Tova Perlman
Final Project Report
MUSA 650
5.9.21

Identification of Coffee Cultivation in Brazil

Section 1: Introduction:

Coffee cultivation is an important part of the economy in Brazil and sustains various communities across the country. Thus, a predictive model that can classify satellite images and whether they are good for coffee cultivation or not good for coffee cultivation is important to identifying key areas that can create new economic capital for farmers. Additionally, it will help future farmers avoid trying to cultivate coffee crops in areas not conducive to the crops and investigate different crop management techniques.

Section 2: Data:

The data set is a "composition of scenes taken by SPOT sensor in 2005 over four counties in the State of Minas Gerais, Brazil: Arceburgo, Guaranesia, Guaxupé and Monte Santo."[1]It is a balanced dataset with 4 folds of 600 images each and a fifth fold with 476 images. For the purposes of my analysis, I have combined all folds together into one dataset with 2786 images and then split this data into a 60% train data and 40% test data. Our dataset is not large, it is less than 3000 images which can affect accuracy. Each image has a pixel size of 64 x 64. There are three channels to the image from the Green, Red and Near Infrared Bands. This is important to note as most CNN's are trained on RGB images.[2] The images are also of relatively low data quality with blurriness and lots of noise. One way to solve the issue of a small number of pixels is to upscale the data and make the images larger. I chose not to do this because I knew it would be computationally expensive later on.

Three other challenges mentioned about this dataset are its intra-class variance, scenes captured from different plant ages, and the dataset contains "images with spectral distortions because of shadows".[3] For more information on dataset origins, please see this link.

The image data and label data was saved separately in the dataset folder so I had to merge them back together in my notebook. I then scaled the images by dividing by 255 so they would be between 0 and 1. I also did PCA for our first two classifiers. I did not set a number of components and decided to cut off the components at 1500 as I saw that 1500 components explained 93% of the variance.

Below, I've visualized some of the images. Figure 1 visualizes them using Montage RGB which changes the colors slightly. Figure 2 shows the images visualized in their true colors. As one can see, the images are relatively blurry and pixelated.

---

1 O. A. B. Penatti, K. Nogueira, J. A. dos Santos. Do Deep Features Generalize from Everyday Objects to Remote Sensing and Aerial Scenes Domains? In: EarthVision 2015, Boston. IEEE Computer Vision and Pattern Recognition Workshops, 2015.

2 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.

3 Ibid.

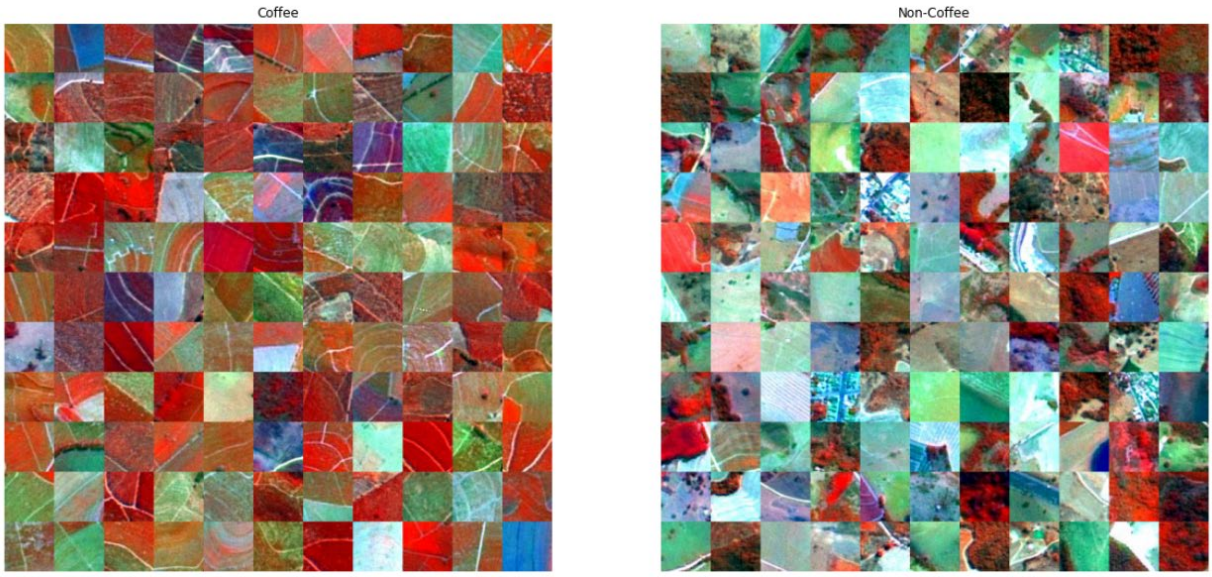


Figure 1

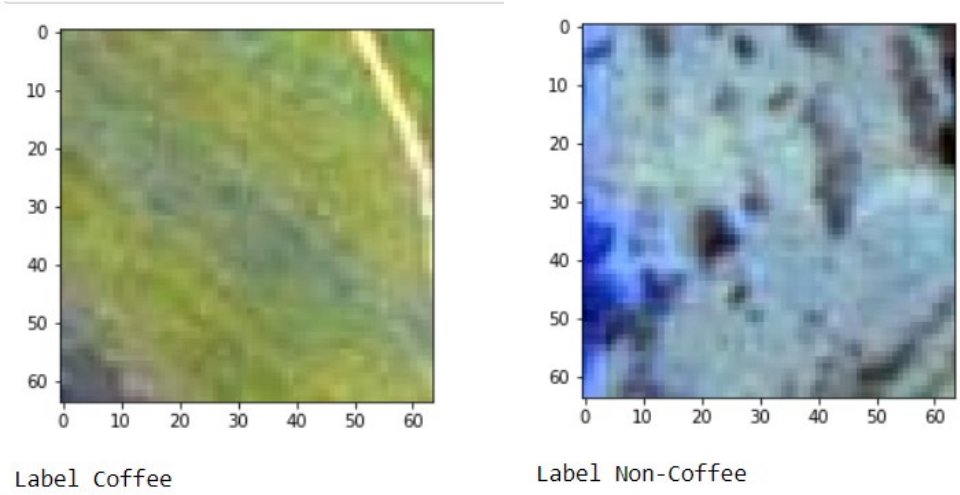


Figure 2

Section 3: Methods:

Our problem is one of binary classification, our images will either be a coffee image or a non coffee image. Our perfomance criterion throughout was accuracy as I wanted to be able to compare my classifier models with my CNN models. Because my image size was relatively small and my dataset was not very large, I was lucky as I was able to tune my CNN with many different parameters.

I first chose to test my images using an SVM classifier as these work well on binary classification problems. Using GridSearch CV with 3 fold cross validation, I found that the optimal hyperparameters for SVM were C: .01, 2 degrees and the Sigmoid Kernel. However, the accuracy from this classifier was only at 51% after outer cross validation was done. I tried to improve this accuracy with a Random Forest in which I found the optimal parameters were a

max depth of 5 and 200 estimators. However, I only improved the accuracy by 3%, getting 54% test accuracy.

Because these models weren't giving me good accuracy, I decided to train a Convolutional Neural Net Model. I chose this model because I knew it would be good at learning images and could have an adequate response to my binary classification problem. My loss type for the CNN was always measured with binary cross entropy. As mentioned before, the metric measure was accuracy.

Section 3.2: CNN Architecture

My initial CNN Architecture set up consisted of four layers with one convolutional layer, followed by a max pooling layer and then a drop out layer. At the end, there were two dense layers. I chose to make the initial number of filters at 16, 64, 128 and 128. I chose this because I wanted to the amount of filters to increase in size but to start out small. Additionally, I chose the kernel size to be (3,3). In fine tuning the model, I played with both the number of filters and the kernel size. The activation network stayed consistent throughout. Figure 3 below features the model architecture set up in more detail.

```
l]: #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')
])
```

Figure 3

Section 3.3: Hyper Parameter Tuning

Once I had the model architecture set up, I wanted to start parameter tuning with a focus on different types of optimizers. I tried six different optimizers: RMSProp, AdaDelta, Adam, AdaGrad, SGD, and Nadam. The main objective of these optimizers is to efficiently calculate the

parameters/weights of our model to minimize the loss function. Currently, Adam is the most popular optimizer used in neural networks today. Within each of these optimizers, I tried different learning rates and momentum values. Testing different learning rates is important because if a learning rate is too small, it will take a neural network too long to converge. If it is too big, the network will not converge at all. Momentum is important because it causes the optimization algorithm to resist changes in direction. Momentum slows the oscillation of our loss especially in high curvature surfaces. If the momentum rate is large, the learning rate should be smaller.[4]

Concurrent with the optimizers, I also experimented with different batch sizes and epoch sizes. For the most part, I remained consistent in choosing 70 for the batch size and 100 for the epoch size with an early stopping callback so the model rarely ran to 100 epochs. An epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. Therefore, we set it at a higher amount in addition to using the early stopping callback with a patience of 5 to monitor the validation loss and stop if it was only oscillating. The smaller the batch size, typically the faster the network trains. However, having a too small batch size could mean that the network is missing vital information to train. Within our fit call, we also chose to have a validation split at .2 to measure our accuracy before we evaluated on our test set.

Lastly, I tried to do Data Augmentation as a preprocessing step to increase the "amount" of data and to improve accuracy. For Data Augmentation, I resized the images to a lareger size and then also did a random flip and random rotation to the images. Figure 4 shows one image and how it has been data augmented.

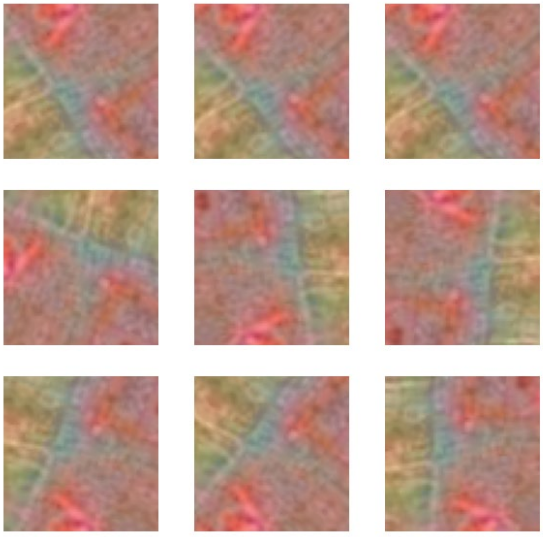


Figure 4

Section 4: Results

The table below lists each combination of various hyperparameters I tried as I was finetuning my network. Figures 5, 6 and 7 also feature the Training and Validation Loss and Accuracy Graphs for the top three performing optimizers: Adam, Nadam, and RMS Prop. Each of these graphs, should show the loss decreasing and then plateauing and then the accuracy

[4] https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0

increasing. For accuracy, a low validation curve in contrast to training accuracy shows strong overfitting.

Combination of Hyper Parameters	Train Accuracy	Validation Accuracy	Test Accuracy
• RMS Prop with default parameters • Batch Size: 70 • Epoch Size: 100	80%	79%	72%
• RMS Prop • Learning rate at .001, rho: 0.9 • Batch Size: 70 • Epoch Size: 100 • Callbacks using Early Stopping after 13 Epochs	91%	87%	68%
• 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	55%	70%	49%
• Changed number of filters in Model architecture (16, 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	88%	86%	75%
• Changed Model Architecture to one layer with 32 filters, flatten, dense layer, kernel initializer of 'he_uniform' • ADAM with default parameters • Batch Size: 128 • Epoch Size: 100	80%	90%	79%

• Callbacks using Early Stopping after 6 Epochs			
• ADAM • Learning rate at .01 • Batch Size: 70 • Epoch Size: 100 • Callbacks using Early Stopping after 30 Epochs	51%	47%	49%
• SGD • Learning rate at .1 • Batch Size: 70 • Epoch Size: 100 • Callbacks using Early Stopping after 100 Epochs	70%	70%	71%
• SGD • Learning rate at .001, epochs=100, momentum=.8, decay rate • Batch Size: 70 • Epoch Size: 100 • Callbacks using Early Stopping after 100 Epochs	71%	69%	71%
• AdaGrad Optimizer • Learning rate of .001 • iIitial_accumulator_value=0.1 • Batch Size: 100 • Epoch Size: 50 •	69%	68%	70%
• AdaDelta • Learning rate at .01 • Batch Size: 70 • Epoch Size: 100 • Callbacks using Early Stopping after 96 Epochs	69%	70%	71%
• 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	85%	80%	78%
• Data Augmentation	51%	64%	49%

• 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			

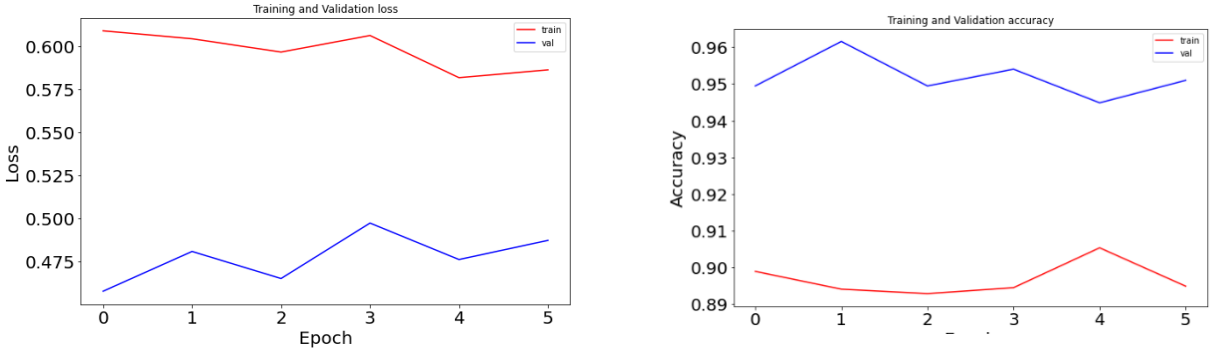


Figure 5: Adam Loss and Accuracy Graphs

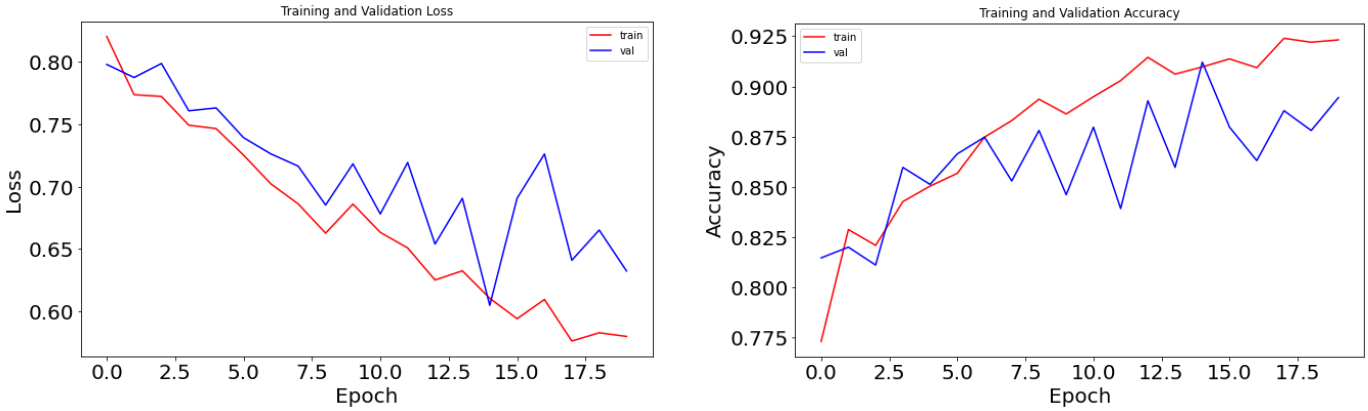
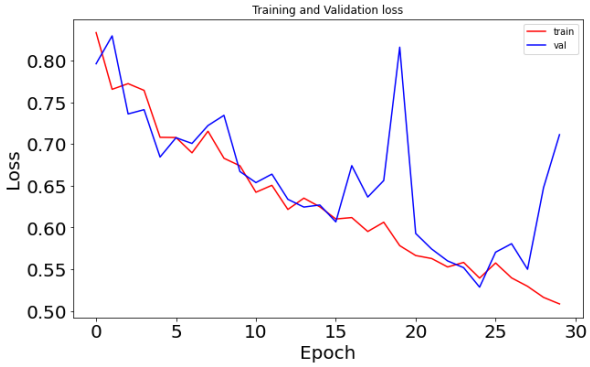


Figure 6: Nadam Loss and Accuracy Graphs



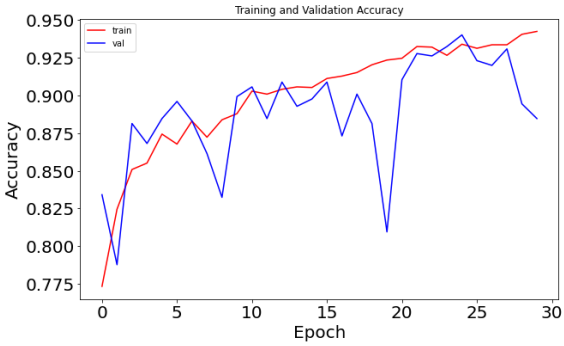


Figure 7: RMS Prop Loss and Accuracy Graphs

Section 5: Discussion

Within CNN, our best accuracy score was with the ADAM optimizer at 79% test accuracy. This result was closely followed by Nadam at 78% and RMSprop at 76%. This accuracy result is significantly higher than both the SVM and Random Forest scores at 57% and 59%. However, I would have liked to get the accuracy higher to the mid 80's or low 90's. It was frustrating when the training accuracy was high but that the network never performed as well on the test set. One challenge to this accuracy score was the data itself. The images were small and blurry. They were abstract landscape images and did not provide as many indicators at difference as other image datasets might have. One thing I could have done to improve this would have been to up-sample the images to a larger pixel size. However, this is computationally expensive and I did not have the timeline to spend on this. I also should have investigated the color channels of the images and created a better understanding around optimizing the accuracy of a Green, Red and Near-Infrared channel dataset. This was one limitation of my modeling.

Another reason why the Adam optimizer experiment worked the best was because the model architecture was less complex than the main one I used and because I deviated in using a bigger batch size. Because I was trying to maintain constant variables throughout my experiments, I chose somewhat arbitrarily at the beginning, to have four layers in my model architecture. Now, I see that the modeling could have benefited from fewer layers and the accuracy might have increased. I also chose 70 as a constant batch size but as I got deeper in my work I realized that a more optimal batch size should be larger, somewhere around 120 perhaps. This is because 70 is too small of a number to train on. The epoch size of 100 was good and the early stopping callback often stopped it before that.

Approaches that didn't work well were the SGD optimizer and Data Augmentation. Both did not produce high test accuracy. Data Augmentation does not work that well on small satellite images. SGD is a less powerful optimizer than Adam, Nadam, or RMSProp. Interestingly, when I used Adam on the standard architecture I had been using, it performed way worse. Thus, a large part of the high accuracy was a smaller layer network. I wish I had realized this earlier and changed my other experiments to reflect this.

In the future, I would like to experiment more with the learning rates for both Adam, Nadam and RMSProp as I identified those as the best performing optimizers. Importantly, none of these optimizers overfit on the test set. Unfortunately, I ran out of time and found that there

was a large number of combinations that could have been tried. I also wish I had been able to re-visualize some of the datasets to check how the labels were being predicted for. In the future, I will work more on evaluating my results.

For future inquiry, I hope to employ transfer learning with pre-trained networks. As I was researching for this report, I found an article that was able to achieve higher accuracy using pre-trained models. With finetuning in GoogleNet, the researchers were able to achieve 94% accuracy. Using CaffeNet and SVM, without fine tuning, they were able to achieve 85%.[5] In the future, I hope to continue to improve on my accuracy scores with these pre-trained models.

Ultimately, I was able to solve the problem I set out to: label coffee and non coffee images correctly. I was able to do this with 79% accuracy. Hopefully, this approach and model can be used in helping the Brazilian government and farmers determine the optimal place to create more coffee plantations and cultivate crops.

Works Cited:

O. A. B. Penatti, K. Nogueira, J. A. dos Santos. Do Deep Features Generalize from Everyday Objects to Remote Sensing and Aerial Scenes Domains? In: EarthVision 2015, Boston. IEEE Computer Vision and Pattern Recognition Workshops, 2015.

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.

Stewart, Matthew. “Neural Network Optimization”, Towards Data Science. https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0

[5] 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.