PLANT SPECIES IDENTIFICATION USING COMPUTER VISION

DATA SCIENCE CAPSTONE, FALL 2018

JACK CRUM

INTRODUCTION

- Endangered Species Act of 1973: *Section 2(a)(2)*
    - "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."
- Expenditures on endangered species conservation plans total more than $800 million annually and include 218 plant species
- Department of Defense (DoD) has the highest density of endangered species 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).

PROBLEM STATEMENT

- Large amounts of available images and data are ignored or overlooked due to lack of automation, manpower, and resources
- Previously developed CNN models have good accuracy with leaves on white backgrounds (Flavia Dataset), not tested with natural background
- The most popular current applications for planet identification still suffer from inabilities to deal with cluttered backgrounds and have single uploads
- Project focus is to save conservation resources by developing a simple automated system for uploading large folders of images, identifying species in images, and sorting based on prediction

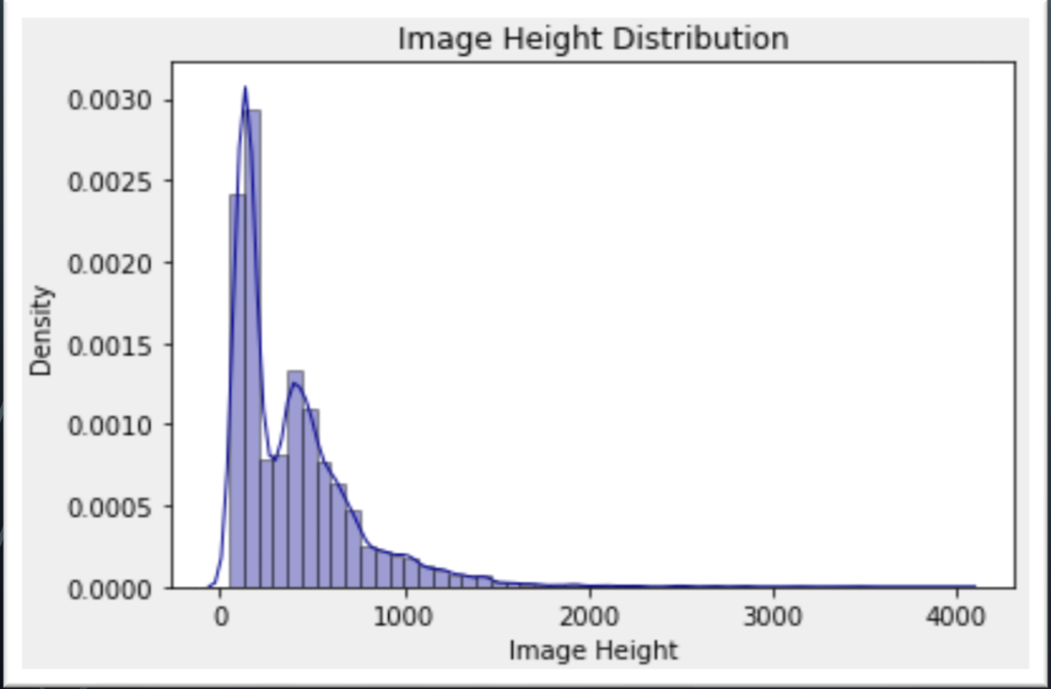
DATASET

- Kaggle Plant Seedlings Classification Dataset
- 12 species in pre-split training/testing datasets
- 4750 training images
- 794 testing images
- Testing set is not labelled
- Uneven class distribution




DATA AUGMENTATION

- All Images were rotated 90˚, 180˚, and 270˚
- Some were horizontally flipped and vertically flipped to create more uniform class distributions
- 50867 total images from original training set: 32429 training,10809 testing, 7629 validation



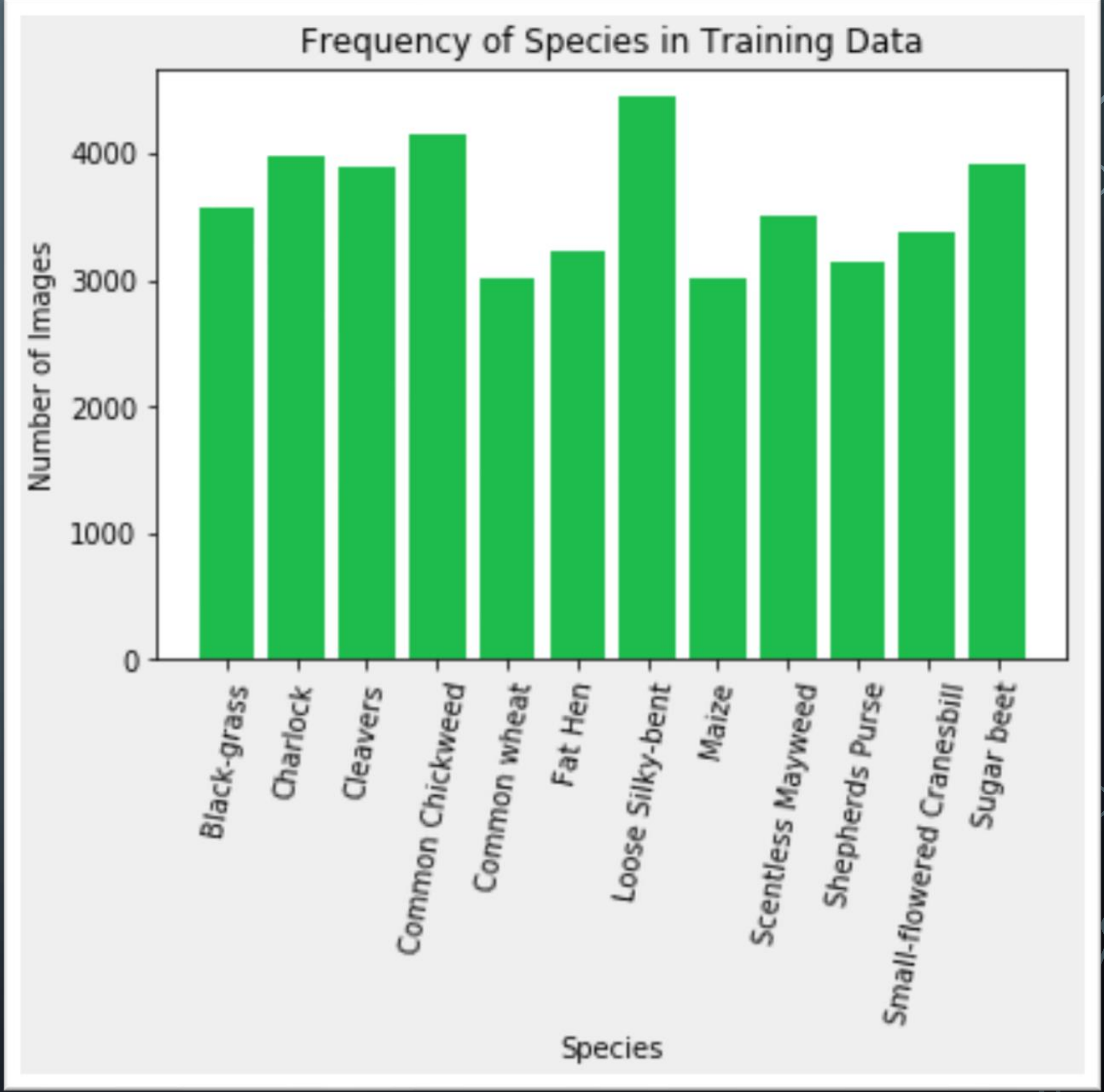


IMAGE PREPROCESSING: NORMALIZATION

- Split into 3 RGB color channels
- Subtracting the minimum pixel value from each pixel, dividing the result by the difference between the max and min, and multiply by 255
- Maps minimum pixel value to 0 and maximum to 255 so the image spans the whole color spectrum
- Increases contrast and negates effects of shadow and lighting

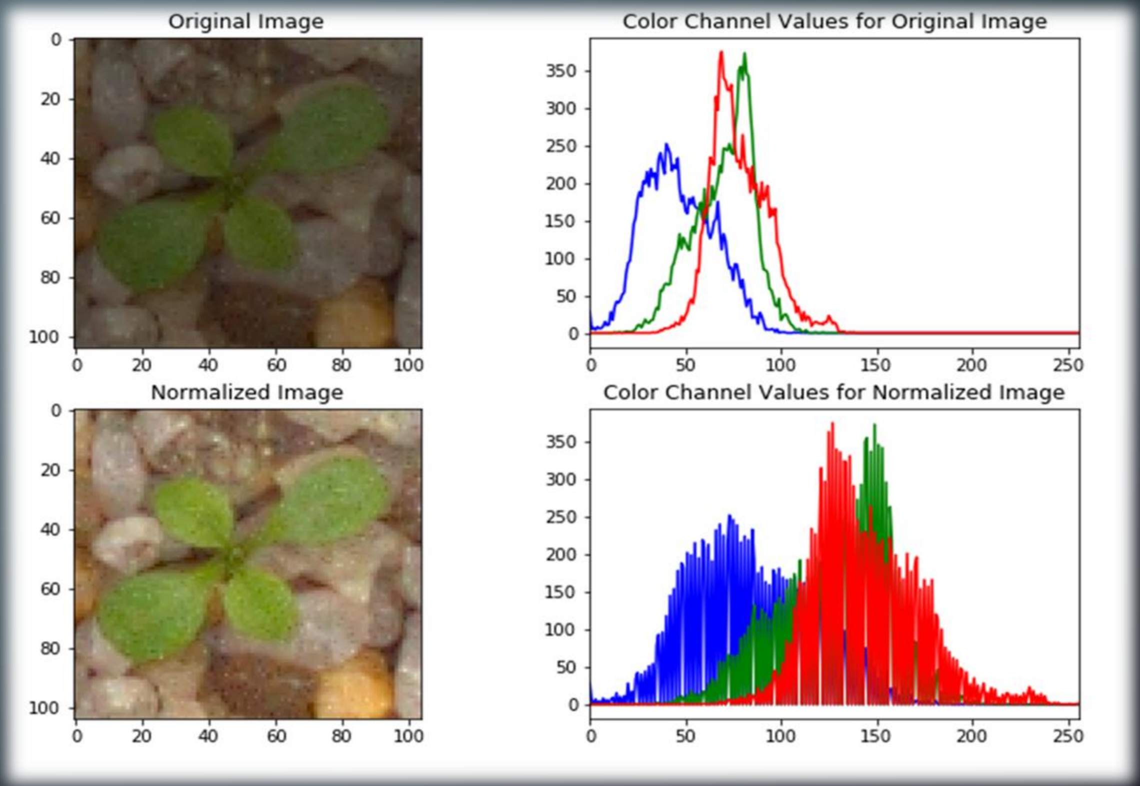


IMAGE PREPROCESSING: MASKING

- Change colorspaces from BGR (Blue, Green, Red) to HSV (Hue, Saturation, Value)
  - In OpenCV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255].
- Threshold the HSV image for a range of green color
  - Chosen range: (40, 0, 0) to (110, 255, 200)

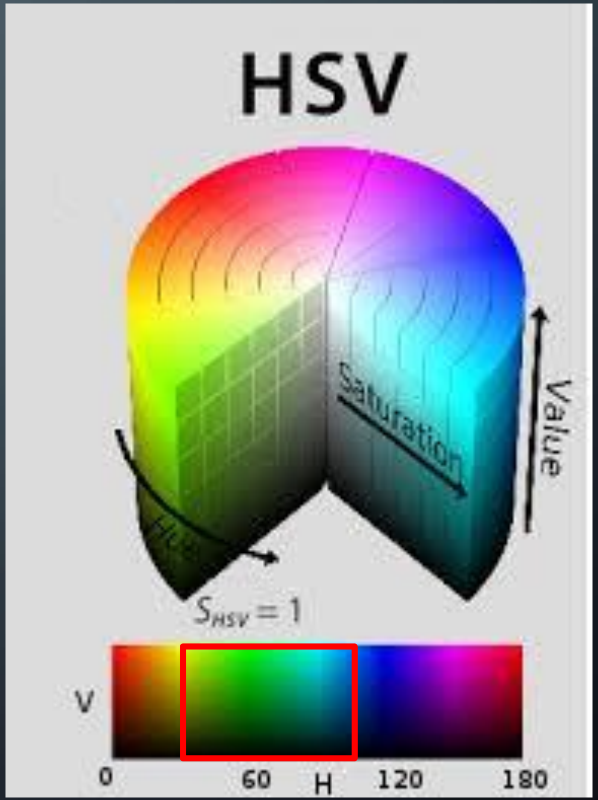
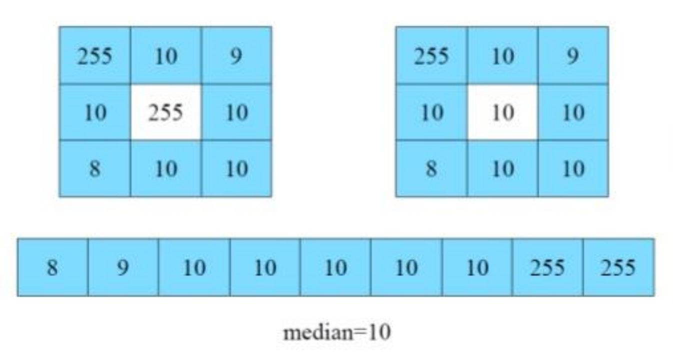


IMAGE PREPROCESSING: MEDIAN BLUR

- Computes median of all pixels under the kernel window and central pixel is replaced with this value
- Highly effective in removing salt-and-pepper noise
- 9x9 kernel chosen for images



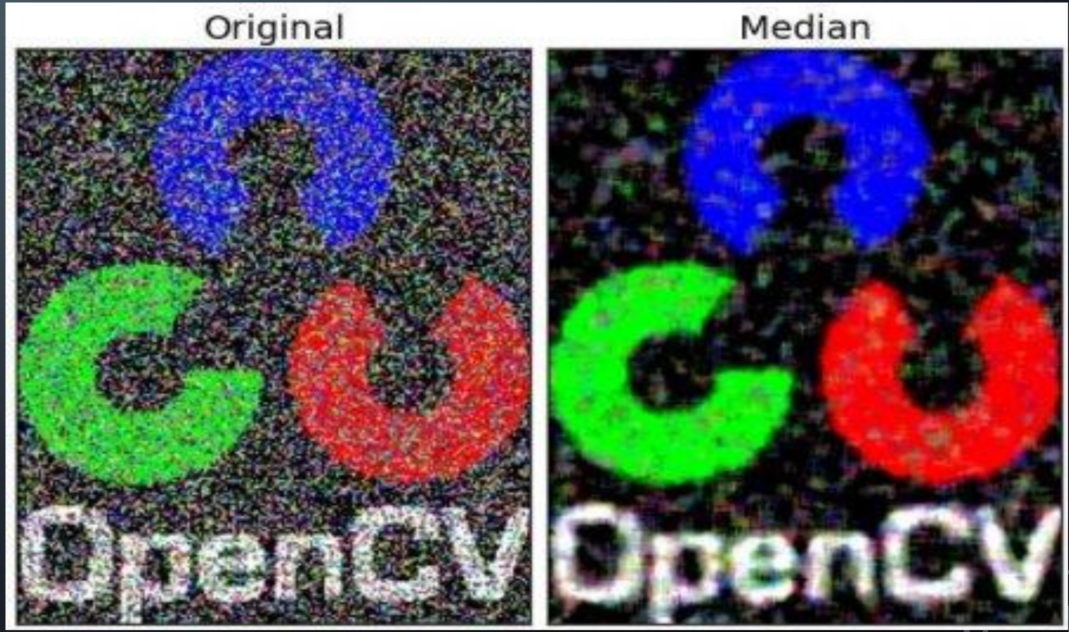
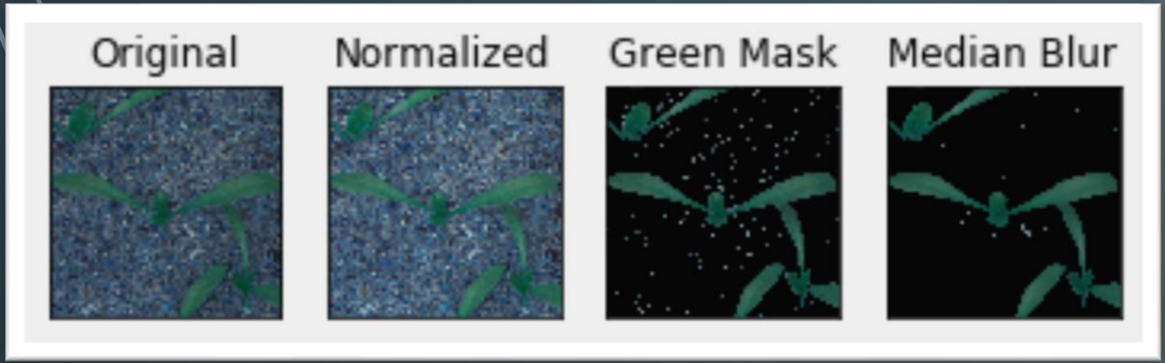
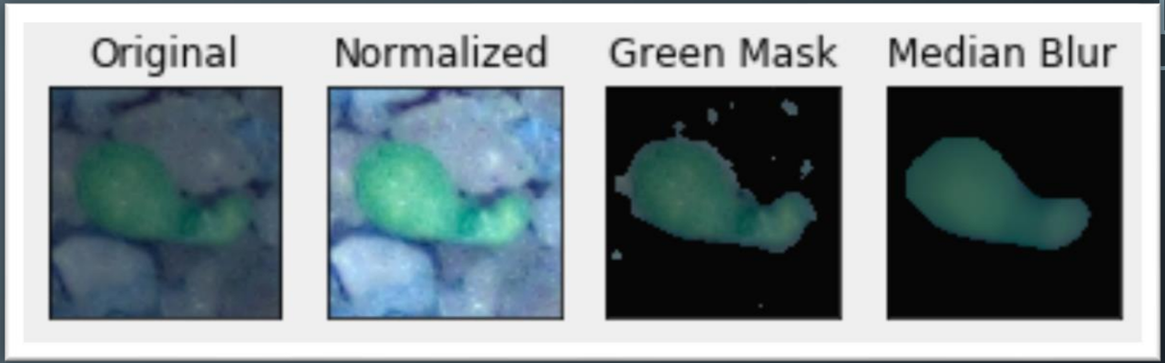
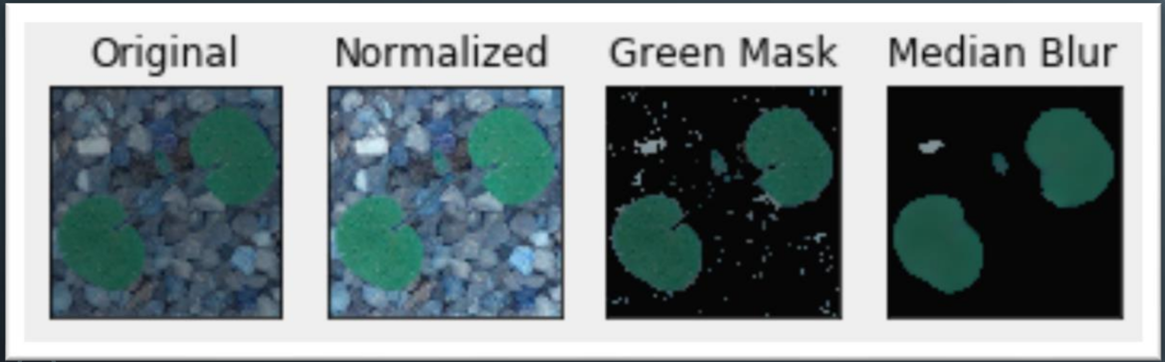
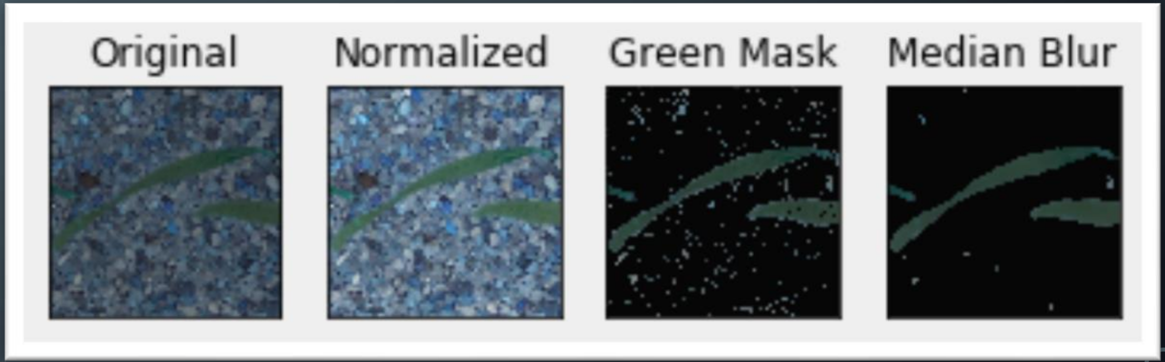


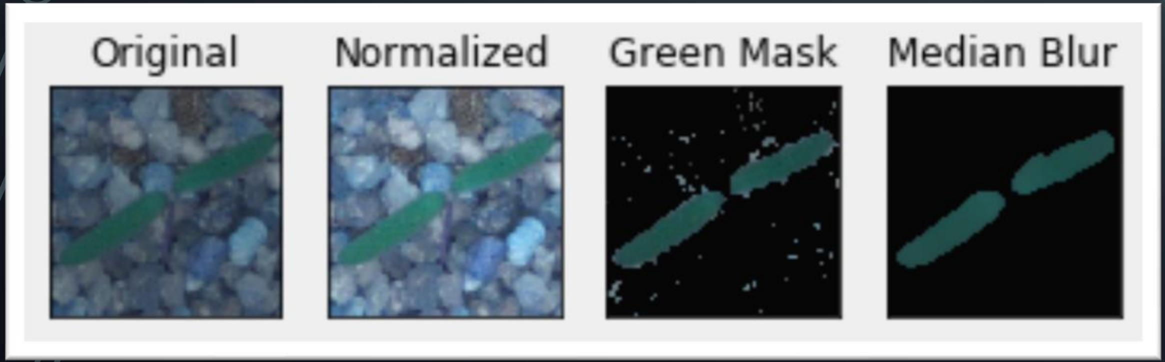
IMAGE PREPROCESSING RESULTS

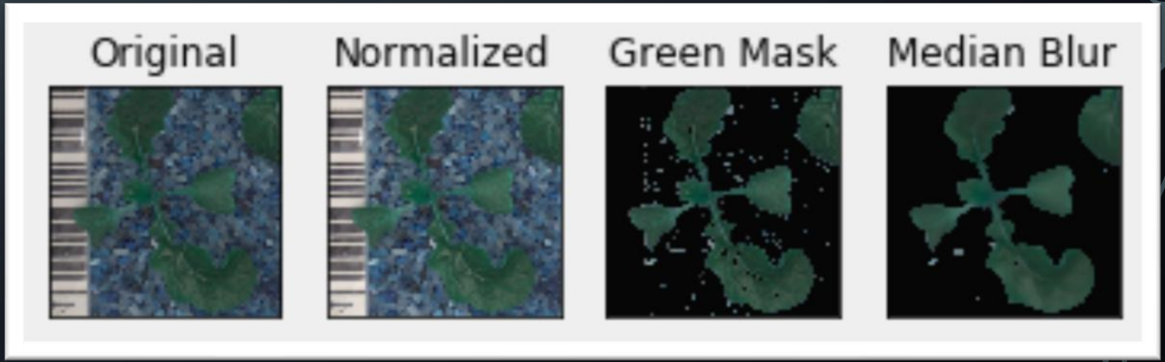








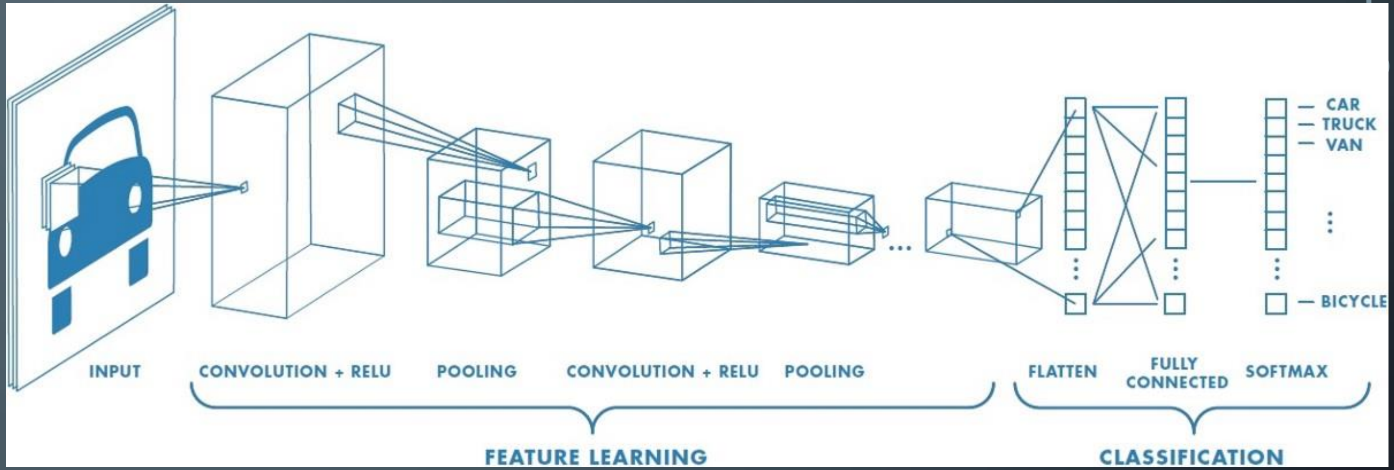


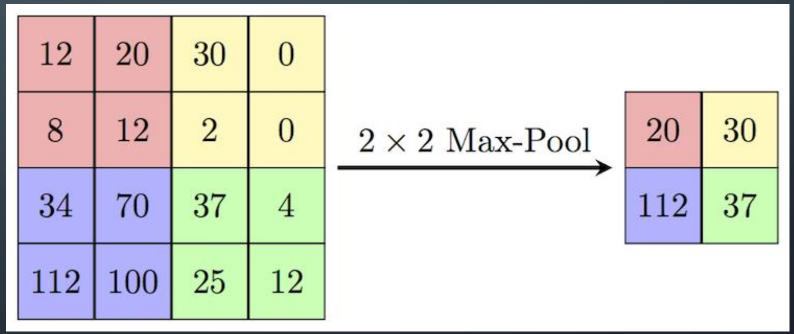


NEURAL NETWORKS

- Convolutional Neural Network
  - Convolutional Layers
    - Stride over image with various number of weight matrices to detect edges and patterns
    - Parameters include: Kernel Size, Stride, Padding
  - Activation
    - ReLU (
  - Max Pooling Layers
    - Shrink images based on maximum

      value in kernel
  - Flatten
    - Flatten into 1d array
  - Fully Connected Layer
    - Fully connected layer
  - Softmax Output
    - Class probability

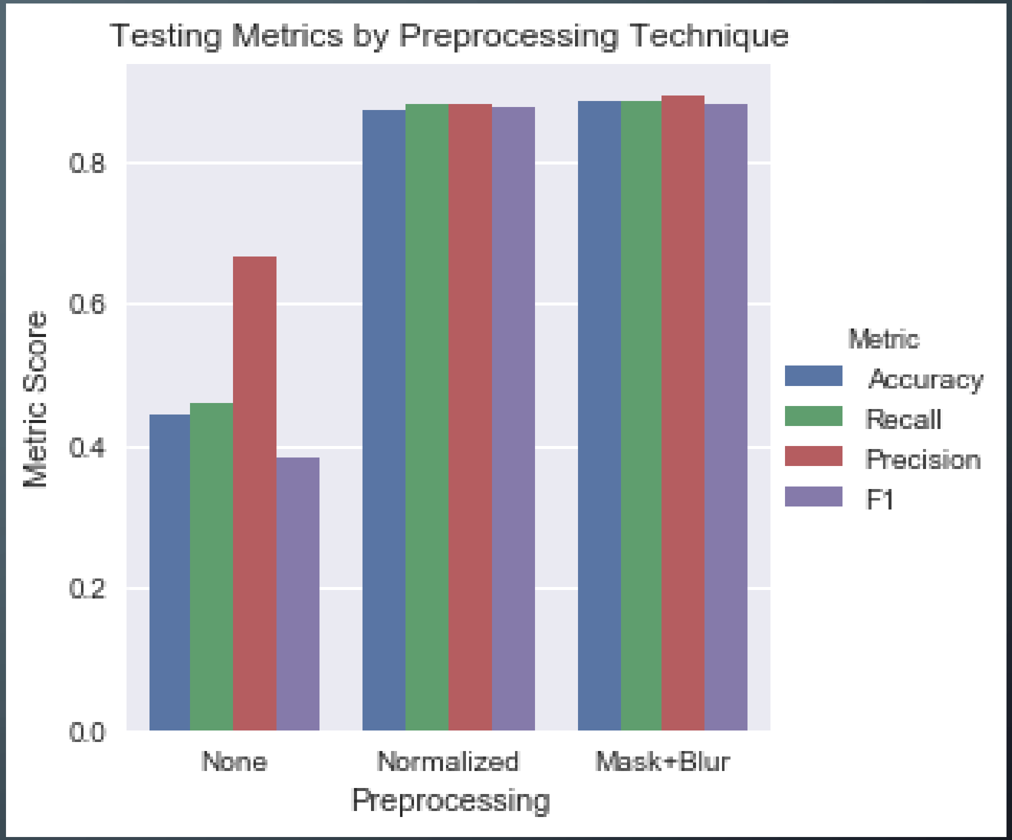


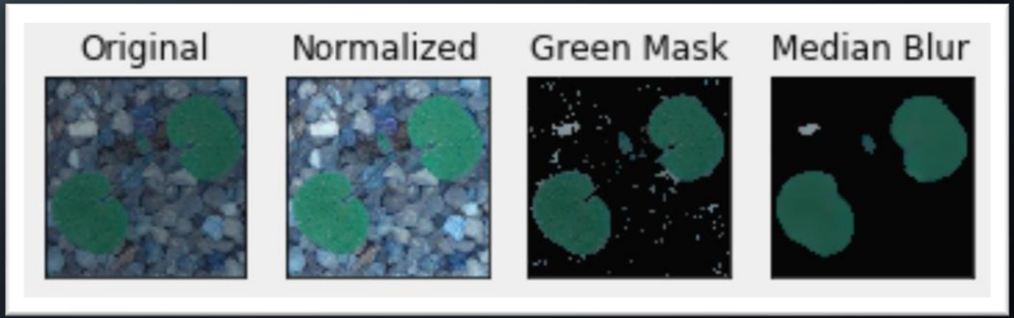


$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

PREPROCESSING TECHNIQUES

- Comparing:
  - No preprocessing
  - Min-max normalization
  - Min-max normalization + mask + median blur
- Full preprocessing produced best results at:
  - 88.2% accuracy
  - 88.1% recall
  - 89.2% precision
  - 88% F1

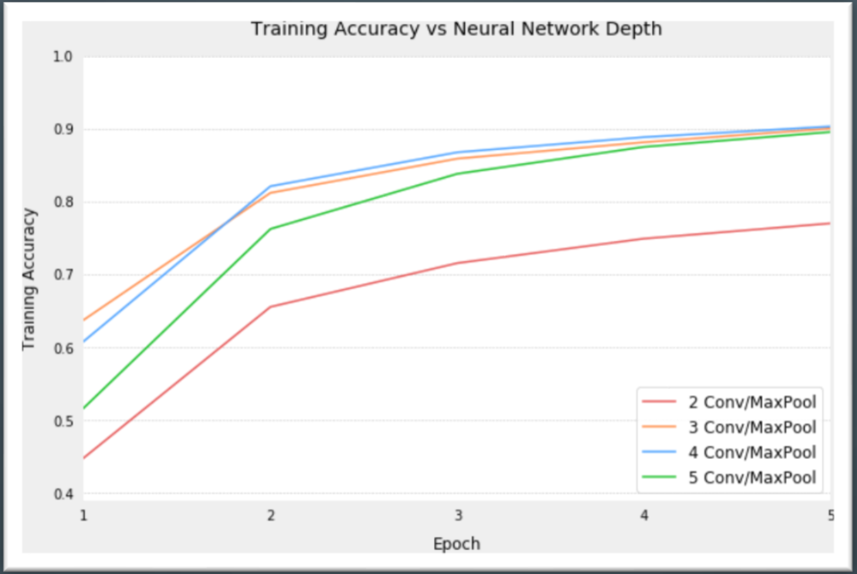


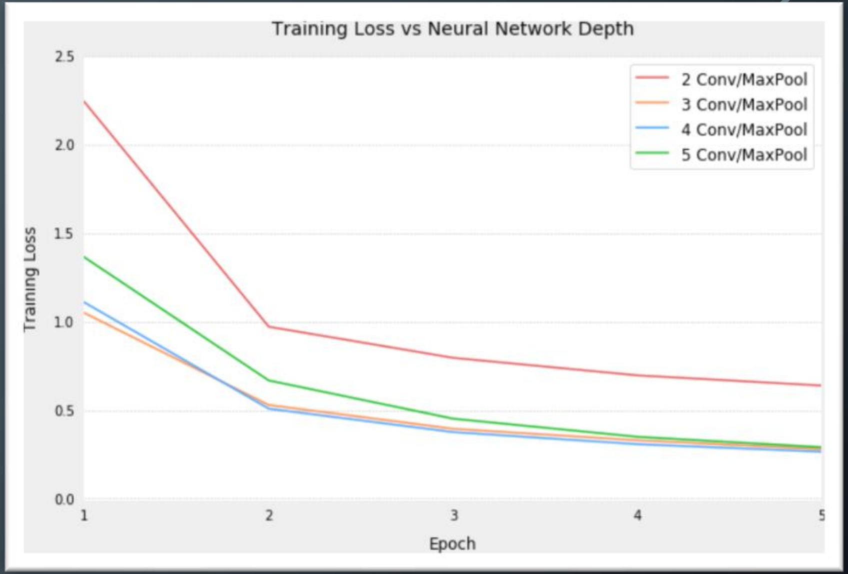


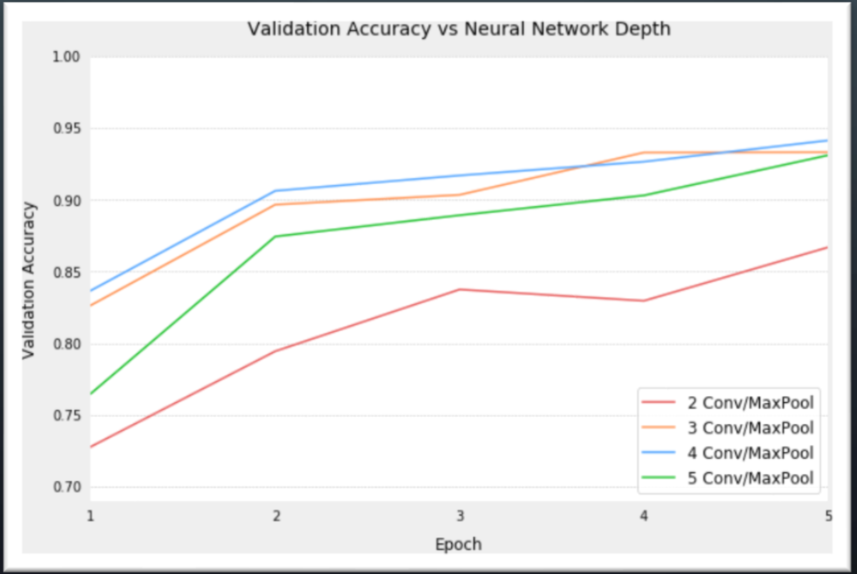
NETWORK ARCHITECTURE: LAYERS

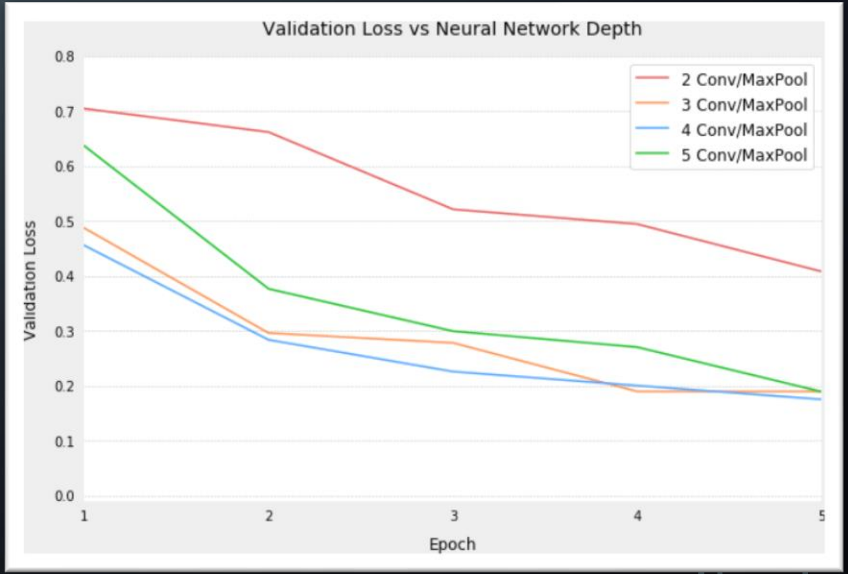
- Conv Block:
  - 1 Convolution Layer
    - 32 Feature Maps
    - 5x5, 3x3 Kernels
  - Batch Normalization
  - ReLU Activation
  - Max Pool (2x2)

- Architecture:
  - Conv Blocks
  - Flatten Layer
  - FC1 Layer (128x1)
  - Dropout(0.1)
  - FC2 Layer (12x1)
  - Softmax Activation





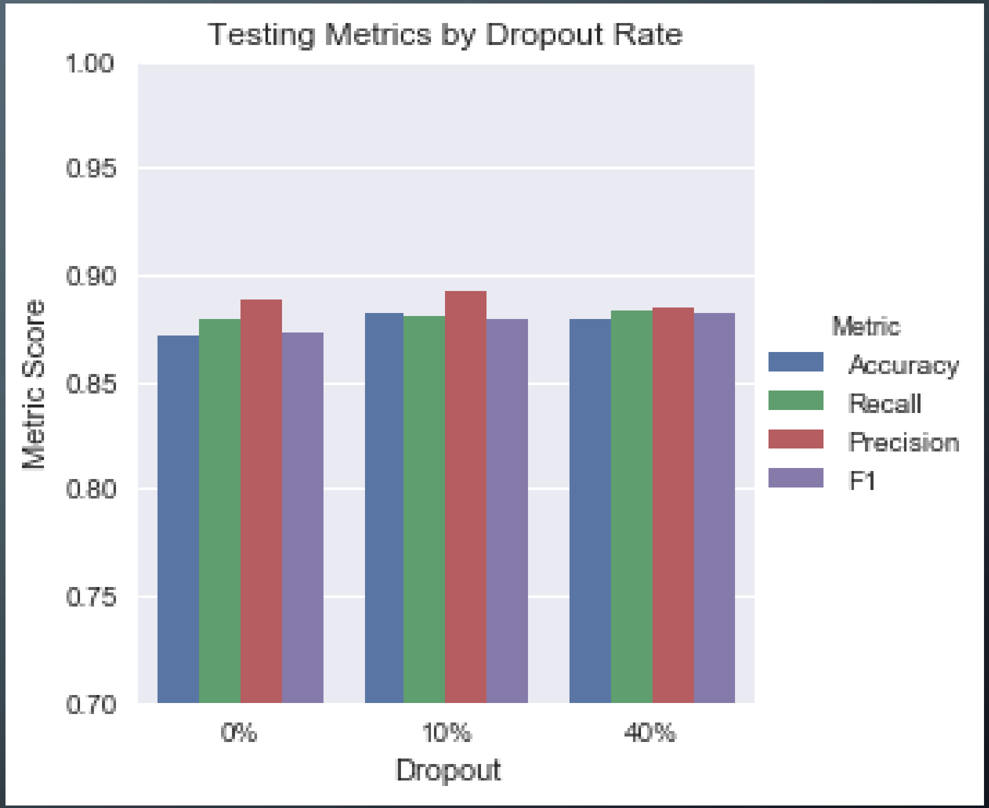




NETWORK ARCHITECTURE: DROPOUT

- Dropout on first FC Layer at three rates:
  - 0% Neuron dropout
  - 10% Neuron dropout
  - 40% Neuron dropout

- 10% Neuron dropout with highest accuracy (88.2%) and highest precision (89.2%)

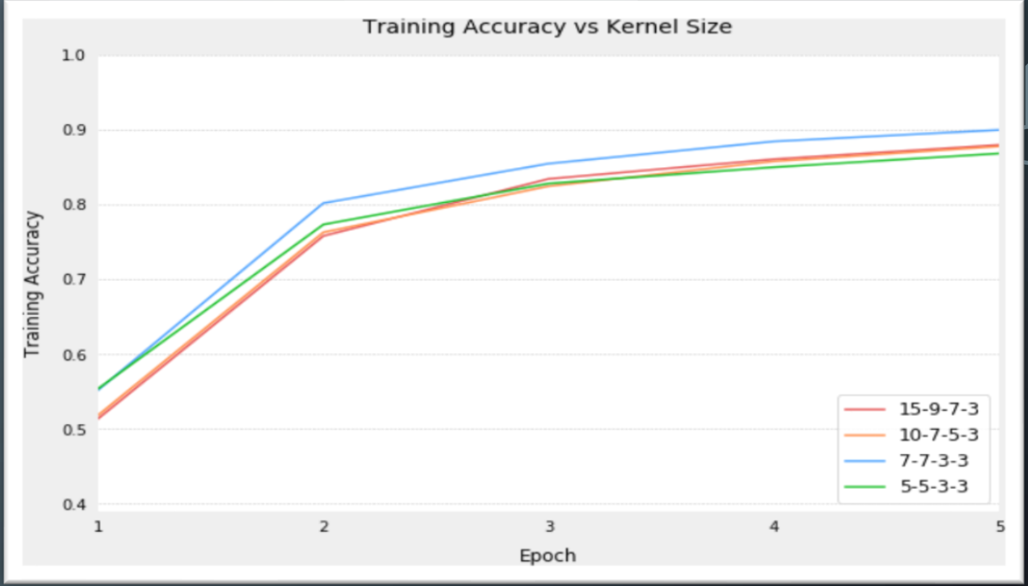


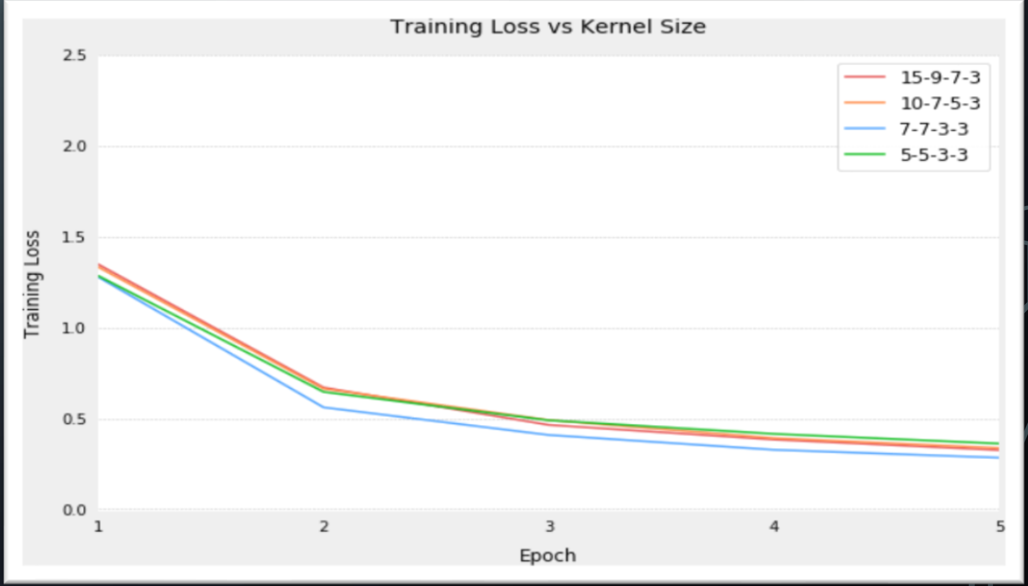
Dropout	Accuracy	Recall	Precision	F1
0%	0.8715	0.8795	0.889	0.873
10%	0.882	0.881	0.892	0.88
40%	0.8795	0.8839	0.885	0.8821

NETWORK ARCHITECTURE: KERNEL SIZE

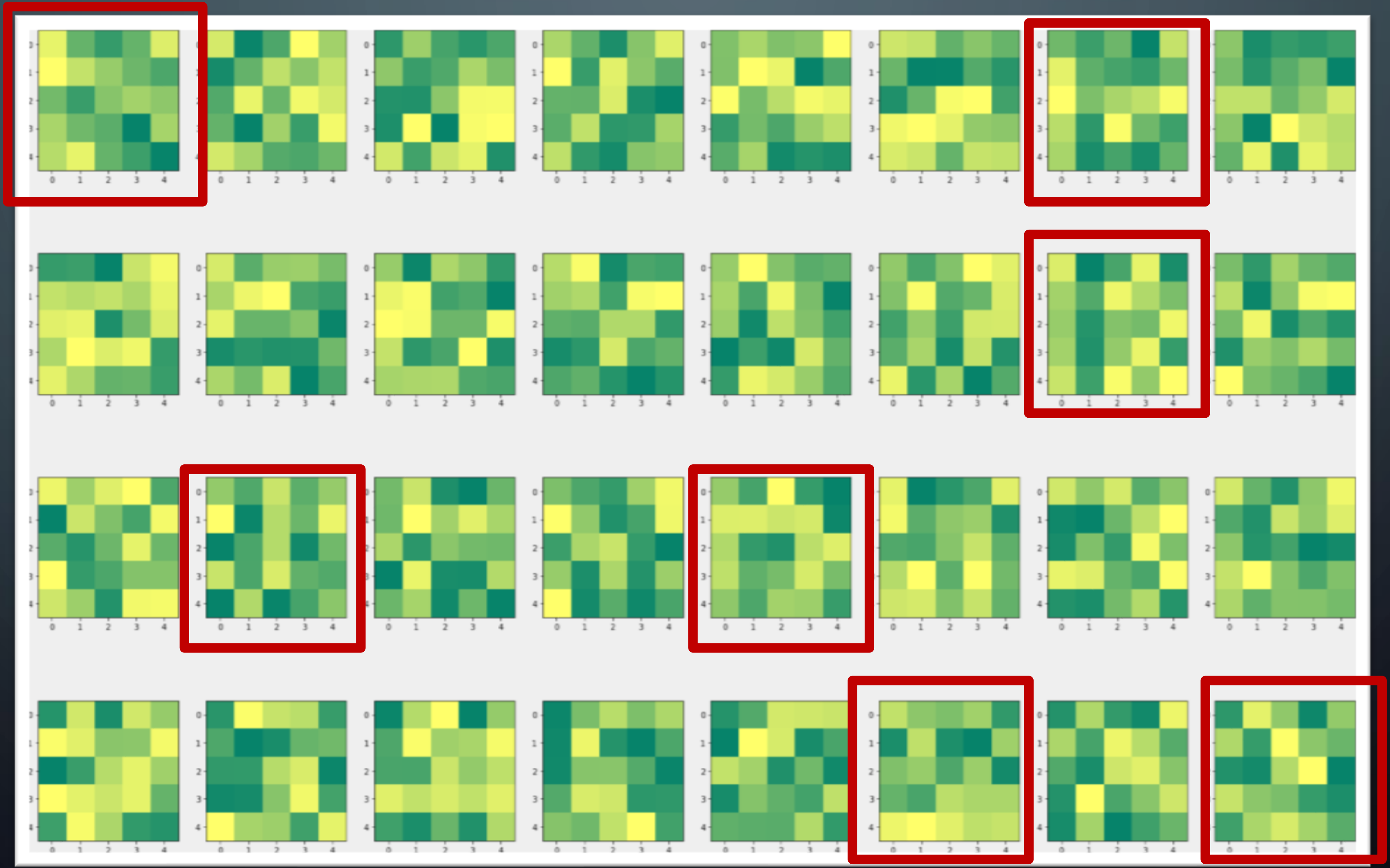
- Four architectures were tested:
  - *15x15-9x9-7x7-3x3*
  - *10x10-7x7-5x5-3x3*
  - *7x7-7x7-3x3-3x3*
  - *5x5-5x5-3x3-3x3.*

		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
	Train Time	2452	1540	1296	1305



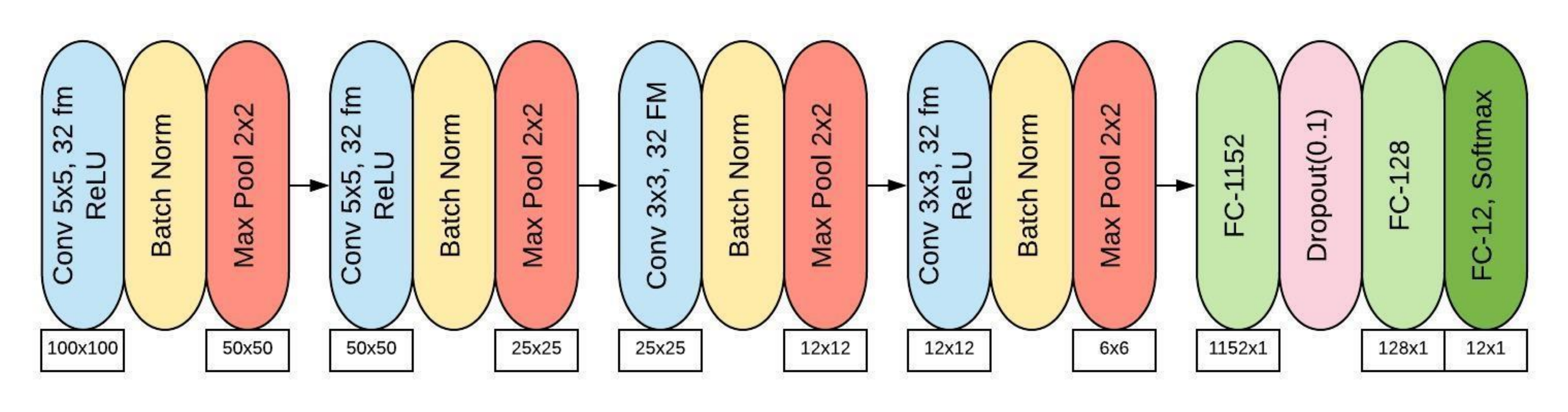


5X5 KERNELS OF FIRST CONVOLUTION LAYER

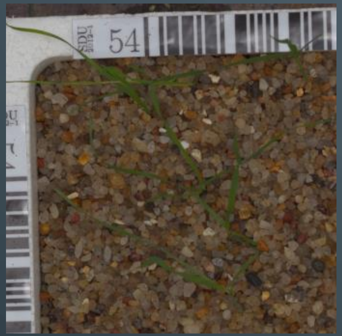


FINAL MODEL

Preprocessing: Min-Max Normalization + Green Mask + Median Blur



Accuracy: 0.8820
Recall: 0.8802
Precision: 0.8923
F1: 0.8808



Black-grass

Loose Silky-bent

Final Model Confusion Matrix

Actual \ Predicted	Black-grass	Charlock	Cleavers	Common Chickweed	Common wheat	Fat Hen	Loose Silky-bent	Maize	Scentless Mayweed	Shepherds Purse	Small-flowered Cranesbill	Sugar beet
Black-grass	314	0	0	0	11	1	563	3	2	0	0	0
Charlock	0	942	23	0	7	6	0	4	0	0	2	10
Cleavers	0	14	944	3	4	1	0	0	4	0	0	6
Common Chickweed	1	0	12	963	3	3	2	2	12	14	11	15
Common wheat	50	0	0	0	685	0	4	10	0	0	0	2
Fat Hen	6	8	11	1	10	731	13	9	0	0	0	18
Loose Silky-bent	84	0	0	0	2	6	1013	1	3	1	0	2
Maize	8	0	1	0	7	0	5	707	0	0	0	23
Scentless Mayweed	2	0	14	5	5	2	4	3	796	19	3	24
Shepherds Purse	0	11	2	5	0	3	0	0	50	704	1	8
Small-flowered Cranesbill	0	18	16	6	1	0	2	2	4	2	792	0
Sugar beet	14	1	7	1	12	3	0	1	0	0	0	942

FLASK APPLICATION

- Python micro web framework to develop web applications
- Connect frontend HTML and JavaScript with backend Python using templates and connect pages using route decorators
- Upload files through upload bar, run each image in file through model, output graphics, and sort on local file system

QUESTIONS?