Fruit Image Classification Using Convolutional Neural Network

Kai Tai Tam

Definition

Ever since the beginning of human existence, fruit have always been considered as healthy food in human diet. There are thousands of diets menu available online; regardless of the purpose of the menus, almost all of them suggest including at least one type of fruits in the diet. The nutritional values of fruits are irreplaceable for human. In fact, the United States spent $6.6 billion dollars on imported fruit in 2015, while they only spent $1.8 billion dollars in 1995 [1]. It shows that American have started to pay more attention to their health, as well as the nutritional value of fruits.

As there are many fruits available in the market today, consumers may not know every fruit and their nutritional value. Nowadays, to learn more about anything you imagined, using a search engine would be a smart choice. However, you would need to know the fruit's name to complete this task. It would definitely be more convenient for people, say consumers in the market, to simply take a picture with their phone and it would know what fruit is in the picture.

Muresan and Oltean [2] conducted a study similar to this problem. They trained a convolutional neural network model on a "dataset of images containing fruits" in order to develop a model that was able to identify fruits from images. They collected data by taken pictures themselves using a camera. They took them in a close distance and removed the background using software technology. While they have an impressive result: achieving more than 90% accuracy in the test set, their data would only be valid if the pictures were taken in a certain angle and distance. An image of a grape, for example, was taken much closer than an image of an orange. It is difficult for general people to utilize this model, because not many people would pull out one particular grape, take a picture in a very close distance, and remove their background. Therefore, their dataset was not applicable for my purpose. I will be constructing a model that would recognize fruits based on pictures that a normal individual would take on their cell phone.

The goal for this paper is to develop a supervised learning algorithm that is able to predict a fruit based on a fruit image provided by users. To achieve such a goal, utilizing image recognition technique is a great choice. As image recognition become more popular, it is possible to train a model to recognize fruits.

As mentioned earlier, there was one big weakness in Muresan and Oltean's study[2]: their inputs were not able to generalize to public. In order to prevent this issue, I downloaded fruit images from Google. In addition, I made sure each label had enough images, with no less than 100 for each label, passing to the training model. I then resized and standardized those images to 100x100 pixels, and randomly separated the dataset into three sets: training set,

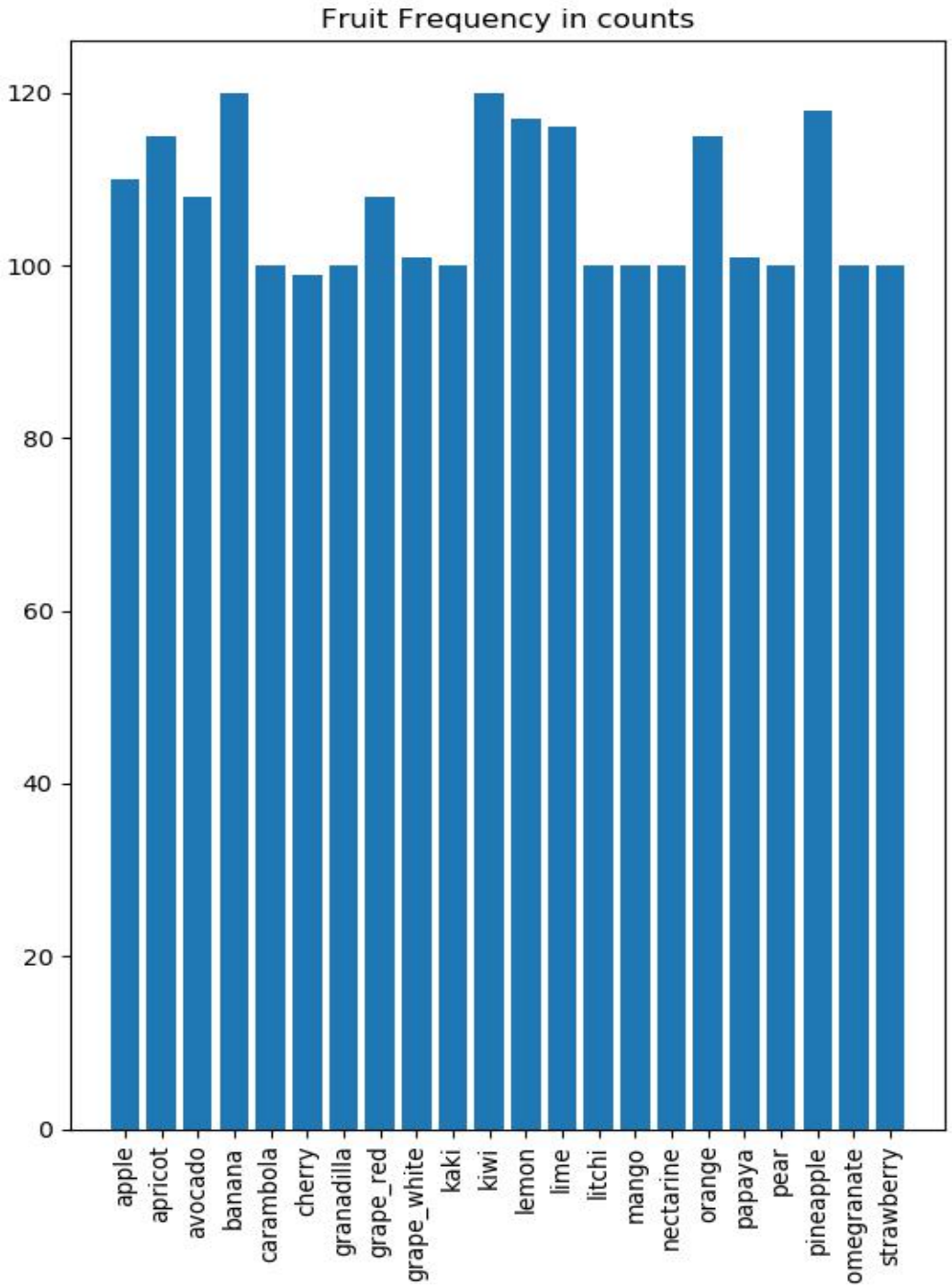
validation set, and test set. I then passed the training set to Convolutional Neural Network in order to train it to make predictions on fruit labels. I used accuracy to keep track how well the model was learning. Accuracy score is a great options for this problem because it shows how often the model predicted correctly. The greater the accuracy is, the higher probability that the model makes accurate prediction. Accuracy was calculated by: the number of corrected labels divided by the total labels. In other words, accuracy indicated the percentage of the model predicting the correct answer.

Analysis

In order to train a model to recognize fruit images, it is essential to obtain fruit image data, along with their names. I utilized an image extraction tool developed by Hardik Vasa [3] and created my own dataset by searching images on Google. This tool downloaded images on Google based on my keywords, and each fruit label had its own folder. I chose 22 fruits from a list of fruits in Wikipedia [4] and searched for their images on Google using the image extraction tool. Even though I tried as precise as possible to obtain images of fruits, there were images that did not match what I searched for, such as searching for oranges when the image only showed the orange color instead of the actual fruit.

Figure 1: Orange shown as Cartoon, and Color

There are approximately 100 images for each category, a total of 2,348 images. The uniform distribution of the data made sure it was balanced and had enough data for validation set and test set.



According to the image above, Kiwi and banana have the most images, both having 120, while Cherry has the fewest, with 99 images. However, the total image count for each label are similar, making it roughly a uniform distribution. The mean of the labels is 106.7, and the standard deviation is 8.0.

Some images can be potentially confusing to the model during training. To exemplify, one of the major difference within each label is inconsistency of the number of fruits. Some images can contain only one pineapples, while other pineapple images can contain dozens. It is important for the model to learn whether there are only one pineapple or multiple pineapples, the label still belongs to pineapple.

Figure 2: Left: image with multiple pineapple, Right: image with one pineapple

When humans identify an object, one of their strategies is to analyze their color, shape, texture, etc. Similarly, Convolutional Neural Network is capable of doing such tasks, and that is why I believe CNN is the right choice for this task. CNN is a deep learning algorithm that is able to preserve spatial relationship between pixels [5]. It starts off with a convolutional layer, which is square window, or kernel, usually 2x2 or 3x3. Each point in that window contains a weight. For every possible pixels window, CNN would calculate the sum of the product of each pixel*weight, and pass it to the activation function. In other words, when I pass on the images to CNN, it computes values based on weights and scans through all possible pixels for that layer, based on the size of the kernel. As it trains the model, the weight will be updated based on which nodes are useful in making a correct prediction.

In addition, I also used max pooling layer in order to decrease the pixel size, after each convolutional layer. The filter searched through all pixels similar to the process of convolutional layer discussed above. Instead of calculating the weight, the filter selected the maximum value among the window with the given size. Max pooling layer is capable to reduce the dimensionality and only keep the most useful pixels. I also included dropout layers between the layers. Dropout layer performed training with some nodes disabled for that layer in order to prevent certain nodes from having too much weight. It is commonly used to reduce the risk of

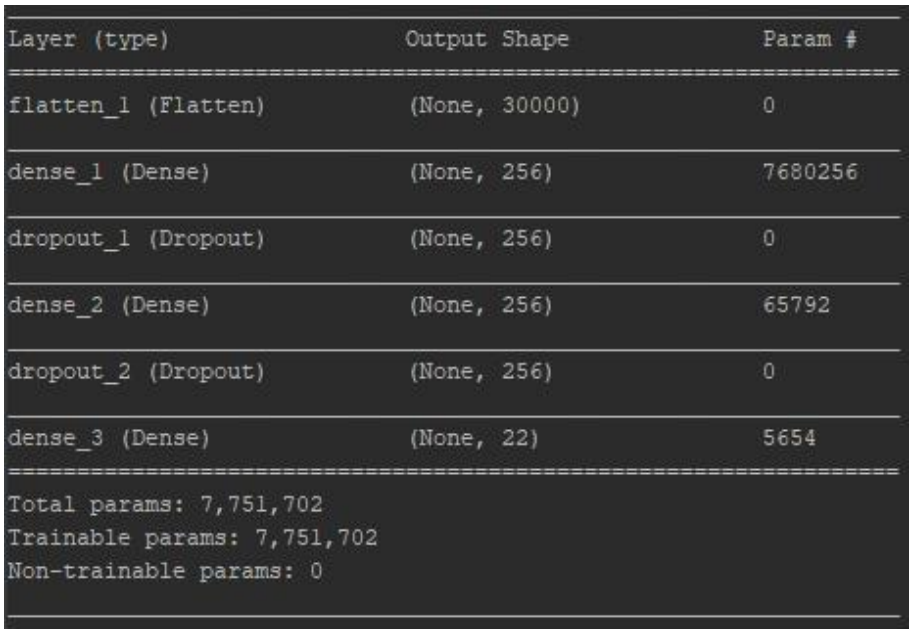
overfitting. I also utilized global average pooling in order to pull the features that the model found useful. Global average pooling layer reduced the dimension to 1 x 1 x depth, where depth is the features the model found useful in predicting labels [6]. Moreover, I added a final layer using “softmax” as the activation. “Softmax” is selected as the final activation because it is structured such that the sum of the probability of each labels are 1.

In addition to building a convolutional neural network model from scratch, I also implemented a pre-trained model in keras using transfer learning. Since training a CNN is time consuming and does not always provide the expected result, using pre-trained model could save us times and could potentially perform better result than the CNN model from scratch.
When training the model, I used categorical cross-entropy loss and accuracy in order to record model performance. Categorical cross-entropy loss is best used when constructing a multi-class classifier. It checks whether the model has done well when classifying a label by comparing to the true label. It returns a lower value if the model has done well in predicting labels. Categorical cross-entropy loss is perfect in this case because each image in the dataset can only belong to one of the twenty two categories. If the model did a great job in predicting fruits, the score for categorical cross-entropy would be low.

I use accuracy to keep track how well the model learned by looking at the accuracy on training set, and how accurate the model was by checking the accuracy on the validation set. It is calculated by the number of corrected labels divided by the total labels. The higher the accuracy in validation set, the better the result in general. As the number of epochs increases, the accuracy in training set and validation set will also increase. paying attention to the accuracy for validation set is essential, as when the accuracy validation set decreases and increases in training set, it maybe a sign of overfitting.

Since the most important concern for my model is to be accurate, it is important to show that the model actually perform better than random guessing. Since there are 22 categories and each category is distributed uniformly, the probability of predicting a correct answer is 1 in 22, or 4.5%. The model should pass 4.5% to indicate that it has in fact learned, or picked up some features, through training.

In addition to random guessing, a benchmark neural network model is also implemented in order to test whether Convolutional Neural Network is a reasonable choice for this problem. I first flatten the images in order to pass them to a neural network model I then added two hidden layers, both with 256 nodes and “ReLU” activation. After each hidden layer, I added a dropout layer of 0.3 in order to prevent from overfitting. I lastly added a final layer which is the same as the number of fruit labels, using “softmax” as the activation. It turned out the validation accuracy for the benchmark model is 5.97%, and the architecture of the model is shown below:



Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 30000)	0
dense_1 (Dense)	(None, 256)	7680256
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65792
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 22)	5654

Total params: 7,751,702
Trainable params: 7,751,702
Non-trainable params: 0

Methodology

As mentioned in the definition section, there are images that do not match what I intended to search for. To deal with this issue, I cleaned up the data manually by going through all images and removing the images that did not contain the fruit at all. I did, however, leave the images the same whether there are multiple fruits or one single fruit.

After the images were resized and standardized, I randomly distributed the dataset into three sets: training set, validation set, and test set. The training set had 80% of the data, while validation had 15% and the test set had the remaining 5%. This ratio is reasonable because given the small size of the data, the model still have enough information for training, while 15% as validation and 5% as final test for performance.

I then passed the training set that was resized to 100 x 100 pixels, to CNN, using “ReLU” as the activation, and increased the depth to 16. I then used max pooling layer,with 2x2 filter and stride of 2, in order to decrease the pixel size by half, after each convolutional layer. I also included dropout layers between the layers to prevent overfitting, and global average pooling layer to pull the features that the model found useful. Lastly, a final layer was added with 22 nodes and “softmax” as activation. The model architecture is shown below:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 100, 100, 16)	208
max_pooling2d_1 (MaxPooling2	(None, 50, 50, 16)	0
conv2d_2 (Conv2D)	(None, 50, 50, 32)	2080
max_pooling2d_2 (MaxPooling2	(None, 25, 25, 32)	0
conv2d_3 (Conv2D)	(None, 25, 25, 64)	8256
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 64)	0
dropout_3 (Dropout)	(None, 12, 12, 64)	0
global_average_pooling2d_1 (	(None, 64)	0
dense_4 (Dense)	(None, 22)	1430

Total params: 11,974
Trainable params: 11,974
Non-trainable params: 0

For the pre-trained model, I implemented a pre-trained model: VGG16, in order to test whether a pre-trained model perform better than a model developed from scratch. VGG16 is a CNN architecture that is "considered to be an excellent vision model," and was "used to win the ImageNet competition in 2014."[7] The last five layers or all layers are set to be untrainable due to our small dataset. I added one fully connected layer with 1024 nodes and "ReLU" as activation, and a final layer with 22 nodes and "softmax" as activation.

I created some functions in order to further reduce and organize my code. For example, "reading_images" took fruit image locations as input. It performed several tasks as follow: resizing all images to 100x100x3 pixels, standardizing images to range 0 and 1, and storing each image and label into two lists, treating them like variables and labels. This function not only read .jpg file, but it also read .png and .jpeg files. Writing this function saved me tremendous time when reading input from folders to folders.

In addition, I also created a function that helped me tune the model. This function took whatever that was provided and train the model. This was very handy as I did not need to re-train the model every time I plan to tune it.

There are many techniques when tuning the model. One of the most effective techniques is grid search. Grid search is an "exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm."[8] In my case, I chose to adjust learning rates to 0.1, 0.01, or 0.001, as well as two optimizers: RMSprop and SGD. In other words, there were six different combinations as it generates a 2x3 table. I would use validation accuracy and select the model with the highest value as the best model for training from scratch.

For tuning the pre-trained model, I utilize grid search again, but with different parameters: instead of adjusting the optimizers, I set the optimizer to be RMSprop across all models and restrict the number of trainable layers: none or last five layers. Similarly, the available learning rates are 0.1,.0.01, and 0.001.

For tuning both models, I set the batch size to be 100 and maximum of epochs as 50. I applied checkpoint and early stop, with patience of 5. Checkpoint stored the model with the highest validation accuracy, while early stop would stop training when the validation accuracy did not pass the past five training, or epochs.

The tuning result for the model trained from scratch is shown below:

	RMSprop	SGD
0.1	3.12%	35.51%
0.01	56.53%	5.68%
0.001	36.93%	5.68%

For the table above, RMSprop performance vary across all levels. However, the best performance for RMSprop is significantly better than SGD optimizer, at 0.01 learning rates. While RMSprop only predicted 3.12% accuracy at 0.1 learning rate, SGD performed its best performance at such learning rate, with 35.51% accuracy. Moreover, when looking at SGD alone, learning rate of 0.01 and 0.001 did not show a sign of learning as the validation accuracy did not improve. On the other hand, RMSprop generally performed a much better task in terms of validation accuracy when the learning rate is less than or equal to 0.01. Even though RMSprop only performed 36.91% validation accuracy when learning rate was at 0.001, it was still better than the best result of SGD.

The tuning result for the VGG16 pre-trained model is shown below:

	Trainable: None	Trainable: Last 5
0.1	3.12%	3.98%
0.01	6.25%	4.55%
0.001	77.84%	4.55%

The pre-trained model performed worse than the model trained from scratch in general. However, the best model recorded in the pre-trained model had the highest accuracy across all models, with close to 78%. The pre-trained recorded the worst result at learning rate 0.1 for non trainable group, while its best performance reached 77.84% accuracy on validation set when learning rate is 0.001. On the other hand, when the model was enable to train the last five layers, the model did not perform well at all, with the highest accuracy of 4.55%.

Results

Both models did a fantastic job on predicting fruits, the highest validation accuracy recorded for model trained from scratch was 56.53%, while the highest recorded for pre-trained model was 77.84%. The pre-trained model definitely performed better if correctly selecting the learning rate and the number of trainable layers, with over 20% higher accuracy than the model trained from scratch. Moreover, the final model also performed significantly better than the benchmark model, with almost 72% improvement. Therefore, the pre-trained model with non-trainable layers and learning rate of 0.001 is selected to be my final model.

The final model has a total of 23 layers, where 19 of them are set to be untrainable. All the untrainable weights are used directly from VGG16. It shows that VGG16 does an excellent job in generalizing the weights to unseen data. This is the beauty of transfer learning: the ability to reuse the weights and still perform at a high level even when the input is truly new. The only trainable layers are the last two layers, which are one fully connected layer, with 1024 nodes and "ReLU" activation, and a final layer with 22 nodes and "softmax" activation.

Applying the model to the test set can test whether the model is robust enough for the problem. Five percent of the data was pulled out and saved in the before the training process begin. Out of 118 images, we recorded 98 correct prediction, or 83.1%. The model seems to do a decent job on predicting the fruits in the test set, meaning that it is not overfitting and is robust enough to generalize the data to unseen images.

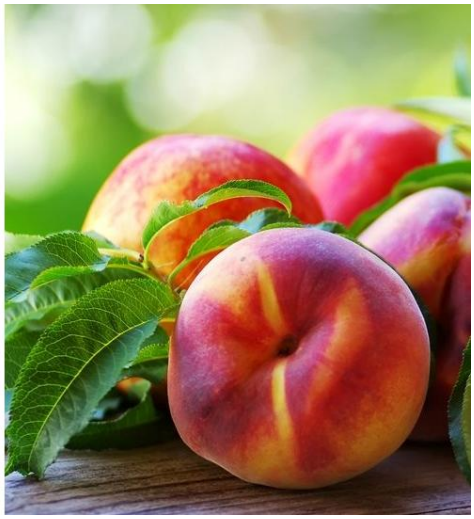


Figure 3: Granadilla predicted as granadilla Nectarine predicted as apricot

Convolutional neural network definitely did a great job in predicting fruits, as it is almost 74% more likely to get to the correct prediction compare to random guessing. In terms of the benchmark model, CNN also clearly outperform regular neural network as well. With closed to 80% accuracy, I believe the model is significant enough to have solved the problem. In fact, If the data had more images, I believe it would perform even better.

Conclusion

Given the research problem is to develop an algorithm that predicts a fruit based on a fruit image provided by users, using images directly from Google might not be the perfect choice. Therefore, to check whether using images from Google is able to generalize and to be robust enough for the problem, I have taken several pictures myself and pass it to the model. The result is shown below:


Figure 4: Apple is predicted as Apple Banana is predicted as Banana


Figure 5: Orange is predicted as Kiwi Orange is predicted as orange

According to the self-taken image above, the model did a great job in predicting the fruit in self-taken pictures. My model correctly predicted three fruits out of four images, which is 75% accuracy. Since recognizing an image is never an easy task to do, it is fascinating to observe what a computer can do with only training for less than 10 minutes. It might even be difficult to human to identify 22 different objects if those objects are truly new to them. I am definitely satisfied with the final model, as the solution fit my expectation and even showed on the self-taken pictures.

There are some improvements that could be made for this project. One of the improvements is to further adjust the learning rate. As we can see in the grid search, learning rate varied a lot from 0.1 to 0.001, meaning that if we could further tune more precisely, we could have got a even better result.

In addition, since this project demonstrated that using transfer learning can achieve a better performance than a model trained from scratch, there are other pre-trained model architecture that could be used, such as VGG19, that might perform even better than VGG16. One technique that I would implement if I knew how it worked and the data allowed is the ability to identify multiple fruits in one image. In real life, it is common that people take pictures with more than one fruit. It would be awesome if my model was able to detect all fruits shown in one single image. It is, however, very hard to train for this dataset, as this dataset is structured in a way that only identify one label for one image.

Given there can be improvements for this model, I think a better solution will exist as I cannot deny other pre-trained model when I haven't even try them. Therefore, if I continue to work on this project in future, I would definitely use the current best model as my new benchmark.

To summarize, Convolutional neural network is a great approach in predicting a fruit based on a fruit image. Downloading images from Google using Hardik Vasa's tool definitely saved me a tremendous of time for the dataset. The resizing and standardizing images also speeded up the training time while not losing much of the detail in images. Convolutional neural network definitely demonstrated its ability to handle images as the accuracy significantly improved after the comparison between the benchmark model and the Convolutional neural network approach.

Reference

1. https://fas.org/sgp/crs/misc/RL34468.pdf
2. https://github.com/Horea94/Fruit-Images-Dataset
3. https://github.com/hardikvasa/google-images-download
4. https://simple.wikipedia.org/wiki/List_of_fruits
5. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
6. https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
7. https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html
8. https://en.wikipedia.org/wiki/Hyperparameter_optimization