Data Science Capstone
Final Report
Jackson Crum

Identifying Plant Species in Images with Deep Learning

Problem Statement

The United States federal government has a major legal responsibility to protect endangered species, threatened species, and critical habitats. Once a species becomes listed as "threatened" or "endangered", it receives special protections by the federal government under the Endangered Species Act. The goal of the Endangered Species Act is to make species' populations healthy and vital so they can be taken off the endangered/threatened species list.[1] Ideally, new technologies for monitoring these species would be incorporated to make the conservation efforts as efficient as possible.

Though some technologies such as motion sensor camera traps and drone/satellite imagery have been incorporated into the efforts, extracting information from these pictures remains a tedious manual task conducted solely by highly-trained and specialized botanists.[2] Large amounts of available images and data are often ignored or overlooked due to lack of manpower and resources. In response to this problem, this research focused primarily on saving conservation resources by developing an automated or semi-automated system for identifying species in images and providing conservationists with a simple application for the overall identification and sorting process.

Background

The Endangered Species Act of 1973 requires federal agencies to ensure that any action they authorize does not threaten any listed endangered species or result in the damage or destruction of critical habitats of those species. Section 2(a)(2) states that it is “the policy of Congress that all Federal departments and agencies shall seek to conserve endangered species and threatened species and shall utilize their authorities in furtherance of the purposes of this Act.”[3] The

[1] "Endangered Species | National Wildlife Federation." The National Wildlife Federation. Accessed September 20, 2018. https://www.nwf.org/Educational-Resources/Wildlife-Guide/Understanding-Conservation/Endangered-Species.
[2] Norouzzadeh, Mohammad Sadegh, et al. "Automatically Identifying, Counting, and Describing Wild Animals in Camera-trap Images with Deep Learning." *Proceedings of the National Academy of Sciences* 115, no. 25 (November 17, 2017).
[3] U.S. Congress. Senate. *Endangered Species Act of 1973*. HR 37. 93rd Cong., 1st sess. Introduced in Senate June 12, 1973. https://www.fws.gov/international/pdf/esa.pdf.

Department of Defense (DoD) has the highest density of endangered species listed as threatened or endangered of any other federal agency, with 340 out of 420 large military installations requiring active conservation management plans to protect 492 listed species (274 animals, 218 plants). Expenditures on these conservation plans total more than $800 million annually.[4]

Related Research

Species detection and recognition are still difficult, time-consuming, and expensive challenges and researchers have yet to produce a method that provides an efficient solution for all situations. Research conducted by faculty of the Electrical Engineering Department at University of Zilina compared a CNN model to well-known image recognition methods, including PCA, LDA, and SVM models. It was concluded that the CNN outperformed all conventional models and produced a significant increase in accuracy.[5]

A joint university study tested various state-of-the-art neural network models, including AlexNet, GoogLeNet, VGG, and ResNet models, on Serengeti wildlife images. The study showed that deep neural networks performed well on the wildlife dataset, although performance was worse for rarer animals, and saved 99.3% on manual labor while maintaining human-level identification accuracy. While neural networks have proven successful in the task of image recognition, they rely on large datasets and perform worse on small training datasets.[6]

Trends and future directions of automated plant species identification were reviewed by biogeochemists and software engineers from the Max Planck Institute for Biogeochemistry and Technische Universität Ilmenau. CNN models with 6, 17, and 26 layers were used to classify the Flavia dataset and produced accuracies of 94.69%, 97.9%, and 99.65%, respectively. However, the Flavia dataset is a constrained set of leaf images taken against a white background and

---

[4] United States. Department of Defense. Natural Resources. By Peter Boice. February 2013. Accessed September 2018. http://www.dodnaturalresources.net/files/te__s_fact_sheet_2-21-13.pdf.

[5] Trnovszky, Tibor, Patrik Kamencay, Richard Orjesek, Miroslav Benco, and Peter Sykora. "Animal Recognition System Based on Convolutional Neural Network." *Advances in Electrical and Electronic Engineering* 15, no. 3 (September 2017). Accessed September 2018. doi:10.15598/aeee.v15i3.2202.

[6] Norouzzadeh, et al. "Automatically Identifying, Counting, and Describing Wild Animals in Camera-trap Images with Deep Learning."

without any stem, meaning natural images may result in worse accuracy. Despite intensive research on automated plant species identification, very few studies have resulted in approaches that can be used by the general public. The most popular current applications for planet identification still suffer from inabilities to deal with cluttered backgrounds (LeafSnap) or low top-5 accuracy (Pl@ntNet).[7]

Frameworks

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, including both classic and state-of-the-art computer vision algorithms, and supports a number of image formats. OpenCV is written natively in C++ but also has Python, Java and MATLAB interfaces.

A number of OpenCV functions and algorithms were used for this project. OpenCV's *imread()* function was used for loading image files as numpy arrays in the BGR colorspace (OpenCV's default colorspace). Other functions and algorithms were used to augment the dataset through rotation and flipping and preprocess the data, including normalization, smoothing, converting colorspaces, developing color masks.[8]

Keras

Keras is high-level, open source neural network library written in Python that is capable of running on top of TensorFlow, CNTK, or Theano. Keras provides intuitive objects and functions that make it simple to develop deep learning models. Keras contains numerous implementations of commonly used neural network building blocks such as convolution and max pooling layers, activation functions, optimizers, and a number of tools to make working with image easy.[9]

---

[7] Wäldchen, Jana, Michael Rzanny, Marco Seeland, and Patrick Mäder. "Automated Plant Species Identification—Trends and Future Directions." *PLOS Computational Biology*14, no. 4 (April 05, 2018). doi:10.1371/journal.pcbi.1005993.

[8] "OpenCV Library." About - OpenCV Library. Accessed December 03, 2018. https://opencv.org/.

[9] "Keras: The Python Deep Learning Library." Keras Documentation. Accessed December 03, 2018. https://keras.io/.

PyTorch

PyTorch is an open-source deep learning library that is a Python wrapper on Torch, a scientific framework based on Lua. PyTorch utilizes tensors and tensor computations similar to Numpy with easy GPU usage that supports by CUDA. Numpy arrays can be cast to tensors and tensors cast to numpy arrays with a small number of commands. PyTorch provides simple function calls the construct, train, optimize, and backpropagate convolutional neural networks.[10]

Flask

Flask is a Python micro web framework that supports extensions that can add application features including object-relational mapping, form validation, file uploading, and template rendering. Flask provides a simple routing system between the backend and frontend of the application. On the most basic level, Flask applications consist of routes, the paths that send get and post requests to and from the server, forms for uploading data and user inputs, and templates, which control and display data on the user interface. Template tagging allows for backend objects to be passed and rendered to frontend user interface.[11]

[10] "PyTorch." PyTorch. Accessed December 03, 2018. https://pytorch.org/.

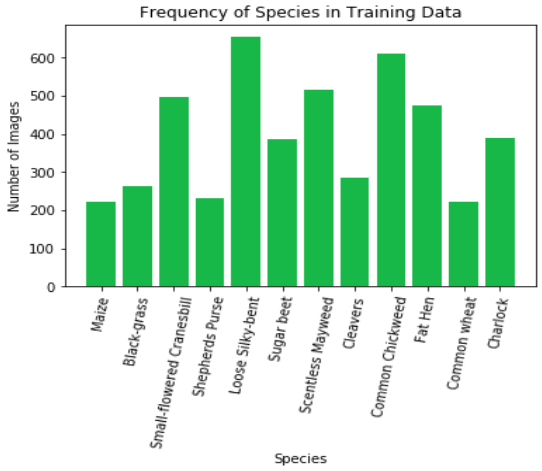
[11] "Welcome." Welcome | Flask (A Python Microframework). Accessed December 03, 2018. http://flask.pocoo.org/.

Data

Original Dataset

The dataset come from the Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, and contained close-up images of approximately 960 unique plants belonging to 12 species at several growth stages. The data consisted of 5544 total color images of the 12 species pre-split into training and testing datasets (4750 labeled training and unlabeled 794 testing images). Only the training dataset was used as the problem is supervised learning. The plant species include Black-grass, Charlock, Cleavers, Common Chickweed, Common Wheat, Fat Hen, Loose Silky-bent, Maize, Scentless Mayweed, Shepherds Purse, Small-flowered Cranesbill, and Sugar Beet. Species classes were non-uniform and ranged from 221 images to 654 images per species. All images had equal height and width dimensions. Image dimensions ranged from 51 pixels to 3991 pixels, with the majority of images having dimensions between 200 and 1000 pixels. Images were stored in a single train folder with a subfolder for each species. The image file name was the image ID and did not have any indication of species. All images were in PNG format.

Original Species Distribution



Data Augmentation

As the image dataset was small and non-uniform in class distribution, the dataset was augmented to provide more images for training. All images were initially rotated 90˚, 180˚, and 270˚ and vertically flipped to increase the dataset size 8-fold. To adjust for class imbalance, all images from classes that were highly underrepresented (Black-grass, Cleavers, Common Wheat, Maize, and Shepherds Purse) were horizontally flipped. For classes that were moderately underrepresented (Sugar Beet and Charlock), half of the images were horizontally flipped.

Finally, labeled training, validation, and test datasets were created from this augmented dataset. Within each new dataset folder, a subfolder for each species was created. The images were shuffled and the first 15% of the images were resorted into their respective species subfolder in the validation dataset. The images were shuffled again and the first 25% were resorted into their respective species subfolder in the testing dataset. The final dataset consisted of 32,439 training images, 10,813 testing images, and 7,632 validation images. The original and augmented class distributions can be seen below.

Species	Original Count
Black-grass	263
Charlock	390
Cleavers	287
Common Chickweed	611
Common wheat	221
Fat Hen	475
Loose Silky-bent	654
Maize	221
Scentless Mayweed	516
Shepherds Purse	231
Small-flowered Cranesbill	496
Sugar beet	385
Total	4750

Species	Final Count
Black-grass	4208
Charlock	4680
Cleavers	4592
Common Chickweed	4888
Common wheat	3536
Fat Hen	3800
Loose Silky-bent	5232
Maize	3536
Scentless Mayweed	4128
Shepherds Purse	3696
Small-flowered Cranesbill	3968
Sugar beet	4620
Total	50884

Original and Augmented Dataset Class Distributions

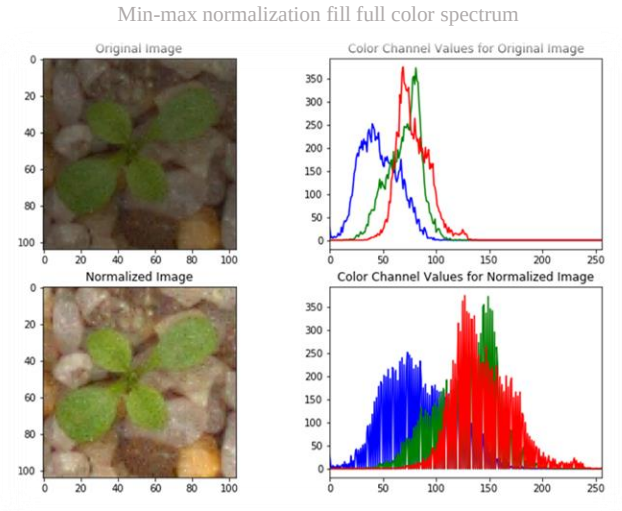
Preprocessing

Building an effective neural network model requires careful consideration of the network architecture as well as the input data format and preprocessing techniques relevant to the input images. The image pixels in this dataset each have 3 values, one for each color channel corresponding to red, green, and blue (RGB), and pixel values ranging from 0 to 255, with 0 being pure black and 255 being pure white. The OpenCV Python library has a number of functions that allow for easy image preprocessing.[12]

The first preprocessing technique considered in the image preprocessing stage was data normalization. Data normalization is an important step which ensures that each pixel has a similar data distribution and makes convergence faster while training the network. Normalization is done

[12] "OpenCV Library." About - OpenCV Library. Accessed December 03, 2018. https://opencv.org/.

by subtracting the minimum pixel value from each pixel, dividing the result by the difference between the maximum and minimum pixel value, and multiplying each value by 255. The resulting data is then projected onto a range of [0, 255], with minimum domain value mapped to 0 and maximum domain value mapped to 255. This acts to stretch the pixel values to span the entire color range, which increases contrast and helps to eliminate the effect of lighting levels, which vary throughout the image dataset.

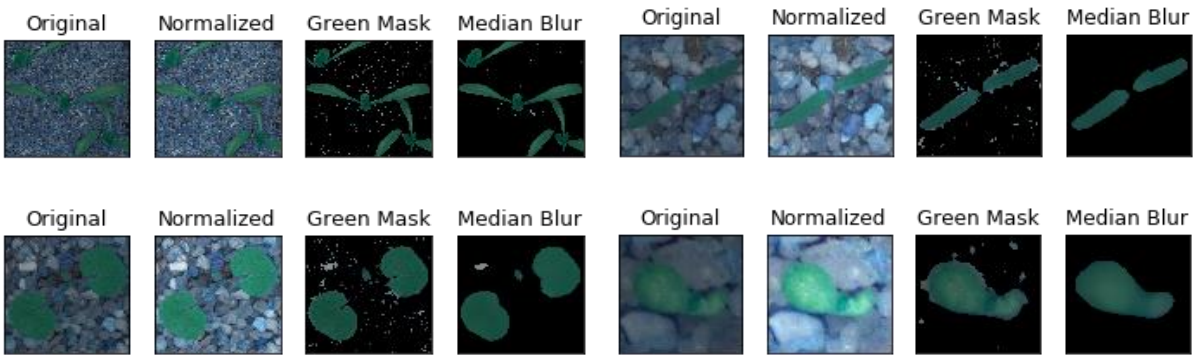


Next, masking was tested to separate the green plants from the non-green background. The plant seedlings are photographed on a pebbly background, a feature heavy background that can cause confusion in image identification. The images are converted from BGR color space to HSV (Hue, Saturation, Value) color space. HSV separates the image intensity from the color information. In this color space, shadow will primarily influence the value and saturation components, while the hue is primarily influenced by colors. Converting the HSV color space allows for the mapping the mask on solely the hue component while spanning the full range of saturation and hue. In this manner, using OpenCV's inRange function, all pixels inside the hue range of the green of the plant seedlings are identified and mapped onto an all-black image of the same dimensions to produce an unchanged seedling on a black background. Using the HSV color space means that shadow and lighting do not affect the masking process.

Finally, median blurring was tested to smooth the background and remove noise from the image. The median filter runs through each pixel individually and replaces each pixel with the median of n-neighboring entries, n being the dimensions of the filter chosen. Median blurring is one of a number of smoothing operations employed in image preprocessing and is particularly good at removing speckle noise while preserving edges. As leaf shape will most likely be a primary factor in image class labeling, it is vital to preserve the edges of the leaves. The pebbly

background of the images also produces a significant amount of speckle noise that median blurring effectively eliminates.[13]

The images preprocessing techniques were employed in their respective order and produced final images as seen below.



Plant Seedling Images at Each Stage of Preprocessing

Convolutional Neural Network Architecture

Benefits of CNN

The main benefits of a convolutional neural network over other neural network designs, such as multilayer perceptrons, is the ability of the CNN to work with matrix-formatted inputs and to isolate and analysis local features with relatively few parameters. Where MLP would require each image to be flattened to a 1-dimensionl input array with a length of tens or even hundreds of thousands (the square of the image dimension), the CNN accepts the image in its original matrix form. After flattening, the MLP requires each node to be connected to every other node in the next layer all they way down the network. For a 100 x 100 image, this means that a network with an input layer with 100^2 neurons, one hidden layer of the same size as the input, and 12 output nodes would require over 1.2 billion weight parameters. The CNN drastically shrinks the number

[13] "OpenCV Library." About - OpenCV Library. Accessed December 03, 2018. https://opencv.org/.

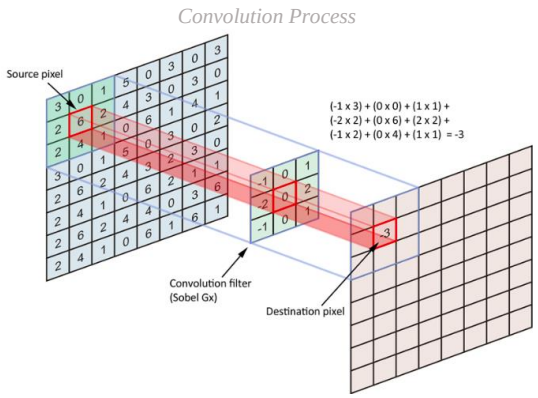
of parameters by running small sets of weight values over the input values in the form of kernels and stride.

Input Layer

The input layer is a four-dimensional matrix. In the case of image analysis, the dimensions represent image height, image width, number of color channels, and batch size. Grayscale images have a single color channel while color images have three color channels, one each for red, blue, and green. Batch size is used to updating weight and bias parameters using the average output of n-number of inputs. Using all training data (1 batch) to update parameters is highly computational expensive. Updating after each input creates noise if the sample is not a good representation of the whole data. Mini-batches are used to compromise between efficiency and noise.

Convolutional Layers

The convolution layer consists of a set of learnable kernels (also called filters). The kernels consist of randomly initialized weight values in a 3-dimensional matrix (height, width, number of color channels) and are typically of size 3x3, 5x5, or 7x7, but can be scaled up to detect more complex features. In the feedforward process, the kernels slide (convolve) over each element of the input matrix and compute the sum of the element-wise multiplication between the kernel and the local image pixel values. This results in a single value that maps to the new feature map. Each feature map is the result of the computed values of a single kernel convolving over the whole of the input matrix. A choice of *n* kernel creates *n* feature maps (new images), increasing the depth of the next layer's input.



Aside from the number of kernels, convolutional layers have a number of adjustable hyperparameters that control output size. Stride is the number of pixels the kernel jumps with each calculation. The size of stride results in an output *1/stride* the size of the original input, with larger strides resulting in smaller outputs. Padding is a parameter that is used to maintain input size. The process of convolution results in a single value

at the center of the kernel, which means that the edges of the input are cut off and the output is *n-1* (n representing the height and width of the kernel) dimensions smaller than the input. For example, a 28x28 input with a 3x3 kernel and no padding will result in a 26x26 output as it omits the outermost layer of pixel values. To prevent this, layers of zeros or other values are added around outside of the input so that the first position of the kernel is centered on the upper-left pixel value. The number of padding layers is calculated by *(n-1)/2* with n representing kernel size.

Convolution layer are used to detect features of an image. In this process, the patterns of the weight values of the kernels are compared to the patterns of the input matrix and the convolving of similar patterns results in feature extraction. Lower level features, such as vertical, horizontal, and diagonal lines, are detected in the earlier convolution layers whereas higher level features, such as plant leaf shape, are detected in the deeper convolution layers of the network.

ReLU Activation

Following the convolution layer, a ReLU (Rectified Linear Units) activation layer is applied. The ReLU layer applies the function $f(x) = max(0, x)$ to all of the values in the input, where x represents the input value. Essentially, the function sets all negative inputs to 0 and sets the output as the input for positive inputs. This function is used to calculate gradients for parameter updating and. ReLU results in a network that is computationally efficient and fast training due to the simplicity of the function derivation and the avoidance of vanishing gradient (gradient values decreasing exponentially) by maintaining a constant derivative.

Batch Normalization

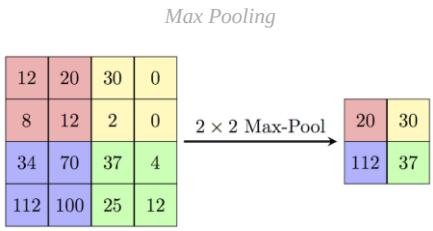
Batch normalization normalizes the output of the ReLU activation layer by subtracting the batch mean and dividing by the batch standard deviation. Batch normalization lessens the likelihood of the gradients decaying (moving to 0) or exploding (moving toward infinity) as well as the chances of the model becoming dependent of a small number of features with high weight values due to exploding gradients. Higher learning rates can also be used because batch normalization ensures stabilization in activation output.

Max Pooling

A pooling layer is applied after the ReLU activation function to downsample (shrink) the input to reduce the number of trainable parameters. Maximum pooling applies a filter of size *n x n* and stride *n* and calculates the maximum of that filter as the output. The output of this layer is *n*-times smaller than the output. For example, as seen in Figure, a maximum pooling of 2x2 reduces the output to half the size of the input and the number of parameters by 75%. Since typical images contain tens-to-hundreds of thousands of input values and associated trainable parameters, reducing these sizes is vital to training viability.

Flattening Layer

For the purposes of classification, the final layer of the CNN is a 1-dimensional array of the class probabilities. This requires that the output of the convolutional layers also be converted to a 1-dimensional array. Following the final pooling layer, the output is flattened into a 1d array of a length equal to the number of feature maps multiplied by the output size of the final layer. For example, a final layer with a 6 x 6 output and 32 feature maps results in a 1 x 1152 array.



Fully Connected Layers

The fully connected layers act as a multilayer perceptron in which every neuron in the input layer is connected with weights and bias to every neuron in the output layer. These layers serve as the classification layers of the network by extracting important features related to each class. The final fully connected layer in the model is a *1 x n* array where *n* is the number of classes.

Softmax Activation

The softmax function normalizes the final output vector to values between 0 and 1 and divides each output value by the sum of the vector so that the sum of the vector equals 1. The result is a categorical probability distribution for the input class, with the index of the maximum value being the label of the most probable class.

Dropout

During each feedforward pass in the training stage, all nodes in a certain layer are nodes are either dropped with probability *1-p* or kept with probability *p*. Fully connected layer consist of the majority of parameters in the network and this cause neurons to develop co-dependency with each other during training which leads to the weakening of individual neuron power and leads to over-fitting of training data. Randomly turning off neurons allows for the neurons of true importance to be determined and weighted appropriately.

Loss

Loss is a measure the quality of the parameters based how well the network classified the input, the difference between the output and the target labels. The entire purpose of training is to find parameters that minimize loss in the model. As this is a classification problem, cross entropy is utilized to find the minimum of the loss function. Cross entropy, or log loss, compares the probability of a correct prediction to the actual prediction and punishes both errors, meaning highly confident and wrong answers are scored worse than less confident and wrong answers.[14]

Optimization

The purpose of optimization is to move towards weight and bias parameters that minimizes the loss function. Optimization calculates gradient descent and discovers the direction of steepest descent towards the minimum of the loss function along which to update the weight vector. The Adam optimizer was used for all networks.

Experimentation

Loading Images

Keras requires the images to a be in a specific folder layout for model training. There are three parent folders, one each for training, testing, and validation datasets, with each dataset folder

[14] Loss Functions¶." Gradient Descent - ML Cheatsheet Documentation. Accessed December 03, 2018. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

having a child folder for each image class that contains the images of only that class. Images were read and loaded directly from the folder path using the Keras ImageDataGenerator class and it's "flow_from_directory" method. The ImageDataGenerator creates batches of tensor image data with real-time custom preprocessing and augmentation and loops over the data indefinitely. The "flow_from_directory" method takes the path to a directory, and generates batches of normalized data. The method allows for the specification of a number of parameters, including image input size, batch size, and class labels, which can be inferred from child folder labels. For this project, images were loaded with an input size of (100,100,3) and in minibatches of 100 images as various trials showed these values to provide reasonable training time without a significant drop in image quality.

Loading datasets into PyTorch requires a different process from Keras to load images for analysis. PyTorch requires the construction of custom image data loader wrapped around Torch's Dataset class. Within this custom class, the data was loaded and preprocessed and then run through PyTorch's DataLoader class. This created a PyTorch dataset of images and labels that can be iterated over for minibatch training.

Keras Model Construction

The model is initiated as an empty Keras Sequential model constructor. The Sequential constructor allows for the addition of linear, ordered stacks of layers through the simple passage of a list of layer instances.

The convolution section of the model was built with a series of convolution-pooling blocks. The convolution-pooling block consisted of a convolution layer, a ReLU activation layer, a batch normalization layer, and a max pooling layer. In Keras, the first convolution layer of the model requires an initial input shape tuple of (height, width, color channels), which was (100,100,3) for this project. Each convolution layer has arguments including number of feature maps, kernel size, stride length, and padding setting, which can be set to automatically preserve input size depending on kernel size. The max pooling layer requires a set pooling kernel, which is the standard *2x2* for the model.

The classification layers of the model are constructed with an initial flattening layer and a two dense (fully connected) layers. The first dense layer is built with 128 neurons as compromise

between number of parameters and training time. The final dense layer has 12 neurons (one for each plant species) and is activated with a softmax function to produce class probabilities.

Prior to training the model, the learning process is configured using Keras's "compile" method. The compiler takes as arguments one of various existing Keras optimizers, a learning rate, a loss function, and a list of metrics.

Finally, the model is fit on the training and, optionally, the validation batches. The "fit_generator" method takes in as arguments the training and validation batches, the number of epochs, parameter updating steps per epoch (*total images / batch size*), and any callbacks placed in the model, such as early stopping.

PyTorch Model Construction

To create data that can be easily passed to the develop custom data loader, the images were first run through a program that extracts each images file path and, from that, extracts the folder name as the species. The species label is then changed to the corresponding numeric label through a reference dictionary. The full image path and numeric label are placed into a two-dimension Pandas DataFrame and saved as a csv. This csv is then passed to PyTorch data loader with the image paths being read by OpenCV. Images are then resized and run through each step of preprocessing to produce dataset to run through the PyTorch neural network.

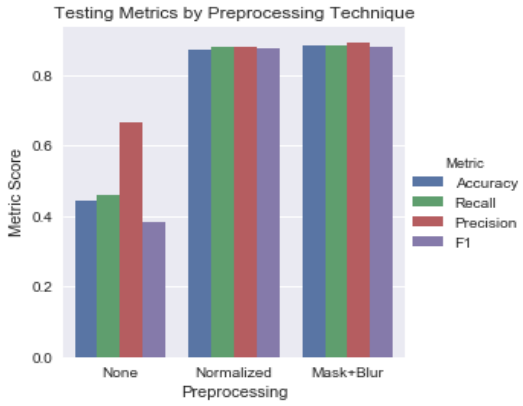
PyTorch requires images to be in a different input shape than Keras. While Keras requires the color channels to be the final dimension, PyTorch requires the color channels to be the second dimension (ex. (3,100,100) for a color image of size *100x100*), while batch size is the first dimension.

Torch provides the torch.nn package creates a set of neural network layers, called modules, that have trainable weight and bias parameters and produce an output from input data. The convolutional layer takes number of color channels, number of feature maps, kernel size, and number of padding layers, which, unlike Keras, has to be manually calculated by the equation *padding=(kernel-1)/2*. Fully connected layers (called Linear layers in PyTorch) take in number of input neurons and output neurons and the input size must equal the output size of the previous layer.

Calculating loss and optimizing parameters is very simple in PyTorch. The loss is calculated automatically by calling a PyTorch default loss function and the image labels and backpropagation is fully calculated by the "loss.backward()" function. PyTorch allows for the easy conversion of numpy arrays to tensors for running on any available GPU's and conversion back to numpy arrays for metric calculations.

Preprocessing

Models were trained at each stage of preprocessing to validate the use of each technique. Each network had four convolution blocks with 32 feature maps, batch normalization, and *2x2* maxpool, with a *7x7-7x7-3x3-3x3* kernel layout. Images were initially trained without any preprocessing. Next, the images were trained after being normalized. Third, the images were trained after being normalized and having a black mask of all pixels with HSV value outside the range (40, 0, 0) to (110, 255, 200) mapped onto the normalized image. These values were chosen through manual testing and deemed the optimal range as this mask isolated the entirety of the plant seedling with minimal background present. Finally, images were trained after having undergone median blur smoothing with a *9x9* kernel size, also deemed optimal through manual testing.



Metrics for Preprocessing Techniques

The network trained on the original images performed poorly, with only precision being above 50%. Normalizing the images produce drastically better results, increasing all metrics to 87-88%. Adding the green mask and median blur brought accuracy up to 88.2% and marginally improved the other metrics as well.

Number of Layers

Initial experimentation trained several models with a varying number of convolutional-pooling blocks with static classification layers and fully preprocessed images. Each convolution layer in the block had of 32 feature maps, 5x5 kernel, and 2x2 maxpool. The two-block model followed

the data-size structure 100-50-25 with a flattening layer with 20,000 neurons and had 2,589,996 total trainable parameters. The three-block model followed the data-size structure 100-50-25-12 with a flattening layer with 4608 neurons and had 629,196 total trainable parameters. The four-block model followed the image-size structure *100-50-25-12-6* with a flattening layer with 1152 neurons and had 196,204 total trainable parameters. The five-block model followed the data-size structure 100-50-25-12-6-3 with a flattening layer of with 288 neurons and had 94,988 total trainable parameters.

In the analysis of training accuracy and loss, the four-block model performed the best, with a training accuracy of 0.903 and training loss of 0.264 after 5 epochs. Training time was reduced with the addition of each layer as the shrinking final image shape drastically reduced total parameter numbers.

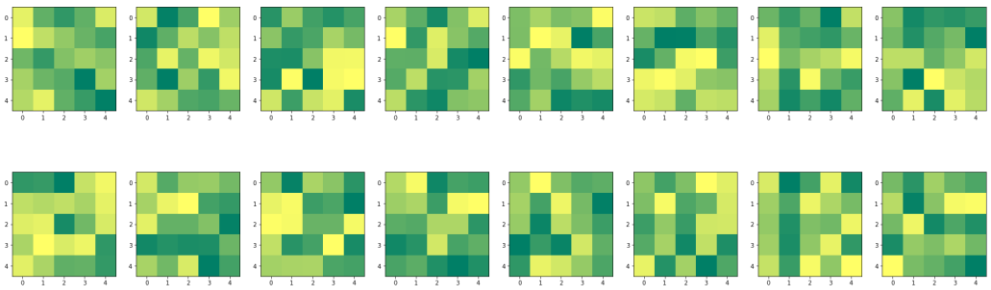
		2 CONV/POOL	3 CONV/POOL	4 CONV/POOL	5 CONV/POOL
TRAINING	Accuracy	0.7701	0.9005	0.9032	0.8632
	Loss	0.6379	0.2788	0.2643	0.3769
VALIDATION	Accuracy	0.8669	0.9331	0.9413	0.8885
	Loss	0.4083	0.1895	0.175	0.2972
	Parameters	2,589,996	629,196	196,204	94,988

Kernel Size

Kernels of various sizes, from *3x3* to *15x15*¸ were tested on the fully preprocessed, four-convolutional block network. Picking the right kernel size is vital as small filters tend to underfit and large filters tend to overfit due to the number of parameters created. As images in a convolutional neural network shrink in size as they move down, kernel sizes were also decreased. Four architectures were tested: *15x15-9x9-7x7-3x3*, *15x15-9x9-7x7-3x3*, *7x7-7x7-3x3-3x3*, *and 5x5-5x5-3x3-3x3*. Each convolution layer in the block had 32 feature maps, batch normalization, and *2x2* maxpool.

		15-9-7-3	10-7-5-3	7-7-3-3	5-5-3-3
TRAINING	Accuracy	0.8795	0.8776	0.871	0.8898
	Loss	0.325	0.3365	0.3542	0.3049
VALIDATION	Accuracy	0.9194	0.8651	0.9014	0.9248
	Loss	0.2221	0.335	0.2708	0.203
	Parameters	313,708	244,364	205,854	196,204

The network with the smallest kernels (*5x5-5x5-3x3-3x3*) performed the best, with an accuracy

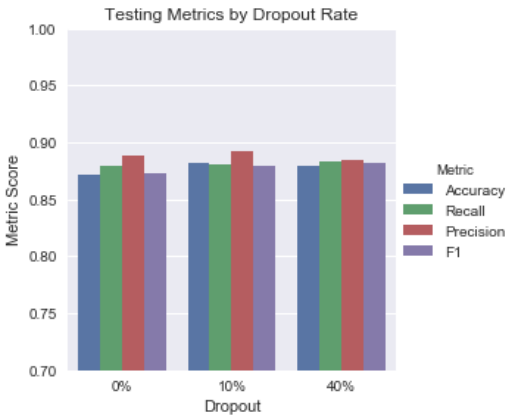


First 16 kernels of first convolution layer of the best model (yellow = higher weight value)

of almost 89%, followed by the network with the largest kernels (*15x15-9x9-7x7-3x3*) at 88% accuracy. The results indicate the kernel size does not seem to have a considerable effect on network outcome. However, training time in significantly affected by kernel size as a result of the increase in total trainable parameter with larger kernels, with the larger kernel network taking nearly double the time to train each epoch than the small kernel network (40.86 minutes and 21.75 minutes, respectively).

Dropout

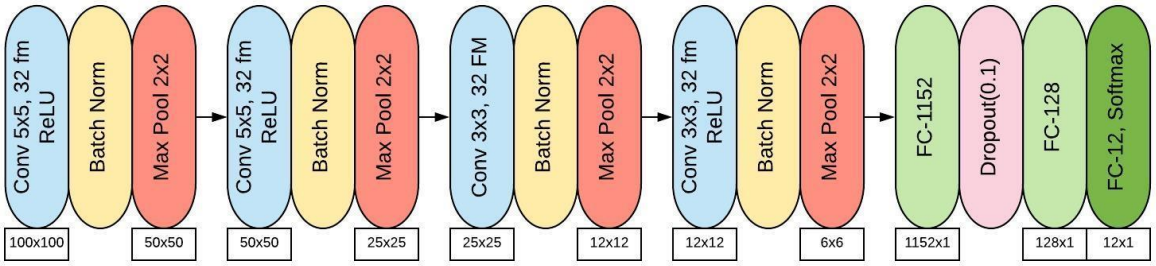
Models were trained with dropout on the first fully connected layer at three dropout rates: 0%, 10%, and 40%. All three networks of different dropout rates had similar metrics, with the 10% dropout network resulting in the best accuracy of 88.2% and precision of 89.2%.



Final Architecture and Metrics

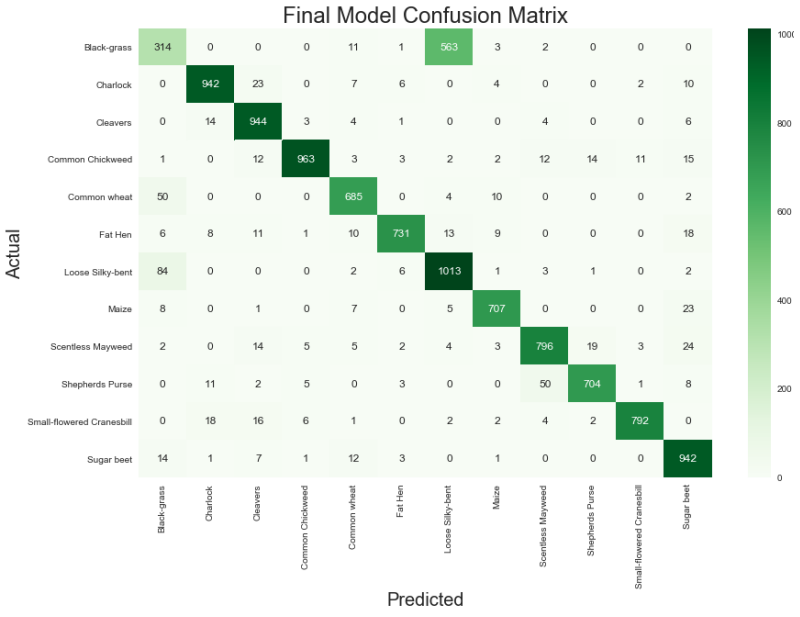
The final model architecture was constructed with the top-performing model attributes. All input images are preprocessed with min-max normalization, green masking, and median blurring. The model was built with four convolutional blocks with the first two blocks having *5x5* sized kernels and the last two blocks having *3x3* sized kernels. ReLU activation, batch normalization, and *2x2* max pooling layer were used after each convolution. After the convolution layers, three

fully connected layers were added were in the structure *1152x1-128x1-12x1*. Dropout of 10% was placed on the first fully connected layer. The output layer used softmax function to produce class probability outputs. Categorical cross entropy was used to calculate loss and Adam was used to optimize the network.



Final Network Architecture with Image Sizes

The final model predicted images with an accuracy of 88.20%, a recall of 88.02%, a precision of 89.23%, and an F1 score of 88.08%. As shown in the confusion matrix, most species are predicted with very high accuracy. The exception is Black-grass and Loose Silky-bent. Black-grass has a recall (*true positive / (true positive + false negative*) of 35.12% (314/894), with the majority of incorrectly predicted Black-grass being labelled as Loose Silky-bent. This means that only 35.12% of Black-grass images in the dataset are correctly identified as Black-grass. All other species had above 90% accuracy, recall, precision, and F1.



Flask Application

The final model was finally incorporated into a simple Flask application for predicting and sorting a folder of new plant seedling images. The application incorporates a Python backend with a JavaScript and HTML frontend. The Bootstrap toolkit was used to produce to user friendly frontend aesthetics and functionality. The application uses functions created with Keras and PyTorch to load in the trained models, run on preprocessed images, return predictions, and sort the submitted directory into species folders. Prediction metrics are displayed once sorting is complete.

The application was clocked at a typical upload, predict, and sort speed of 8.8 images/second. This far out performs any human classifier in terms of speed. However, prediction error is still high. Future developments of the application would include incorporating PyTorch predictions, a more user-friendly interface, metric visualizations, and long-term storing in a No-SQL database.

References

"Keras: The Python Deep Learning Library." Keras Documentation. Accessed December 03, 2018. https://keras.io/.

"Loss Functions¶." Gradient Descent - ML Cheatsheet Documentation. Accessed December 03, 2018. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

Norouzzadeh, Mohammad Sadegh, Anh Nguyen, Margaret Kosmala, Alexandra Swanson, Meredith S. Palmer, Craig Packer, and Jeff Clune. "Automatically Identifying, Counting, and Describing Wild Animals in Camera-trap Images with Deep Learning." *Proceedings of the National Academy of Sciences* 115, no. 25 (November 17, 2017). Accessed September 2018. doi:10.1073/pnas.1719367115.

"OpenCV Library." About - OpenCV Library. Accessed December 03, 2018. https://opencv.org/.

"PyTorch." PyTorch. Accessed December 03, 2018. https://pytorch.org/.

Trnovszky, Tibor, Patrik Kamencay, Richard Orjesek, Miroslav Benco, and Peter Sykora. "Animal Recognition System Based on Convolutional Neural Network." *Advances in Electrical and Electronic Engineering* 15, no. 3 (September 2017). Accessed September 2018. doi:10.15598/aeee.v15i3.2202.

United States. Department of Defense. Natural Resources. By Peter Boice. February 2013. Accessed September 2018. http://www.dodnaturalresources.net/files/te__s_fact_sheet_2-21-13.pdf.

U.S. Congress. Senate. *Endangered Species Act of 1973*. HR 37. 93rd Cong., 1st sess. Introduced in Senate June 12, 1973. https://www.fws.gov/international/pdf/esa.pdf.

Wäldchen, Jana, Michael Rzanny, Marco Seeland, and Patrick Mäder. "Automated Plant Species Identification—Trends and Future Directions." *PLOS Computational Biology*14, no. 4 (April 05, 2018). doi:10.1371/journal.pcbi.1005993.

"Welcome." Welcome | Flask (A Python Microframework). Accessed December 03, 2018. http://flask.pocoo.org/.