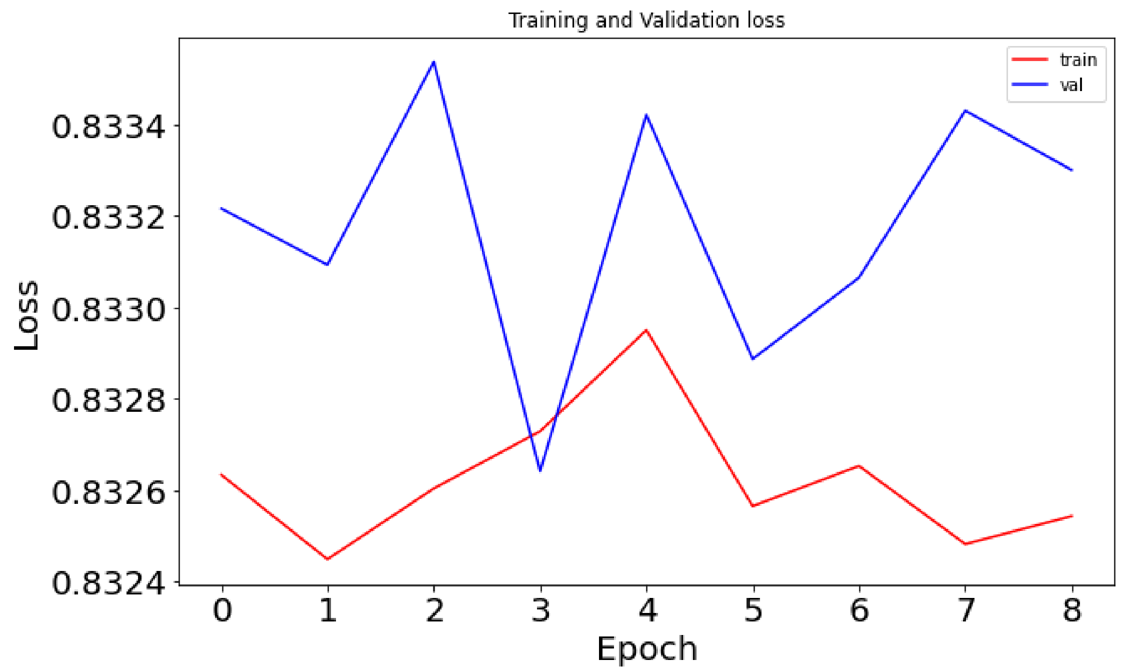


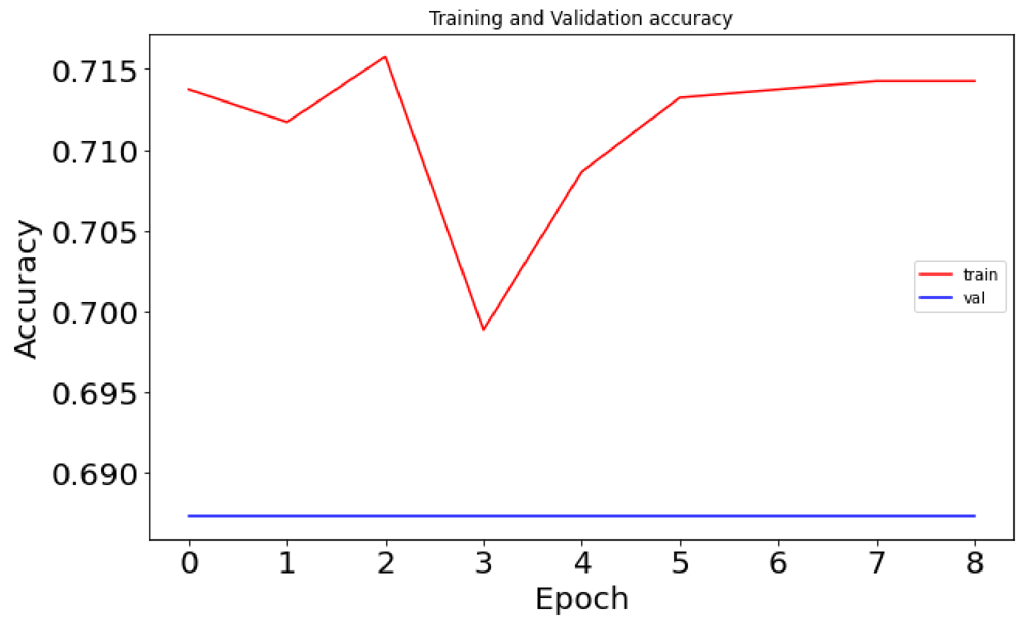
Experiment 8:

Parameters:

- ADAM
  - Learning rate at .01
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 30 Epochs

- Highest Training Accuracy: 51%
- Val Accuracy: 47%
- Test Accuracy: 49%





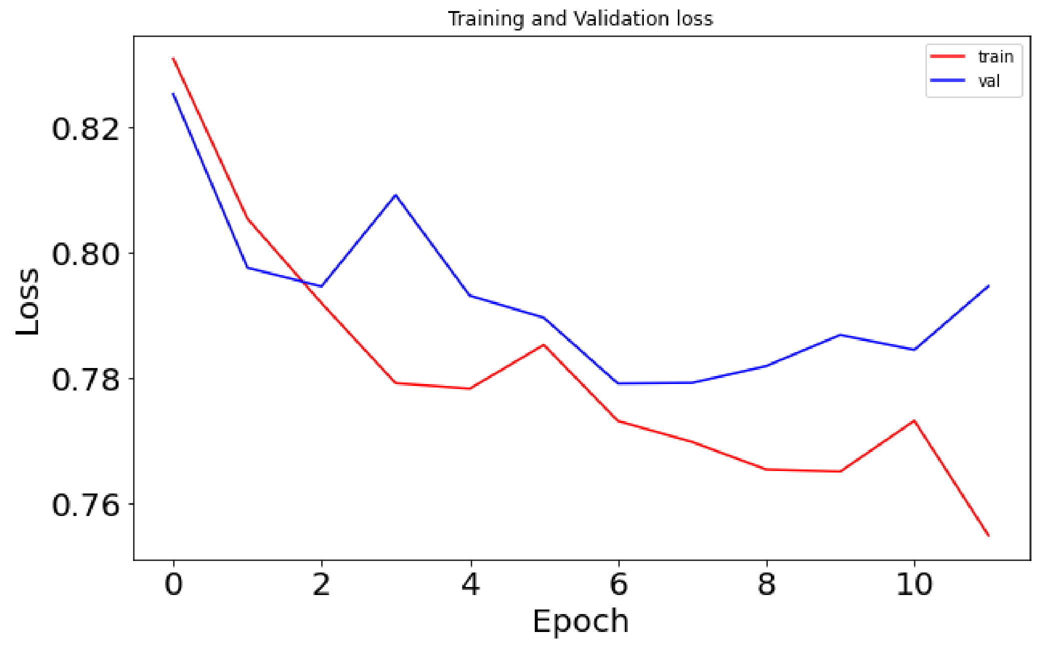
Conclusion: Our test accuracy is much worse here. Maybe has to do with the learning rate and batch size.

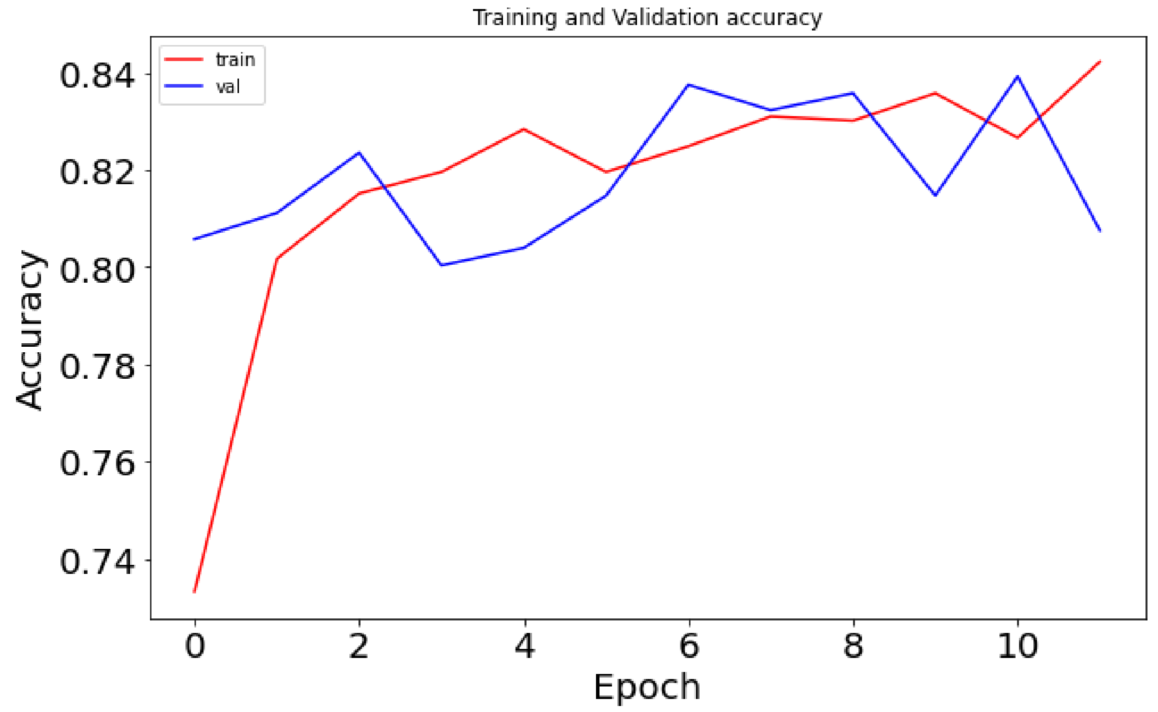
Experiment 9:

Parameters:

- SGD
    - Learning rate at .1
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 100 Epochs

- Highest Training Accuracy: 70%
- Val Accuracy: 70 %
- Test Accuracy: 71%





Conclusion: Acceptable test accuracy here but not our best. The model trained for all the epochs.

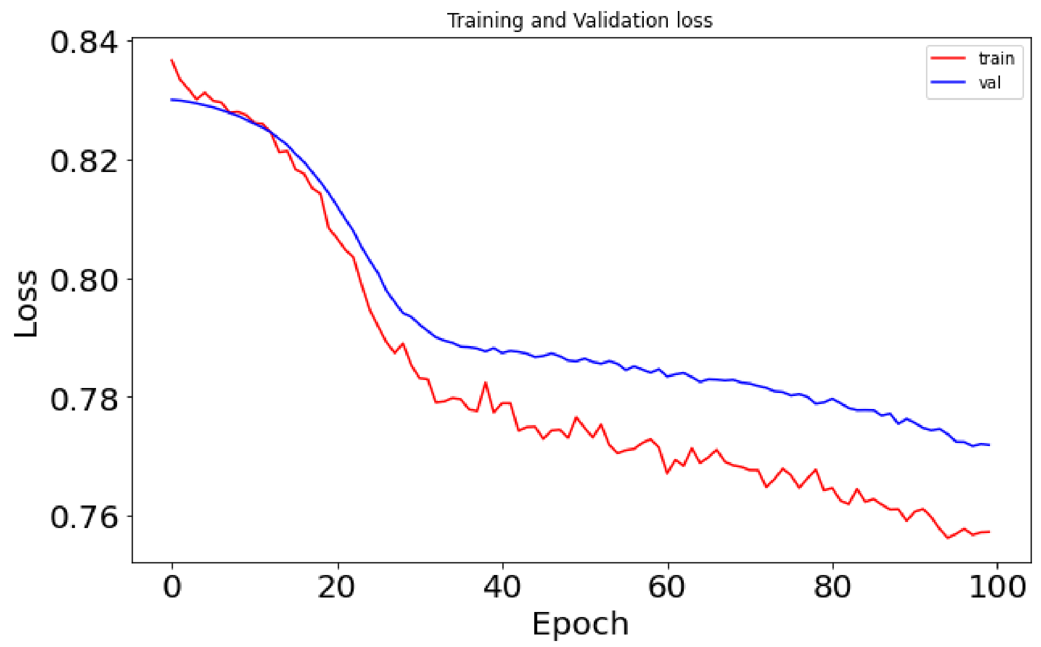
Experiment 10:

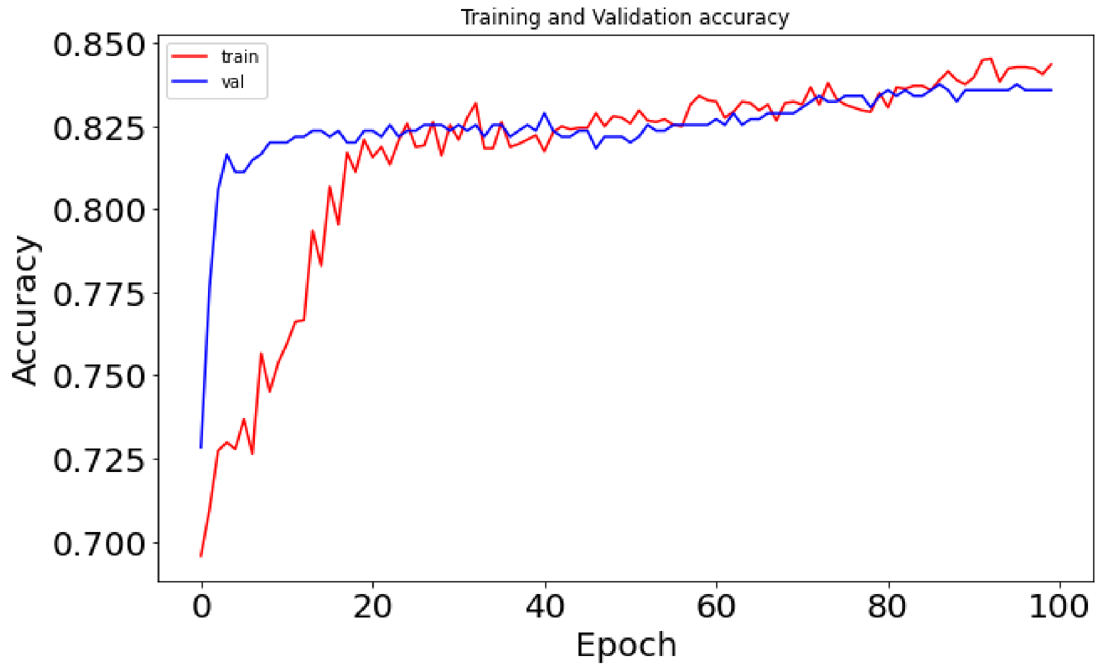
Parameters:

- SGD
  - Learning rate at .001, epochs=100, momentum=.8, decay rate
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 100 Epochs

- Highest Training Accuracy: 71%
- Val Accuracy: 69 %
- Test Accuracy: 71%

Conclusion: Not overfit! Loss and Accuracy oscillate a lot.





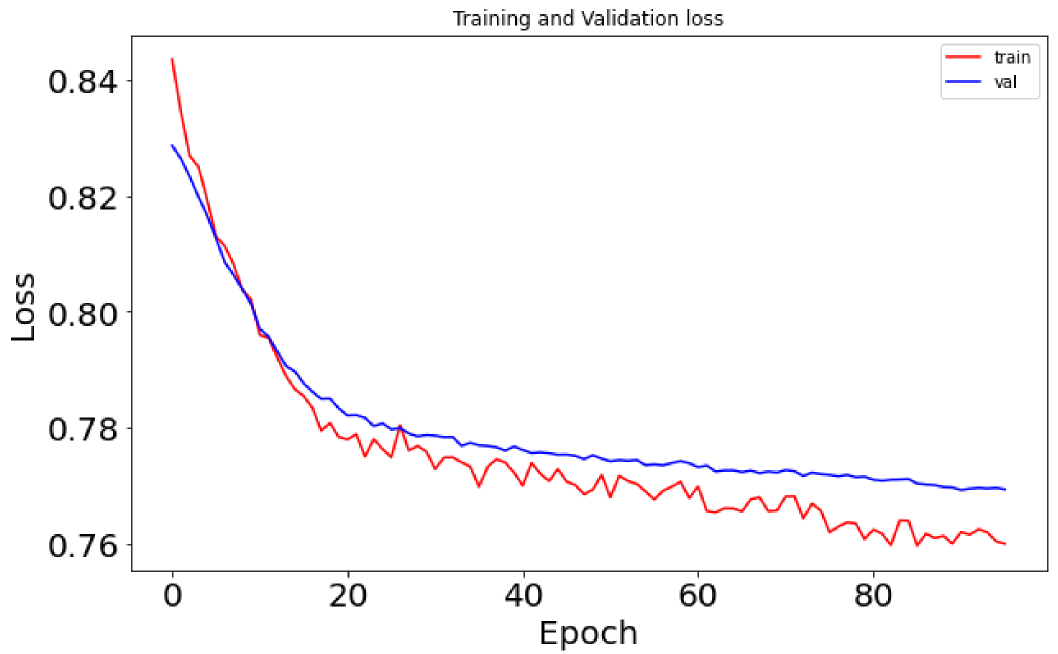
Experiment 11:

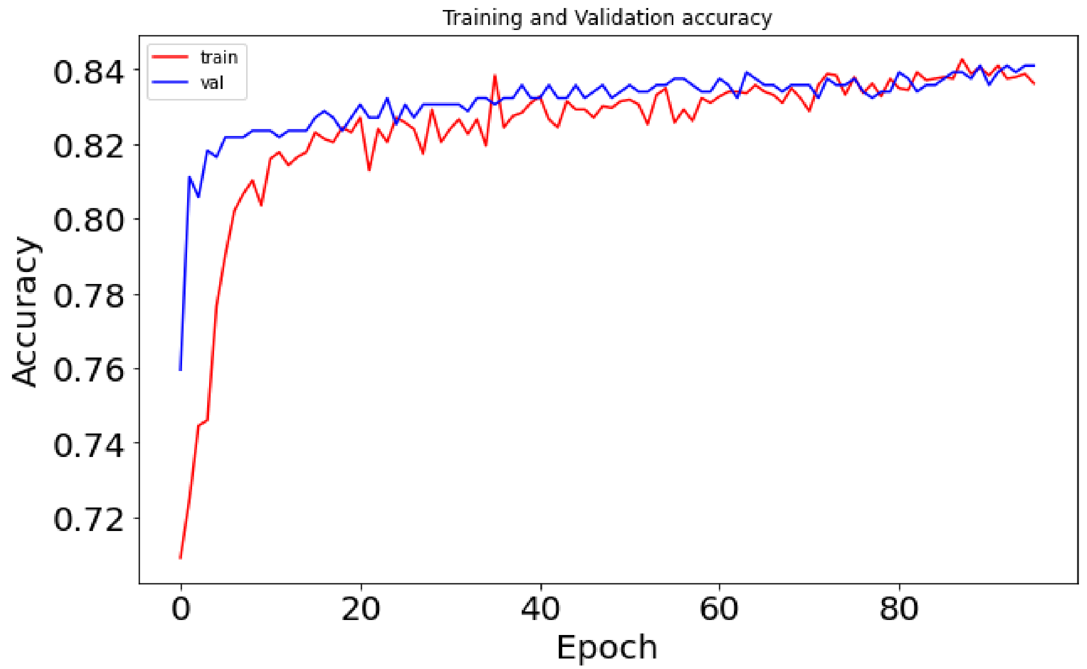
Parameters:

- AdaDelta
    - Learning rate at .01
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 96 Epochs

- Highest Training Accuracy: 69%
- Val Accuracy: 70 %
- Test Accuracy: 71%

Conclusion: Acceptable test accuracy here but not great. Needed to run for most Epochs.

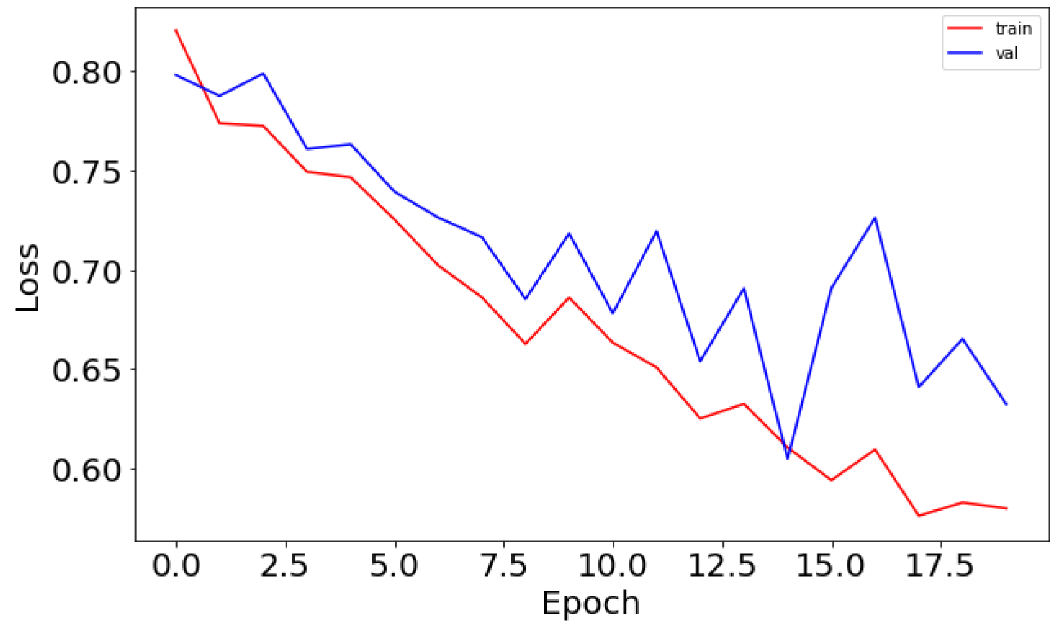


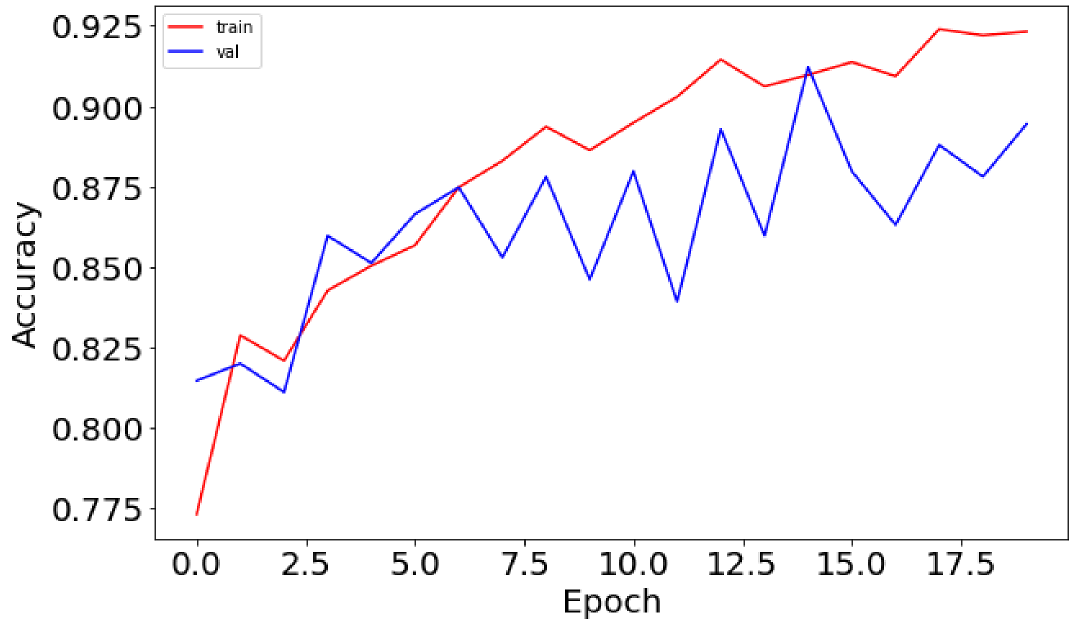


Experiment 12:

Parameters:
- Nadam
  - Learning rate at .001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
- Batch Size: 70
- Epoch Size: 100
- Callbacks using Early Stopping after 20 Epochs

- Highest Training Accuracy: 85%
- Val Accuracy: 80 %
- Test Accuracy: 78%





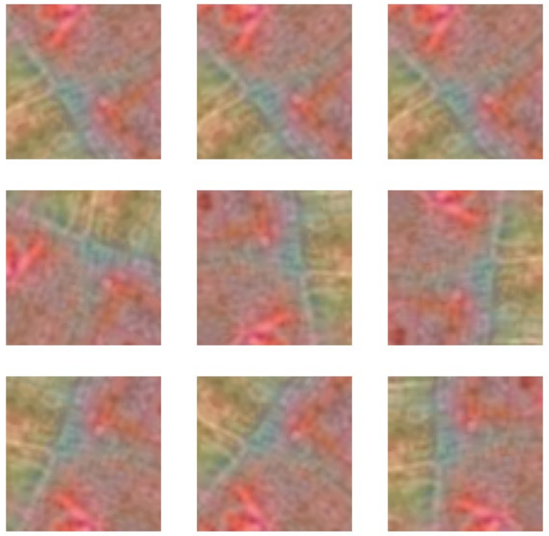
Conclusion: Nadam works great! This is one of our best test accuracies. Notice the small learning rate. For next time, make the batch size greater.

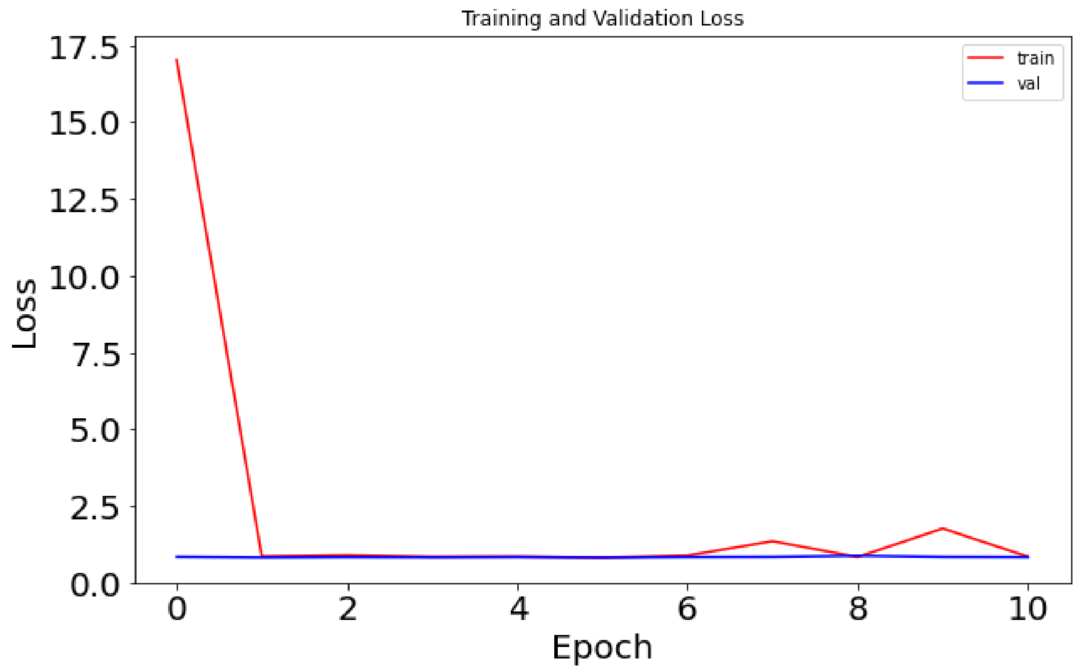
Experiment 13:

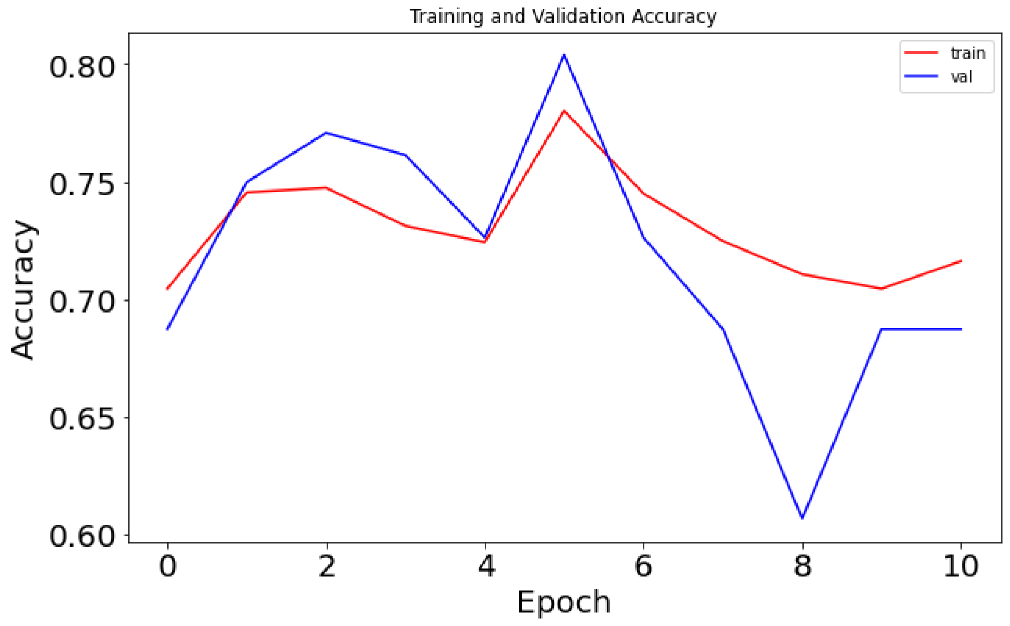
Parameters:

- Data Augmentation
    - Rescale
    - Flip and Rotate
- RMSProp
    - Learning rate at .01, rho at .9
- Batch Size: 100
- Epoch Size: 100
- Callbacks using Early Stopping after 11 Epochs

- Highest Training Accuracy: 51%
- Val Accuracy: 64 %
- Test Accuracy: 49%







Conclusion: Pictured above is one image flipped and rotated. However, this did not help increase our accuracy. Probably because low quality satellite images don't do as well with data augmentation

Conclusion

A Reminder of the Parameters tested:

Optimizers:

- RMSProp
  - Learning Rates: .01,.001,.0001
- AdaDelta
- Adam
  - Learning Rates: Default, .001
- SGD
- Nadam

Data Augmentation

Batch Size: 70, 100, 120

Epoch Size: 50, 100

Number of filters in model architecture: (16, 64, 128, 128)

(32, 64, 128, 128)

The filter size: (3,3) or (5,5)

Number of layers in model architecture: 4, 5

Best Recommendation:

We recommend using Adam with the default learning rate and large batch and epoch size with four layers. This got us the highest test accuracy at 79%.

Further Inquiry:

In further iterations of this project, we would have employed more transfer learning with already constructed models. Our research indicates that using CaffeNet or VGG16 with SVM would have gotten use higher results around 85%. If we had used GoogleNet with fine tuning our results could have reached 94.1% accuracy.

Works Cited

For more information on dataset origins, please see this link.

O. Korzh, G. Cook, T. Andersen and E. Serra, "Stacking approach for CNN transfer learning ensemble for remote sensing imagery," *2017 Intelligent Systems Conference (IntelliSys)*, 2017, pp. 599-608, doi: 10.1109/IntelliSys.2017.8324356.