On Image Preprocessing Techniques in Classification with Neural Networks

Tyler G. Hudson

Texas State University, San Marcos, TX

E-mail: [*tgh22@txstate.edu*](mailto:tgh22@txstate.edu)

# Abstract

Classification of images is one of the most commonly-discussed functions of deep neural networks, especially with the recent invention of the convolutional neural network and various modifications thereof to imitate neurons within the human eye. In spite of their many strong points, Neural Networks of any kind can benefit from data preprocessing in order to decrease the amount of noise and constrain the information in an image to that which is relevant.

In this paper I discuss the effects of a battery of different image preprocessing techniques on the output accuracy of an Xception neural network pretrained on the imagenet dataset and using a binary subclass network sublayer. I experiment on the Seedling imageset from Kaggle using various image editing methods ranging from image equalization to transformation to image masking to delete background information.

This study shows that images subjected to equalization by several means are less suited for classification. Images subjected to random transformations using Keras’ ImageDataGenerator were usually classified slightly better to those left alone. Images of seedlings segmented using various methods are much more accurately classified by the network.

# Introduction

Image categorization has been an exceptionally useful function of neural network architecture for much of its recent history following the advent of the convolutional model and its later iterations. Discussions in neural network classification often float around the architecture of the network itself and not what can be done to sanitize data before it enters the network.

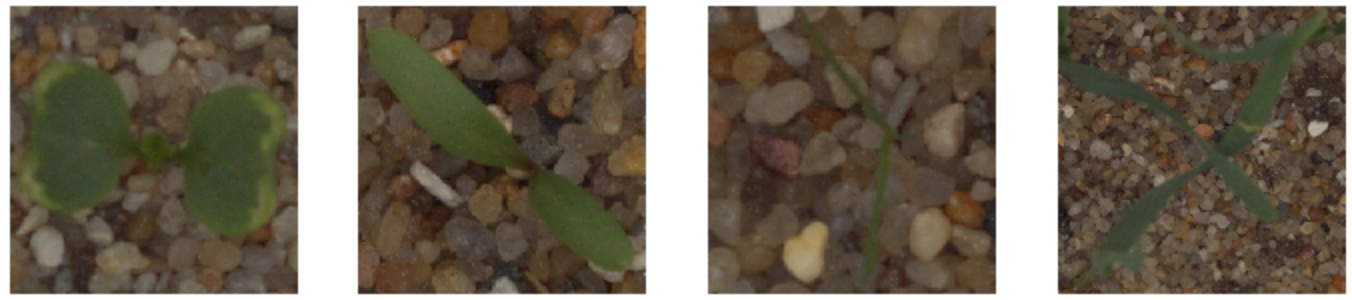
In this paper I attempt to quantize the effects of a small handful of preprocessing methodologies using accuracy on a training set as an indication of the successfulness or failure of a method.

Below are listed the methods which I have implemented or discovered being implemented on the Kaggle playground competition for this particular dataset:

|  |  |  |
| --- | --- | --- |
| 1. | Unedited Images | Images with no preprocessing. |
| 2. | Histogram Equalized | Each image was run through the histogram equalization method. |
| 3. | HSV Histogram Equalized | Each image was converted to HSV, run through the histogram equalization method, and converted back. |
| 4. | Gábor Vecsei’s algorithm (5) | Images pre-segmented using Kaggler Gábor Vecsei’s image segmentation algorithm were used in place of the ordinary imageset. These were segmented by converting to HSV and then masking off all areas that were not within some range of a particular shade of green which Vecsei selected. |
| 5. | CIELUV Color Thresholding (3) | Images were segmented at runtime using CIELUV color thresholding. |
| (x) | ImageDataGenerator | Images were rotated, flipped, stretched, and zoomed at random using Keras’ ImageDataGenerator. All of the above techniques were tested with and without this. |

The intention of using these methods was to test several very different classes of techniques against each other to get a general idea of how these classes of technique effect classification accuracy on the network.

# Data





*Above: (Top) A sample of images in the seedling dataset, (Bottom) A sample of seedling images segmented using Gábor Vecsei’s algorithm (5)*

The data used in this project was taken from the Kaggle Seedling Classification playground training dataset as well as a dataset generated from the previous dataset and segmented using Gábor Vecsei’s algorithm. This dataset contains 4,750 images of 12 different species of seedling with the challenge of classifying the images accurately into their correct species. Images were split into two groups, one consisting of approximately 10% of the images as a test set and the rest were kept as a training set. The images were then resized to 300x300 pixels and converted to 300x300x3 arrays of floats. With each run of the program, the data was pre-processed using some combination of the above-mentioned techniques.

# Model

The neural network that I used was constructed in Keras from three parts:

## Image result for xception networkXception Pretrained Network

*Left: An image describing the Xception neural network architecture (2)*

An Xception network pretrained on the imagenet dataset was used as the head of the network, extracting features from the image and arranging them into a vector of numerals using global average max pooling. This head network is set to not be trainable.

## 12 Sparse Binary Subclass Networks

Sparse Binary Subclass Networks are the name I will be giving to this section of the network because I haven’t seen this technique used previously. The Xception head of the network feeds its resulting formation into twelve parallel dense subnetworks, each consisting of a dense network with 512 outputs using the tanh function, a dropout layer with a 0.25 parameter, a batch normalization layer, and a dense network with a single output using the sigmoid function. These subnetworks were intended to read the information and each guess whether or not the pattern output by the Xception network indicated that the plant belonged to a particular class or subclass. It’s unclear if these networks achieved their intended effect, but the sparse binary subclass networks were the most accurate network architecture I was able to produce given my current skill with neural nets. In any case, these networks are each binarily classifying the plant or some aspect of it and outputting a single float between 0 and 1.

## Concatenation Layer

The concatenation layer concatenates the 12 outputs of the above Sparse Binary Subclass Network layer and runs them through a final dense neural network with softmax activation for the final guess at the image class.

## Optimizer

This neural network uses the Adadelta optimization scheme.

## Loss Function

The loss function for the network is categorical crossentropy.

# Methodology

I wrote a program which contains a series of Boolean values and integers which allow a user to select combinations of pre-processing types by hand. The program then runs through each pre-processing technique and then trains the network on the resultant imagery. Fitting occurs with 10 epochs. Afterward, the test set of data is predicted, the maximum prediction is selected, and this outcome is compared to the ground truth. A percentage accuracy is output into the program and recorded by hand. To account for outliers in accuracy and get an idea of how the model performs on average, the tests were run twice and the results are averaged together.

# Results

The table below details the results of the experiment, separated into columns indicating whether or not an ImageDataGenerator was used.

|  |  |  |
| --- | --- | --- |
| *average accuracy by pre-processing method* | Without generator | with generator |
| unedited | 29.68% | 33.81% |
| histogram equalization | 8.90% | 12.58% |
| hsv histogram equalization | 12.72% | 14.37% |
| Vecsei’s method | 53.47% | 54.87% |
| Cieluv thresholding | 50.44% | 49.04% |

*Above: (Top) Raw percentages of average accuracy by preprocessing type, separated by whether or not an ImageDataGenerator was used (Bottom) The same data in a histogram for ease of visualization*

# Conclusions

The above graphs demonstrate clearly that the equalization methods on both RGB and HSV files removes necessary data for classification from the image, resulting in drastically less (sometimes apparently random) classification of the images, whereas the image thresholding segmentation techniques on HSV files provided an automated method to preprocess image data for classification in a way that showed significant (~20%, in this case) improvement in classification accuracy. The ImageDataGenerator transformations showed, in most cases, a slight improvement in overall performance with no noticeable impact on performance.

# Problems & Drawbacks

## Selecting an Architecture

I tried a handful of models in the course of this project, including a more simple convolutional network, other pretrained models, and several different dense bodies interpreting the output of whichever convolutional head network I was working with at the time. Some models would converge at around 14% test accuracy without regard for pre-processing type, and even the selected model failed at any point to exceed 60% accuracy on the test set. While my best results were well in excess of random guessing, there are undoubtedly more elegant solutions to this classification problem.

## Selecting Algorithms

I tried easily as many optimizers as architectures. This testing took some time and was prohibitive. Also, the dense networks each needed an activation function which took some experimentation to select.

## Technical Difficulties

Getting Keras set up with tensorflow-gpu appropriately turned out to be a tricky issue which resulted in some frustrating delays to project completion. The program would crash if the wrong libraries were loaded in the wrong place, and if the GPU failed to activate, training would extend to several hours or the program would again fail out.

## The Selected Architecture

The selected architecture has a very uneven error surface due to the training of twelve independent subnetworks simultaneously and therefore it can be difficult to tell if it has converged or not. Additionally, the model seems to be very sensitive to randomized starting weights and may not always converge to a small accuracy range on every run.

# Future Works

This experiment has opened several lines of inquiry which I hope to follow regarding image classification and pre-processing.

## A more complete study of segmented pre-processing

* + This experiment turned up promising results with seedlings in automated segmentation. However, the drawback of these segmentation algorithms is that they assess a pixel’s vector similarity to a particular color or set of colors in HSV format. Automated image segmentation in this way may not be useful for datasets in which the classes of objects are of different colors or a single object class has multiple possible colors or combinations of colors. (e.g. cars)

## A more complete study of image editing techniques of different kinds.

* + This experiment covered only histogram equalization techniques outside of the automated image segmentation. While it is clear that these techniques are lossy, it bears further investigation whether or not other techniques that modify the color balance of an image might help to improve accuracy.

## A better network

* + The neural network that I have used is admittedly inaccurate compared to all possible network architectures. I would like to revisit this subject with a better understanding of image classification architectures and a more accurate and/or representative architecture to work with.

# Works Cited

1. Anish, J. (2017, November 05). The Idea of preparing your own Dataset for convolutional neural network. Retrieved from https://becominghuman.ai/the-idea-of-preparing-your-own-dataset-for-convolutinal-neural-network-49ec097b8313
2. Chollet, F. (2017, April 04). Xception: Deep Learning with Depthwise Separable Convolutions. Retrieved from https://arxiv.org/abs/1610.02357
3. Chu Te, I. (n.d.). Background Removal (CIELUV Color Thresholding). Retrieved from https://www.kaggle.com/ianchute/background-removal-cieluv-color-thresholding/notebook
4. Sebastian Ruder. (2018, April 16). An overview of gradient descent optimization algorithms. Retrieved from [http://ruder.io/optimizing-gradient-descent/index.html#visualizationofalgorithms](http://ruder.io/optimizing-gradient-descent/index.html)
5. Vecsei, G. (n.d.). Plant Seedlings Fun with Computer Vision. Retrieved from <https://www.kaggle.com/gaborvecsei/plant-seedlings-fun-with-computer-vision/notebook>
6. D., M., Zeiler. (2012, December 22). ADADELTA: An Adaptive Learning Rate Method. Retrieved from https://arxiv.org/abs/1212.5701