Fruit Classification Using Convolutional Neural Networks

Liam Eynan, liam.eynan@richmond.edu
Stephen Owen, stephen.owen1@richmond.edu

1 Abstract

Deep learning has become one of the dominant fields of computer science, and convolutional neural networks have become a favorite tool of those practicing deep learning. For this project, we created a model (that we have decided to name F.I.C.T.I.O.N. which stands for Fruit Image Classification That Is Ostensibly Nutritious)[1], that as the name would imply, able to take in pictures of various fruits (and vegetables!) and classify them using a convolutional neural network. Using the Fruit-360 data set as our training data and Keras as our back end, our model is able to achieve a 98.84% testing-accuracy. We have also implemented a system for uploading user images, modifying them to achieve a similar format to the testing data, and classify them on a one-off basis.

2 Introduction

You can rarely go anywhere in the computer science space without seeing something relating to machine learning, be it in the academic or, more recently, the industrial front (as evident by the astounding number of "machine learning start-ups", and the prevalence of buzz words like "deep learning," "neural network," "Artificial Intelligence," and thinks of the like). This is, however not without good reason. Machine learning has completely changed the way we approach certain computational tasks and has opened up countless advances in technology that were previously thought impossible.

At its core, machine learning is advanced applications of procedures from statistics, and many algorithms in the machine learning space are almost directly taken from statistics. Take regression, for example. Regression is a statistical method of parameter estimation in which you make an attempt to fit some function to a data set (the function used is depending on the generalized shape of the data, common ones are linear, logistic, and quadratic) and use this resulting function to make predictions about other potential data points that may lie outside of the observed data. As machine learning grew as a field, there was a widespread obsession over these new processes called "artificial neural networks."

Using these artificial neural networks involved using other algorithms to extract the "features" we look for in a data set, and then feeding the information extracted from that into a "network" of nodes that would apply various linear-algebra functions to the input and eventually give you an output. This output is compared to the output of the training data, and then the network would go back and correct itself if it preformed sub-optimally (more on this later). As people began to play with these more and recognize their computational power, it was realized that not only were neural networks better at the actual computations, but were also better at doing the feature extraction step as well. This process of using neural networks for both learning and feature extraction brought with it the widespread fascination and desire to use what would be named deep

[1] This name is a nod at old Cartoon Network show *Kids Next Door*, where the names of inventions made on the show always had over the top names that formed an acronym.

learning wherever possible.
As people studied deep learning more, people found certain network structures excelled at certain tasks. Convolutional neural networks became one of the go-to applications of deep learning because of it's ability to handle visual/image-based data in a highly intuitive and efficient way.
This kind of work has numerous practical applications in the real world (as evident by the fact that it seems like you can't go anywhere in the tech industry without hearing the words "machine learning"). Jokes aside, machine learning, specifically classification algorithms, have proven to be extremely useful tools in our society. They are used for medical diagnostics, autonomous cars, fraud detection, and so many other applications. While building an algorithm specifically designed for classifying fruit may not necessarily be the most "useful" program in the world, the core technology behind it is the foundation of many solutions to the worlds growing list of computing problems.

3 Related Work

One of the central uses for CNN based learning in the past has been visual classifiers, such as this very project. Sakib, Ashrafi, and Sidique wrote "Implementation of Fruits Recognition Classifier using Convolutional Neural Network Algorithm for Observation of Accuracies for Various Hidden Layers," a paper on a similar project, and their previous work was greatly informative in creating our own classifier. A related field is that of image segmentation, such as separating layers of images and locating the desired subject. The development of CNN image segmentation strategies summarized by Sultana, Sufian, and Dutta's "Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey." By combining the two techniques, an AI that can locate and identify fruits of unknown location could be made. Of course, CNN classifiers are not only useful for fruit identification. The same idea behind fruit classification, an image classifier, can be used in plenty of real-life situations. One common and important of these is medical diagnosis. CNN classifiers have been used to diagnose and find the severity of diseases such as diabetic retinopathy and pulmonary tuberculosis, by using medical imaging technology to get a visual of the affected area and using the teaching the model to classify severity. Similar classification strategies have been used to make great strides in natural language processing, as described in "Very Deep Convolutional Networks for Text Classification" by Conneau, Schwenk, Cun, and Barrault.

4 Formulation

This project is formulated as a classification problem, so we use deep learning to accomplish this. We opt to use deep learning because deep neural networks automate the extraction of image features, which simplify design and accelerate model construction. More specifically, we train and use a multi-layered convolutional neural network because of their dominance and universal acceptance as the gold standard for handling visual data.
As mentioned, using a deep-learning based approach simplifies the design, because it allows the network to automatically extract features from the data set on its own without any direction from us.

5 Approach

5.1 Data Set

We use the Fruit-360 data set for our model.[2] This data set provides 120 different types of labeled fruits (and vegetables) with a white background (which ended up being a hindrance, and will be discussed in the conclusions) in a standardized 100 $\times$ 100 pixel format. This data set contains a total of 82213 images showing off the fruits at various different angles. We used 60498 of these images as our training set and 20622 as our testing set (the rest of the images were a random assortment of images that were not in the standardized format, such as pictures of multiple types of fruit together), which represents an approximately 75% train/test split of the data.

[2]Available for free at https://www.kaggle.com/moltean/fruits courtesy of Mihai Oltean

Figure 1: Standardized Strawberry 1

Figure 2: Standardized Strawberry 2

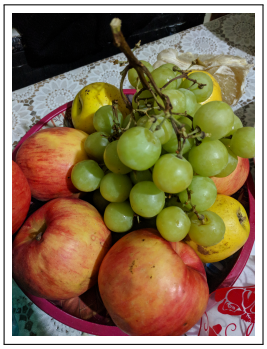


Figure 3: An example of a "non-standardized" photo

The Above photos highlight the benefit of using a data set like this. For each fruit or vegetable, there are at least 150 images showing the fruit in multiple different angles, which aids in preventing model over-fitting, because it learns to recognize both of these images as "Strawberries" rather than only being able to categorize something that looks like, say, figure 2, for example.

5.2 Language & Tools

5.2.1 Python

We did all of our programming in Python 3.7 and interfaced with the language through the use of Jupyter notebooks. We chose this because Python is the standard programming language for doing machine learning of any variety. Jupyter notebooks were used because they allow programming in "blocks" where certain segments of code can be executed in any order, which makes prototyping, editing, and parameter-tweaking much more efficient than standard functional programming.

5.2.2 Numpy

Numpy is a python library that has become the gold standard for number manipulation and array based applications. They provide a wealth of sub-libraries and functions for almost any array-based application and Numpy has become the backbone of many commonly used modern python libraries and projects.

5.2.3 Keras

Keras is one of the most popular frameworks for doing machine learning tasks in python. At its core, it is a front end for tensorflow, which is Google's API for doing many common machine learning tasks. We chose Keras because it provide a plethora of easy to use tools and makes model construction and modification as simple as possible. Models with Keras are built sequentially with every next layer being added to the model with a simple *model.add()* method. This sped up our process of building and modifying the model significantly.

5.2.4 Other various tools

Some other tools we used were:

- **matplotlib**: This is a widely used python library for handling graphic functionality of data in python programs. We used this library for plotting our accuracy and loss graphs.
- **sklearn (formally known as scikit-learn)**: While this in itself, is a fully functioning library for applying machine learning models, we only needed this library for

its input capabilities, as it has useful functions built in for handling data that is categorized with folders on a computer. We also utilized its splitting features which intelligently splits the training data into train / validation sets.

- **PIL (formally known as Python Image Library)**: This, as the name would imply, is an image library for python that provided us with many useful features for manipulating non-data set images to formats that would be accepted by our model.

5.3 Model Breakdown and Procedure

Our model breakdown is as follows:

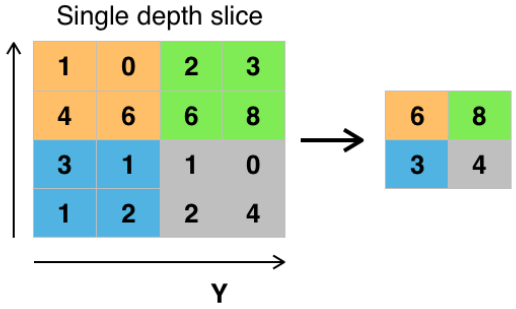
Layer (type)	Output Shape	Param
conv2d_1 (Conv2D)	(None, 100, 100, 16)	208
activation_1 (Activation)	(None, 100, 100, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 50, 50, 16)	0
conv2d_2 (Conv2D)	(None, 50, 50, 32)	2080
max_pooling2d_2 (MaxPooling2	(None, 25, 25, 32)	0
conv2d_3 (Conv2D)	(None, 25, 25, 64)	8256
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 64)	0
conv2d_4 (Conv2D)	(None, 12, 12, 128)	32896
max_pooling2d_4 (MaxPooling2	(None, 6, 6, 128)	0
conv2d_5 (Conv2D)	(None, 6, 6, 256)	131328
max_pooling2d_5 (MaxPooling2	(None, 3, 3, 256)	0
dropout_1 (Dropout)	(None, 3, 3, 256)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 150)	345750
activation_2 (Activation)	(None, 150)	0
dropout_2 (Dropout)	(None, 150)	0
dense_2 (Dense)	(None, 120)	18120

5.3.1 Convolution Layer

The convolution filter is the main feature that separates convolutional neural networks to standard neural networks. At a *very* high level, the convolution layer consists of a filter that "scans" around the image and picks out features that correspond to certain spacial locations of the image. Most convolutional neural networks will have multiple convolution layers with different filter sizes, because different sized filters will be able at picking up different sized features in the original image.

5.3.2 Pooling Layer

Pooling layers perform a kind of downsampling. It essentially takes the input and partitions it according to preset parameters and applies a downsampling function. The most common pooling method is max-pooling, in which the max value of each partition is outputted, as seen below:[3]



5.3.3 Dropout

A dropout layer randomly drops (sets to 0) some of the input data. This helps prevent over fitting by invalidating some of the data and ensure the model doesn't get too attached to the training data and can still perform well on generalized data the model hasn't seen before.

5.3.4 Flatten

Flatten layers flatten input, which is useful for cases where we what do reduce the dimensionality of an input (such as converting a two-dimensional image to a one-dimensional array of outputs).

5.3.5 Activation

An activation layer is a layer that applies some function to an input which decides whether or not to fire the "neuron" and to what extent. The most common activation function (and the one we ended up using) is the ReLU function, which

[3] Image taken from Wikipedia: Convolutional Neural Networks

stands for rectified linear unit, and is defined as:

$$f(x) = max(0, x)$$

Other common activation functions are the hyperbolic tangent: $f(x) = tanh(x)$ and the sigmoidal function:$f(x) = (1 + e^{-x})^{-1}$

5.3.6 Output

The final layer of the neural network is a fully connected layer with a size of 120 (the number of possible classifications in the data set), and applies a softmax function to the output. The softmax function takes in all of the inputs and subsequently fits them to a probability distribution proportion to the input such that

$$\sum_{i=0}^{n} x_i = 1$$

where x_i is the value of the i^{th} output node. This means that each output node is a proportion the corresponds to the "amount" that the neural network thinks the input matches each of the possible outputs.[4] With this, we can run *argmax(output)* to get the index of the labels that corresponds to the network's guess of the input.

5.3.7 Optimizer

Optimizers interact with both the loss function and the model parameters by updating said parameters according to how the loss function performed. There are many options for optimizer choice such as stochastic gradient descent and RMS-prop, but we chose to go with Adam, as it is regarded as a very high performing and reliable optimizer.

5.3.8 Loss

Loss functions can be essentially thought of as cost functions from search/optimization problems. The "loss" value is simply the output of this said function. Loss functions often signify error, and as a result with neural networks, we seek to minimize this value. As with the optimizers, there are a number of different options available to choose from, and for our model we went with categorical cross-entropy. CCE is defined as:

$$L(y, \hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} \times log(\hat{y}_{ij}))$$

6 Experiments & Analysis

6.1 Model Performance

We followed guidelines from our sources when constructing our initial model and managed to get an initial accuracy of around 97%. This admittedly was a good result in and of itself, but we wanted to distinguish our model from the other efforts so we made various attempts at tuning the model to increase accuracy and ended up with a final accuracy of 98.84% testing accuracy.
From the experiments we ran, we found that adding an additional convolutional layer gave us a small increase, and fine tuning the adam parameters also gave us tiny benefits in test accuracy. We found that for Adam, a learning rate of $\alpha = 0.001$ was very high-performing. We trained our model for 20 epochs (this will be explained later). and our final model's accuracy and loss graphs are as follows:

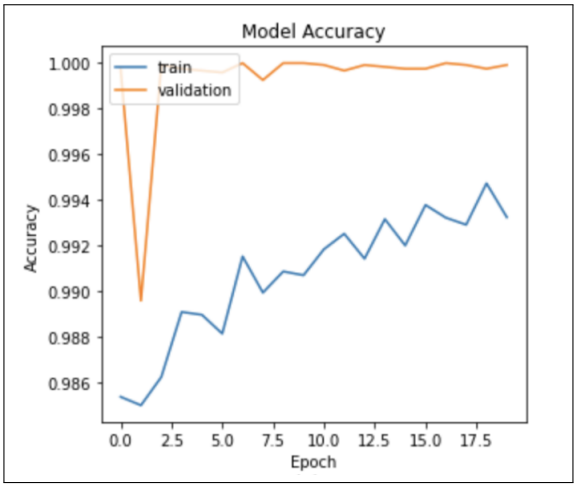


Figure 4: Accuracy vs. Epoch

[4]This is a very handwave-y description of how neural networks work, and if you want to learn more I highly recommend 3blue1Brown's videos on neural networks on Youtube

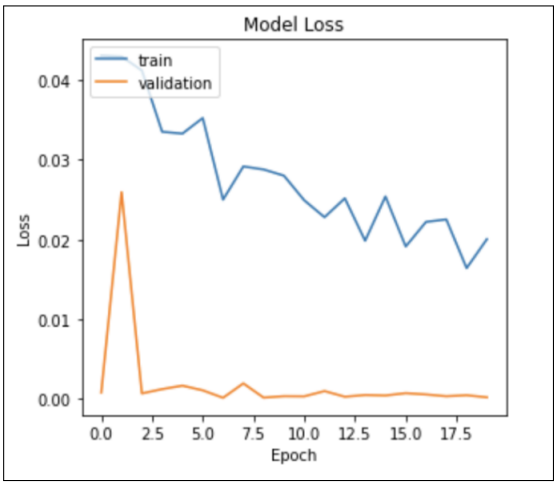


Figure 5: Loss vs. Epoch

For an example of seeing the model perform on a randomized set of 25 examples from the test set, see the appendix. This image also highlights a common shortcoming in image classification, where at certain angles, objects can look like other objects, which caused an understandable incorrect classification in the example in the appendix.
We also created a python script that, using this trained model, is able to classify images that the user supplies. From the test of this system on a couple hand-picked images from the web, it seems to perform very well, with prediction confidence often exceeding 95%.

7 Conclusions

7.1 Summary

This was a really fun introduction to the world of image processing and convolutional neural networks. Given the time frame, I'm quite pleased with the results, while the system doesn't do everything we wanted it to do (see: Shortcomings), We're very pleased with the

7.2 Shortcomings

The statement that the model performs well is a little misleading. It's true, that on images from this data set, and images that *look like* they came from this data set, the model performs admirably. However, on data that didn't look like the images from the data set, the model performs...less than stellar. For fun the model was fed a meme image that had absolutely nothing to do with fruit, and it was 99% sure that it was a pineapple. So our model isn't lacking in accuracy, but rather *robustness*. The input data was all formatted exactly the same, so our model has no concept of the fruit it's trying to classify if there is anything else in the photo aside from a white background, which presents a very clear idea of ways to improve this model. We tried to mitigate this by creating a way of standardizing the input photos. We need to convert them to a 100×100 resolution anyway, but we wanted to try and find a way to make user provided images look like the test set, which would involve focusing on the piece of fruit and then removing the background and replacing it with solid white. We did extensive research into ways of accomplishing this, but the methods we found often required some kind of user input along with the image (such as general location of where the subject was, etc), which is something we wanted to avoid. Implementing something like this would not be easy and would likely require an entire separate neural network to intelligently detect the subject of a photo (which is much easier said than done).
Another shortcoming came in the form of model training time. As described, we only trained for 20 epochs, which is relatively small. However, training 20 epochs still took a couple hours each time, which presented a very limited time frame for experimenting with various model parameters.

7.3 Future Work

There are a number of obvious way this system could be improved. The first and most obvious improvement to make is to have it be able to recognize "unclean" data (i.e. images that aren't just a single piece of the fruit in question with a white background). A potential way to help mitigate this problem would be to create some webscraper

8 Appendix

8.1 Work Breakdown

Work was split pretty evenly among the two of us, Liam focused a bit more on the research aspect, while Stephen focused a bit more on the programming aspect, but the work was split pretty evenly.

8.2 Final Notes

The code is available on github at:

`https://github.com/xbarjoe/fiction`

Thanks Dr. Park for a great semester!

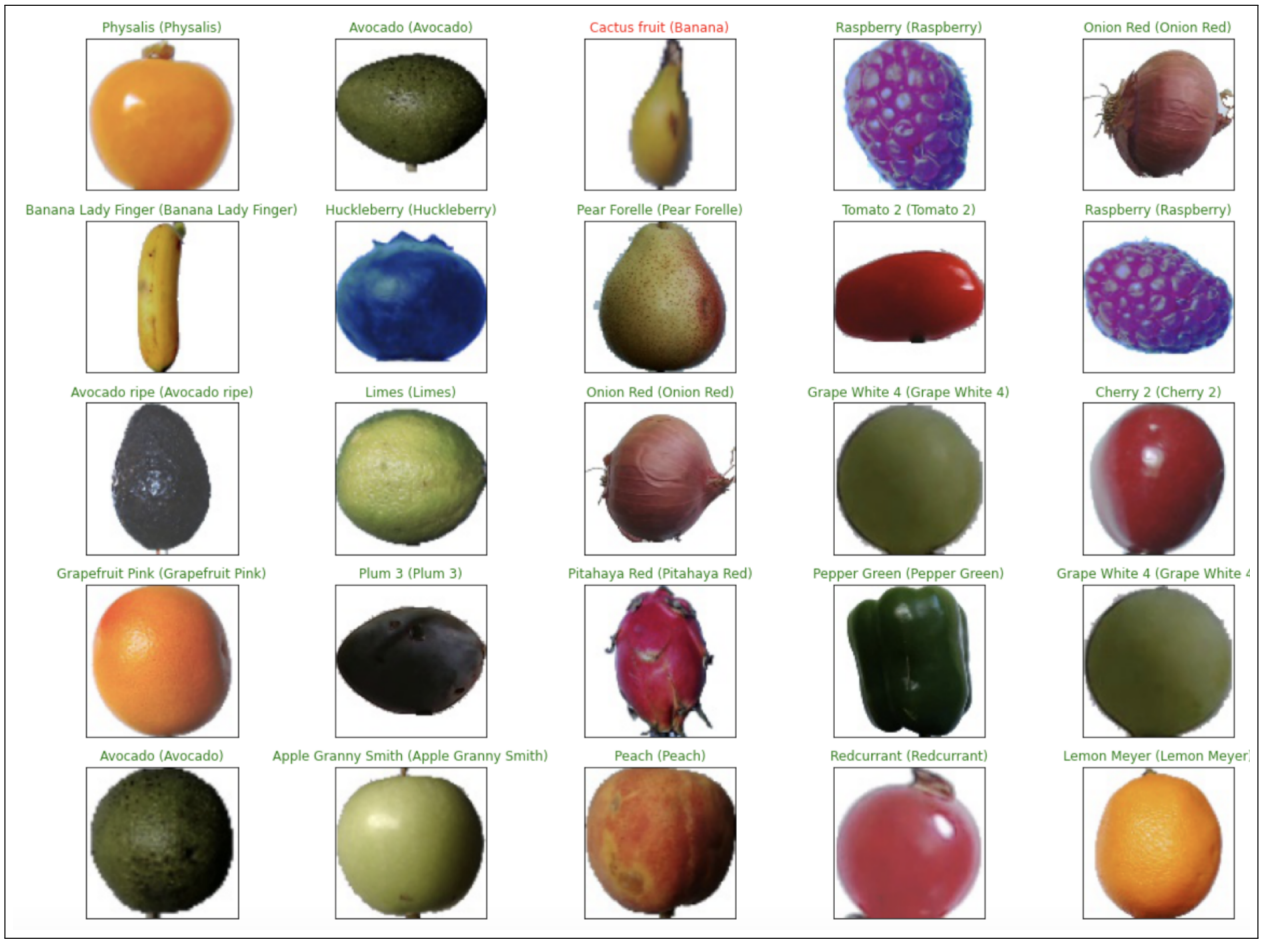


Figure 6: Here we can see the value that the model predicted, and then the true value in parentheses, with correct classifications in green, and incorrect classifications in red