**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). CNN’s 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 ran in the cloud using a Google Cloud Platform instance configured with 1 GPU. Potential applications of this network include automating Customs and Border Protections processes or for use in the agricultural industry (Vibhute and K. Bodhe, 2012).

# The Dataset

A Kaggle dataset with over 20,000 images of 33 types of fruit were used (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 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 1 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\_subset1’ and it only includes 5 different fruit types. Fruits\_data\_subset1 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\_subset1 dataset.

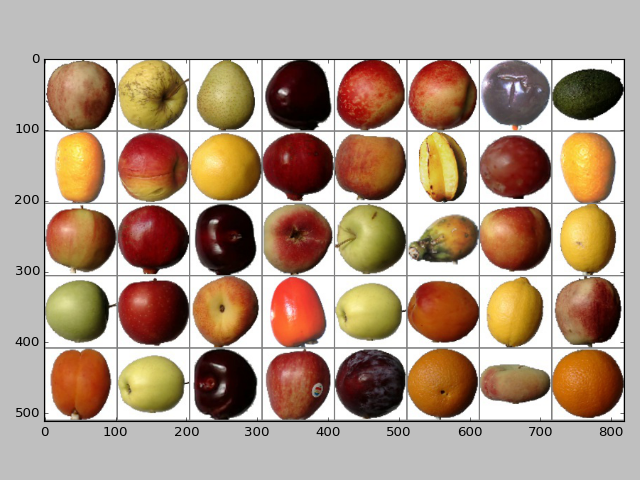


Figure 1. A mini batch of data from overall dataset

# Convolutional Neural Network

Using the torch.nn package from pytorch.nn we implemented a 2-layer sequential module. Pytorch was the preferred framework given our familiarity with it and the lack of prior research on this dataset using it. As briefly mentioned above, the file was run in a GCP cloud environment for access to a GPU.

The first layer is a nn.Conv2d layer with 3 input channels (Red, Green, Blue channels), 32 ouput channels \*\*\*Insert what it means and why we chose that\*\*\*, a kernel size of 5x5, and a padding of 2. \*\*\*\*Insert here why we chose 5x5 and padding of 2\*\*\*\* 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. Finally, we flatten the output of the second 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.

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\_subset1’ and only included pictures of red apples, cherries, grapes, kiwi’s, and quinces. These particular fruits were strategically chosen because they look very different and it should be relatively easy for a CNN to distinguish the difference.

# Results

# Summary

# 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].

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].