Machine Learning Engineer Nanodegree

Report

Matthew Crummey
9-28-2019

Contents

Table of Tables

Table of Figures

Definition

Project Overview

The continued integration of automation into everyday life will require further refined computer vision. Computer vision is the process in which computers are trained to interpret and understand the visual world [1]. Some of the industries that use computer vision are: Manufacturing, Health Care, Insurance and Retail [1, 2]. One of the applications of computer vision within retail is the recognition of food, which is used in cashier-less stores like Amazon GO [2]. By being able to accurate identify fruit from images taken from video cameras, a store can know two main things: 1. When a customer has "purchased" it and 2. When the shelf is running low on a fruit.

Problem Statement

The main objective of this project is to use machine learning to be able to identify fruits within images, thus this is a computer vision problem. The input is an RGB picture of a piece of fruit and the output is the prediction vector of what type of fruit it is. The problem is to train an algorithm that when given an image of a fruit, can accurately identify what type of fruit it is. The solution to this problem will be a Convolutional Neural Network (CNN) model built in python using Keras on the Tensorflow backend. This model will accept a 100 pixel by 100 pixel RGB image as input and output the name of the predicted fruit.

Evaluation Metrics

The first evaluation metric for this project will be the Accuracy Score of the testing set, after training and validation have been done. This accuracy score is produced by Keras itself with metric specified as *accuracy* during the compile step of the model creation. When specifying *accuracy* as the metric, Kera itself will decide between *binary accuracy*, *sparse categorical accuracy*, or *categorical accuracy* [3]. For this model, Keras would decide to use categorical accuracy because the size of the final layer is not 1 (for binary accuracy), and then the loss function would not be used as to determine if the data is sparse enough. An analysis of the sparseness of the data will be completed in the Data Exploration section.

The second evaluation metric will be the Accuracy Score of internet sourced images. This will be done because the images used for testing and training have a white background and I want to see how well the model does with images with backgrounds.

Analysis

Data Exploration

The chosen data set is the Fruit-360 dataset from Kaggle.com (https://www.kaggle.com/moltean/fruits) [4]. This data set has approximately 70000 100 pixel by 100 pixel RGB images of fruit that are already split into a training and testing set. I will split the training set into a training set and a validation set to allow the model to take advantage of validation during training.

Both the training and testing have subfolders with the name of each fruit as the name of the subfolder. Within the dataset different pieces of fruit are given different numbers (Apple Golden 1 vs Apple Golden 2); for this project I will strip out the numbers so all Apple Golden will be trained, validated and tested the same.

Figure 1: Example of Apple Golden 1

Figure 2: Example of Apple Golden 2

Figure 3: Example of Apple Granny Smith

Originally, without the above trimming, there are 103 targets; after the trimming there are 87 targets. The highest count of any one target in either the training set or target set is 5% of the total images. I do not believe this upsets the distribution of the images.

As mentioned above, the sparseness of the dataset must be determined so the proper loss function can be used. To do this, the average intensity of all the pixels in each image were determined (1 is full intensity - white, 0 is no intensity – black). Ideally the average intensity would be in the 0.4-0.6 with a low standard deviation as that would mean that on average the pixels were some variation between white and black. The results of the training, validation, and testing set can be seen below in Table 1.

Table 1: Pixel Intensity by Dataset

Data Set	Average	Standard Deviation
Training	0.5845	0.0717
Validation	0.5834	0.0724
Testing	0.5947	0.0715

Also, the max intensity for a class was found to be for Strawberry Wedges with an average intensity of 0.6164 and a standard deviation of 0.0533. The minimum intensity for a class was found to be for Mangostan with an average intensity of 0.4992 and a standard deviation of 0.0402.

Visual Exploration

In the Figure 4 below, the distribution of the number of images in each fruit category has been plotted for the full training set (before split into training and validation set). A large majority (~85%) of the classifications of fruit have been 300 and 550 images in the training dataset, whereas there is only one classification (Tomato) that has more than 2000 images. I do not believe this will cause an overfit of the model, because this does not even represent 5% of the total number of images within the training set.

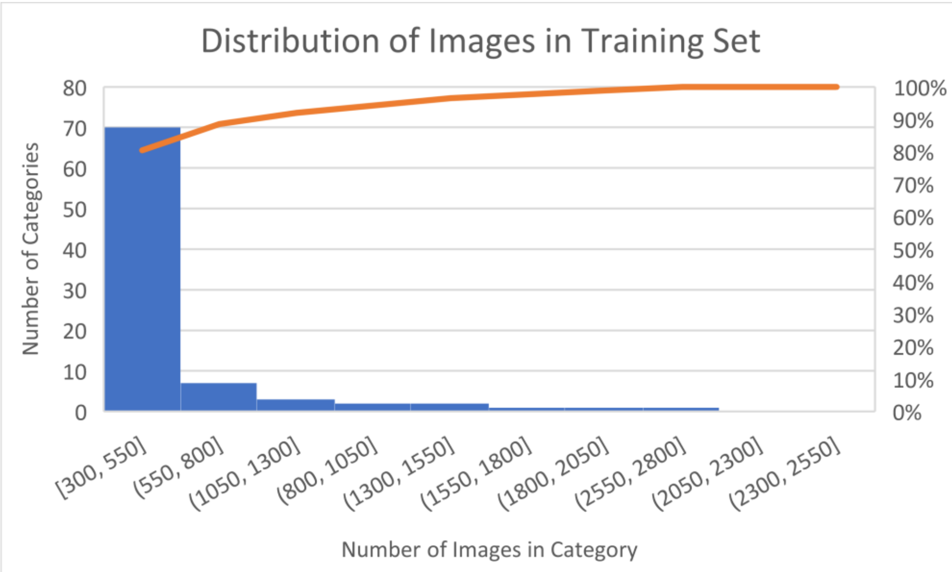

Figure 4: Plot of Distribution of Number of Images in the Training Set

The same also holds true for the testing set. A large majority of the images have a similar range of images with only a few being outside that range. This can be seen in Figure 5.

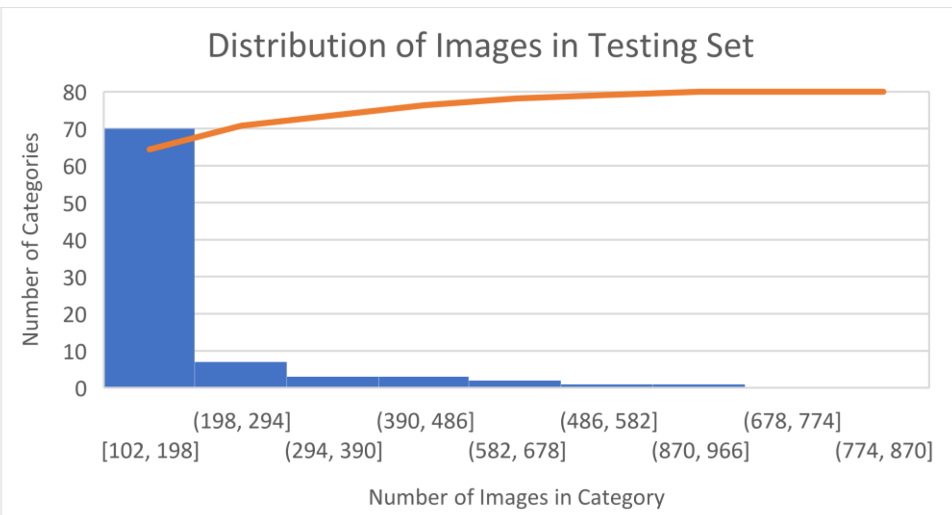

Figure 5: Plot of Distribution of Number of Images in the Testing Set

Doing a cursory glance over the dataset there does not appear to be any images that are mislabeled, blurry or otherwise corrupted that would cause issues with training.

Algorithms and Techniques

The main algorithm that will be used is the Convolutional Neural Network (CNN). This type of neural network is ideally suited for image classification because it combines the neural network architecture with the feature finding power of convolution [5]. Convolution across the images with various filters also different features to be determined by each filter. A CNN typically consists of pairs of convolutional layers and pooling layers followed by at least one densely connected layer [5]. The input into the first convolutional layer would be the dimensions of the image and the output from the last densely connected layer would be a one-dimensional array with the same length as the number of classifiers [5]. This output would contain one entry per node with the probability that the node corresponds to the image that was classified.

Convolutional Layer

A convolutional layer works by using a filter (for images a 2D array of initially randomized values) to try to detect features of an image. As the filter convolves around the image, each element of the filter is multiplied against its corresponding element in the image. This brings up the issue of padding; as when trying to convolve around an edge or at a vertex, the filter is going to be technically out of bound of the image. This can be resolved in many ways, but for the purposes of this project I have chosen to deal with it by filling those out-of-bound elements with zeroes.

1	0	0	0	1
0	1	0	1	0
0	0	1	0	0
0	1	0	1	0
1	0	0	0	0

0	1	0
1	1	0
1	0	1

Figure 6: Example Image (left) and Filter

For example, given the image (in array form) and the filter in Figure 6, doing the first convolution calculation (top left corner), with zero padding, and a stride of one, would look like below (filter application in green):

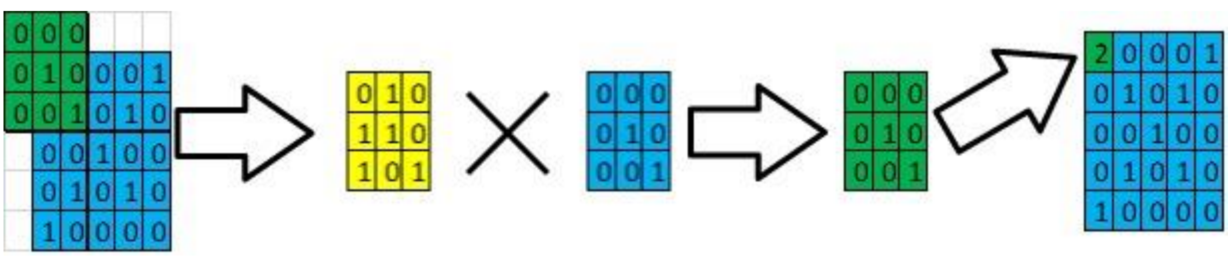

Figure 7: Example of Convolutional Step

Applying the entire filter over the image would result in the below figure:

2	1	2	0	2
1	2	1	2	2
1	1	3	2	1
0	2	2	1	1
1	2	0	1	0

Figure 8: Example of fully convoluted array

Pooling

Pooling is the process in which an aggregation function is used to reduce the dimensionality of the data. This helps in feature identification because as the aggregation occur, the important parts of the arrays will become more prominent. In this project max pooling has being chosen as the pooling layer of choice. In this type of pooling, the output of the previous is divided into sections (as determined by the size and stride of the pooling layer). Then the max value of that section is found and used to produce a new smaller array. Using Figure 8 as a starting point, using valid padding (no padding), with a size of 2 and a stride of 2, a max pooling layer would divide the array as follows:

2	1	2	0	2
1	2	1	2	2
1	1	3	2	1
0	2	2	1	1
1	2	0	1	0

Figure 9: Sectioned Array for Max-Pooling

The yellow elements in Figure 9 are not included in the pooling calculations because no padding is being used in this case. Thus, the output of this max pooling layer would be:

2	2
2	3

Figure 10: Output of example max pooling layer

As we can see, the dimensionality of the array has been reduced by at least the size of the pooling layer filter.

Dropout

The use of dropout is a relatively new concept in terms of densely connected layers. A densely connected layer is a layer in which every node of that layer is connected to every node of the previous layer. While this is a very good thing for CNNs because it combines features together that would not be combined otherwise, it can lead to overfitting of the model [6]. An example of a densely connected layer can be seen in Figure 11.

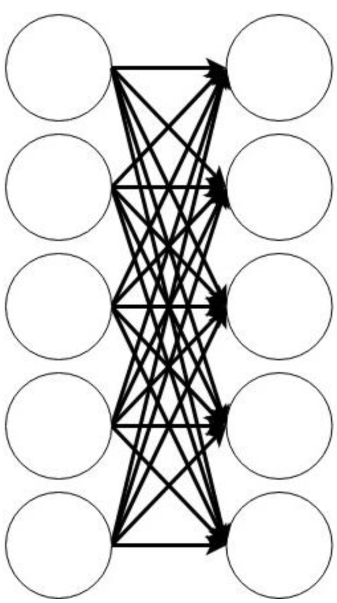

Figure 11: Example of two fully connected dense layers

The process of dropout means that only a certain percentage of nodes will be connected to the node from the previous layer. So, for example, a dropout of 0.4 (40% of nodes are not connected) could look like the below figure:

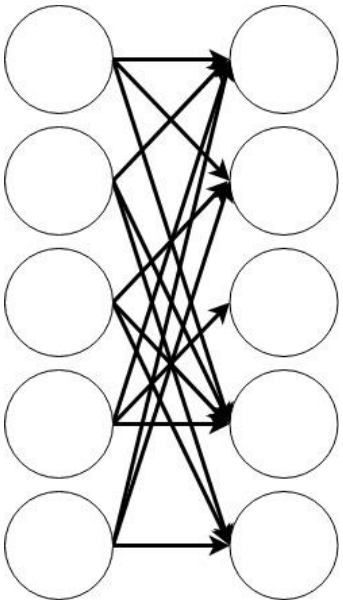

Figure 12: Two dense layers with 40% dropout

When using dropout, the nodes that connected to each other change with every propagation. Also, as dropout is applied randomly, some nodes might still be fully connected still, while others are completely disconnected from all other nodes.

Benchmark

The benchmark model for this project will be the best performing configuration that was used by Mureşan and Oltean in their creation of the dataset that I am using. Their best performing network had a training accuracy of 99.62% and a test accuracy of 95.88% with the following configuration[1]:

Table 2: Simplified Configuration of Best Performing CNN for Mureşan and Oltean [4, p. 21]

Layer	**Filter Size**	**Number of Nodes/Filters**
Convolutional/Pooling	5x5	16
Convolutional/Pooling	5x5	32
Convolutional/Pooling	5x5	64
Convolutional/Pooling	5x5	128
Fully Connected		1024
Fully Connected		256

They used the *adam* optimizer and *categorical crossentropy* for the loss. Thus, this model is the model that I will base my initial model on and iterate on. Since I will not be able to replicate their model 100%

[1] For the full network configuration please see [3, pp. 35-39]

accurately, I will use the internet sourced fruit accuracy as a benchmark solely on my iteration to determine if the model is getting better or not.

Methodology

Data Preprocessing

In order to use an image in a machine learning application, the image must be converted into a form that a machine can understand, into a zero or one, or in this case and array of arrays of zeroes and ones. For this problem, each of the images are 100px by 100px RGB images. Thus, they are first loaded in as a 100 by 100 by 3 matrix, with each layer representing one colour and each cell of the layer representing the saturation of it's layer's color in that cell. For example, Figure 13 below is a 5 pixel by 5 pixel RGB image and must be converted into a 5 by 5 by 3 matrix.

Figure 13: Example 5 pixel by 5 pixel RGB image

Each layer of the 5 by 5 by 3 matrix can be seen below in Figures Figure 14 through Figure 16.

120	186	241	205	226
167	197	234	243	249
204	247	232	123	118
153	253	204	143	67
151	221	58	214	235

Figure 14: Red layer of Figure 13

226	238	251	179	202
212	235	247	233	246
217	253	249	74	164
80	243	210	228	149
87	177	227	255	244

Figure 15: Green layer of Figure 13

252	225	243	204	224
192	194	193	231	253
208	227	213	95	153
172	244	196	205	207
201	90	24	154	249

Figure 16: Blue layer of Figure 13

Therefore, we can see that each pixel in the example image can be represented by a red, green, and blue saturation number. So, for example, the top left pixel in the example image has a RGB value of (120, 226, 225), which logically follows considering that the actual color of the pixel is a teal colour which itself is primarily a mix of green and blue.

After every image has been decomposed into its three-color layers, it is best to normalize the values in each layer to make the fitting of the model quicker and more efficient. In the RGB image format, the maximum saturation of any layer (color) is 255 and the minimum is 0, so it would make sense to divide every cell by 255 to normalize the value between 0 and 1 instead of 0 and 255.

The final step in the preprocessing is to expand the dimensions of the array from three dimensions to four dimensions, where the fourth dimension is the sample size. This is a requirement of the Keras 2D convolutional layers that are used in the model [7].

Implementation

After preprocessing the data, the implementation of the actual CNN involves:

1. Define the model architecture and parameters

2. Define the loss function and the testing metric
3. Fit the model and keep track of the loss and if it has improved save that model as the best model
4. Iterate through step 3 for a set number of epochs or until an appropriate loss value has been produced.

Refinement

Model Refinement

The first step of the refinement was to determine if any layers should be added or removed from the model. The layers that I looked at were the 2D Convolutional, 2D MaxPooling, dense, and dropout layers. The convolutional and maxPooling layers are the hallmarks layers of CNNs, so I knew that changing the number of pairs of those would be crucial to improving the performance. Adding in dense layers and dropout layers after the convolutional layers might assist in the classification. Thus, I decided to test both the number of convolutional/maxPooling pairs, the number of dense/dropout pairs after the flatten operation and the sizes of each layer. For this process I first created all the model and then tested them one by one.

It should be noted that there are some restrictions on the size and number of the maxpooling layers. As each maxpooling layer used will decrease the size of the length and width of the tenors, only a certain size and number of them can be used before a 1 by 1 tensor is produced. For example, use the four-layer benchmark model with a maxpooling pool size of four for each layer, the size of the layers would go from 100 by 100 to 25 by 25 (1^{st} pooling) to 6 by 6 (2^{nd} pooling) to 1 by 1 (3^{rd} pooling) at which point the model size cannot be pooled anymore. For this reason, the maxpooling size wasn't increased past three for any number of maxpooling layers and was left at two for the five max pooling layer models.

I tested the following combinations for a grand total of 396 models[2]:

- Kernel size of Conv2D layers (3,5,7)
- Pool size of MaxPooling2D layers (2,3) [except where noted above]
- Number of Conv2D and MaxPooling2D layers (3,4,5)
- Size of 1^{st} dense layer (1024, 2048)
- Size of 2^{nd} dense layer (256, 512)

The models were created in *modelRefinmentCreator.py* and were tested in *modelRefinmentTester.py*

After testing the combinations above using five training epochs (total process took approximately 30 hours), there were three models that shared the highest internet image accuracy score of 19%. Out of those models, the one that has the highest test accuracy (98.69%) is the one I am going to use for the next stage of refinement. That model has the following architecture:

[2] I only tests kernel and layer sizes in ascending order away from the benchmark, so I never created a model that had a kernel size of 5 on the first layer and 3 on the second, or 7 on the third and 5 on the second. Otherwise there would have been close to 100,000 models to test

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=3, padding='same', activation='relu',input_shape=(100, 100, 3)))
model.add(MaxPooling2D(pool_size=3))
model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(2048, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(812, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(87, activation='softmax'))
```

Figure 17: Best Model after Model Refinement

The full result set can be found in Appendix 1: Model Refinement Results.

Variable Refinement

After determining a better model, the next step in refinement is determining the best parameters to be used in the model. In order to do this, I will implement a grid search-like operation on various parameters to determine the best option for those parameters. The following parameters can be modified:

- Conv2D (per layer)
  - Number of filters
  - Strides
  - Kernel Size
  - Padding
  - Activation
  - Use bias
  - Kernel initializer
  - Bias initializer
  - Kernel regularizer
  - Bias regularizer
  - Activity regularizer
  - Kernel Constraint
  - Bias Constraint
- MaxPooling2D (per layer)
  - Pool_size
  - Strides
  - Padding
- Dropout (per layer)
  - Rate
  - Noise shape
  - Seed (for random seed)
- Dense (per layer)

- Units
- Activation
- Use bias
- Kernel initializer
- Bias initializer
- Kernel regularizer
- Bias regularizer
- Activity regularizer
- Kernel Constraint
- Bias Constraint

- Compile
  - Optimizer (along with the hyperparameters for each unique optimizer)
  - Loss
  - Metrics
  - Loss weights
  - Sample weight mode
  - Weighted Metrics

Just as with the model refinement, due to the sheer number of parameters that can be tuned a much smaller set must be used. The parameters I chose are the dropout rates for the two dropout layers along with the optimizer. I chose the dropouts due to research that showed that changing the dropout rate can have a significant impact on the performance [6]. I chose to include the optimizer because it is one of the only single parameters that can be changed that will affect the entire way the model fits. Also, because the model takes about three and a half minutes to fit and test (on my computer), I didn't want to choose too many parameters or values for those parameters to keep the total testing time to within a reasonable length of time. The values that I chose to test on can be seen below in Table 3.

Table 3: Parameters and Values to Grid Search

Parameter	**Values to Test**
First Dropout	0.2, 0.3, 0.4, 0.5
Second Dropout	0.2, 0.3, 0.4, 0.5
Optimizer	SGD, RMSprop, Adam, Adagard

I chose the values of 0.2 to 0.5 for the dropouts based on the findings by Srivastava Et al. [6][3]. I chose to use the four optimizers listed because I have experience in using them in other projects. The metrics I am using to determine the best parameters are the same as I used to test the benchmark model:

1. the accuracy of the prediction of the testing set
2. the accuracy of predicting internet sourced images

[3] Srivastava Et al. indicate a dropout rate of 1 as meaning no dropout [7], whereas dropout is implemented in Keras where a rate of 0 means no dropout [6]

The code that I created to do this grid search is in *parameterTuning.py* where I go through the same initial stages of creating a CNN of loading in the tenors. I then created a pandas data frame with all combination of the above parameters (64 in total). I then iterate through each row of the data frame creating, compile, fitting, and testing a model with those parameters. I then record the test accuracy, the time taken to fit the model and the accuracy against the internet images. The full result set can be seen in Appendix 2: Variable Refinement Results.

Results

The top ten results from the grid search in decreasing accuracy against the internet sourced images can be seen in Table 4.

Table 4: Top Ten Results from Grid Search (descending internet accuracy)

index	dropout1	dropout2	optimizer	Train Accuracy	Test Accuracy	Learning Time	Internet Accuracy
4	0.2	0.2	Adam	0.99146	98.58224	03:12.3	19.04762
27	0.3	0.4	RMSprop	0.991527	97.99384	04:11.9	19.04762
3	0.2	0.2	RMSprop	0.993518	97.51751	04:13.1	19.04762
23	0.3	0.3	RMSprop	0.991571	98.16195	04:12.4	14.28571
5	0.2	0.3	Adagrad	0.97531	96.42477	03:52.8	14.28571
40	0.4	0.3	Adam	0.989027	98.54861	03:12.1	11.90476
39	0.4	0.3	RMSprop	0.991305	98.39731	04:12.5	11.90476
51	0.5	0.2	RMSprop	0.990619	98.14514	04:11.5	11.90476
48	0.4	0.5	Adam	0.989823	97.90417	03:13.1	11.90476
19	0.3	0.2	RMSprop	0.992168	97.52872	04:11.2	11.90476

The best model after variable refinement is still the best model from the model refinement. It seems that overall *adam* and *RMSprop* and low levels of dropout are the best for the internet accuracy score.

Results

Model Evaluation and Validation

The final model is a three-layer 2Dconvolutional with maxpooling layers followed by two layers of dropout and densely connected layers. The model was compiled with the adam optimizer and loss function of categorial crossentropy. The training accuracy of the final model was 99.146%, with a test accuracy of 98.58%, training time of 3 minutes and 12 seconds, and an accuracy against the internet sourced images of 19%. The final architecture of the model can be seen below in Figure 18.

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=3, padding='same', activation='relu',input_shape=(100, 100, 3)))
model.add(MaxPooling2D(pool_size=3))
model.add(Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(2048, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(812, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(87, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 18: Final Model Architecture

I used the internet sourced images of fruit to measure the robustness of the model. However, what I found was that due to the limitations of the initial data set, the model did tend to overfit to the type of image that it was trained on; 100px by 100px with a white background. In Table 5, I have selected a few rows from the full table of the prediction of all the internet sourced images to highlight some interesting cases.

Table 5: Excerpt from Final Model Internet Sourced Image Prediction

path	target	prediction	Prediction Percent	Target Prediction Percent
fruits/internet_resized\Apple_Red_1.jpg	Apple Red	Apple Pink Lady	0.654492	8.57E-12
fruits/internet_resized\Apple_Red_3.jpg	Apple Red	Cherry	0.603045	0.328073
fruits/internet_resized\Avocado_1.jpg	Avocado	Avocado ripe	0.760063	7.67E-08
fruits/internet_resized\Banana_3.jpg	Banana	Banana	0.503727	0.503727
fruits/internet_resized\Cantaloupe_1.jpeg	Cantaloupe	Cantaloupe	0.997193	0.997193
fruits/internet_resized\Cantaloupe_2.jpg	Cantaloupe	Cantaloupe	0.977307	0.977307
fruits/internet_resized\Cantaloupe_3.jpg	Cantaloupe	Cantaloupe	0.949618	0.949618
fruits/internet_resized\Cherry_3.jpg	Cherry	Pear Red	0.99999	7.71E-17
fruits/internet_resized\Pear_Red_1.jpg	Pear Red	Pear Red	0.949801	0.949801
fruits/internet_resized\Pepper_Green_1.jpg	Pepper Green	Pepper Green	1	1
fruits/internet_resized\Pepper_Green_2.jpg	Pepper Green	Banana Lady Finger	0.998981	0.000441
fruits/internet_resized\Tomato_1.jpg	Tomato	Tomato	0.923479	0.923479
fruits/internet_resized\Tomato_2.jpg	Tomato	Pear Red	1	9.99E-25

For example, all the Cantaloupe images were classified correctly, but none of the Apple images were. It is also interesting to note that in the data set there was a distinction between Avocado ripe and Avocado, but as I did not know the ripeness of the avocados in the images, I found I classified them generically. Overall, I think that fruits with distinct shapes, colours or surfaces were more easily and accurate classifiable (Cantaloupe and Banana for example), then fruits that have a common size, colour or shape (Apple, Red Pear, Cherry and Tomato). The full version of the Table 5 can be found in Appendix 3: Internet Sourced Images Predictions.

Justification

The model itself I believe is good, but I think after going through the training/validation/testing process, the images are all too alike to provide any meaningful real-world data for a model. Another shortcoming in the data is that all information about size is lost when all the images were made to be 100px by 100px. I believe this is why such things as real-world Apples, Cherries, and Pears were misclassified as each other because when all scaled down to 100px by 100px almost all information about the size of the fruit is lost.

However, when compared to the benchmark model, the model that I refined did perform better similarly against the training set (99.15 vs 99.62%) and better for the testing set (98.58% vs 95.88%). I believe these accuracy differences stem from the fact that my model used one less layer so it didn't overfitting as much versus the images as it might with three layers.

Conclusion

Free-Form Visualization

Figure 19: Apple Red (left); identified as Apple Pink Lady

Figure 20: Cantaloupe (left) identified as Cantaloupe

Figure 21: Pepper Green (left) identified as Pepper Green

Figure 22: Pepper Green (left) Identified as Banana Lady Finger

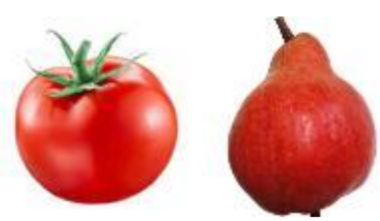

Figure 23: Tomato (left) identified as Pear Red

Figure 24: Cherry (left) identified as Pear Red

In the above images it can be seen that in some cases the features were very similar to each other and understandably why the model may have gotten the classification wrong (Figure 19). However, there were other cases in which the difference does not make logical sense as in Figure 22 where the model classifies a Green Pepper as a type of banana when it have also identified a Green Pepper Correctly.

Reflection

The process using for this project can be broken down in the following way:

1. A problem was identified and a dataset that was applicable to that project was found and downloaded
2. The data was preprocessed into a format that Keras can utilize
3. A benchmark model was referenced and recreated from other literature
4. The exact layout of the model and the parameters were iterated to determine the best combinations
5. Internet sourced images of fruit were used to determine how effect the final model was

The most difficult step in the process was step number 4 for a few reasons. First, determining what parameters and changes to the model should be tested proved to be difficult because it is impossible to fully understand the impact of a change to this type of model. Second, the actual testing of these parameters, while not technically difficult (once GPU memory issues were resolved), were time consuming, with each test of the model taking approximately 5 minutes. The third reason why step four was so difficult was because of the random nature of GPU training. Even though a seed number can be set for keras, there are still things that a GPU may do to add randomness to a model.

Improvement

The biggest improvement that I could see for this model would be additional data items that include real world images of fruits in real world resolutions. I believe that the biggest downside of the dataset that I used was that each image of the same type of fruit was normally just the fruit rotated a few degrees and that all fruit almost completely filled the image, so that the size difference between a cherry and a red pear could not be appreciated by the model. This might also mean that features that are present on most pieces of a typical piece of fruit may have been missing from the fruit in this dataset.

Further, this provokes a question of what makes a good, well-rounded representative data set, and if a specific issue is trying to be solved, how specific does the dataset need to be. For example, in this problem of fruit identification in stores, would the model require images of fruit only in how they are laid out in a store, or would generic images like the one in the dataset I use still provide some use.

Final Thoughts

Overall, I did have a lot of fun with the problem as it really made me think critically about how a dataset can or should be used to solve a particular problem. I also enjoyed the challenge of trying to determine the best parameters and model features to try and tune to improve the performance of the model. I do believe that I have reached the limits of the dataset I used to solve this problem, and that in order to get an accuracy against real world fruits greater than 20%, a bespoke data set would have to be created.

References

[1] SAS, "Computer Vision," 2019. [Online]. Available: https://www.sas.com/en_us/insights/analytics/computer-vision.html. [Accessed 20 07 2019].

[2] CB Insights, "Beyond Amazon Go: The Technologies And Players Shaping Cashier-Less Retail," 09 10 2018. [Online]. Available: https://www.cbinsights.com/research/cashierless-retail-technologies-companies-trends/#why. [Accessed 20 01 19].

[3] T. Gabriel, "General Terminology," 26 7 2018. [Online]. Available: https://github.com/sagr4019/ResearchProject/wiki/General-Terminology#difference-between-accuracy-and-categorical_accuracy. [Accessed 29 09 2019].

[4] M. Oltean and H. Muresan, "Fruit recognition from images using deep learning," *Acta Universitatis Sapientiae, Informatica,* vol. 10, no. 1, pp. 24-42, 2018.

[5] skymind, "A Beginner's Guide to Convolutional Neural Networks (CNNs)," [Online]. Available: https://skymind.ai/wiki/convolutional-network. [Accessed 20 07 2019].

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from," *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[7] F. Chollet, "Keras Convolutional Layers - Conv2D," 2015. [Online]. Available: https://keras.io/layers/convolutional/#conv2d. [Accessed 7 September 2019].

[8] F. Chollet, "Keras Core Layers - Dropout," 2015. [Online]. Available: https://keras.io/layers/core/#dropout. [Accessed 7 September 2019].

Appendix 1: Model Refinement Results

Test Number	Train Accuracy	Test Accuracy	Learning Time	Internet Accuracy
1	0.98714602	96.74978986	04:58.2	11.9047619
2	0.986393809	94.41300084	05:13.7	4.761904762
3	0.987123907	94.43541608	05:26.8	11.9047619
4	0.983517706	95.80274587	05:31.2	4.761904762
5	0.985951304	95.63463155	05:35.2	2.380952381
6	0.988805294	97.06920706	09:21.0	2.380952381
7	0.991238952	97.0916223	04:08.4	19.04761905
8	0.991415918	96.31269263	03:55.4	7.142857143
9	0.992765486	96.04931353	03:54.1	4.761904762
10	0.988761067	96.67133651	03:13.0	11.9047619
11	0.987035394	95.33202578	03:26.3	2.380952381
12	0.98993361	95.02942001	03:51.7	9.523809524
13	0.986836255	95.78033062	04:06.4	4.761904762
14	0.982101798	95.58980106	04:02.6	7.142857143
15	0.988561928	98.16195013	02:59.4	9.523809524
16	0.990929186	97.84253292	02:46.4	2.380952381
17	0.990995586	97.59596526	02:40.4	9.523809524
18	0.990597367	97.56234239	02:38.1	16.66666667
19	0.987831831	96.27906977	02:42.3	4.761904762
20	0.985840678	95.80834968	02:59.2	7.142857143
21	0.984955728	95.13589241	03:21.3	2.380952381
22	0.983716786	95.2927991	03:33.6	4.761904762
23	0.983871698	97.27654805	03:32.5	7.142857143
24	0.988296449	97.84813673	02:33.5	7.142857143
25	0.990420341	98.45895209	02:21.3	9.523809524
26	0.990376115	97.89296722	02:16.8	9.523809524
27	0.989977896	98.54300925	02:16.2	4.761904762
28	0.985088468	97.70243766	02:34.8	7.142857143
29	0.982035398	95.8755954	02:51.7	4.761904762
30	0.981438041	96.00448305	03:14.6	9.523809524
31	0.979955733	96.25105071	03:25.5	7.142857143
32	0.98252213	98.20678061	03:25.2	9.523809524
33	0.986482322	97.05799944	02:24.3	9.523809524
34	0.990398228	97.78649482	02:12.8	4.761904762
35	0.987721264	97.06360325	02:07.8	11.9047619
36	0.987411499	99.07537125	02:09.3	11.9047619
37	0.985088468	97.0355842	02:33.8	9.523809524
38	0.982964575	97.60156907	02:51.8	7.142857143

39	0.981415927	94.83889045	03:13.9	7.142857143
40	0.985641599	96.44158027	03:25.4	11.9047619
41	0.980265498	96.18380499	03:24.8	4.761904762
42	0.98904866	96.8786775	02:23.2	4.761904762
43	0.98707962	98.4141216	02:12.1	7.142857143
44	0.988230109	97.21490614	02:08.2	11.9047619
45	0.985752225	99.46203418	02:09.5	9.523809524
46	0.989977896	96.85065845	04:35.9	7.142857143
47	0.988340735	94.9733819	04:52.4	7.142857143
48	0.988318563	95.35444102	05:08.8	9.523809524
49	0.987699091	95.10226954	05:12.2	2.380952381
50	0.98612833	93.83020454	05:11.6	11.9047619
51	0.987854004	97.27094424	04:28.8	11.9047619
52	0.991128325	97.31017092	04:02.1	4.761904762
53	0.988606215	97.13084898	03:48.7	11.9047619
54	0.993230104	96.74978986	03:49.0	2.380952381
55	0.988252223	96.57046792	03:10.7	7.142857143
56	0.984646022	94.73241804	03:30.3	9.523809524
57	0.985951304	96.38554217	03:51.1	2.380952381
58	0.98422569	95.97086018	04:05.2	7.142857143
59	0.985752225	95.77472681	04:04.7	4.761904762
60	0.989469051	98.69991594	02:59.9	7.142857143
61	0.990973473	95.61221631	02:51.1	4.761904762
62	0.989535391	97.59036145	02:41.9	2.380952381
63	0.991592944	98.66068927	02:40.9	7.142857143
64	0.988318563	97.11964136	02:45.2	9.523809524
65	0.985154867	96.32390025	03:01.2	2.380952381
66	0.983561933	95.75231157	03:25.0	7.142857143
67	0.982013285	96.63210983	03:37.5	9.523809524
68	0.983185828	95.69066966	03:36.4	11.9047619
69	0.989380538	96.09414402	02:36.7	2.380952381
70	0.989181399	98.1507425	02:22.3	4.761904762
71	0.990575194	97.92098627	02:18.1	4.761904762
72	0.990973473	98.8344074	02:18.1	7.142857143
73	0.987256646	96.23984309	02:37.0	7.142857143
74	0.982367277	96.83384702	02:55.8	4.761904762
75	0.980000019	95.06304287	03:15.7	4.761904762
76	0.983008862	97.18128327	03:28.1	4.761904762
77	0.97986728	96.07733259	03:26.9	11.9047619
78	0.986371696	97.5679462	02:25.6	9.523809524
79	0.986946881	97.9377977	02:14.7	2.380952381

80	0.990132749	97.988232	02:10.9	9.523809524
81	0.98707962	98.04987391	02:11.4	2.380952381
82	0.98530972	98.38049874	02:37.3	14.28571429
83	0.982212365	97.17567946	02:55.4	7.142857143
84	0.98081857	93.26982348	03:15.2	2.380952381
85	0.980398238	94.09358364	03:28.7	4.761904762
86	0.980995595	94.3401513	03:26.5	7.142857143
87	0.988340735	97.18688708	02:25.6	9.523809524
88	0.990508854	98.91846456	02:14.5	4.761904762
89	0.985398233	99.08097506	02:10.1	2.380952381
90	0.989358425	97.8313253	02:12.0	7.142857143
91	0.99075222	96.95152704	05:57.6	4.761904762
92	0.988119483	94.56430373	06:09.9	2.380952381
93	0.989601791	96.04931353	06:20.0	4.761904762
94	0.988119483	96.0325021	06:24.5	4.761904762
95	0.986681402	96.7722051	06:24.0	9.523809524
96	0.988827407	96.68254413	05:59.9	11.9047619
97	0.990464628	98.65508546	05:20.3	2.380952381
98	0.990663707	97.04118801	05:03.8	7.142857143
99	0.99163717	97.01877277	05:01.6	7.142857143
100	0.984668136	97.34379378	03:47.7	2.380952381
101	0.989181399	97.82011768	04:03.8	0
102	0.983738959	97.76407957	04:23.6	7.142857143
103	0.983783185	94.56990754	04:35.8	7.142857143
104	0.98721236	93.98150743	04:36.6	7.142857143
105	0.991128325	97.95460913	03:37.4	4.761904762
106	0.989535391	96.68814794	03:39.5	4.761904762
107	0.991482317	98.22359204	03:20.0	4.761904762
108	0.992101789	97.4614738	03:17.6	9.523809524
109	0.990796447	94.05435696	03:01.7	2.380952381
110	0.987455726	97.47268142	03:17.1	7.142857143
111	0.983871698	94.62594564	03:41.0	4.761904762
112	0.986017704	96.20061642	03:49.9	4.761904762
113	0.983783185	94.68758756	03:51.2	4.761904762
114	0.98816371	98.97450266	02:50.8	16.66666667
115	0.990464628	97.48388904	02:41.8	11.9047619
116	0.991792023	96.97954609	02:35.5	4.761904762
117	0.990309715	97.62958812	02:37.5	7.142857143
118	0.986747801	95.78033062	02:45.3	2.380952381
119	0.984048665	97.31577473	03:03.1	4.761904762
120	0.98238939	95.51134772	03:23.5	9.523809524

121	0.984734535	97.06360325	03:34.5	4.761904762
122	0.982411504	95.06864668	03:34.5	7.142857143
123	0.989889383	97.89296722	02:33.7	11.9047619
124	0.988296449	98.44214066	02:20.8	4.761904762
125	0.988340735	98.22359204	02:17.1	9.523809524
126	0.98980087	97.94900532	02:17.7	7.142857143
127	0.986858428	96.22303166	02:44.7	0
128	0.983805299	97.90417484	03:01.3	2.380952381
129	0.983207941	97.12524517	03:24.2	4.761904762
130	0.98075223	94.62034183	03:35.7	4.761904762
131	0.98252213	97.32137854	03:34.4	9.523809524
132	0.989513278	96.25105071	02:31.5	4.761904762
133	0.987854004	98.03306248	02:21.2	9.523809524
134	0.989424765	97.5119081	02:17.8	2.380952381
135	0.987787604	98.65508546	02:17.7	7.142857143
136	0.990154862	93.9702998	05:57.1	4.761904762
137	0.985619485	96.0325021	06:19.8	4.761904762
138	0.987035394	95.18072289	06:28.3	7.142857143
139	0.982854009	95.48893247	06:27.0	14.28571429
140	0.986061931	95.6626506	06:29.1	0
141	0.991061926	95.83076492	05:50.2	16.66666667
142	0.992322981	95.44410199	05:22.7	14.28571429
143	0.98884958	96.19501261	05:11.0	4.761904762
144	0.989889383	97.9377977	05:09.1	11.9047619
145	0.985354006	96.07733259	03:51.7	4.761904762
146	0.983893812	96.32390025	04:08.4	11.9047619
147	0.984646022	96.33510787	04:28.8	9.523809524
148	0.987477899	96.38554217	04:41.8	11.9047619
149	0.985752225	93.23620062	04:41.5	4.761904762
150	0.991194665	97.08601849	03:39.5	16.66666667
151	0.990199089	97.90417484	03:28.7	7.142857143
152	0.990420341	97.36620902	03:21.7	9.523809524
153	0.991061926	98.69431213	03:18.7	19.04761905
154	0.988296449	97.0355842	03:02.5	2.380952381
155	0.982986748	96.64331746	03:20.7	2.380952381
156	0.985840678	94.34575511	03:46.3	4.761904762
157	0.980022132	95.79714206	03:55.6	2.380952381
158	0.986637175	95.05743906	03:54.3	9.523809524
159	0.987256646	97.5679462	02:53.2	0
160	0.989557505	97.45026618	02:43.8	11.9047619
161	0.990221262	97.91538246	02:39.0	9.523809524

162	0.990154862	97.05799944	02:37.9	9.523809524
163	0.984314144	95.45530961	02:47.5	2.380952381
164	0.983982325	95.46091342	03:04.4	4.761904762
165	0.985110641	96.6657327	03:27.1	4.761904762
166	0.983384967	95.19193051	03:38.4	7.142857143
167	0.983738959	96.99635752	03:38.4	7.142857143
168	0.988185823	98.05547772	02:37.7	9.523809524
169	0.988407075	97.54553096	02:26.5	14.28571429
170	0.988053083	97.36620902	02:20.2	2.380952381
171	0.988805294	99.08657887	02:21.2	4.761904762
172	0.984668136	96.5592603	02:47.5	4.761904762
173	0.982035398	98.16195013	03:05.3	2.380952381
174	0.981150448	96.66012889	03:27.6	2.380952381
175	0.982234538	95.6626506	03:39.8	0
176	0.985663712	94.39618941	03:38.6	9.523809524
177	0.987278759	97.89296722	02:36.5	4.761904762
178	0.987411499	98.59344354	02:24.9	14.28571429
179	0.989889383	98.71672737	02:21.4	7.142857143
180	0.987831831	98.25721491	02:23.7	4.761904762
181	0.983584046	95.41047913	05:09.4	2.380952381
182	0.981924772	93.44354161	05:28.2	4.761904762
183	0.984048665	91.9585318	05:39.4	2.380952381
184	0.978473425	88.12552536	06:17.0	4.761904762
185	0.981371701	95.3992715	07:12.1	7.142857143
186	0.984469056	94.79966377	04:57.8	7.142857143
187	0.991902649	98.94648361	04:30.6	9.523809524
188	0.988053083	95.81395349	04:05.4	16.66666667
189	0.989469051	96.0829364	03:28.4	9.523809524
190	0.982035398	95.70748109	05:10.7	7.142857143
191	0.982854009	90.1036705	05:29.4	0
192	0.982322991	90.49593724	05:42.8	2.380952381
193	0.972765505	91.73998319	06:19.0	0
194	0.979579628	91.6391146	07:13.2	7.142857143
195	0.98258847	96.83384702	04:57.7	9.523809524
196	0.985398233	97.8313253	04:34.7	4.761904762
197	0.986725688	97.99943962	04:06.3	7.142857143
198	0.987898231	96.47520314	03:30.2	4.761904762
199	0.986017704	95.91482208	05:30.2	4.761904762
200	0.979601741	93.63967498	05:49.9	0
201	0.978517711	90.44550294	05:59.1	0
202	0.978694677	93.37629588	06:38.9	0

203	0.981747806	90.0532362	07:33.3	2.380952381
204	0.985619485	96.78901653	05:21.4	11.9047619
205	0.985929191	96.88428131	04:52.8	7.142857143
206	0.988694668	94.96217428	04:22.7	4.761904762
207	0.989380538	96.01569067	03:49.5	7.142857143
208	0.984646022	97.43345475	05:35.3	7.142857143
209	0.980707943	91.57186887	05:54.1	2.380952381
210	0.980066359	91.71196414	06:03.2	2.380952381
211	0.981747806	93.11852059	06:44.9	11.9047619
212	0.980884969	92.15466517	07:38.9	0
213	0.985707939	96.15578593	05:25.0	4.761904762
214	0.983650446	96.9291118	04:55.8	4.761904762
215	0.98442477	95.88119922	04:27.6	2.380952381
216	0.985110641	97.50630429	03:50.5	9.523809524
217	0.985420346	94.24488652	04:28.9	0
218	0.984889388	93.7013169	04:47.4	0
219	0.98442477	95.30961054	04:58.2	7.142857143
220	0.986703515	95.14149622	05:42.1	4.761904762
221	0.985575199	92.00336229	05:41.7	2.380952381
222	0.987477899	94.6539647	04:18.9	9.523809524
223	0.988561928	97.70243766	03:56.9	11.9047619
224	0.988495588	95.68506584	03:27.3	14.28571429
225	0.98986727	97.53992715	03:25.5	7.142857143
226	0.985995591	96.54244887	03:15.9	9.523809524
227	0.986349583	95.12468479	03:35.2	7.142857143
228	0.984446883	94.37937798	03:55.9	7.142857143
229	0.985000014	93.10731297	04:20.9	11.9047619
230	0.982079625	94.99579714	04:19.9	11.9047619
231	0.987035394	96.89548893	03:06.5	7.142857143
232	0.988827407	97.43905856	02:55.0	7.142857143
233	0.988340735	98.12832726	02:34.1	9.523809524
234	0.988296449	97.76407957	02:33.1	7.142857143
235	0.983915925	95.39366769	02:50.6	2.380952381
236	0.982256651	96.66012889	03:08.2	7.142857143
237	0.97973454	95.48893247	03:32.0	0
238	0.983362854	94.83328663	03:56.8	9.523809524
239	0.983296454	96.42476884	03:55.2	2.380952381
240	0.988694668	96.19501261	02:41.5	4.761904762
241	0.986482322	96.23423928	02:29.1	7.142857143
242	0.986438036	97.99943962	02:19.0	4.761904762
243	0.986592948	97.96581676	02:17.8	14.28571429

244	0.984933615	97.13084898	02:45.5	4.761904762
245	0.980530977	95.32642197	03:03.9	2.380952381
246	0.978606224	95.4497058	03:26.0	0
247	0.979845107	96.77780891	03:47.7	9.523809524
248	0.982854009	95.53376296	03:47.7	2.380952381
249	0.983849585	97.78649482	02:35.9	9.523809524
250	0.984247804	96.16699356	02:23.5	14.28571429
251	0.984181404	97.27094424	02:14.5	7.142857143
252	0.985354006	97.64639955	02:30.3	9.523809524
253	0.980553091	95.92042589	03:12.0	4.761904762
254	0.979358435	95.10787335	03:24.4	4.761904762
255	0.981128335	92.19389185	03:44.9	0
256	0.980354011	96.39674979	03:59.2	11.9047619
257	0.982964575	96.47520314	03:57.9	4.761904762
258	0.986659288	97.69123004	02:54.7	4.761904762
259	0.986393809	96.34071168	02:41.7	14.28571429
260	0.983230114	96.88428131	02:31.1	7.142857143
261	0.984181404	98.21238442	02:34.5	9.523809524
262	0.984092891	96.45839171	05:00.2	9.523809524
263	0.986991167	93.89745027	05:35.2	2.380952381
264	0.984557509	95.37125245	05:21.0	0
265	0.98265487	94.26730177	06:00.1	4.761904762
266	0.982632756	94.25609414	05:59.8	11.9047619
267	0.987035394	96.70495937	04:34.8	4.761904762
268	0.989159286	96.42476884	04:08.7	4.761904762
269	0.990265489	98.67189689	03:40.3	9.523809524
270	0.990840733	97.7248529	03:41.2	9.523809524
271	0.985287607	93.93667694	03:22.5	14.28571429
272	0.984535396	93.10170916	03:45.1	2.380952381
273	0.982035398	94.25609414	04:10.6	4.761904762
274	0.985663712	93.89184646	04:32.6	11.9047619
275	0.98157078	93.4379378	04:26.8	0
276	0.985508859	97.74726814	03:08.9	7.142857143
277	0.988230109	98.58223592	02:56.9	4.761904762
278	0.987676978	98.520594	02:37.4	11.9047619
279	0.989092946	97.09722611	02:37.7	7.142857143
280	0.985752225	95.69627347	02:57.0	2.380952381
281	0.983584046	95.07425049	03:11.6	7.142857143
282	0.979601741	95.01821238	03:48.4	0
283	0.982765496	95.32081816	04:13.3	14.28571429
284	0.984004438	91.9585318	04:11.9	7.142857143

285	0.985774338	97.66881479	02:56.2	9.523809524
286	0.990265489	99.27710843	02:42.1	2.380952381
287	0.988473475	97.71364528	02:32.3	4.761904762
288	0.985354006	96.21742785	02:24.3	7.142857143
289	0.981438041	96.41916503	02:51.9	7.142857143
290	0.979822993	96.13337069	03:12.7	4.761904762
291	0.979137182	96.78901653	03:40.2	9.523809524
292	0.977632761	96.40795741	04:06.6	0
293	0.976460159	93.99271505	04:05.1	0
294	0.984601796	95.15830765	02:50.6	9.523809524
295	0.98530972	97.84813673	02:52.1	4.761904762
296	0.983805299	98.53740544	02:34.6	4.761904762
297	0.985575199	97.6744186	02:42.6	7.142857143
298	0.982942462	94.90613617	02:56.7	2.380952381
299	0.981194675	94.98458952	03:14.4	2.380952381
300	0.97966814	96.37433455	03:47.3	9.523809524
301	0.978274345	94.27850939	04:02.7	7.142857143
302	0.97980088	95.56178201	03:55.1	2.380952381
303	0.986769915	97.37181283	02:38.5	2.380952381
304	0.984115064	96.5592603	02:24.9	11.9047619
305	0.986393809	98.33006444	02:15.1	7.142857143
306	0.986681402	97.49509667	02:16.2	0
307	0.985641599	95.82516111	05:09.5	7.142857143
308	0.984778762	94.19445223	05:25.2	0
309	0.985840678	96.79462034	05:39.6	4.761904762
310	0.981438041	93.37069207	06:18.5	4.761904762
311	0.984092891	95.04623144	06:17.1	9.523809524
312	0.986858428	96.51442981	04:58.2	14.28571429
313	0.988274336	97.248529	04:35.6	9.523809524
314	0.991194665	96.68254413	04:03.5	7.142857143
315	0.993871689	97.2989633	04:04.4	0
316	0.986570776	95.62342393	03:34.6	7.142857143
317	0.983185828	96.64331746	03:52.6	0
318	0.980088472	95.75231157	04:14.0	7.142857143
319	0.983606219	94.90053236	04:37.2	9.523809524
320	0.984822989	94.17203699	04:37.6	11.9047619
321	0.98701328	94.14962174	03:21.7	2.380952381
322	0.98891592	96.25665453	03:11.6	0
323	0.989026546	96.49201457	02:52.6	4.761904762
324	0.987809718	98.26281872	02:52.7	9.523809524
325	0.985929191	96.41356122	02:57.4	4.761904762

326	0.985929191	93.34267302	03:14.0	7.142857143
327	0.982699096	94.04875315	03:37.7	2.380952381
328	0.981747806	95.36004483	04:01.6	9.523809524
329	0.980420351	91.8520594	04:01.3	0
330	0.986238956	96.94031942	02:46.7	11.9047619
331	0.988761067	98.24600728	02:34.6	9.523809524
332	0.986814141	97.04118801	02:25.8	11.9047619
333	0.990243375	96.78341272	02:24.6	0
334	0.983738959	98.10591202	02:48.9	11.9047619
335	0.981659293	96.35191931	03:05.8	4.761904762
336	0.975508869	94.64275707	03:30.2	4.761904762
337	0.977544248	95.52255534	03:51.6	7.142857143
338	0.983075202	97.29335948	03:51.3	7.142857143
339	0.982035398	92.51330905	02:39.3	14.28571429
340	0.986393809	98.13393107	02:27.7	11.9047619
341	0.983274341	96.0829364	02:17.0	7.142857143
342	0.982610643	96.43037265	02:17.0	7.142857143
343	0.982101798	96.16699356	02:52.2	0
344	0.980000019	95.03502382	03:07.7	4.761904762
345	0.98170352	97.80891006	03:31.3	4.761904762
346	0.978362858	94.09918745	03:52.8	2.380952381
347	0.978008866	94.91734379	03:52.6	0
348	0.986504436	96.58727935	02:39.2	7.142857143
349	0.986504436	95.9260297	02:29.9	4.761904762
350	0.985685825	96.16138975	02:18.1	9.523809524
351	0.984092891	97.5679462	02:17.4	2.380952381
352	0.986305296	96.23984309	05:16.2	11.9047619
353	0.988495588	93.23059681	05:31.4	2.380952381
354	0.987190247	95.03502382	05:45.6	4.761904762
355	0.984491169	92.32277949	06:20.0	0
356	0.98707962	94.80526758	06:21.9	9.523809524
357	0.987544239	96.6657327	05:01.7	2.380952381
358	0.987455726	97.29335948	04:38.0	11.9047619
359	0.991924763	96.44718409	04:05.6	7.142857143
360	0.990221262	97.04679182	04:09.7	2.380952381
361	0.987168133	93.3875035	03:38.8	2.380952381
362	0.984004438	96.4527879	03:58.2	16.66666667
363	0.982765496	93.30905015	04:18.7	7.142857143
364	0.981659293	92.70383861	04:42.0	2.380952381
365	0.980464578	95.60100869	04:42.4	4.761904762
366	0.989446878	98.00504343	03:30.4	4.761904762

367	0.984756649	95.83076492	03:15.1	7.142857143
368	0.988495588	96.13337069	02:57.8	2.380952381
369	0.987300873	98.22359204	02:56.3	2.380952381
370	0.980707943	97.21490614	03:02.8	9.523809524
371	0.981283188	95.72989633	03:19.0	9.523809524
372	0.979513288	95.00700476	03:42.3	9.523809524
373	0.97524339	91.17399832	04:07.0	2.380952381
374	0.986615062	95.1302886	04:05.6	9.523809524
375	0.985840678	99.0473522	02:52.8	0
376	0.990265489	95.7130849	02:41.8	7.142857143
377	0.986703515	98.22359204	02:30.7	4.761904762
378	0.98809737	98.34127207	02:33.8	4.761904762
379	0.980884969	98.04987391	02:53.2	11.9047619
380	0.971769929	94.26730177	03:10.8	0
381	0.978318572	93.14093584	03:32.8	2.380952381
382	0.97157079	92.42364808	03:55.4	2.380952381
383	0.980553091	91.94732418	03:54.0	0
384	0.984203517	96.7217708	02:44.0	2.380952381
385	0.98272121	95.69066966	02:51.2	14.28571429
386	0.987787604	96.9851499	02:36.3	7.142857143
387	0.984491169	96.73858224	02:36.5	7.142857143
388	0.98170352	97.51751191	02:59.8	19.04761905
389	0.97688055	94.86130569	03:28.8	11.9047619
390	0.97789824	96.06612496	03:36.5	9.523809524
391	0.982809722	93.34267302	03:56.5	7.142857143
392	0.977699101	94.39618941	03:57.7	4.761904762
393	0.985044241	96.35191931	02:42.3	0
394	0.986216843	96.34071168	02:29.7	4.761904762
395	0.985619485	97.54553096	02:21.0	4.761904762
396	0.985553086	96.71056318	02:23.4	2.380952381

Appendix 2: Variable Refinement Results

Test Number	dropout 1	dropout 2	optimizer	Train Accuracy	Test Accuracy	Learning Time	Internet Accuracy
1	0.2	0.2	Adagrad	0.979159	97.65761	03:54.2	9.52381
2	0.2	0.2	SGD	0.994137	96.03811	02:29.1	0
3	0.2	0.2	RMSprop	0.993518	97.51751	04:13.1	19.04762
4	0.2	0.2	Adam	0.99146	98.58224	03:12.3	19.04762
5	0.2	0.3	Adagrad	0.97531	96.42477	03:52.8	14.28571
6	0.2	0.3	SGD	0.99458	97.14766	02:27.4	0
7	0.2	0.3	RMSprop	0.994071	98.52059	04:11.1	2.380952
8	0.2	0.3	Adam	0.993341	98.27963	03:13.9	4.761905
9	0.2	0.4	Adagrad	0.975531	97.19809	03:50.6	9.52381
10	0.2	0.4	SGD	0.99292	97.18689	02:28.0	2.380952
11	0.2	0.4	RMSprop	0.993186	98.03867	04:11.7	9.52381
12	0.2	0.4	Adam	0.992589	97.94901	03:11.0	2.380952
13	0.2	0.5	Adagrad	0.953296	96.07733	03:50.2	4.761905
14	0.2	0.5	SGD	0.98792	96.85626	02:27.5	0
15	0.2	0.5	RMSprop	0.99135	96.7666	04:12.1	4.761905
16	0.2	0.5	Adam	0.990022	98.10591	03:12.1	0
17	0.3	0.2	Adagrad	0.975708	96.06052	03:51.5	7.142857
18	0.3	0.2	SGD	0.995465	97.33819	02:27.0	0
19	0.3	0.2	RMSprop	0.992168	97.52872	04:11.2	11.90476
20	0.3	0.2	Adam	0.989358	97.00196	03:12.1	7.142857
21	0.3	0.3	Adagrad	0.961394	95.5954	03:50.5	4.761905
22	0.3	0.3	SGD	0.992655	96.15579	02:27.4	0
23	0.3	0.3	RMSprop	0.991571	98.16195	04:12.4	14.28571
24	0.3	0.3	Adam	0.991018	97.47829	03:12.2	4.761905
25	0.3	0.4	Adagrad	0.969912	96.761	03:50.9	4.761905
26	0.3	0.4	SGD	0.989137	96.27347	02:27.4	4.761905
27	0.3	0.4	RMSprop	0.991527	97.99384	04:11.9	19.04762
28	0.3	0.4	Adam	0.989381	97.28215	03:12.5	7.142857
29	0.3	0.5	Adagrad	0.950708	96.25105	03:51.2	7.142857
30	0.3	0.5	SGD	0.987235	95.8756	02:27.5	4.761905
31	0.3	0.5	RMSprop	0.989889	98.92407	04:10.9	9.52381
32	0.3	0.5	Adam	0.989226	97.98823	03:13.0	9.52381
33	0.4	0.2	Adagrad	0.966571	95.83076	03:50.9	2.380952
34	0.4	0.2	SGD	0.991748	94.87812	02:28.4	2.380952
35	0.4	0.2	RMSprop	0.992323	98.37489	04:10.5	9.52381
36	0.4	0.2	Adam	0.989049	97.21491	03:11.9	4.761905
37	0.4	0.3	Adagrad	0.957013	96.06052	03:51.2	4.761905

38	0.4	0.3	SGD	0.990907	96.50883	02:28.1	0
39	0.4	0.3	RMSprop	0.991305	98.39731	04:12.5	11.90476
40	0.4	0.3	Adam	0.989027	98.54861	03:12.1	11.90476
41	0.4	0.4	Adagrad	0.950221	96.54245	03:50.6	7.142857
42	0.4	0.4	SGD	0.989027	97.24853	02:28.4	2.380952
43	0.4	0.4	RMSprop	0.990022	98.58784	04:11.5	7.142857
44	0.4	0.4	Adam	0.989093	98.14514	03:13.0	4.761905
45	0.4	0.5	Adagrad	0.932633	96.25105	03:51.2	4.761905
46	0.4	0.5	SGD	0.987677	97.39423	02:27.7	4.761905
47	0.4	0.5	RMSprop	0.988805	98.32446	04:12.3	9.52381
48	0.4	0.5	Adam	0.989823	97.90417	03:13.1	11.90476
49	0.5	0.2	Adagrad	0.949823	94.5643	03:51.2	4.761905
50	0.5	0.2	SGD	0.990044	96.83385	02:27.5	7.142857
51	0.5	0.2	RMSprop	0.990619	98.14514	04:11.5	11.90476
52	0.5	0.2	Adam	0.989491	98.59344	03:11.5	9.52381
53	0.5	0.3	Adagrad	0.929513	95.06865	03:50.9	2.380952
54	0.5	0.3	SGD	0.989712	97.31577	02:27.4	2.380952
55	0.5	0.3	RMSprop	0.98885	97.60717	04:09.9	7.142857
56	0.5	0.3	Adam	0.988142	97.37742	03:12.7	11.90476
57	0.5	0.4	Adagrad	0.914403	92.63099	03:51.2	7.142857
58	0.5	0.4	SGD	0.98469	96.64892	02:27.1	0
59	0.5	0.4	RMSprop	0.988717	98.54861	04:10.4	7.142857
60	0.5	0.4	Adam	0.987876	97.94901	03:15.0	7.142857
61	0.5	0.5	Adagrad	0.907588	94.37938	03:51.5	7.142857
62	0.5	0.5	SGD	0.977633	96.464	02:27.3	0
63	0.5	0.5	RMSprop	0.988761	96.49762	04:11.5	7.142857
64	0.5	0.5	Adam	0.984889	97.96021	03:16.1	7.142857

Appendix 3: Internet Sourced Images Predictions

path	target	prediction	Prediction Percent	Target Prediction Percent
fruits/internet_resized\Apple_Red_1.jpg	Apple Red	Apple Pink Lady	0.654491544	8.57E-12
fruits/internet_resized\Apple_Red_2.jpg	Apple Red	Pear Red	0.999998689	7.81E-17
fruits/internet_resized\Apple_Red_3.jpg	Apple Red	Cherry	0.603045106	0.328073
fruits/internet_resized\Avocado_1.jpg	Avocado	Avocado ripe	0.760063231	7.67E-08
fruits/internet_resized\Avocado_2.jpg	Avocado	Avocado ripe	0.601834297	6.93E-06
fruits/internet_resized\Avocado_3.jpg	Avocado	Pepper Green	0.996228456	8.34E-08
fruits/internet_resized\Banana_1.jpg	Banana	Physalis with Husk	0.914720714	0.001867
fruits/internet_resized\Banana_2.jpg	Banana	Pear Monster	0.495245576	0.00409
fruits/internet_resized\Banana_3.jpg	Banana	Banana	0.503727078	0.503727
fruits/internet_resized\Cantaloupe_1.jpeg	Cantaloupe	Cantaloupe	0.997193277	0.997193
fruits/internet_resized\Cantaloupe_2.jpg	Cantaloupe	Cantaloupe	0.977306902	0.977307
fruits/internet_resized\Cantaloupe_3.jpg	Cantaloupe	Cantaloupe	0.94961828	0.949618
fruits/internet_resized\Cherry_1.jpg	Cherry	Pear Red	0.999910116	1.29E-09
fruits/internet_resized\Cherry_2.jpg	Cherry	Pear Red	0.999989748	9.63E-08
fruits/internet_resized\Cherry_3.jpg	Cherry	Pear Red	0.999990463	7.71E-17
fruits/internet_resized\Grape_Pink_1.jpg	Grape Pink	Pepper Green	0.999930739	1.70E-17
fruits/internet_resized\Grape_Pink_2.jpg	Grape Pink	Strawberry Wedge	0.999443471	4.23E-14
fruits/internet_resized\Grape_Pink_3.jpg	Grape Pink	Cherry Rainier	0.671511292	1.34E-10
fruits/internet_resized\Grape_White_1.jpg	Grape White	Physalis with Husk	0.999682069	6.91E-13
fruits/internet_resized\Grape_White_2.jpg	Grape White	Physalis with Husk	0.634479284	2.93E-10
fruits/internet_resized\Grape_White_3.jpg	Grape White	Cantaloupe	0.947030187	2.27E-08
fruits/internet_resized\Lemon_1.jpg	Lemon	Carambula	0.999988437	1.73E-14
fruits/internet_resized\Lemon_2.jpg	Lemon	Quince	0.705528498	2.36E-10
fruits/internet_resized\Lemon_3.jpg	Lemon	Banana Lady Finger	0.99985528	6.42E-10
fruits/internet_resized\Orange_1.jpg	Orange	Carambula	0.854445279	8.52E-15
fruits/internet_resized\Orange_2.jpg	Orange	Lemon Meyer	0.846696794	1.66E-14
fruits/internet_resized\Orange_3.jpg	Orange	Pepper Green	1	4.33E-25
fruits/internet_resized\Pear_Red_1.jpg	Pear Red	Pear Red	0.949800968	0.949801
fruits/internet_resized\Pear_Red_2.jpg	Pear Red	Pepper Green	1	6.76E-21
fruits/internet_resized\Pear_Red_3.jpg	Pear Red	Granadilla	0.999974847	2.31E-08
fruits/internet_resized\Pepper_Green_1.jpg	Pepper Green	Pepper Green	1	1
fruits/internet_resized\Pepper_Green_2.jpg	Pepper Green	Banana Lady Finger	0.998981297	0.000441
fruits/internet_resized\Pepper_Green_3.jpg	Pepper Green	Pepper Green	1	1
fruits/internet_resized\Pineapple_1.jpg	Pineapple	Banana Lady Finger	0.996977329	6.17E-09
fruits/internet_resized\Pineapple_2.jpg	Pineapple	Banana Lady Finger	0.539131999	2.69E-08
fruits/internet_resized\Pineapple_3.jpg	Pineapple	Carambula	0.902884543	9.56E-09
fruits/internet_resized\Strawberry_1.jpg	Strawberry	Pear Red	0.99985528	4.23E-08
fruits/internet_resized\Strawberry_2.jpg	Strawberry	Pear Red	0.999899387	7.19E-13

fruits/internet_resized\Strawberry_3.jpg	Strawberry	Pear Red	1	2.90E-11
fruits/internet_resized\Tomato_1.jpg	Tomato	Tomato	0.923478901	0.923479
fruits/internet_resized\Tomato_2.jpg	Tomato	Pear Red	1	9.99E-25
fruits/internet_resized\Tomato_3.jpg	Tomato	Pear Red	0.999998689	6.71E-11