RipeOrWrong: Using Deep Learning Networks to Determine the Quality of Fruits and Vegetables Using Thermal Imaging

Tyler Kaye
Adviser: Alan Kaplan

Abstract

This paper details the design, development, and evaluation of RipeOrWrong, an Android application that identifies bruised and under/over-ripened fruits and vegetables. The application utilizes the FLIR One thermal imaging camera in order to measure thermal variations below the skin of the fruit / vegetable. Additionally, the application employs a deep convolutional neural network designed with the TensorFlow framework in order to implement the classifier. Although this application focusses on apples, it has been designed and documented so that the implementation of other fruits and vegetables is seamless. RipeOrWrong leverages both thermal imaging and modern machine learning to prevent consumers from purchasing undesirable fruits and vegetables.

1. Introduction

In 2011 the US Department of Agriculture reported that supermarket retailers lose approximately $15 billion in sales of fruit and $18 billion in sales of vegetables due to strict industry standards on quality [6]. These figures have spurned an abundance of research into the automation of identifying bad fruits and vegetables. However, much of this research has focused on how to use machine learning techniques and state of the art equipment to identify poor fruit and vegetable quality at the distribution-side. While this is important research, no application has been created that leverages accessible and widely-used technologies to prevent consumers from buying bad fruits and vegetables. A McKinsey study reported that nearly 25% of the fruits and vegetables in the United States are thrown out by consumers; two-thirds of which are due to poor quality such as being bad,

bruised, or not ripened. When this much food gets thrown out by consumers it becomes a waste of not only money, but also time.

RipeOrWrong leverages a mobile thermal imaging camera and state-of-the-art machine learning techniques to inform a user whether or not they should buy a piece of fruit based on factors such as ripeness, bruising, and other imperfections. Once in the store, a user will be able to open the RipeOrWrong application on their smartphone and plug in the FLIR One thermal imaging camera to see the thermal properties of the fruit or vegetable that they are thinking of purchasing. Then, when the image capture button is clicked, the neural network reads in the thermal image and processes it to produce a recommendation for the user. This process will become even simpler as thermal imaging cameras become more widespread and common not only as smartphone accessories, but also as core components of the smartphone architecture as evidenced by the emergence of the CAT-S60 smartphone [13].

Although it is initially designed for just apples, RipeOrWrong provides a framework for the seamless integration of various other fruits and vegetables into the platform. It has been highly modularized and well documented in order to make the process of adding new data to the platform straightforward and comprehensible.

2. Background and Related Work

In the past few years, there has been much research into how to combine modern imaging techniques with state of the art machine learning algorithms to develop grading systems for fruits and vegetables. However, these innovations have yet to be brought to the consumer as they require highly-specialized imaging equipment. For example, in [17], researchers used a deep convolutional net with a sophisticated boosting algorithm named AdaBoost, which combines the output of several "weak" learning algorithms together into a weighted sum that becomes the final output of the classifier. Additionally, [17] developed a novel set of features and distinction factors for its neural network to specifically look for. However, its algorithm was more geared towards automating the process of fruit sorting and grading in large quantities. Furthermore, the study used non-specialized cameras

to identify surface blemishes on apples as opposed to going beyond the visual spectrum and approaching the problem using thermal or spectral imaging as done in [20] and [14] and [15].

In [2], researchers used a non-specialized camera to identify the coloring of tomatoes in order to classify them into ripeness stages. [2] ultimately concluded that the visual spectrum is not sufficient to accurately classify and grade tomatoes, and it cited the necessity of more detailed yet non-invasive sensing methods for the evaluation of fruit and vegetable quality. [5] developed an application that uses a flash camera to detect the ripeness of a fruit. Although this motivation is similar to RipeOrWrong, its results were ultimately unsatisfactory as the flash did not reveal anything beneath a very thin layer of skin. Thus, it seems to be clear that non-specialized cameras that capture only the visual spectrum are not capable of grading fruit, and thus other equipment must be used.

[20] details the chemical nature of fruit and the process of fruit decomposition viewed through different spectrums. Once it concluded that more advanced sensing techniques must be used in order to classify fruits and vegetables, [20] utilized spectroscopy, multispectral imaging, and hyperspectral imaging techniques to develop a set of "chemometrics" to identify fruit quality. [14] also uses hyper-spectral imaging, but in this case it aimed to identify cherries with pits in them. The challenge with both of these experiments is that they both used highly-sophisticated and fine-tuned equipment as opposed to tools and sensors accessible to every day consumers.

Two applications that do in fact use a smartphone to determine the ripeness of apples are WiSci [19] and SciO [15]. WiSci and SciO are less of computing applications and more of hardware projects to make portable and wireless spectrometers capable of measuring the chlorophyll composition of certain fruits. This differs greatly from RipeOrWrong in two important ways. Firstly, the application utilizes a spectrometer, which is a highly specific and costly piece of equipment, as opposed to a thermal imaging camera. This is an important distinction as spectrometers will always be highly specialized and expensive sensors. However, the emergence of the FLIR One thermal camera extension piece for smartphones and the development of the CAT-S60 smartphone [13], the first fully-functional smartphone with a thermal imaging camera, suggests that the integration of thermal imaging technology may become commonplace among smartphones. Thus, is it important to

leverage popular and cost-effective technology to solve this problem as opposed to using complicated and costly sensors. Secondly, WiSci [19] and SciO [15] do not leverage any type of machine learning, but rather they are simply sensors that read the levels of chlorophyll present in the fruit and convert this measurement into a ripeness index.

[16] details the process for collecting data on the ripeness of fruits using a higher-end thermal imaging camera; however, its process is not as efficient or utile as RipeOrWrong. The experiment calculates the heat capacity of more ripened fruits by heating them up and observing the thermal variation over four minutes as it cools down. This time-series data can be used to determine the heat capacity of the fruit or vegetable in question and this quantity can be converted into a ripeness index based on its chemical composition. This process is not only unsatisfactory and infeasible for the average consumer in the supermarket, but also it is a potentially destructive method for testing fruits and vegetables. Similarly, in [7], researchers used a thermal sensor to convert the radiant flux coming from an apple into a measurement of its thermal emissivity and then its ripeness. The average consumer in the supermarket wants to be able to pull out their phone and instantly tell if the fruit is worth buying. While other methods such as the use of a spectrometer, chemical analysis, or heat capacity observance may be more accurate, they are ultimately too time-consuming and inconvenient to be useful. These solutions do not address the core motivation of RipeOrWrong to solve the problem of determining the ripeness of fruits and vegetables while in the grocery store.

[8] and [9] come the closest to emulating the approach of RipeOrWrong. [9] identifies concrete and specific reasons for temperature variations in apples. It uses thermal imaging to detect the transpiration and metabolic heat as well as the heat and water exchange with the surrounding air as variables in its experiment. Additionally, the researchers noted that defective surfaces and bruises undetectable to the human eye cause increased transpiration spots in the thermal images due to a local rise in metabolic heat production. It notes that previous studies have failed to classify the bruising of apples due to the inability of the experiments' thermal cameras to detect small variations in temperatures. However, [9] found success using a camera with a temperature sensitivity of .1 degrees kelvin, which is the exact same temperature sensitivity as the FLIR One thermal imaging

camera. An important distinction noted by [9] was that different types of apples exhibit disparate heat signatures surrounding their bruises, which adds an important complication to the problem. [8] approaches the problem very similarly by searching for thermal variation in horticultural products due to the metabolic heat caused by ripening and the respiration process of fruits and vegetables. Furthermore, [8] found positive results when using a thermal imaging camera with a temperature sensitivity of .1 degrees kelvin. [8] and [9] utilize thermal imaging to detect statistical variations in the temperature distributions of apples; however, both of these studies were only statistical studies aimed at identifying the quantitative differences among fruits of different ripeness levels as opposed to approaching the problem from a machine learning viewpoint in which the most important question is: should this fruit be purchased? Instead, these studies focus on finding empirical reasons for arriving at the conclusion. Furthermore, both of these studies used non-portable thermal imaging cameras as opposed to RipeOrWrong's use of a portable smartphone attachment. Nevertheless, the above researchers' findings regarding the importance of using a thermal imagine camera with a temperature sensitivity .1 degrees suggest that the FLIR One thermal imaging attachment will be powerful and accurate enough for the purposes of this application.

Although there has been plenty of research into the chemical nature of unripened fruit, and even how to apply machine learning techniques to the evaluation of fruits and vegetables in large quantities on the supply-side, the current market lacks a consumer-facing application designed for ease of use and accuracy.

3. Approach

Utilizing the FLIR One thermal imaging camera, RipeOrWrong will leverage cutting edge machine learning techniques to provide users with a recommendation on the quality of fruits and vegetables based on several factors (bruising, ripeness, blemishes, seeds, etc.).

While non-specialized cameras and the human eye can only see light in the wavelengths between 400 nm and 700 nm, the FLIR One camera provides accurate readings in the infrared spectrum. A useful feature of the FLIR One is that it actually contains both a thermal imaging camera and a

non-specialized camera. Merging both of these cameras, the FLIR One overlays the contours of the visible spectrum onto the thermal image. This can be seen below in Figure 1. The camera superimposes the outline of the apple as well as its background onto its own thermal signature.

In its technical specifications [4], Flir describes the grading of fruit as a major use for some of its more high-end industrial infrared cameras. Although the studies cited in [4] utilize highly-specialized imaging equipment, this equipment has same temperature sensitivity as the FLIR One smartphone attachment camera. This suggests that the FLIR One should be sufficiently precise for the purposes of this application. Additionally, the recent release of the FLIR One SDK makes the integration of the mobile thermal camera seamless. The SDK enables the application to show the user the output of the camera and even convert a snapshot into a format consistent with the TensorFlow neural network.

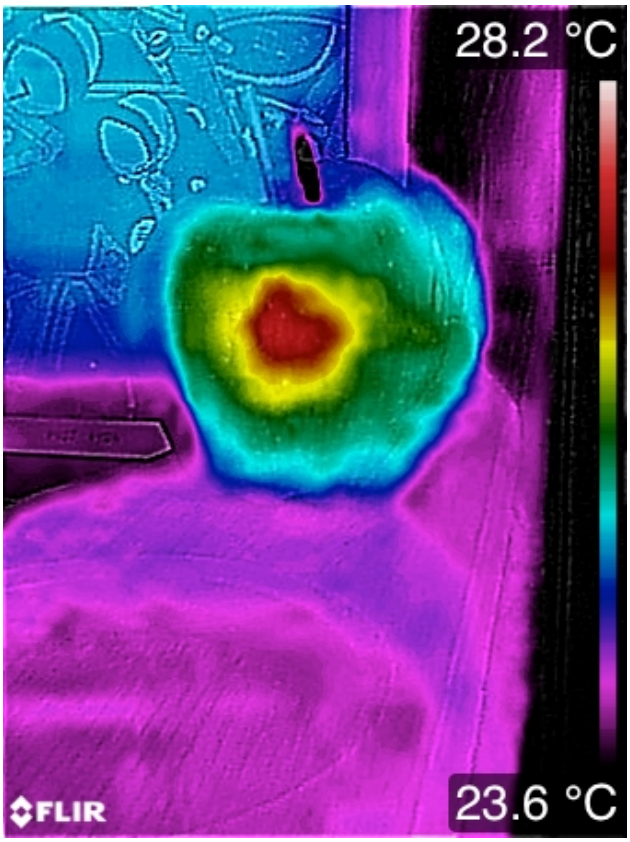

Figure 1: Thermal image of a bruised apple.

A user of RipeOrWrong takes a picture of the fruit they wish to purchase and the application

will utilize the FLIR One SDK to determine the thermal composition of the fruit or vegetable and provide its recommendation on its quality. In order to provide this recommendation, a classifier was developed using a deep convolutional neural network.

This convolutional neural network will utilize TensorFlow, a Google-created framework for deep neural nets. Once programmed and configured with a satisfactory dataset, the algorithm will train a convolutional network capable of grading the quality of different fruits and vegetables. In fact, two different neural networks (FruitNet and GoogLeNet) were trained and tested in order to ensure the accuracy of the classifier.

Although the machine learning model will only provide binary output (yes or no) as to whether or not it would recommend this particular item, it is the intention of the application to eventually develop the complexity necessary to grade the piece of fruit on a more tangible and quantitative scale as opposed to just binary output. The challenge with this second approach is finding or creating a dataset containing thermal images of fruits associated with a specific grade. Nevertheless, if such a data set is found this would be the ideal solution.

Although this application focusses on identifying and grading the quality of apples, the system has been designed such that the addition of new fruits and vegetables to the model is as effortless and straight-forward as possible. Thus, the modularized approach to solving this problem is a very important part of the solution itself. After dropping a folder containing the initial images into the directory, several scripts can be run to generate a complete dataset using the ImagePipeline, convert the images into the necessary format, upload them into the TensorFlow model, and finally retrain the neural network. This approach ensures that it is easy and straightforward to add new fruits and vegetables into the capabilities of the application. The basic structure of adding a new set of images into the model is outlined in Figure 2.

4. Implementation

The implementation of this project consists of four main components:

1. Crafting the Android application

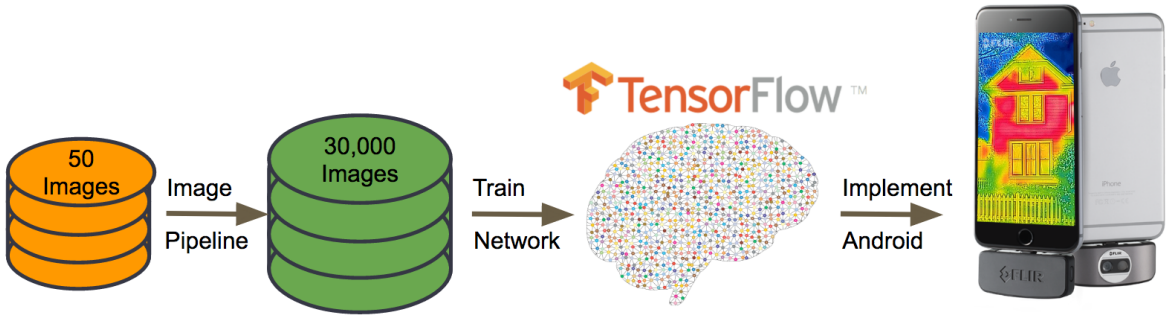

Figure 2: Modules for this project

2. Implementing the classifier as a Python server
3. Creating a data set large enough to train a neural network
4. Designing and training a neural network capable of grading fruits and vegetables

The complete system architecture is shown in Figure 3.

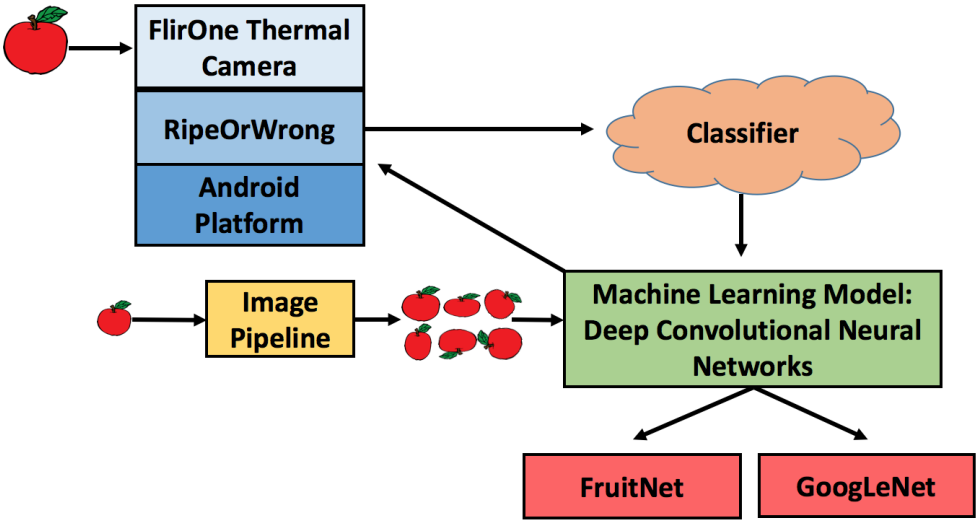

Figure 3: Overall System Architecture of RipeOrWrong

4.1. Developing the Android Application

The stack of the Android application can be seen below in Figure 4. Overall, there were two main challenges in the implementation of the Android application.

4.1.1. Utilizing the FLIR One SDK The FLIR One SDK works by a series of delegate methods defined in the main activity of the application. First, the `Device.startDiscovery()` method must be called to determine if the FLIR One camera is currently connected to the Android phone.

Next, the manifest file must be changed to give the application access to the USB port as well as the ability to automatically open the application when the FLIR One is plugged into the phone. Finally, the `Device.streamDelegate` must be set to the current activity and the `onFrameReceived()` method must be implemented to display the current thermal image on the screen and allow the user to interpret the image in several different thermal frame types. Much of the source code for the basic Flir functionality was taken from the FLIR One Android example application, which was helpful to use as a starting point for the project [3]. Furthermore, the SDK aided the implementation of complex functionality such as viewing the fruit through different lenses and measuring the surface temperature of the piece of fruit.

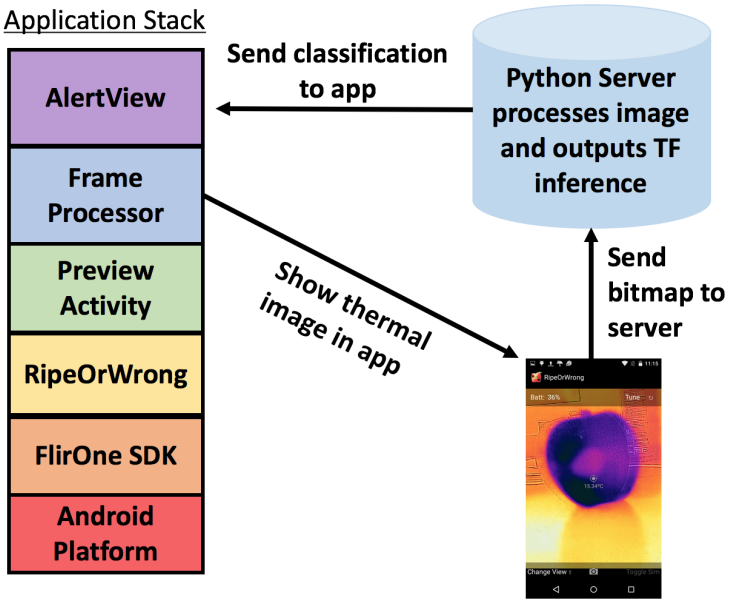

Figure 4: Basic stack of RipeOrWrong

4.1.2. Designing the Application The application has one main activity: PreviewActivity. Upon receiving the frame stream from the FLIR One camera, the application has a button that takes a snapshot of the thermal image, encodes it as a bitmap, and sends it to the server through a socket. The application then waits for a response from the server at which point it populates the screen with an AlertView telling the user its recommendation for the current fruit or vegetable and its confidence level in the decision. Furthermore, the application utilizes the key-value storage on the Android device to save and show how much money the application has saved the user. This calculation is

derived using the formula: number of fruits the classifier deems “bad” multiplied by the average price of a piece of fruit. The basic functionality of the application can be seen in Figure 5 and the AlertView displaying the output of the neural network can be seen in Figure 6

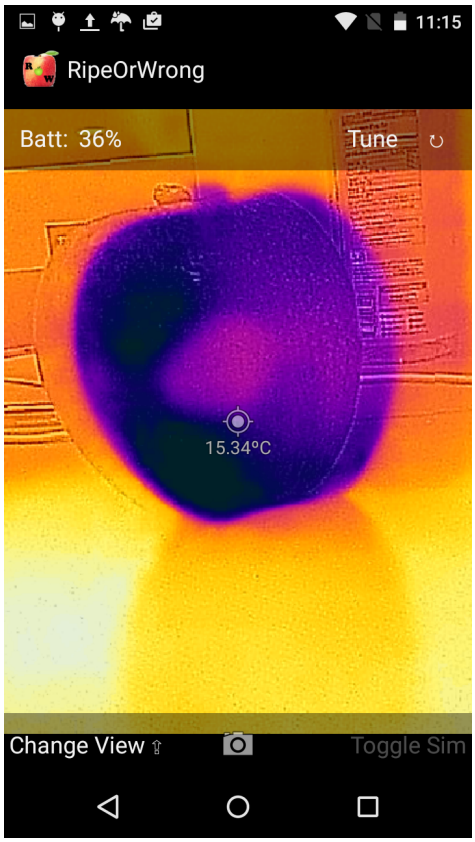

Figure 5: Basic frame stream of thermal apple

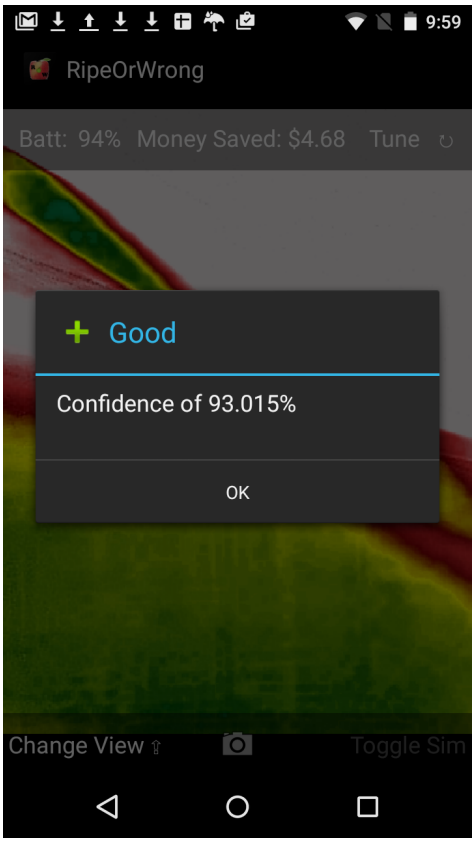

Figure 6: AlertView showing result of the model

4.2. Implementing the Classifier

Although it was the original intent of this application to incorporate the TensorFlow NDK into the Android application to provide real-time classification, ultimately it was decided that putting the classifier in a server was the superior option.

First and foremost, this is due to the complexity of implementing TensorFlow locally on the Android [1]. The NDK is written in C++, so the application cannot be developed in AndroidStudio and the project must be built using Bazel instead of the standard Android build tools. Furthermore, the sample project includes several-hundred files to build a simple application that classifies what the camera sees. Thus the adaptation of this sample application into RipeOrWrong would have been

an exhaustive process without the help of AndroidStudio and the emulator, which cannot simulate a FLIR One camera being plugged into it. Finally, formatting the TensorFlow computational graph into the exact format that the Android NDK wants it to be is challenging and largely undocumented. While implementing the TensorFlow inference locally would make the application faster and provide real-time classification, it was ultimately too challenging to develop an application within the constraints that it imposes.

However, there are many advantages to not implementing the classifier locally in the Android application. First, the application becomes much more lightweight. The TensorFlow sample Android application has an APK size of 58 MB, 54 of which come from the TensorFlow computational graph. Thus, removing this from the application greatly reduces its size. In addition, by not including the NDK, the project does not use Bazel and thus the application can be developed, built, and debugged in AndroidStudio. More importantly, the Flir simulator can be used to test the application instead of exporting the APK each time testing was necessary. Finally, this approach makes the platform more seamless in the addition of new fruits and vegetables into the model. Instead of remaking and distributing the application each time the neural network is re-trained, the model can instead be a separate module operating in the cloud and thus easy to change. In order to change the neural network or add new features, a new model must be put into the server directory along with a text file of the descriptions of its outputs (good apple, bad apple, etc).

Therefore, a Python server was developed to accept incoming socket connections, read the image into memory, resize it to be the proper input dimensions, and finally run it through one iteration of the TensorFlow computational graph in order to produce an inference. Finally, the classifier sends back the result (good or bad) along with its confidence level. This server was hosted on Amazon Web Services EC2 due to its ease of use.

The next step of implementing the classifier was to design and train a neural network for the application. However, prior to this it was necessary to gather a robust dataset that will be capable of training the network.

4.3. Generating and Extending the Dataset

Initial research suggested that no full datasets of thermal images of fruits were open sourced and available for use. Although this meant that a custom dataset needed to be collected, this turned out to be an important advantage in terms of creating a model highly specific and tailored to images taken with a FLIR One thermal imaging camera. Even if a dataset could be found, the resolution and temperature sensitivity would in all likelihood be different. Although it would take extra time, creating a custom dataset would ultimately lead to more representative data and thus a more promising neural network for the application.

The main challenge was to create a dataset large enough to train a deep neural network. Modern neural networks require thousands of labeled examples to ensure optimal generalization performance. This is because the model needs to learn how to identify a bad or good fruit based on its features and thermal signature regardless of its relative position, perspective, or rotation. If there are not enough images in the dataset, then the model will run into the problem of over-fitting, in which the training error will approach zero while the testing error (the error on images not previously seen by the model) will increase as seen in Figure 7. This phenomenon occurs either when the model is too large and it just memorizes the inputs and their respective desired outputs, or when the dataset is too small and the above happens regardless of size of the model.

In order to counteract the problem of over-fitting, a technique known as fabricating "fake data" was used for this experiment. This process involves generating a set of permutations of each image in the data set using different perspectives, rotations, translations, and distortions. An image pipeline was created to generate this fake data.

The first step of this process was to take 50 photos of apples using the FLIR One camera and label them as either good (1) or bad (0). Although it is the intention of this application to one day provide a more quantitative metric by having a dataset of ratings rather than a binary schema, it was ultimately too challenging to develop a consistent grading platform without taking chemical measurements of the fruits, and thus a binary scheme was used. Some apples were intentionally bruised for the purposes of the photo, some were bought when they were not ripe enough, and some

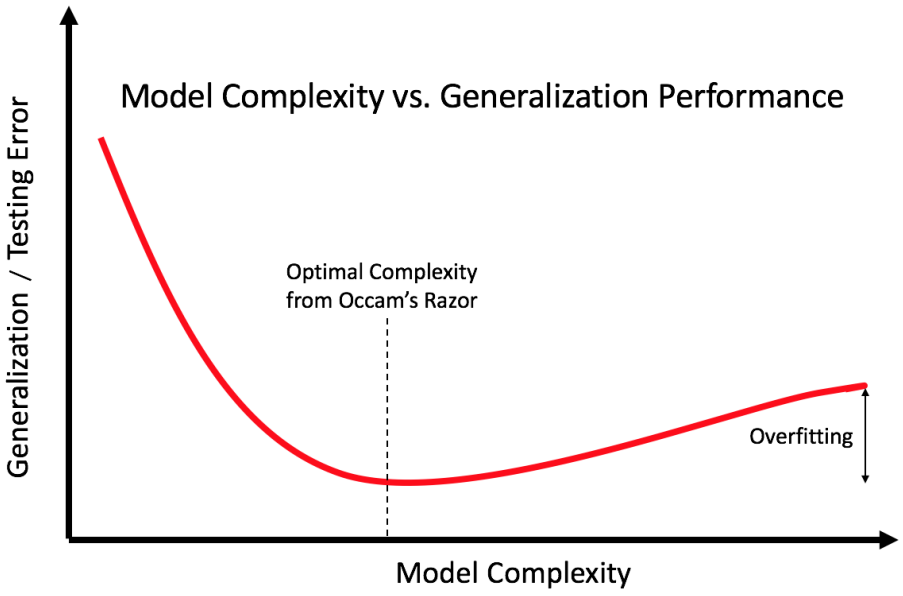

Figure 7: Overfitting: Model Complexity vs. Generalization Performance

were left in a warm area for two weeks to intentionally over-ripen and potentially rot. This ensured that the dataset included a diverse group of apple qualities. Figure 1 shows a sample thermal image of a bruised apple. In this photo it is clear that a bruised apple gives off a very specific heat signature. This is because a bruise is an area within the fruit with excess water. This water has a higher metabolic rate than the remaining composition of the apple, and thus the bruised area becomes approximately two degrees warmer than the rest of the apple [20].

The first step of the ImagePipeline was to use a SeamCarver program [10], which is a content-aware resizing program. The SeamCarver program is given two randomly selected numbers (the number of rows and columns to remove) and then the algorithm finds a seam (or rather the path of least importance) in the proper orientation through the image and then removes that seam. The program randomizes the order in which the horizontal and vertical seams are removed to ensure that as many possible permutations of the image are created.

Once all of the seams have been identified and removed, the image is resized using a smooth image re-scaling library so that the images used are all the same height and width. For the purposes of this neural network, the number of pixels in the final output was determined to be 128 x 128 pixels. It was important for the images to be square so that they could be transposed and rotated. Additionally, 128 pixels is an ideal size given the tradeoffs between memory, feature detection

capability and computational complexity. Finally, each permutation of the images outputted by the SeamCarver program was flipped along its x-axis, y-axis, and then both-axis's, before being rotated to the left and the right.

In order to ensure the strength, quality, and quantity of the dataset, the SeamCarver program generated 100 permutations of each image, each of which was fed into the program generating a total of 6 permutations of the image rotated / transposed. Thus, a dataset of just 50 labeled images synthesized a data set of size 50 x 100 x 6 = 30,000 images. As the dataset was now of a similar size to that of MNIST (a dataset of handwritten digits), its size was now sufficient for the purposes of this application. All of this was to ensure that the model learned translational and rotational invariance. This means that the model should be capable of detecting poor fruit quality regardless of the fruit's relative position, angle, or perspective. Figure 8 and 9 show twelve samples of the 600 permutations of an image generated by the ImagePipeline.

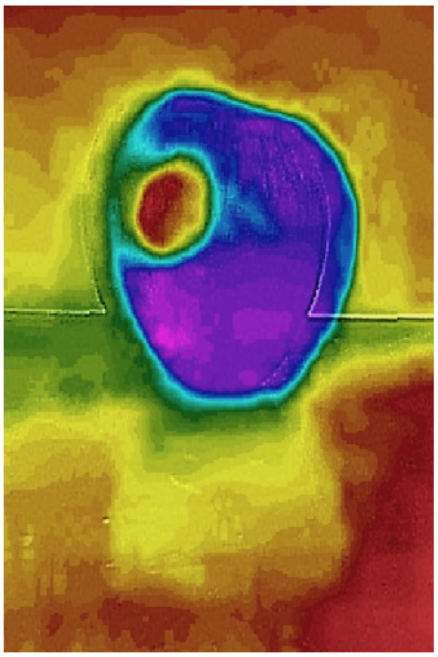

Figure 8: Original Image of Bruised Apple

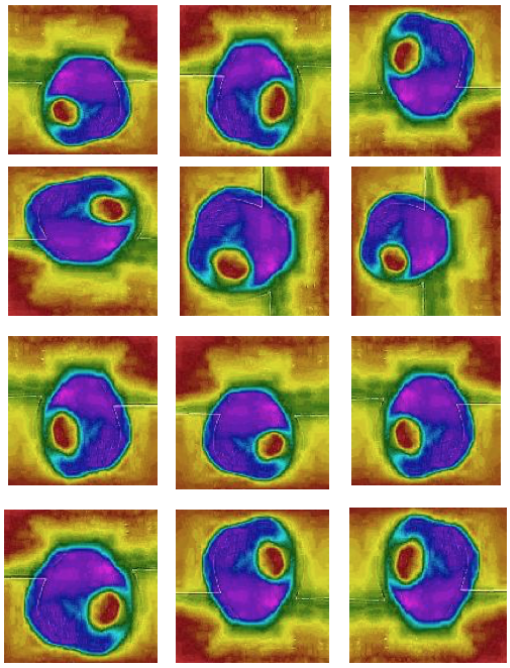

Figure 9: 12 Sample Permutations of the Image

Finally, the dataset had to be split into a training set and a testing set. The training set is the set of examples that the model learns from and what contributes to the training error, whereas the testing set is the set of examples that the model does not learn from but rather uses to evaluate how the

model will perform on examples it has not seen (the testing or generalization error). The field of machine learning has come to accept 10% to be an adequate percentage of the dataset to be in the testing set. Therefore, the order of the images in the dataset was completely randomized and then 3000 images were selected and set aside as the testing set.

The final step in this process involved converting the dataset of 30,000 images into an easily readable binary format. First the images were read into a NumPy array and reshaped to follow the format of one input byte followed by 128x128 bytes representing the red pixel value in the RGB color scheme, then the same for green and then blue. A module of Python called Pickle was used to convert the array of labels and the matrix of pixel values into an easily compressed binary dictionary. The values were then spread over 6 files in order to ease the computational complexity and memory requirements of the step. Figure 10 shows several images from the dataset along with their corresponding labels after being un-pickled in the Python program that defines the convolutional network.

This process was made to ensure that the addition of additional models for the application would be seamless. Detailed instructions have been posted in the README file of the Github Repository for the project [11]. In summary, one must simply collect the dataset, run the shell script to generate the sample images, then run another script to convert the photos into a .tar.gz file that the model will easily accept.

4.4. Developing the Neural Network

Although other machine learning methods were tested (Support Vector Machines, Multilayer Perceptrons, and Recursive Neural Networks); ultimately convolutional neural networks were used due to their superiority in learning from images by developing feature-maps capable of detecting bruises and other beneath-the-surface imperfections. Ultimately, two entirely different models were trained and tested for this application to ensure that the model was as accurate as possible. The key factors in determining which model would be used are cross-entropy loss, training error, validation error, training time, ability to re-train, and generalization performance.

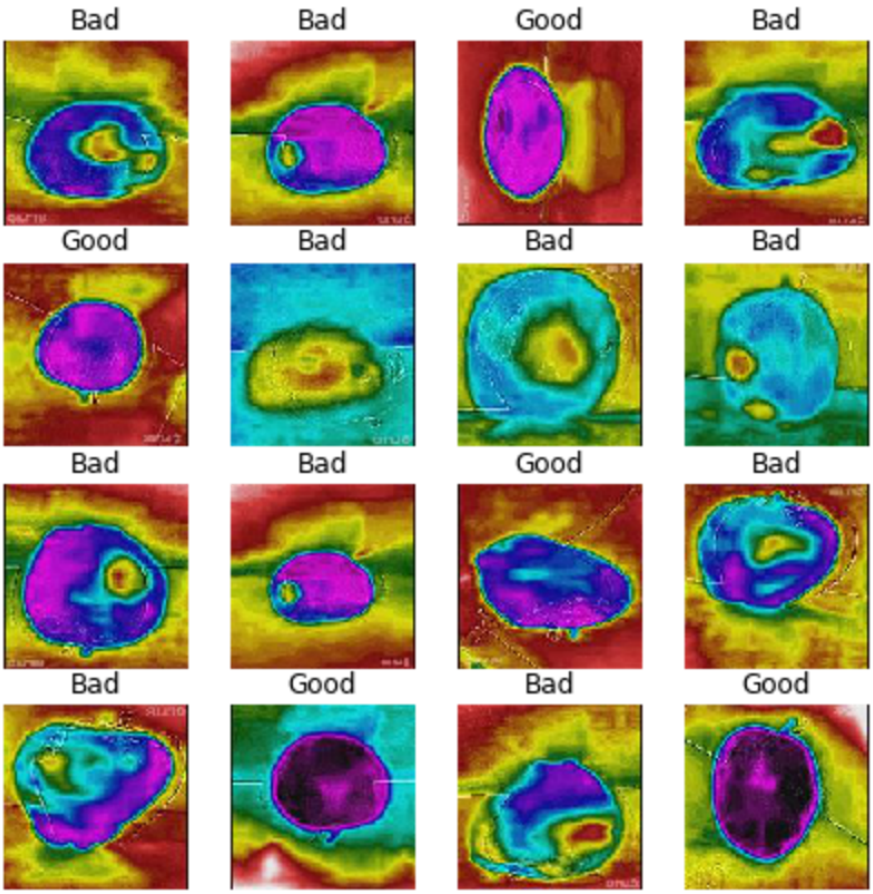

Figure 10: Sample of randomized images along with their labels

4.4.1. FruitNet FruitNet is a custom neural network designed for this application. Its architecture draws heavily from AlexNet, the network that first won the ImageNet classification competition, due to its simplicity and efficacy [12].

The first subsection of the architecture for the model involved a convolutional layer followed by a pooling layer followed by a normalization layer. In order, the convolutional layer develops kernels capable of defining a set of feature maps for the image. Then the max-pooling layer maintains translation invariance by pooling the convolutions to allow for noise and other translations of the image. Finally, the local responsive normalization layer increases the sensory perception of the neurons in the convolutional layers. FruitNet then contains two more subsections identical to the previous one in order to develop more complex feature maps. All three of the convolutional layers use 3x3 kernels as this size has come to be accepted by the machine learning community as the

best kernel size in the tradeoff between computational complexity and the granularity of the filter. Finally, the model employs a 3-layer multilayer perceptron in which the final layer employs a softmax function (cross-entropy function) with only two outputs so that the model can provide a recommendation. The softmax function works by computing the function below on each of the output nodes and then selecting the node with the highest result as the predicted output of the network.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

A summary of this architecture can be found in Figure 11.

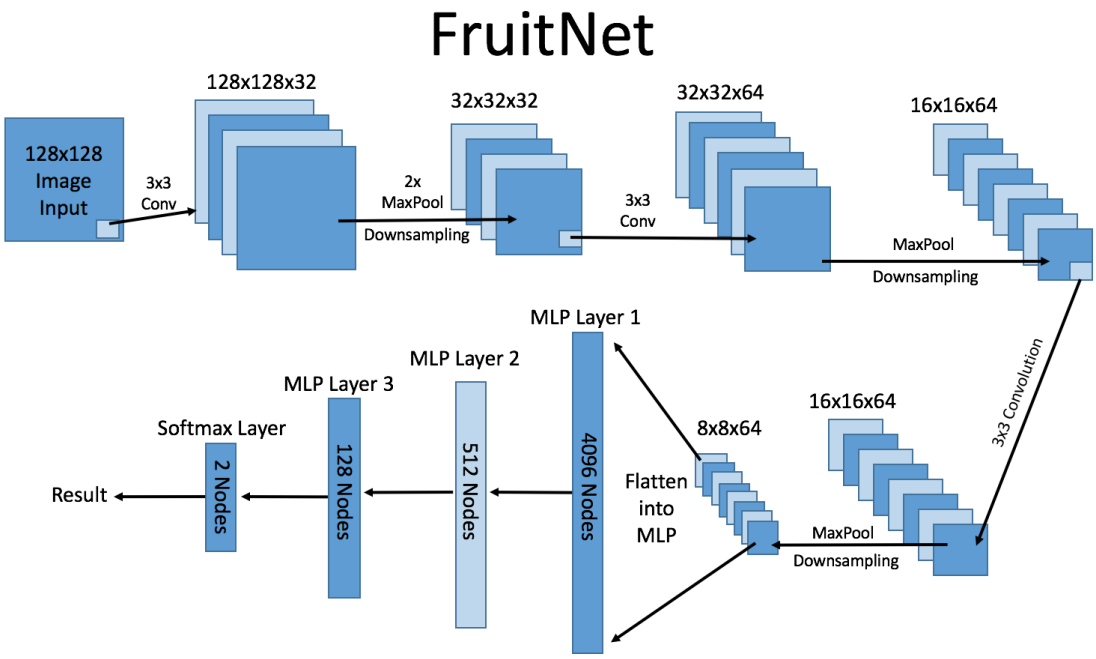

Figure 11: This is the architecture of the Convolutional Net: FruitNet

Once the basic architecture was decided, testing began immediately. At first the testing was focused on finding the proper hyper-parameters, such as the learning rate parameter, learning rate decay factor, and weight matrix initializations. Ultimately the optimal hyper-parameters were found by trying out different variations of the parameters. The model was tested several times with

different hyperparameters before finding the optimal combination. The results are summarized in Table 1.

Hyperparameter	Value
Learning Rate	.001
Learning Rate Decay Factor	.9
Dropout Rate	.9
Batch Size	128
Weight Initialization	Xavier Initialization

Table 1: Final Hyperparameter Values

Next, the architecture had to be adjusted to optimize the width (number of synapses per layer) of the layers in the MLP (Multilayer Perceptron) and the number of feature maps in the convolutional layers. The optimal results are described in Table 2:

Layer	Description
First Convolutional Layer	96 Kernels of size 3x3
First Pooling Layer	2x2 Max Pooling
Second Convolutional Layer	2048 Kernels of size 3x3
Second Pooling Layer	2x2 Max Pooling
Third Convolutional Layer	4098 Kernels of size 3x3
Third Pooling Layer	2x2 Max Pooling
First MLP Layer	512 Node Fully Connected Layer
Second MLP Layer	256 Node Fully Connected Layer
First MLP Layer	2 Node Softmax Layer

Table 2: Final Architecture Parameters

Once both the hyper-parameters and the architecture were deemed to be optimal, the model was trained until completion. The generalization error and the training error were closely monitored during training to ensure that the model stopped training at the optimal time. This is because generally when the model has seen each image roughly ten times, the model begins to experience overfitting as it simply memorizes the examples that it sees. When this phenomenon occurs, the model experiences a continuing decrease in training error (as expected); however, it is accompanied by an unwanted increase in generalization error. This phenomenon is pictured in Figure 12. In

order to prevent this loss of performance, a technique known as "early stopping" was used to stop the training when the model began to experience the above unwanted behavior.

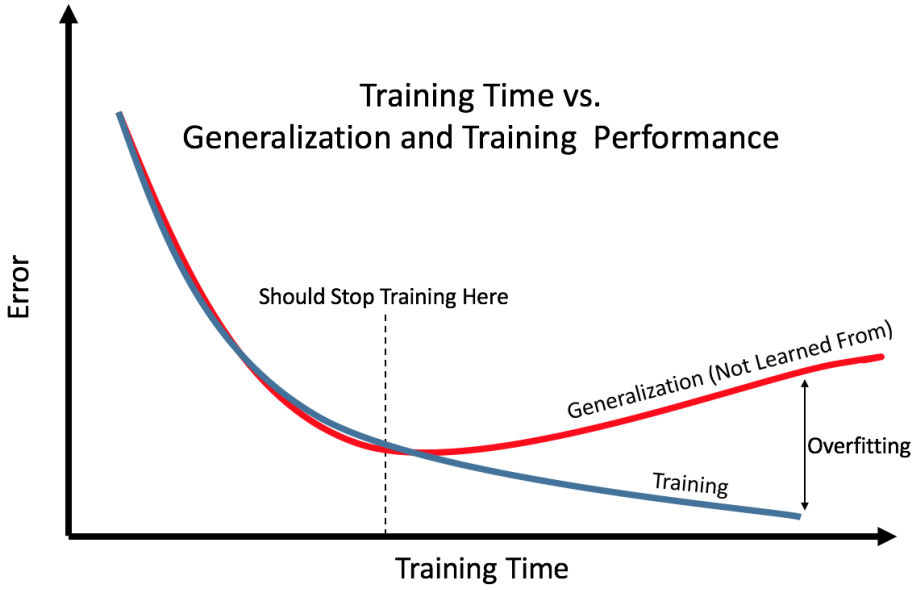

Figure 12: Depiction of the problem of overtraining

Ultimately, the model obtained a final training error of 0% and a testing error of 0%.

The final step for this sub-project was to output a binary-encoded representation of the entire TensorFlow computational graph to be used in the classifier. A script was used to ease this over-complicated process by keeping track of not only the structure of the neural net, but also the values in its weight matrices and feature maps. This file `inception-graph.pb` must be put in the same directory as the running server for it to perform inference on incoming images.

4.4.2. GoogLeNet In order to ensure the capability of RipeOrWrong to detect bruising and other below-the-surface imperfections of fruits and vegetables, a second network architecture was tested. Instead of re-creating another architecture similar to FruitNet (which had already been optimized), a model known as the GoogLeNet Inception Network was re-trained. (GoogLeNet is a play on words with Google and LeNet, a famous convolutional network)

GoogLeNet is a deep and wide neural network that won the 2014 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC14). Its full architecture is shown in Figure 13. The key to GoogLeNet's success lies in its three main strengths. Firstly, it is 22 layers deep which enables it

to develop complex feature maps. Secondly, the model incorporates a diversity of kernel sizes in its convolutions as it performs 1x1, 3x3, 5x5, and even 7x7 convolutions. This diversity enables the model to maintain translation invariance in its convolutions and develop a more diverse and expressive set of feature maps than a net like FruitNet. Finally, GoogLeNet's distributed and multi-leveled architecture fully leverages parallelism in order to reduce the computational complexity of the graph [18]. This enables the network to be trained and perform inference quickly despite its large size.

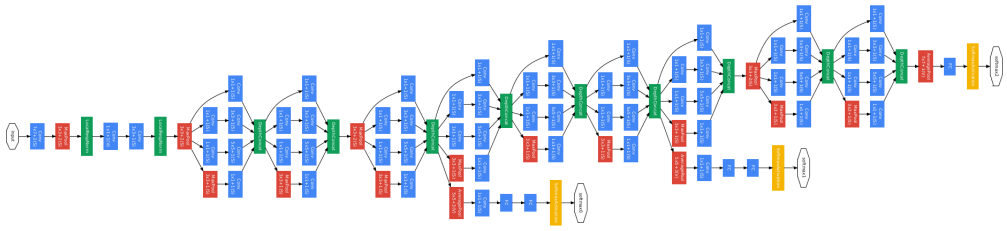

Figure 13: Overall architecture of GoogLeNet

In addition, the creators of this network wrote a script to re-train the model which involves stripping off the final softmax layer and replacing it with a new layer defined by the user who creates the new model. A blog post, "Tensor Flow for Poets", was especially helpful in understanding and performing this re-training [21]. The advantage of this method is that the model begins with a very sophisticated and trained set of feature maps and only has to retrain the final layer to output the desired result. Therefore, the re-training process takes a few minutes as opposed to a few days / weeks.

5. Results

5.1. FruitNet vs. GoogLeNet

In evaluating the performance of FruitNet relative to GoogLeNet several key factors were taken into account including the total cross entropy loss, the training / validation error, time of training, and overall ease of use.

In terms of cross entropy loss, as we can see in Figure 14, FruitNet achieved a slightly lower cross entropy loss function over time. The cross entropy function is a great metric to measure how far away the model's final softmax layer is from the correct answer. As seen below, the cross entropy loss function quantifies not only the classification error of the model but also how close its softmax outputs are to being the ideal one-hot vector.

$$E(y,\hat{y}) = \sum_{i=1}^{N} y_i \log(\hat{y})$$

Where y_i is the one-hot encoded vector describing the desired output and $\hat{y}$ is the softmax output of the network described by:

$$\hat{y} = \sum_{j=0}^{N} \frac{e^{x_j}}{\sum_i e^{x_i}}$$

When the training accuracy is high, the cross entropy loss function is more of a measurement of how confident the model is with its prediction.

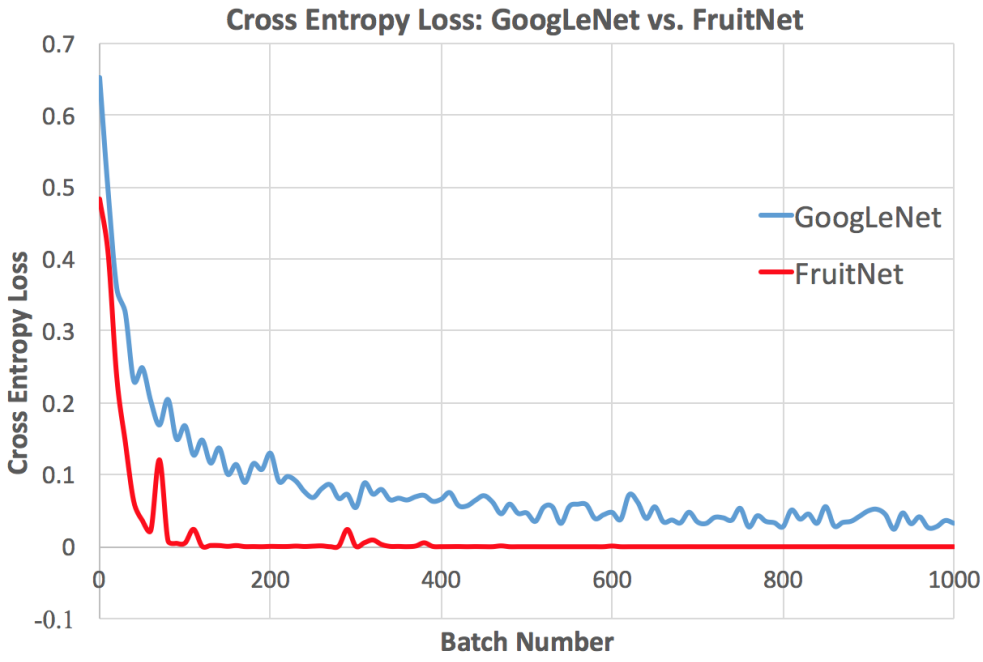

Figure 14: Cross entropy loss of FruitNet vs. GoogLeNet

Thus, although FruitNet may be more confident in its inferences than GoogLeNet, we can see in

Figures 15 and 16 that the accuracies of both models converge to 100% on both the training set and the validation set, meaning that both models begin to correctly label the dataset with nearly zero error.

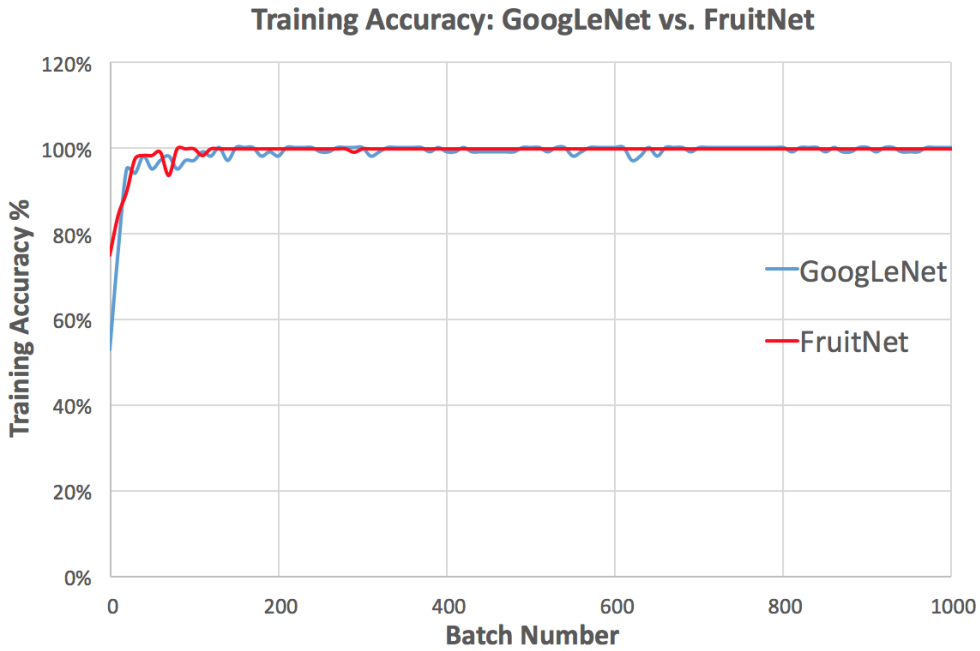

Figure 15: Training accuracy of FruitNet vs. GoogLeNet

However, this distinction between training set and test set may not be a valid measurement of the generalization performance of the two models. The generalization performance of the model is supposed to be a prediction of how well the model will perform on future images it has not learned from. This is generally done by leaving a subset of the data out of the training set and periodically testing them to get a generalization error metric without learning from them. However, in this application, most of the images in the test set are some permutation of several-hundred images in the training set with slightly different seams carved out. While generating this "fake data" was critical to the training of the model, it provides an overly-optimistic estimation of the model's generalization abilities. This can be observed in Figure 17. The residuals between the classification performance on the testing set and the training set for both GoogLeNet and FruitNet are centered around zero as opposed to a positive number as we would expect. Thus suggests that they may not be entirely

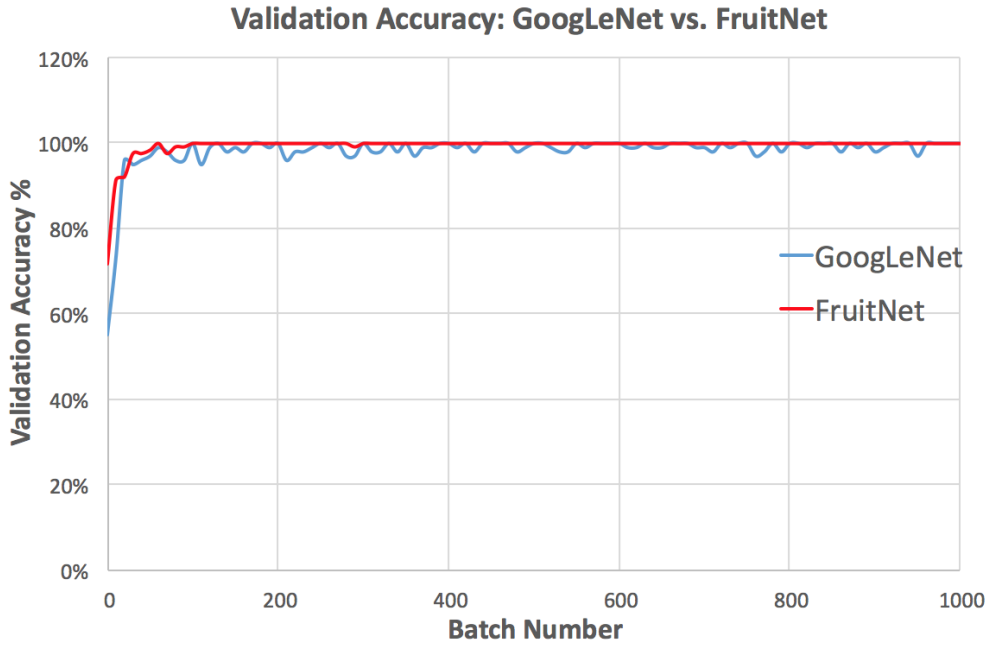

Figure 16: Validation / generalization accuracy of FruitNet vs. GoogLeNet

independent sets. Thus, the nets still perform at a high level, but the generalization performance in Figure 16 is not representative of either model's generalization capabilities.

While both models perform well, they each come with their own set of advantages and disadvantages.

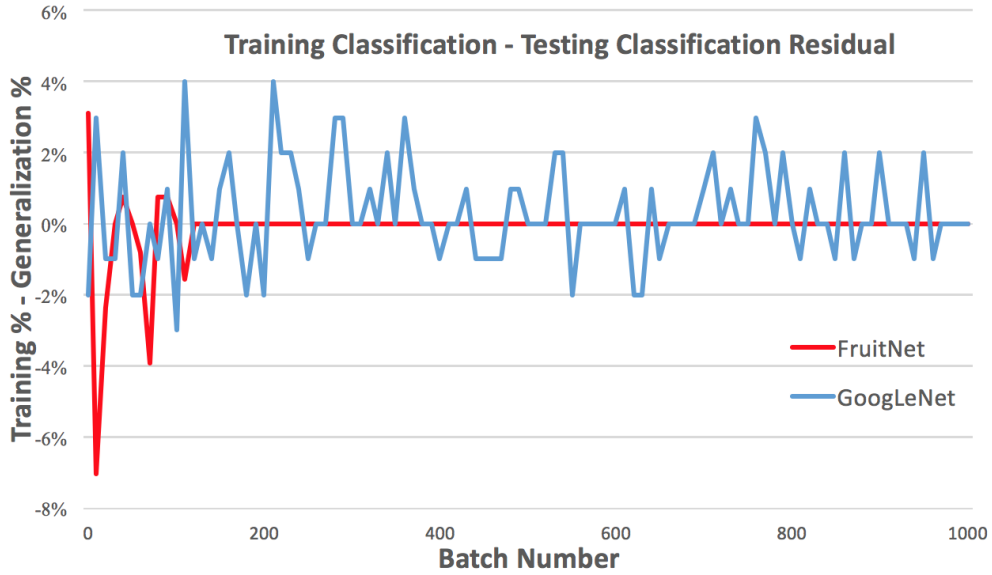

Figure 17: Residuals of Training Performance - Testing Performance

5.1.1. Advantages of FruitNet: FruitNet was trained only on thermal images and is thus more specifically tailored to read in thermal images. This is an important advantage because the network develops feature maps optimized to detect thermal features. Furthermore, FruitNet achieves a lower cross entropy loss. This means that the model is more confident in its inferences. Additionally, the model is smaller, which decreases the computational complexity of the neural network and leads to faster inference.

5.1.2. Advantages of GoogLeNet: GoogLeNet achieves an identical error rate on both the training set and the validation set as FruitNet. Furthermore, it takes only 20 minutes to train as opposed to FruitNet which takes hours or even days to complete. Additionally, GoogLeNet most likely has superior generalization skills due to its diversity of kernel sizes in its convolutional layers. This means that the model is more translationally invariant and can perform just as good on data it has not seen.

5.1.3. Solution: Ultimately, GoogLeNet was chosen to be used in this application due to its ease of use and superior generalization capabilities. The ability to re-train a model that begins with fully functional feature maps enables the application to continue to add new fruits and vegetables with ease. Instead of taking days to pickle / unpickle the data before fully training FruitNet, GoogLeNet allows for the seamless incorporation of new fruits and vegetables without sacrificing accuracy.

5.2. Results: Ability to Detect Bruising and Ripeness

The application is very successful at detecting bruising and other under-the-surface blemishes on apples as well as similarly shaped fruits (peaches, plums, and pears). In fact, the below Figure 3 shows the results using the Android application. As seen in Table 3, the application is accurate in reporting that a piece of fruit is bad given that it is indeed bad or bruised. However, it is also more prone to report good apples as "bad" than reporting bad apples as "good". Thus the model is more susceptible to report false negatives than to report false positives. While this is not ideal, it is preferable to the application reporting "bad" fruits as "good".

This susceptibility to false negatives is very likely because of the fact that the majority of the

images in the dataset were taken with a blank backdrop and thus the bruise appears as a brightly colored heat spot. This becomes a problem when using the application because heat sources in the background (people, hands, etc.) are likely to be mistaken for bruises. Thus, better results would be expected if the image was stripped of its background leaving only the thermal image of the apple on a black backdrop.

Desired Value	Total Number	# of "bad" classifications	# of "good" classifications
Good Apples	10	3	7
Bad Apples	10	9	1

Table 3: Final results on apples

While the application was successful in identifying blemished fruits, it was never successful in classifying the ripeness of fruits. This may be in large part due to the inability of the dataset to accurately label the desired value. Furthermore, the image quality was down-sampled not only to send it to the server, but also to pass it through the neural network in which the input size must be tapered. Perhaps if the image quality was not decreased and the dataset was properly labeled it would be possible to determine the ripeness of the fruit or vegetable.

6. Future Work

Given more time there are some optimizations and improvements that would make RipeOrWrong more accurate and efficient. Firstly, a background stripping algorithm would be added either in the Android application or in the classifier server in order to eliminate some noise from the input and ensure that the heated objects in the background of the photo (people, hands, objects, etc.) do not influence the result of the model. Next, although there are challenges associated with creating the application using the TensorFlow NDK, given enough time this is the better option. This is because the model can then use the full-resolution image as input, perform the classification in real time, and not need phone service or Wi-Fi. Although it would be challenging to alter the model once the application utilizes a native TensorFlow graph, the strengths outweigh the weaknesses of this

approach. Nevertheless, this approach is much more complicated, so in order to get the application working in a reasonable amount of time, the classifier was put into a server instead.

Finally, given enough time it would be ideal to develop a dataset capable of training the net to give a true ripeness estimation. This would require more time to label the fruits and vegetables as well as creating a wider network in order to not lose any of the image quality. Furthermore, the metadata of the thermal image could be read in as additional input (pure temperature readings as opposed to pixels) in order to get a more accurate and precise classifier.

7. Summary

RipeOrWrong leverages machine learning techniques along with the FLIR One thermal imaging camera in order evaluate the quality of fruits and vegetables. The classifier for the application utilizes a deep convolutional neural network capable of complex feature detection. Ultimately two models were tested, GoogLeNet and FruitNet, to ensure the accuracy of the network before GoogLeNet was selected as the model in the network. Although both models achieved a training classification error of 0%, GoogLeNet was chosen due to its superior generalization capability as well as its ability to quickly re-train itself. While the final application was successful in its detection of bruises and other beneath-the-surface blemishes, it was unable to determine the ripeness of the fruit or vegetable. RipeOrWrong is optimized for the grading of apples, but it has been modularized such that the implementation of new fruits or vegetables is a seamless process.

References

[1] J. Alammar, "Supercharging android apps with tensorflow (google's open source machine learning library)." Explorations in touchable pixels and intelligent androids, 2015.

[2] N. El-Bendary *et al.*, "Using machine learning techniques for evaluating tomato ripeness." Arab Academy For Science, Technology, and Information, 2014.

[3] Flir, "Android platform guide – flir one/cat s60." Flir, 2015.

[4] FLIR, "R&d and industrial applications for near infrared (nir) cameras," in *Technical Note*, vol. T820185, 2015.

[5] B. R. Guerra and A. H. D. Véle, "Smartphone ripeness app to simplify produce purchases." Monterrey Institute of Technology and Higher Education's, 2014. Available: http://www.freshfruitportal.com/news/2014/03/10/smartphone-ripeness-app-to-simplify-produce-purchases/

[6] D. Gunders, "Wasted: How america is losing up to 40 percent of its food from farm to fork to landfill." Natural Resources Defense Council, August 2012.

[7] H. Hellebrand *et al.*, "Chances and shortcomings of thermal imaging in the evaluation of horticultural products." Institute of Agricultural Engineering Bornim, 2000.

[8] H. Hellebrand *et al.*, "Horticultural products evaluated by thermography." Institute of Agricultural Engineering Bornim, 2014.

[9] H. Hellebrand *et al.*, "Bruises and ripeness of apples studied by thermal imaging." Institute of Agricultural Engineering Bornim, 2000.

[10] J. Hug, M. Ginsburg, and K. Wayne, "Programming assignment 7: Seam carving." Princeton University, 2017. Available: https://www.cs.princeton.edu/courses/archive/fall14/cos226/assignments/seamCarving.html

[11] T. Kaye, "Ripeorwrong repository," 2017. Available: https://github.com/tkaye407/RipeOrWrong/

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks." University of Toronto, 2012.

[13] N/A, "Cat s60 gsm smarphone." CAT, 2016. Available: http://www.catphones.com/en-us/phones/s60-smartphone

[14] J. Qin and R. Lu, "Detection of pits in tart cherries by hyperspectral transmission imaging," in *Proceedings of the 33rd Annual ACM SIGUCCS Conference on User Services*, vol. 48. American Society of Agricultural and Biological Engineers, 2005, pp. 1963–1970.

[15] L. Rothman, "This pocket-sized sensor will tell you when fruit is ripe." Modern Farmer, 2014. Available: http://modernfarmer.com/2014/07/handheld-device-might-indispensible-shopping-future/

[16] S. Sumriddetchkajorn1 and Y. Intaravanne, "Two-dimensional fruit ripeness estimation using thermal imaging." International Conference on Photonics Solutions and the National Electronics and Computer Technology Center, June 7th 2013. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1696495

[17] T. Susnjak, A. Barczak, and N. Reyes, *A Decomposition Machine-learning Strategy for Automated Fruit Grading*, vol ii ed. San Francisco, California: World Congress on Engineering and Computer Science, 2013.

[18] C. Szegedy *et al.*, "Going deeper with convolutions." Google Inc and University of North Carolina and Chapel Hill and University of Michigan, Ann Arbor and Magic Leap Inc, 2015.

[19] A. Wahi, A. Das, and I. Kothari, "Wisci: Wireless spectrometer." hackaday.io, 2016. Available: https://hackaday.io/project/13422-wisci-wireless-spectrometer

[20] H. Wang *et al.*, *Fruit Quality Evaluation Using Spectroscopy Technology: A Review*, may ed. Hangzhou 310058, China: Zhejiang University, 2015.

[21] P. Warden, "Tensorflow for poets." Pete Warden's Blog, February 28, 2016.