I. Definition

Project Overview

Computer vision is a science and engineering domain concerned with automating tasks involving the human visual system. These tasks include identifying objects within images and summarizing scenes in images. Studies conducted in the 1970s provided some of the foundations of computer vision algorithms including edge extraction and line labeling. The 1990s saw statistical learning techniques applied to the field that resulted in improved facial recognition (Eigenfaces). More recently, neural networks have been applied to computer vision tasks. Neural network techniques attempt to model the process by which humans and animals process images, namely, through a series of processing steps that recognize structural features in the early phases before combing those structures into more abstract objects in the later stages (see here for an example). One of the applications of computer vision is recognizing and reading information stored in bar codes. Using this capability, a person can take a picture of a bar code and be presented with information about the product. Fitbit has applied this technology to food tracking. With this technique, users can quickly record what they eat throughout the day. However, the current system does not allow for easily tracking food items without barcodes, such as whole fruits and vegetables. Some groups have used support vector machines to classify fruits (Zhang 2012, Rocha 2010). Rocha 2010 used specific features from the images and paired combinations of these features with different classification algorithms to achieve 90% classification accuracy. Zhang 2012 used a similar approach in which the color histogram, texture, and shape features were used with principal component analysis and support vector machines to achieve a maximum accuracy of 88%. In recent years, convolutional neural networks have become the gold standard for image recognition and I would like to apply these updated techniques to the problem of fruit and vegetable image classification. CNNs are considered the current gold standard because they have 1) a high learning capacity that can be controlled by varying the model depth and breadth[1], 2) make accurate assumptions regarding images (stationarity of statistics and pixel location dependencies) [1], 3) have fewer connections and are easier to train compared to similarly sized fully connected deep neural networks[1].

Images of produce to address this problem were obtained from the Fruit Image Dataset and Supermarket Produce dataset.

Problem Statement

The problem is: given an input image of a fruit or vegetable, can we classify the type of fruit or vegetable with greater than 90% accuracy utilizing convolutional neural networks? Fruits and vegetables can take on a variety of shapes, colors, and sizes; even for a particular type of fruit. There are also multiple types of specific kinds of fruits, such as fuji apples compared to granny smith apples. For this project, I am concerned with classifying a greater number of fruit types, without specifying specific subtypes. To solve the problem, I will utilize existing databases of fruit and vegetable images. CNNs are a good tool to solve this problem because of their ability to extract relevant features from images without the need for extensive feature engineering. However, lots of data is necessary for a CNN to appropriately extract features of a given dataset. More data available during training contributes to the model's ability to extract the relevant features. The features extracted are a function of the layers making up a particular network. Several top-performing network architectures have been created and trained on large image datasets and are now available for use in other applications, e.g., VGG16, VGG19, ResNet50, and Inception. These top-performing models are now available with pre-trained weights, allowing them to be applied to different applications. I intend for the final solution to the problem to rely on implementing the VGG16 architecture and pre-trained weights to the task of produce classification. I have chosen the VGG16 because of its relatively simple architecture compared to the other examples above. The VGG16 is shallower and narrower than the other options which should help to avoid overfitting the relatively small dataset I am using for this problem.

Metrics

I will utilize overall accuracy, intra-class accuracy, F1 score to determine the final performance of the model. Overall accuracy is defined as the total number of positive predictions divided by the total number predictions (accuracy = # correct predictions / # predictions). Intra-class accuracy is defined as the number of true positives of a specific class divided by the number of predictions made for that class (accuracy_i = # correct predictions_i / # predictions_i, where i denotes a particular class in the dataset). The intra-class accuracy accounts for the different numbers of examples there are for each class and allows to evaluate if the model is biased against specific classes. F1 score is a good metric to use when the dataset has imbalanced classes such as this one because it takes into account precision and recall for each class. Essentially, the F1 score penalizes models that would predict only the majority class of an imbalanced dataset. The F1 score is calculated as F1 = 2 * (precision * recall) / (precision + recall), where precision is the (true positive/ (true positive + false positive)) and recall is (true positive / (true positive + false negative)). F1 scores are bound between 0 and 1, with higher scores indicated better performance.

II. Analysis

Data Exploration and Visualization

A dataset containing images from the Supermarket Produce dataset[2] and the Fruit Image Dataset[4]. The Supermarket Produce dataset images were captures on a clear background at a resolution of 1024x768 pixels and downsampled to 640x480 pixels. The dataset consists of 15 different classes The Fruit Image Dataset images were collected from the internet and represent 30 different kinds of fruit. The final dataset for this project was generated by combining several of the classes (e.g., fuji apples and granny smith apples combined into a single apple class) and eliminating others (cashews). The data consisted of 34 classes of fruit/vegetables and a total of 3373 images. Figure 1 shows the number of figures in each class.

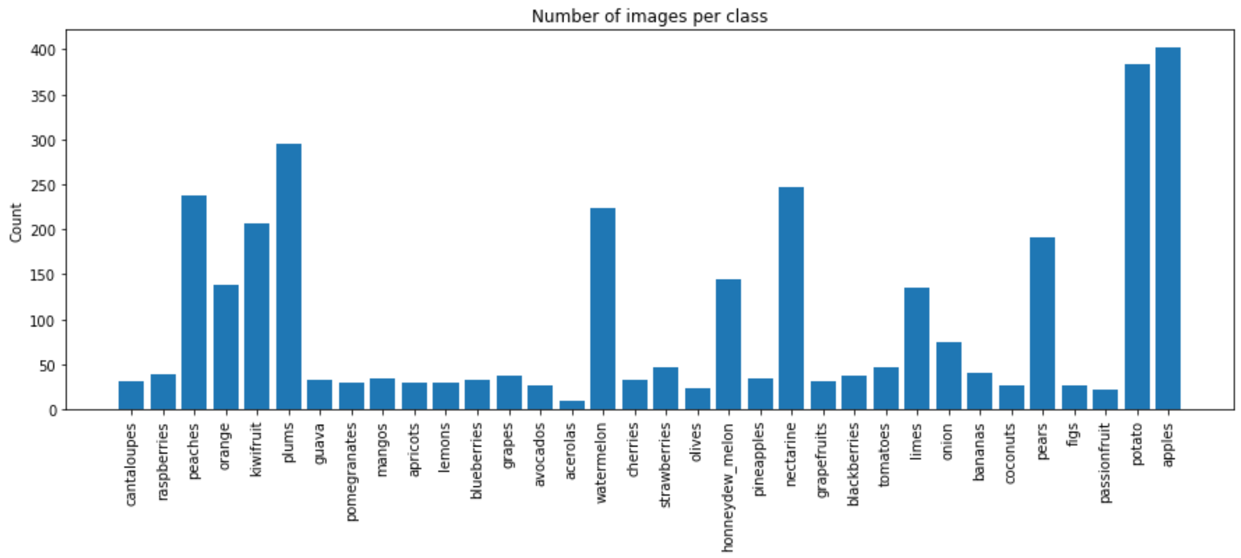

Figure 1 The number of images used for each class of fruit/vegetable.

Several example images from each source dataset are shown in figure 2. There is a clear difference between the images collected by the different authors. Rocha 2010 sought to create a program that could identify fruit at the checkout counter of grocery stores. The images from that study represent the fruit in a specific scenario. The Skrjanec 2013 images represent more general representations of fruit. They include images of fruit in bowls, on trees, sliced open, and from multiple perspectives. Combining these datasets is likely to reduce the performance of the final model, as the Skrjanec images have a high amount of variability and only a small number of examples for each class (~32). These images could be eliminated from my dataset to increase performance. However, my goal in the project is identifying a greater number of classes, as opposed to maximizing accuracy.

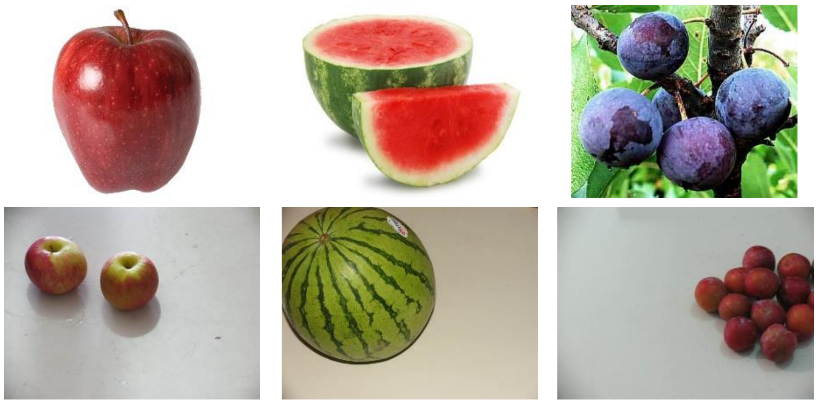

Figure 2 Image examples from the two datasets used in this classification task. The top row shows examples Skrjanec 2013, the bottom row comes from Rocha 2010. The images are of apples, watermelons, and plums.

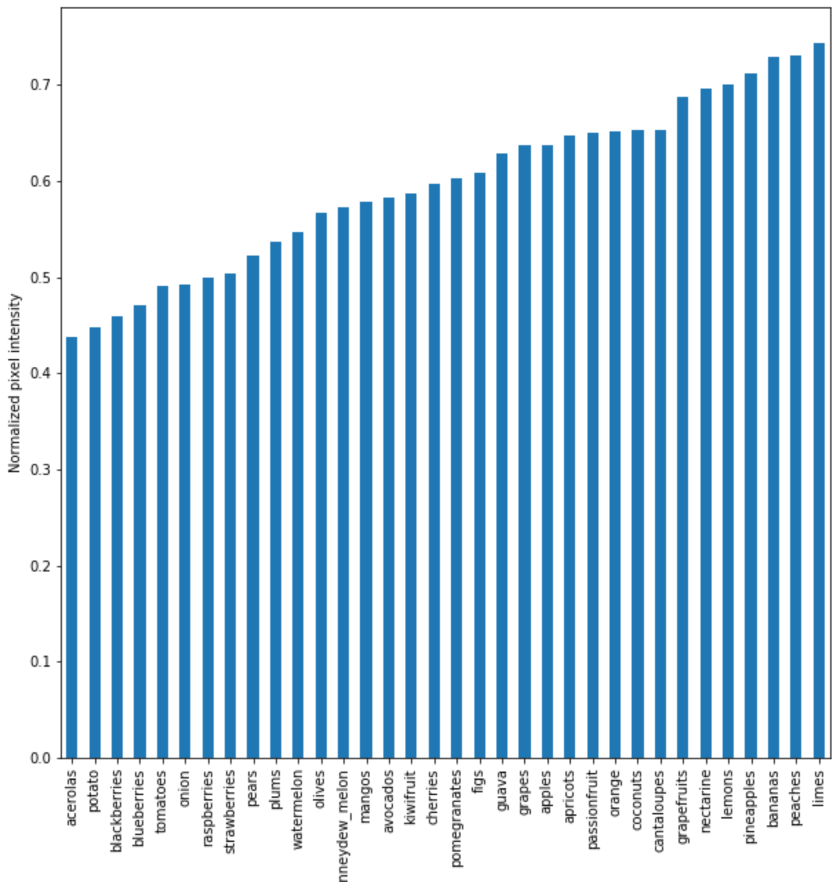

Figure 3 Shows the average normalized pixel intensities of images from each class.

The model accepts normalized pixel intensities as input. Figure 3 shows the average pixel intensity for each class of images. Limes, peaches, and bananas have the highest average pixel intensity and acerolas and potatoes have the lowest average pixel intensity.

Algorithms and Techniques

Convolutional neural networks have several strengths and weaknesses. The weaknesses include lack of theory explaining how much data or layers are needed to get a particular level of performance, they are computationally expensive, there are lots of hyperparameters to tune, and they require lots of data. The strength of convolutional neural networks is the assumption that locality in the data is important. This works well for image data, as pixels that are near to each other should be close to each other in the image. Thus, convolutional neural networks are a better choice for image data compared to other machine learning algorithms, such as support vector machines (which we use as a baseline classifier).

The convolution operation determines the amount of agreement between two pieces of data. For image classification, this means that the pixel values within the receptive field of the filter are multiplied by the weights of that filter. The product of the pixels and weights are summed together, resulting in one number that represents the "agreement" between the weights and the pixel values at that location. This convolution operation results in a feature map. The resulting feature map then becomes the input for the next operation in the network. Max pooling can be applied to the feature maps to select the largest value in a defined region and pass that value on to the next layer of the network[8].

One of the ways to prevent overfitting when using this small dataset is by using dropout. Dropout randomly eliminates some of the connections between neurons during training. This causes the network to make redundant connections and learn multiple representations of the data.

The classification task was performed using the VGG16 convolutional neural network architecture. The VGG16 network consists of five convolution layers followed by several fully connected layers. The operations performed in each of the layers is shown in figure 3. The VGG16 network was created by Krizhevsky et. al. and trained on 1.2 million images with 1000 different classes. Instead of training an entirely new version of the VGG16 architecture on my dataset, I utilized the training done by Krizhevsky using a technique called transfer learning. I used the existing VGG16 weights to extract features from my images and then used those features to train a new fully connected layer on top of the convolutional layers. Changes to the default input and output to the VGG16 network were made to accommodate my chosen dataset. The input size was increased from 224x224 to 250x250. I chose a larger input because the images in my dataset were of varying heights and widths and I wanted to avoid overly aggressive downsampling. Some early exploratory manipulation of the images indicated that the important image information was lost when the images were made to small. The output layer of the VGG16 network shape was changed from 1000x1 to 34x1 to be compatible with the 34 classes in my dataset. A softmax classifier determined the class from the final output layer. I chose to use the categorical crossentropy loss function. This function measures the probability error in discrete classification tasks[7]. The architecture of the VGG16 CNN is shown in figure 3.

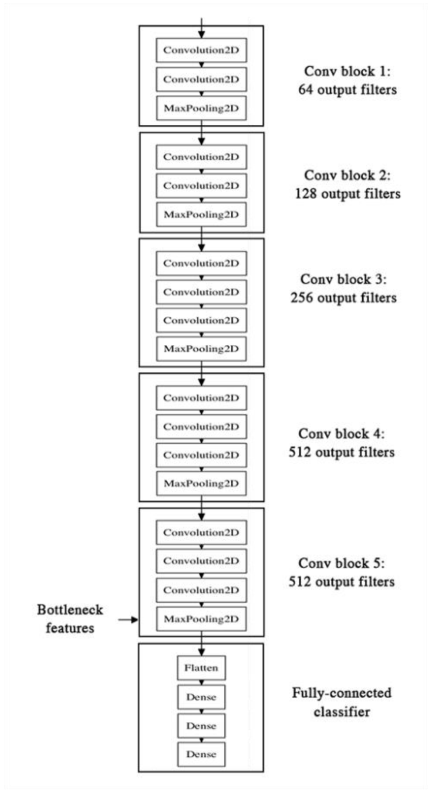

Figure 4 VGG16 network architecture

Benchmark

To benchmark the performance of the model, I used a dummy classifier that simply predicts the most frequent class in the training set. Apple was the most common class in the training set with 241 examples. Simply predicting that all images in the validation set are apples results in a predictive accuracy of 35.7%. I also used the default support vector machine algorithm from sklearn as a second benchmark. Using the normalized pixel values as features, the support vector machine algorithm reached an accuracy of 38.6%.

III. Methodology

Data Preprocessing

I downloaded the images in compressed formats. The first step was to unzip the individual image files and place each class of images in its own directory. Images from similar classes were combined (fuji apples and granny smith apples were both labeled as apples). I then tested that the files were usable by looping though the files and attempting to open the jpeg files. Several files were not able to be opened and were removed from the dataset. I then created a directory structure that would easily allow images to be retrieved from disk at training and test time. Next,

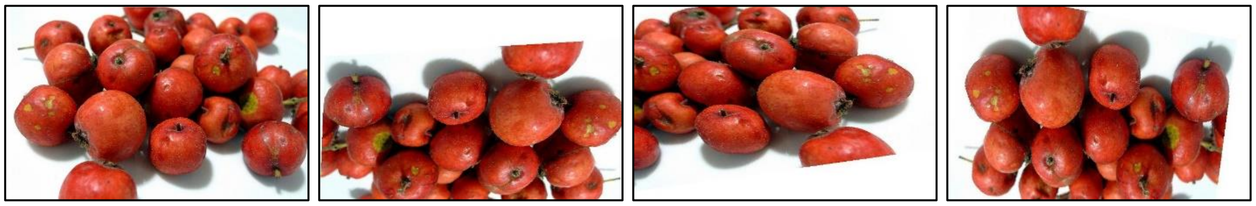

Figure 5 Original (left) and transformed examples of a single training image.

```
Data
…Train
…….Class 1
…Valid
…….Class1
…Test
…….Class1
```

I removed Windows thumbnail files from each of the directories holding the images and then split the images into training, validation, and test sets using a 60/20/20 split. Because the dataset I am using is relatively small, I applied several transformations to training data when being fed to the learning algorithm. This helps to protect against over fitting by preventing the algorithm from seeing the exact same image during the training epochs. Training images had the following transformations randomly applied: 1. Rotation (+/- 40 degrees), 2. Random horizontal shifts (up to 20%), 3. Random vertical shifts (up to 20%), 4. Shear (up tp 0.2 radians), 5. Horizontal flips, and 6. Vertical flips. When transformations resulted in the original image being outside the boundaries, the "empty" parts of the new image were filled with white pixels. Examples of images after these transformations are shown in figure 4. Additionally, all images (train/validation/test) were resized to 250 by 250 pixels and pixel values were normalized to between zero and one.

Implementation

I implemented the CNN for the classification task using the Keras (v2.0.6) library with Tensorflow (v1.1.0) backend for Python (v3.5.3). Keras provides a high level API for constructing deep learning architectures. The initial model consisted of 3 convolutional layers followed by a fully connected layer and output layer. The convolutional layers each had a 2-dimensional convolutional operation with a kernel size of (3x3), a rectified linear activation unit (ReLU), 2-dimensional max pooling with a (2x2) receptive field. The kernel in the first two layers had a depth of 32, and the final kernel had a depth of 64. The fully connected layer had 64 hidden nodes with ReLU activation connected to 34 output nodes with a softmax activation function. I chose categorical cross-entropy as the loss function, RMSprop for the gradient descent optimizer, and calculated accuracy as the output metric. RMSprop adjusts the learning rate during the training by dividing the gradient by a running average of its recent magnitude (http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf). Batch size during training was set to 16.

To reduce overfitting on the relatively small dataset, I utilized dropout and early stopping during training. Half of the connections between nodes in the fully connected layer were randomly removed during training. All models were trained for 50 epochs. All results for the model report the accuracy that occurred during the epoch with the smallest loss value, which always occurred before the 50th epoch.

The Keras library simplified the implementation steps of the building the models. The most time-consuming steps of the process were transforming the data at different steps of the process. For example, preparing the data for Keras was taken care of by using the built-in data generator functions. However, when using sci-kit learn for the baseline support vector machine, I needed to build the pipeline to convert the image filenames into vectors of pixel values that could be fed to sci-kit learn's support vector machine algorithm.

Refinement

Improving on the initial model consisted of two strategies: adding class weights to handle the unbalanced classes and using transfer learning with pre-trained models. After training the initial model, I calculated the class weights and used that information in the second trial with the architecture described above. In the next phase of refinement, I moved on to using the VGG16 network architecture. The first phase of fine-tuning using the VGG16 architecture utilized the 5 convolutional layers as feature extractors. The training and validation images were passed through the convolution layers one time. The resulting feature arrays were then used to train the fully connected network. I used this strategy without and without providing the class weight information.

IV. Results

Model Evaluation and Validation

The final model utilizes transfer learning with class weights to achieve a validation accuracy of 80.61% using the VGG16 architecture. This model also had the highest F1 score, although it was not significantly different from the VGG16-FE model ($p > 0.05$).

Table 1 – Validation accuracy and test accuracy of tested models.

Model	Accuracy - Validation	F1 Score	Test Accuracy
3-layer CNN	62.67% +/- 1.75%	0.52 +/- 0.03	
3-layer CNN_CW	53.79% +/- 1.49%	0.53 +/- 0.01	
VGG16-FE	81.30% +/- 0.58%	0.80 +/- 0.01	
VGG16-FE_CW	80.61% +/- 1.44%	0.81 +/- 0.01	80.67%

I further evaluated the VGG16-FE_CW model with previously unseen test data. Accuracy on these new images was 80.67% with intra-class accuracies shown in table 2. Several classes of fruit were unable to be learned by this model at all. Acerolas, cherries, lemons, and olives were never predicted by the model. Even though this model takes class weight into account, the small amount of data is likely affecting the ability of the model to learn the characteristics of some classes.

Table 2 – Accuracy class accuracy of VGG16-FE_CW evaluated on test data.

Class	Intra-class Accuracy

acerolas	0.00
apples	90.12
apricots	33.33
avocados	16.67
bananas	87.50
blackberries	87.50
blueberries	71.43
cantaloupes	14.29
cherries	0.00
coconuts	66.67
figs	66.67
grapefruits	85.71
grapes	37.50
guava	28.57
honneydew_melon	100.00
kiwifruit	97.62
lemons	0.00
limes	62.96
mangos	14.29
nectarine	98.00
olives	0.00
onion	93.33
orange	85.71
passionfruit	60.00
peaches	85.42
pears	89.74
pineapples	71.43
plums	89.83
pomegranates	83.33
potato	84.42
raspberries	62.50
strawberries	50.00
tomatoes	70.00
watermelon	93.18

Justification

The best CNN was more than 2x more accurate than the baseline SVM model (80.95% vs. 43.80+/-0.04%). The accuracy of the CNN is significantly better as determined by a t-test comparing test accuracies of the two models. These results are significantly better than the baseline model. While the CNN model implemented here did perform better than the baseline models, it did not perform better than the accuracy achieved by Rocha 2010. However, the reduced performance of my solution is likely attributable to having a greater number of classes with fewer examples per class compared to Rocha 2010.

V. Conclusion

Free-Form Visualization

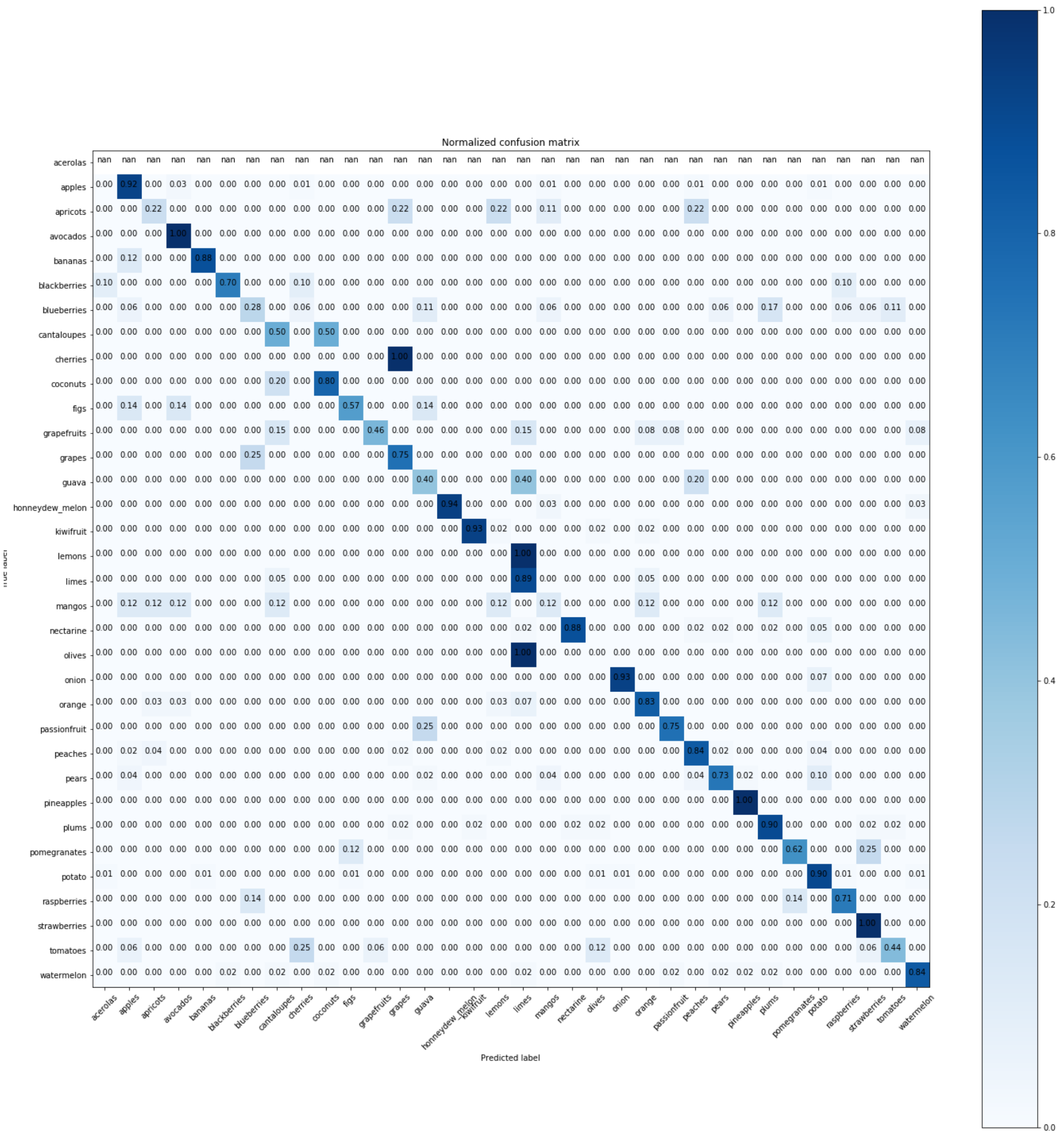

Figure 6 Shows how images in the test set were labeled. The true label is shown on the y-axis and the predicted label is shown on the x-axis. The color of the square indicates how accurate the classification of that class was, with darker colors indicating more accurate classification. Looking across the row shows the labels that were assigned to images in a given class.

The confusion matrix in figure 5 shows how well the classifier worked on the individual classes. The diagonal line of the matrix indicates instances when the predicted label matched the true label. The darker colors along the diagonal indicate good classification of these labels. Apricots, blueberries, grapefruits, guava, lemons, mangos, olives, and tomatoes had classification accuracies that were less than 50%. Lemons and olives proved particularly difficult for the model, all of these images were predicted to be limes. Cherries were also all incorrectly labeled as grapes. The label acerola was never predicted by the model (which is why there is a row of 'nan' in the figure). In general, the model had difficulty with produce that has similar colors and sizes. A much larger dataset is likely needed for the model to learn the difference between produce that share similar characteristics, e.g., a bunch of grapes and a bunch of cherries.

Reflection

Produce are generally not labeled with barcodes, making it harder to quickly scan the item for use with diet tracking apps. I sought to create an image classification algorithm that could identify fruits and vegetables in images to allow for easy diet tracking. I utilized a convolutional neural network to classify up to 34 different kinds of produce. The final CNN model highlights the value of deep learning. Given enough images, these kinds of models can learn the relevant features without the difficult task of feature engineering. My model was able to achieve 80% accuracy with a relatively small number of images. Often times, deep learning models are trained with tens of thousands of images. When providing the pixel values to the baseline SVM model, the model only provided a slight bump above a majority classifier. In order to achieve the type of results provided by Rocha 2010 (SVM with 90% accuracy), considerable research investment is needed for feature engineering and selection. Deep learning models turn the task into a problem of collecting enough data and running the computations on faster computers to by-pass the feature engineering problem.

The final results achieved by my model fall short of the goal of 90% accuracy. I have focused only on accuracy of the top prediction. Using top-3 or top-5 accuracy is likely to bring the accuracy closer to 90%. A system which outputs the top 3 or 5 choices to the user for final selection would still represent an improvement over the current system which requires the user to type and search for the food they would like to record.

Improvement

I see three potential ways to improve the implementation of my model. One, different model architectures that have been pre-trained on large datasets could be used in place of the VGG16 architecture I used. VGG19, ResNet50, Inception, and Xception could all be used in place of VGG16. It is possible that one of these works better than VGG16 for my application. A comparison of each model architecture on my data would allow for the selection of the model with the greatest accuracy. Two, I could use a larger dataset. I have limited myself to datasets that were readily available. However, scraping publicly available images from Flickr would allow me to build a larger dataset and presumably increase the number of classes I can train and the accuracy of the model. Three, I could implement the feature engineering performed by Rocha 2010. At the moment, the implementation of their feature extraction techniques is beyond me. It would be interesting to see if their technique scales to a greater number of classes and more variation in the pictures. Their dataset uses images that are consistent with regards to lighting, photo angles, fruit size, etc. I have added images to the dataset that do not fit the restraints they placed on their original image capture process. Using my current model as a benchmark against each of these potential improvements would set the stage for building toward a better, more accurate model.

References

1. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton; ImageNet Classification with Deep Convolutional Neural Networks
2. Anderson Rocha, Daniel C. Hauagge, Jacques Wainer, Siome Goldenstein; Automatic fruit and vegetable classification from images
3. Yudong Zhang and Lenan Wu; Classification of Fruits Using Computer Vision and a Multiclass Support Vector Machine
4. Škrjanec Marko; Automatic fruit recognition using computer vision, Bsc Thesis, (Mentor: Matej Kristan), Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2013
5. Building powerful image classification models using very little data; https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
6. Save and Load Your Keras Deep Learning Models, https://machinelearningmastery.com/save-load-keras-deep-learning-models/
7. https://www.tensorflow.org/api_docs/python/tf/nn/softmax_cross_entropy_with_logits
8. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/