Fruit Classification Using Convolutional Neural Network

Henry Tappa, Ryan Kinsey, Eric Goldman

The George Washington University

Introduction

Image classification tasks using Convolutional Neural Networks (CNN) have been an active area of research within recent years (Li et al., 2017). CNNs take advantage of the unique way computers interpret images. A computer interpretation of a color image is expressed as a matrix of values, ranging from [0, 255], representing the specific color of the pixels on the "RGB" scale (Udofia, 2018).

A key component of CNN is the utilization of Kernels. Kernels are typically a smaller matrix than the input image matrix that work to identify key components of an image. These kernels are convolved over the image to obtain a feature map of the specific features that appear in the image (Saxena & Rarr, V. 2019).

For this project we decided to build a fruit classification network using a 2-dimentional CNN architecture. The CNN was built in Python using Pytorch as the framework and was run in the cloud using a Google Cloud Platform instance configured with one GPU. Potential applications of this network include automating Customs and Border Protections (CBP) processes or for use in the agricultural industry (Vibhute and K. Bodhe, 2012).

In 2018, CBP intercepted 2.5 tons of prohibited fruits from Mexico in the state of Ohio. The fruits were prohibited due to the potential for foreign diseases and pests to enter the border. Currently, agriculture specialists inspect by hand the cargo that enter their territory. Deep Learning has the potential to inspect the fruit through imagery. This would enable the CBP to inspect a higher percentage of products and potentially catch a higher rate of illegal fruit (Vanderhorst, 2018).

The Dataset

The data set that we used was the "Fruits 360 dataset," a Kaggle dataset with over 20,000 images of 33 types of fruit (Kaggle.com, 2017). The images are full-color, 100x100 pixels and treated as such in the network. No cropping, resizing, or grey-scaling was necessary. The pictures were taken on a white background and the camera was rotated incrementally 360° around the fruit taking pictures every 20°.

The only image manipulation was to normalize the images with a mean and standard deviation of 0.5. The result is that the normalized image is in a range from [-1, 1] for each of the color channels. The purpose of normalization is to reduce the skewness of the images so that the CNN can better learn the data in hopes of faster convergence.

The data, as structured, was split into a training set with 15,506 images and a testing set with 5,195 images. Figure 1 contains a single mini-batch of images in the dataset. Given the size of the overall dataset and the potential latency issues associated with it, we felt it necessary to first implement our network on a subset of the data. We called this dataset "fruits_data_subset" and it only includes 5 different fruit types. This collection is split into a training set with 2,432 images and testing set with 814 images. Figure 2 contains 1 mini-batch of images in the fruits_data_subset dataset.

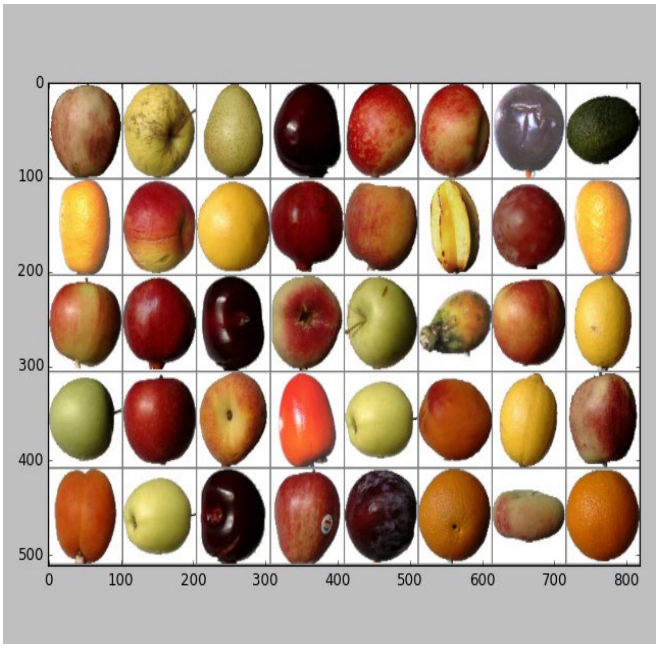

Figure 1. Mini-Batch of Data from Overall Dataset

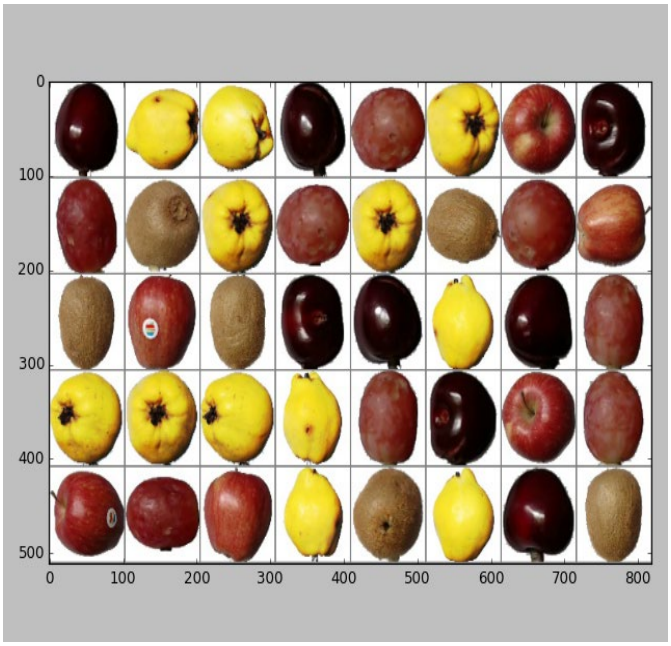

Figure 2. Mini-Batch from 'fruits_data_subset'

Convolutional Neural Network

Using the `torch.nn` package from `pytorch.nn`, we implemented a two-layer sequential module in `fruits_cnn_subset.py` and `fruits_cnn_all_2layer.py`, and a three-layer sequential module in `fruits_cnn_all_3layer.py`. Pytorch was the preferred framework given our familiarity with it and the lack of prior research on this dataset using Pytorch. As briefly mentioned above, the file was run in a Google cloud environment for access to a GPU.

The first layer is a `nn.Conv2d` layer with 3 input channels (Red, Green, Blue channels), 32 output channels, a kernel size of 5x5, and padding of 2. Output channel size, kernel size, and padding were initially chosen at random. After many iterations trying different combinations, these appeared to be the optimal values. `BatchNorm2d` was applied in order to normalize the mini-batches, allow for faster convergence, and reduce the need for a dropout layer (Ioffe and Szegedy, 2015). Following the convolution, a non-linear activation function was applied, in this case ReLU, which changes all negative inputs to 0 allowing for faster convergence and protecting against the vanishing/exploding gradient problem (Xu et al., 2015). Lastly, a 2-dimentional max pooling layer is used to reduce the size of the feature map as well as decrease the parameters and computation in the network (Yamashita, 2018).

The second layer is nearly identical to layer one, aside from the differences in the input and output channels. The input channel for the second layer equals the output of the first layer, 32. Additionally, the output of the second layer is set at 64. In the `fruits_cnn_all_3layer.py` file we added a third layer which is nearly identical to the second one, but with the output doubling to 128. Finally (for all three files), we flatten the output of the previous layer into a vector and feed it into a fully connected layer. The output of the fully connected layer applies a linear transformation to the incoming data and outputs the target class.

For our specific CNN implementation, Cross-Entropy Loss is the preferred loss function. Cross-Entropy works by penalizing an incorrect prediction probability. Simply put, the goal is to have a loss as close to 0 as possible. However, if the loss results in any value other than 0 it means that the predicted probability of a certain target class could be incorrect. The model will then update the weights and biases appropriately in hopes of driving the loss downward and thus becoming more confident in the predicted class.

The ADAM optimizer was used within our CNN model due to its effectiveness in similar image classification tasks. On a theoretical level, ADAM is superior to classic stochastic gradient descent because it computes individual learning rates for different parameters (Kingma and Ba, 2014). Stochastic gradient descent, on the other hand, has only one arbitrary learning rate for all parameters in the network. The result is that ADAM is able to converge much more quickly than other optimizers while maintaining accuracy (Kingma and Ba, 2014).

Experimental Setup

To begin with, we trained and tested our model on a subset of the overall data. As briefly mentioned above, this dataset was called “fruits_data_subset” and only included pictures of red apples, cherries, grapes, kiwi’s, and quinces. These particular fruits were strategically chosen because they look very different from one another and it should be relatively easy for a CNN to distinguish the difference. Also, we were worried about latency given that the overall dataset would be training and testing on over 20,000 images. Essentially, we wanted to get our network working properly before running on a more complex dataset.

Once our model was running properly, we trained and tested it on the overall dataset. This dataset included all images for all 33 fruit classes. As expected, the model took much longer to train and test, taking 110 seconds to run, whereas the model took only 23 seconds to run on the smaller subset. Though we were happy with the results (which will be discussed in the next section), we decided to add a third layer to our model to see if we could even further improve the model’s accuracy. This took only slightly longer to run, at 137 seconds.

Results

Our first model, which uses a 2-layer network on a subset of the data (5 classes: red apples, cherries, grapes, kiwi’s, quinces) reaches 100% accuracy after just 6 epochs. Figure 3 shows the confusion matrix associated with this 5-fruit class model.

	Apple Red 1	Cherry	Grape	Kiwi	Quince
Apple Red 1	**164**	0	0	0	0
Cherry	0	**164**	0	0	0
Grape	0	0	**164**	0	0
Kiwi	0	0	0	**156**	0
Quince	0	0	0	0	**166**

Figure 3. Confusion matrix for fruit_data_subset

Our second model, which uses the same 2-layer network on the whole dataset (33 classes) and the same number of epochs, has an accuracy of 97%. Figure 4 shows a table representing the number of fruit class misclassifications when using this model.

	Apple Red 1	Apple Red Yellow	Braeburn	Cherry	Golden 3	Granny Smith	Nectarine	Peach
Apple Red 2			**10**					
Braeburn							**36**	
Nectarine		**19**						**13**
Peach	**17**							
Pear					**26**	**4**		
Plum				**11**				

Figure 4. Number of Misclassifications, 2-Layer Network

The images below (Figure 5) give a visual representation of these misclassifications. Just by looking at these images, it's clear that these fruits are very difficult to discern from one another, even for the human eye.

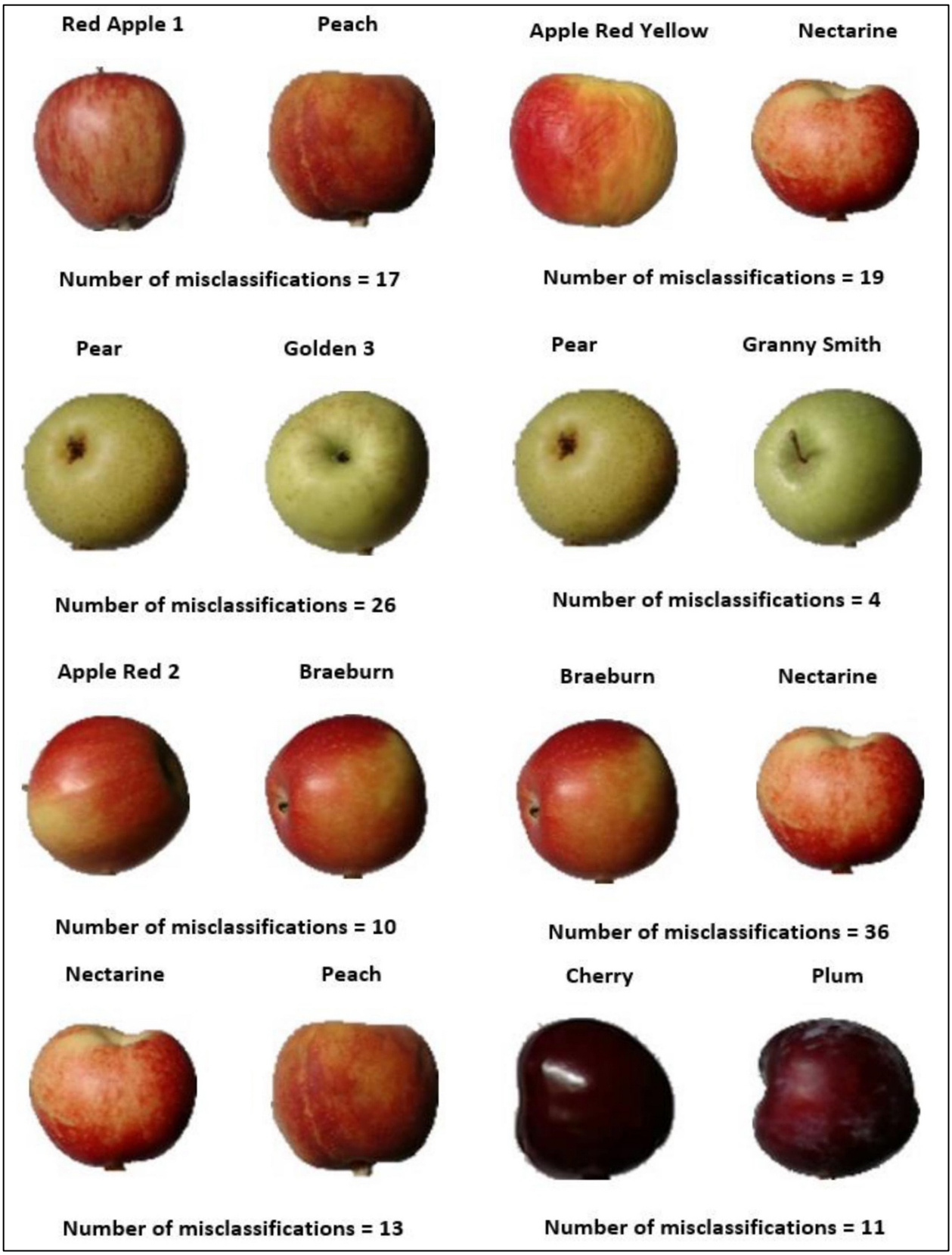

Figure 5. Fruit Misclassifications, 2-Layer Network

Increasing the epochs did not substantially change the accuracy of this model. Therefore, we added a third layer to see if training the network even further could have a significant effect on the accuracy. The new 3-layer model had an accuracy of 99%, and the only misclassifications were between Braeburn and Nectarine, with 33 misclassifications. Though the model did learn to correct itself with all of the fruits that were misclassified in the 2-layer network, it still had trouble distinguishing between these two fruits.

Given the way our training and testing for-loops were written in the code, the initial output was not intuitive. It took me some time, but we figured out that the loss and accuracy data are plotted per mini-batch training. Our dataset was split using minibatches with about 2,300 mini batches that were then split up into iterations per epoch. Knowing this we reformatted the axes of our plots with appropriate information. The end result represents the model's learning as it advances in epochs.

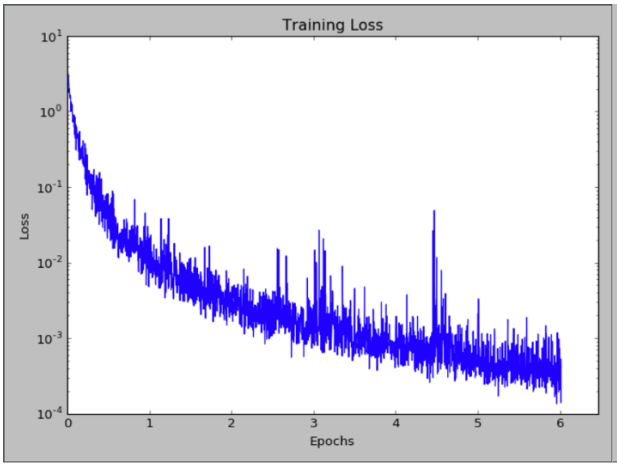

Figure 6. Cross-Entropy Loss, 2-Layer Network

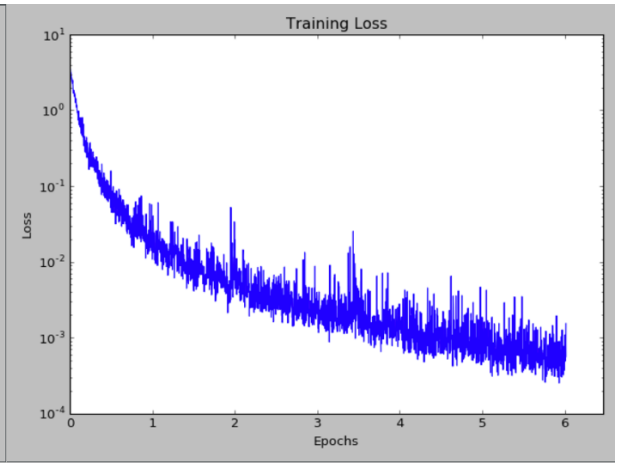

Figure 7. Cross-Entropy Loss, 3-Layer Network

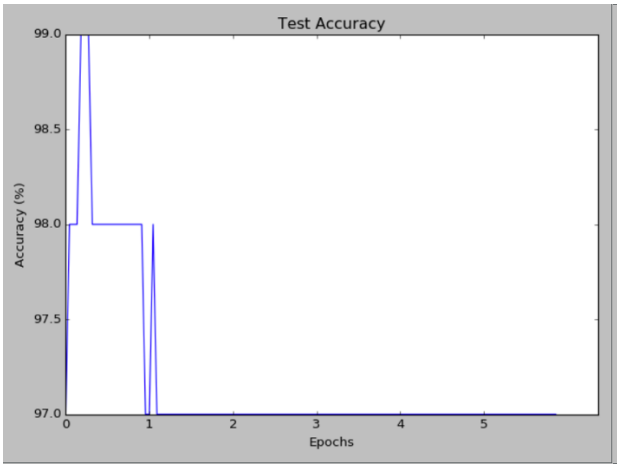

Figure 8. Test Accuracy, 2-Layer Network

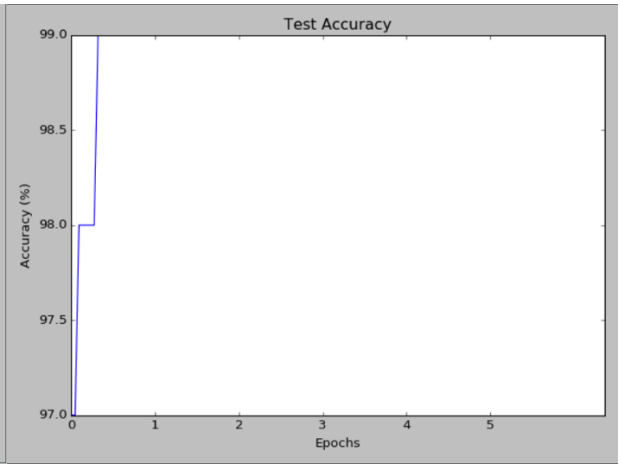

Figure 9. Test Accuracy, 3-Layer Network

Summary

Convolutional Neural Networks are an important field for computer vision and offer a powerful yet accessible way for researchers to use computers to interpret images. Fruit classification is a key use for CNNs and can potentially automate the inspection process of fruits for Customs and Border Patrol, allowing a higher and more accurate inspection rate. The CNN model that we developed for this project was able to classify the test set from the Fruits 360 dataset of 5,195 images with 99% accuracy, which we determine to be a very successful classification rate. Further research would involve investigating the misclassifications for “Braeburn” and “Nectarine” and determining how these could be mitigated. The images in the Fruits 360 dataset were specifically processed for the purpose of machine learning, so further research could also involve testing the network on raw, unprocessed photographs of fruit, and making further adjustments based on the results of these classifications.

Works Cited

W. Li, G. Wu, F. Zhang and Q. Du, "Hyperspectral Image Classification Using Deep Pixel-Pair Features," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 844-853, Feb. 2017. doi: 10.1109/TGRS.2016.2616355

Udofia, U. (2018). [online] Available at: https://www.researchgate.net/publication/325803364_A_Study_on_CNN_Transfer_Learning_for_Image_Classification [Accessed 17 Apr. 2019].

Saxena, A. & Rarr, V. (2019). Convolutional Neural Networks (CNNs): An Illustrated Explanation - XRDS. [online] XRDS. Available at: https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/ [Accessed 20 Apr. 2019].

Kaggle.com. (2017). Fruits 360 dataset. [online] Available at: https://www.kaggle.com/moltean/fruits/version/2 [Accessed 6 Apr. 2019].

Vanderhorst, D. (2018). CBP intercepts prohibited Mexican fruit [online] Available at: https://www.thepacker.com/article/cbp-intercepts-prohibited-mexican-fruit [Accessed 6 Apr. 2019].

Vibhute, A. and K. Bodhe, S. (2012). Applications of Image Processing in Agriculture: A Survey. International Journal of Computer Applications, 52(2), pp.34-40.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. [online] arXiv.org. Available at: https://arxiv.org/abs/1502.03167v3 [Accessed 16 Apr. 2019].

Xu, B., Wang, N., Chen, T. and Li, M. (2015). Empirical Evaluation of Rectified Activations in Convolutional Network. [online] arXiv.org. Available at: https://arxiv.org/abs/1505.00853 [Accessed 17 Apr. 2019].

Yamashita, R., Nishio, M., Do, R.K.G. et al. Insights Imaging (2018) 9: 611. https://doi.org/10.1007/s13244-018-0639-9

Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980 [Accessed 19 Apr. 2019].