Identifying Fruits with Machine Learning

Martin Acob
Mathematics and Computer Science
California State University, Sacramento
Sacramento, California
martinkarlacob@csus.edu

Aaron Cheung
Computer Science
California State University, Sacramento
Sacramento, California
aaron97c@gmail.com

ABSTRACT

A few years ago, the consensus within the field of computer science was thought it would be impossible for a computer to classify between fruits. Given the capabilities of machine learning, we are capable of doing this task. Machine learning models can always be improved in its ability to predict. This project was created by utilizing a dataset from kaggle.com, a website for data scientists and machine learners. The dataset in question was "Fruits 360", which consisted of over 80,000 images of fruits and vegetables at a multitude of angles. They used convolutional neural networks to classify 120 different types of vegetable and fruits. Our take on this project is to use four different fruits (apple, grape, pear, and tomato), create different convolutional networks, and apply transfer learning because of the decreased amount of inputs. Within the projected we implemented 4 different models, which were 3 convolutional neural networks and 1 transfer learning. The three CNNs: a CNN from mini project 3, the best CNN model found through various parameter tuning, and our variation on their CNN model. They measured their performance through accuracy and for RGB images they were able to get an accuracy of 95.89%. Our best model was our transfer-learning model with an accuracy of 98.29%.

1 Introduction

There are different takes on machine learning models and not one machine-learning model can solve all different. They are shaped depending on what data you are handling and the desired output. These models are improved through data preprocessing and parameter tuning. The data we are using is from kaggle.com through a problem called "Fruits 360." The problem tries to classify 120 different types of fruits and vegetables, which their data set is contained 82,213 images of fruits and vegetables with different rotations for each. For our project, we took four different fruits out of 120 different types of fruits and vegetables from the original dataset making our dataset to 12,275 images. Then we created three different convolutional neural networks. First, we decided to use the best convolutional neural network from a previous project to see how it would perform on this dataset. Second, among different takes on parameter tuning we are showing the best model. Third, we attempted recreating the neural network defined on the paper. Since we shortened the original dataset, we found it appropriate to apply transfer learning. We worked through the project side by side from finding the data, data preprocessing, parameter tuning, writing the report, and PowerPoint. Section 2 will define the problem in more depth. Section 3 will go into detail the builds of the convolutional neural networks and transfer learning. For example, which layers we used, how each layers are defined, their activation functions, and how we applied transfer learning. Section 4 will cover the methodology and the results of our experiment. Section 5 will talk about where we based our project on. Section 6 will be our conclusion. Section 7 will explain how we divided the work. Section 8 will talk about what we learned in this project.

2 Problem Formulation

The problem we were addressing was having a neural network successfully identify a fruit from an image. We ended up utilizing a subset of the original dataset considering that it was too big; in other words, we cut down the original number of 120 different categories to only 4 different categories. We did this because had we stuck with the 120 different categories of fruits and vegetables, the amount of time we had in order to do this project would not be able to accommodate for it. In any case, the 4 different categories of fruit that we chose out of the 120 original categories of fruits and vegetables were Apple, Grape, Pear, and Tomato, which sums up to be 12,275 out of 82,213 images of fruits and vegetables from the original dataset.

Since we are utilizing a subset of the original data, we planned on using a convolutional neural network (CNN) used in a previous mini project and creating our own CNN for this project as an attempt to follow an ideal formula for how the layers should be ordered. We also created another CNN that attempts to recreate the neural network that the original authors of a paper who have created this dataset. We also planned to utilize transfer learning with a model from mini project 3 called VGG16.

3 System/Algorithm Design

3.1 System Architecture

For this project, we are implementing two types of machine learning models, convolutional neural networks (CNN) and transfer learning. Convolutional neural networks are a kind of deep neural networks that are used image classification and object detection. Transfer learning is a concept in machine learning, where we take a pre-existing trained neural network and use it to train and test our data. The reasoning for this is at times there isn't enough data collected to completely build an effective, so we take an existing neural network that was trained on a data that shares the somewhat similar characteristics. For this project, we are using the CNN model VGG16 that were trained on the CIFAR dataset. For all the models in this project, Convolution2D layer, MaxPooling, Flatten, and Dense were used. For activation functions, we used tanh, Sigmoid, and ReLU. All these different models were compared on its capability of predictability.

3.2 CNN1

3.2.1 Model Build

```
cnn_1 = Sequential()

cnn_1.add(Conv2D(128, kernel_size=(4, 6), strides=(2, 2),
                 activation='relu',
                 input_shape=(100, 100, 3)))

cnn_1.add(MaxPooling2D(pool_size=(1, 1)))

cnn_1.add(Conv2D(90, kernel_size=(6, 4), strides=(2, 2),
                 activation='sigmoid'))

cnn_1.add(MaxPooling2D(pool_size=(1, 1)))


cnn_1.add(Flatten())
cnn_1.add(Dense(128))
cnn_1.add(Activation("sigmoid"))
cnn_1.add(Dense(256, activation="tanh"))
# cnn_1.add(Dropout(0.5))
cnn_1.add(Dense(128, activation="sigmoid"))
# cnn_1.add(Dropout(0.5))
cnn_1.add(Dense(64, activation="relu"))
# cnn_1.add(Dropout(0.5))
cnn_1.add(Dense(32, activation="tanh"))
cnn_1.add(Dropout(0.25))
cnn_1.add(Dense(4, activation="softmax"))
```

Figure 1: Python Code for the first CNN model

3.2.2 Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 49, 48, 128)       9344
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 48, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 22, 23, 90)        276570
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 23, 90)        0
_________________________________________________________________
flatten (Flatten)            (None, 45540)             0
_________________________________________________________________
dense (Dense)                (None, 128)               5829248
_________________________________________________________________
activation (Activation)      (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 132
=================================================================
Total params: 6,191,550
Trainable params: 6,191,550
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Output of first model summary

3.2.2 Model Training

```
Train on 9118 samples, validate on 3157 samples
Epoch 1/1000
9118/9118 - 325s - loss: 1.3965 - val_loss: 1.3891
Epoch 2/1000
9118/9118 - 319s - loss: 1.3853 - val_loss: 1.3922
Epoch 3/1000
9118/9118 - 310s - loss: 1.3856 - val_loss: 1.3883
Epoch 00003: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f8c10065eb8>
```

Figure 3: Output of first model training with epochs

3.3 CNN2

3.3.1 Model Build

```
cnn_2 = Sequential()

cnn_2.add(Conv2D(128, kernel_size=(8, 8), strides=(2, 2),
                 activation='relu',
                 input_shape=(100, 100, 3)))

cnn_2.add(Conv2D(90, kernel_size=(4, 4), strides=(2, 2),
                 activation='relu'))

cnn_2.add(MaxPooling2D(pool_size=(1, 1)))
cnn_2.add(MaxPooling2D(pool_size=(1, 1)))

cnn_2.add(Dropout(0.125))

cnn_2.add(Flatten())
cnn_2.add(Activation("relu"))
cnn_2.add(Dense(150, activation="relu"))
cnn_2.add(Dense(100))
cnn_2.add(Dense(125, activation="relu"))
cnn_2.add(Activation('relu'))
cnn_2.add(Dense(4, activation="softmax"))
```

Figure 4: Python Code for the second CNN model

3.3.2 Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 47, 47, 128)       24704
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 90)        184410
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 22, 22, 90)        0
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 22, 22, 90)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 22, 22, 90)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 43560)             0
_________________________________________________________________
activation_3 (Activation)    (None, 43560)             0
_________________________________________________________________
dense_10 (Dense)             (None, 150)               6534150
_________________________________________________________________
dense_11 (Dense)             (None, 100)               15100
_________________________________________________________________
dense_12 (Dense)             (None, 125)               12625
_________________________________________________________________
activation_4 (Activation)    (None, 125)               0
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 504
=================================================================
Total params: 6,771,493
Trainable params: 6,771,493
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Output of second model summary

3.3.3 Model Training

```
Train on 9118 samples, validate on 3157 samples
Epoch 1/1000
9118/9118 - 253s - loss: 0.1339 - val_loss: 0.4309
Epoch 2/1000
9118/9118 - 253s - loss: 0.0658 - val_loss: 0.3851
Epoch 3/1000
9118/9118 - 254s - loss: 0.0050 - val_loss: 0.2520
Epoch 4/1000
9118/9118 - 251s - loss: 7.3128e-04 - val_loss: 0.3268
Epoch 5/1000
9118/9118 - 253s - loss: 5.4098e-05 - val_loss: 0.3699
Epoch 00005: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f8c10065cc0>
```

Figure 6: Output of second model training with epochs

3.4 CNN3

3.4.1 Model Build

```
cnn_3 = Sequential()

cnn_3.add(Conv2D(16, kernel_size=(4, 4),
                 activation='relu',
                 input_shape=(100, 100, 3)))
cnn_3.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 1)))

cnn_3.add(Conv2D(32, kernel_size=(4, 4),
                 activation='relu'))
cnn_3.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 1)))

cnn_3.add(Conv2D(64, kernel_size=(4, 4),
                 activation='relu'))
cnn_3.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 1)))

cnn_3.add(Conv2D(128, kernel_size=(4, 4),
                 activation='relu'))
cnn_3.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 1)))

cnn_3.add(Flatten())

cnn_3.add(Dense(49))
cnn_3.add(Dense(256))
cnn_3.add(Dense(4, activation="softmax"))
```

Figure 7: Python Code for the third CNN model

3.4.2 Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 97, 97, 16)        784
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 48, 96, 16)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 45, 93, 32)        8224
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 22, 92, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 19, 89, 64)        32832
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 9, 88, 64)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 6, 85, 128)        131200
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 3, 84, 128)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 32256)             0
_________________________________________________________________
dense_14 (Dense)             (None, 49)                1580593
_________________________________________________________________
dense_15 (Dense)             (None, 256)               12800
_________________________________________________________________
dense_16 (Dense)             (None, 4)                 1028
=================================================================
Total params: 1,767,461
Trainable params: 1,767,461
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Output of third model summary

3.4.3 Model Training

```
Train on 9118 samples, validate on 3157 samples
Epoch 1/1000
9118/9118 - 320s - loss: 0.1457 - val_loss: 0.5024
Epoch 2/1000
9118/9118 - 321s - loss: 0.0083 - val_loss: 0.9838
Epoch 3/1000
9118/9118 - 315s - loss: 0.0288 - val_loss: 0.5105
Epoch 00003: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f8c094dfeb8>
```

Figure 9: Output of third model training with epochs

3.5 Transfer Learning

3.5.1 Model Build

```
model.add(Flatten())


# Add some "Dense" layers here, including output layer


model.add(Dense(40))
model.add(Dense(20, activation='relu'))
model.add(Dense(30))
model.add(Dense(5, activation='sigmoid'))

model.add(Dense(4, activation='softmax')) # output layer

model.compile(loss="categorical_crossentropy", optimizer="adam")

monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=2, verbose=2, mode='auto')

model.fit(new_x_train, y_train, validation_data=(new_x_test, y_test), callbacks=[monitor], verbose=2, epoc
```

Figure 10: Python Code for the Transfer Learning model

3.5.2 Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Figure 11: Model Summary of Transfer Learning model

3.5.3 Model Training

```
Train on 9118 samples, validate on 3157 samples
Epoch 1/1000
9118/9118 - 500s - loss: 0.9463 - val_loss: 0.7251
Epoch 2/1000
9118/9118 - 502s - loss: 0.5073 - val_loss: 0.5233
Epoch 3/1000
9118/9118 - 503s - loss: 0.3469 - val_loss: 0.3327
Epoch 4/1000
9118/9118 - 503s - loss: 0.2292 - val_loss: 0.2367
Epoch 5/1000
9118/9118 - 501s - loss: 0.1584 - val_loss: 0.1767
Epoch 6/1000
9118/9118 - 500s - loss: 0.1158 - val_loss: 0.1419
Epoch 7/1000
9118/9118 - 501s - loss: 0.0879 - val_loss: 0.1194
Epoch 8/1000
9118/9118 - 500s - loss: 0.0685 - val_loss: 0.0958
Epoch 9/1000
9118/9118 - 500s - loss: 0.0545 - val_loss: 0.0838
Epoch 10/1000
9118/9118 - 501s - loss: 0.0441 - val_loss: 0.0790
Epoch 11/1000
9118/9118 - 499s - loss: 0.0361 - val_loss: 0.0652
Epoch 12/1000
9118/9118 - 499s - loss: 0.0299 - val_loss: 0.0596
Epoch 13/1000
9118/9118 - 498s - loss: 0.0250 - val_loss: 0.0579
Epoch 14/1000
9118/9118 - 499s - loss: 0.0210 - val_loss: 0.0554
Epoch 15/1000
9118/9118 - 499s - loss: 0.0178 - val_loss: 0.0459
Epoch 16/1000
9118/9118 - 498s - loss: 0.0151 - val_loss: 0.0445
Epoch 17/1000
9118/9118 - 496s - loss: 0.0129 - val_loss: 0.0444
Epoch 18/1000
9118/9118 - 496s - loss: 0.0110 - val_loss: 0.0329
Epoch 19/1000
9118/9118 - 496s - loss: 0.0094 - val_loss: 0.0521
Epoch 20/1000
9118/9118 - 496s - loss: 0.0081 - val_loss: 0.0303
Epoch 21/1000
9118/9118 - 499s - loss: 0.0070 - val_loss: 0.0384
Epoch 22/1000
9118/9118 - 498s - loss: 0.0058 - val_loss: 0.0571
Epoch 00022: early stopping
<tensorflow.python.keras.callbacks.History at 0x7ff29c469fd0>
```

Figure 12: Figure 6: Output of second model training with epochs

4 Experimental Evaluation

4.1 Methodology

Our approach to this project at the very beginning wasn't as smooth as we liked. In mini project 3, we used the cifar10 dataset, which was downloaded for us through the notebook for easy use. For this final project, however, we did not have such luxury and had to deal with importing the images from the dataset that we chose, which was the Fruits 360 dataset. It should be noted that the original dataset already split the data between training and testing, so we only had to worry about importing the data to our notebook. Of course, this was no easy task considering that we had to figure out how to get these files to Google Drive in the same directory as the Google Colab notebook, which was our experimental setting. Upon further research, we discovered that when we upload the files to the same location as the notebook, it was not really in the same location. What we had to do to alleviate this issue was to authorize the notebook for access to a Google account's Google Drive, and after that was done, the notebook was finally able to access all the data that we have uploaded.

Now comes the most difficult portion of the entire project, which was importing all the data for use in the notebook. Initially, our plan was to create two lists, one for the image and one for the label identifying the image. We did this for both the training set and the test set, which turned out to be quite elegant, however what we did not realize at the time was that we would have to run this code *every time we wanted to work on this project*. This was quite cumbersome considering that importing the data for use in our notebook would take roughly two hours every time. Because this was the case, we found a shortcut to this in that we just saved the numpy arrays generated after creating the training and test sets by utilizing the *np.save()* function.

After running and training each model, the metrics we used to measure their performances and compare and contrast the models were the precision, recall, f1-score, and the accuracy in their classification report.

Each of our CNNs took a different approach in terms of implementing and testing. Our first CNN was from mini project 3, which used an almost random sequence of layers and values for the activation functions and filters/neurons.

For our second CNN, we took a more "careful" approach and recalled that there was an ideal way to order the layers for our models, which were as follows: Conv2D, pooling, dropout, repeating those three as necessary, then flatten and dense layers. Also in contrast to the first CNN, we only used the ReLU activation function for most of the layers.

Finally, our third CNN was just an attempt of recreating the CNN that they used in their paper. What they used was a series of Conv3D layers with max pooling layers that followed each one (there were 4 pairs of these in the original paper), and so what we did was apply what we learned from previous mini projects and used Conv2D instead, mirroring their specifications on their paper as closely as possible.

4.2 Results

Accuracy: 0.2610072853975293
Averaged F1: 0.10804823067950976

	precision	recall	f1-score	support
0	0.00	0.00	0.00	624
1	0.26	1.00	0.41	824
2	0.00	0.00	0.00	832
3	0.00	0.00	0.00	877
accuracy			0.26	3157
macro avg	0.07	0.25	0.10	3157
weighted avg	0.07	0.26	0.11	3157

Figure 13: Classification report of first CNN model

Accuracy: 0.9382324992081089
Averaged F1: 0.9386695047221433

	precision	recall	f1-score	support
0	0.78	0.97	0.87	624
1	1.00	0.80	0.89	824
2	0.97	0.98	0.98	832
3	1.00	1.00	1.00	877
accuracy			0.94	3157
macro avg	0.94	0.94	0.93	3157
weighted avg	0.95	0.94	0.94	3157

Figure 14: Classification report of second CNN model

Accuracy: 0.9116249604054482
Averaged F1: 0.9132586929991344

	precision	recall	f1-score	support
0	0.74	0.92	0.82	624
1	1.00	0.80	0.89	824
2	0.91	0.93	0.92	832
3	1.00	1.00	1.00	877
accuracy			0.91	3157
macro avg	0.91	0.91	0.91	3157
weighted avg	0.92	0.91	0.91	3157

Figure 15: Classification report of third CNN model

Accuracy: 0.982895153626861
Averaged F1: 0.9829187029272352

	precision	recall	f1-score	support
0	0.98	1.00	0.99	624
1	1.00	0.98	0.99	824
2	1.00	0.96	0.98	832
3	0.95	1.00	0.98	877
accuracy			0.98	3157
macro avg	0.98	0.98	0.98	3157
weighted avg	0.98	0.98	0.98	3157

Figure 16: Classification report of transfer learning model

Table 1: Model Predictions

	True Label 1	Prediction 1	True Label 2	Prediction 2	True Label 3	Prediction 3	True Label 4	Prediction 4
CNN1	0	1	1	1	2	1	3	1
CNN2	0	0	1	1	2	2	3	3
CNN3	0	0	1	1	2	2	3	3
Transfer Learning	0	0	1	1	2	2	3	3

As we can see, the first CNN performed the worst out of the 4 models used in this project, and this is most likely because it did not translate well from mini project 3. For the second CNN, it had a much better performance, as we followed an ideal "formula" for how the layers are supposed to be ordered, which are Conv2D, pooling, dropout, repeating those three as necessary, then flatten and dense layers for the. For the third CNN, we attempted to recreate the neural network used for the original dataset, and that performed slightly worse than the second one that we made. Finally, the transfer learning model performed the best out of the four models.

Their original model had an accuracy of 95.89% for the original dataset, which is roughly 3% less than what our transfer learning model got.

5 Related Work

The problem addressed in a paper provided to us by the original dataset is the idea that "existing datasets with images contain both the object and the noisy background. This could lead to cases where changing the background will lead to the incorrect classification of the object." This is why they made the original dataset of approximately 80,000 images, as they wanted to make sure the neural network they were training had no "interference" from other sources of the images, so to speak.

In terms of why they made this neural network, according to their paper they addressed that the whole idea of identifying fruits from images was part of "a more complex project that has the target of obtaining a classifier that can identify a much wider array of objects from images"[1]. As some background information, it is also mentioned that Google (at the time of them writing this

paper) announced that it was working on an application named Google Lens, which tells a user many pieces of useful information about an object that a phone's camera would be pointing at [1]. They mention that the first step in creating such an application is to identify objects correctly [1].

6 Conclusion

In this project we attempted to create convolutional neural networks to produce better predictability than our reference. Even though accuracy is not the best metric to evaluate a model, we used it as the main comparison since our reference used only accuracy. Within our four models the transferling model, which used VGG16's pre-built CNN, had the best performance with an accuracy of 98.29%. While CNN3, which was our implementation of our reference's CNN model, had 91.16% accuracy. According to our reference's paper, their accuracy was at 95.89%. I don't think it is a fair comparison between our best model and their model accuracy because of the different approach of the dataset. While they classified 120 categories, we only categorized 4 classes. Also, we found that the relu function produced the best results and our reference also only used relu. Although, our implementation of their model was outperformed by our CNN2, which was our best CNN model and an accuracy of 93.82%, and transfer learning model. In addition, although CNN1 performed the best in mini-project 3 it did not perform as well in this project. The main difference between CNN2 and CNN3 was the added dense layers towards the end of CNN2, which produced better results. Since we reduced the amount of data originally used, it shows that the concept of using transfer learning when there is not enough data produces the best result versus creating a neural network with not enough data.

7 Work Division

We pretty much planned everything together from start to finish. A lot of the programming was essentially figuring out the "puzzle pieces" from previous mini projects, so it just came down to deciding how to go about the various parts and/or obstacles that we came across. Specifically, if an issue arose for the programming portion, both people would look into the issue to fix and debug as soon as possible.

8 Learning Experience

Overall, it was an interesting experience trying to apply almost everything we have learned from the mini projects into this one final project. The feeling was quite similar to the first mini project since we are technically programming from scratch once again, but this time we get to choose our own dataset rather than have one that has been provided for us. The most useful and important thing that we probably learned was importing data, specifically the images we used in this dataset/project, to our Jupyter notebook on Google Colab. That alone probably took up over half the total time spent on this project, since we had to worry about authenticating a Google email to make sure the notebook had access to the Google drive where all the data and files were, making sure we save the numpy arrays appropriately so we didn't have to repeatedly import the images every time the notebook disconnected or when we wanted to work on it (doing so takes roughly two hours at a time for the subset of images we chose for this project… Imagine doing all ~80,000 images!), and just making sure that everything was working properly in terms of accessing the necessary files through the Jupyter Notebook.

The ability and opportunity to solve problems as they come and go is very important when comes to industry and life in general, and this project seems to be a clear reflection of that.

REFERENCES

[1] H Muresan, M Oltean (2018) Fruit recognition from images using deep learning.
[2] M Oltean, Fruits 360, https://www.kaggle.com/moltean/fruits