Capstone Project
Machine Learning Engineer Nanodegree

Plant Seedlings Classification

Definition

Project Overview

The ability to efficiently differentiate between a weed and crop seedling would allow crop growers and farmers to improve their crop yields and maintain the health of their corner of the environment. In addition, it would aid amateur and hobbyist crop growers in identifying any weeds growing among their crops. This capstone project is based on the Plant Seedlings Classification challenge on Kaggle [1].

Problem Statement

The goal of this project is to correctly identify the species of a weed or crop seedling given its image. Given multiple plant images to be classified, the goal would be to maximize the micro-averaged F1-score produced by the classifier's predictions; this would depend on the number of true positives, false positives, and false negatives reported by the classifier [2]. The classifier should be able to work with any seedling that belongs to one of 12 specific plant species.

Metrics

As mentioned in the Problem Statement, the micro-averaged F1-score will be used to evaluate the accuracy of the benchmark model and any classifiers:

$$F1_{micro} = \frac{2Precision_{micro}Recall_{micro}}{Precision_{micro} + Recall_{micro}}$$

This metric is based on the precision and recall produced by a classifier's test, which are in turn based on the number of true positives, false positives, and false negatives (for each class *k*) produced by the test:

$$Precision_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} TP_k + FP_k}$$

$$Recall_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} TP_k + FN_k}$$

Analysis

Data Exploration and Exploratory Visualization

The dataset used in this project was provided by the Aarhus University Signal Processing Group in collaboration with the University of Southern Denmark [3]. This data comprises 4750 coloured plant images non-uniformly split among 12 different plant species.

While all images are in square format, each image varies in its dimensions; one might be 993 pixels wide by 993 pixels high, while another might be only 198 pixels wide by 198 pixels high. As such, the images will require normalization before they can be used in developing the classifier.

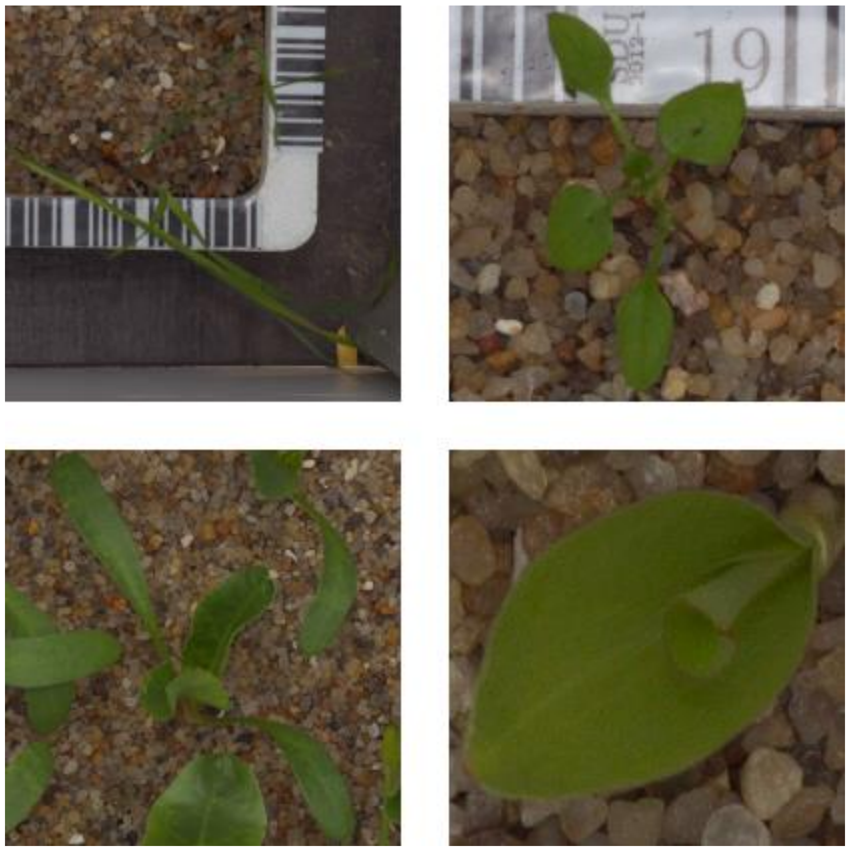

Figure 1: *Sample plant images taken from the Aarhus University dataset that represent one of 12 plant species (top-left, going clockwise, with dimensions in parentheses): Black-grass (993 x 993), Common Chickweed (443 x 443), Maize (198 x 198), Sugar beet (1061 x 1061).*

The distribution of images per species is non-uniform; certain species have more images than others. While this likely won't pose an issue, it would still be worth keeping in mind when training a classifier on such data and subsequently using it in predictions. Such a non-uniform distribution may help explain why the classifier overfit to certain species, if such a case occurs.

In addition, when setting up the training and testing data split, care must be taken to ensure that the random split does not accidentally allocate too many images of a particular species to either the training or testing set. Otherwise, the classifier may have a difficult time training on a particular species, or it may not be able to sufficiently test its accuracy on a particular species.

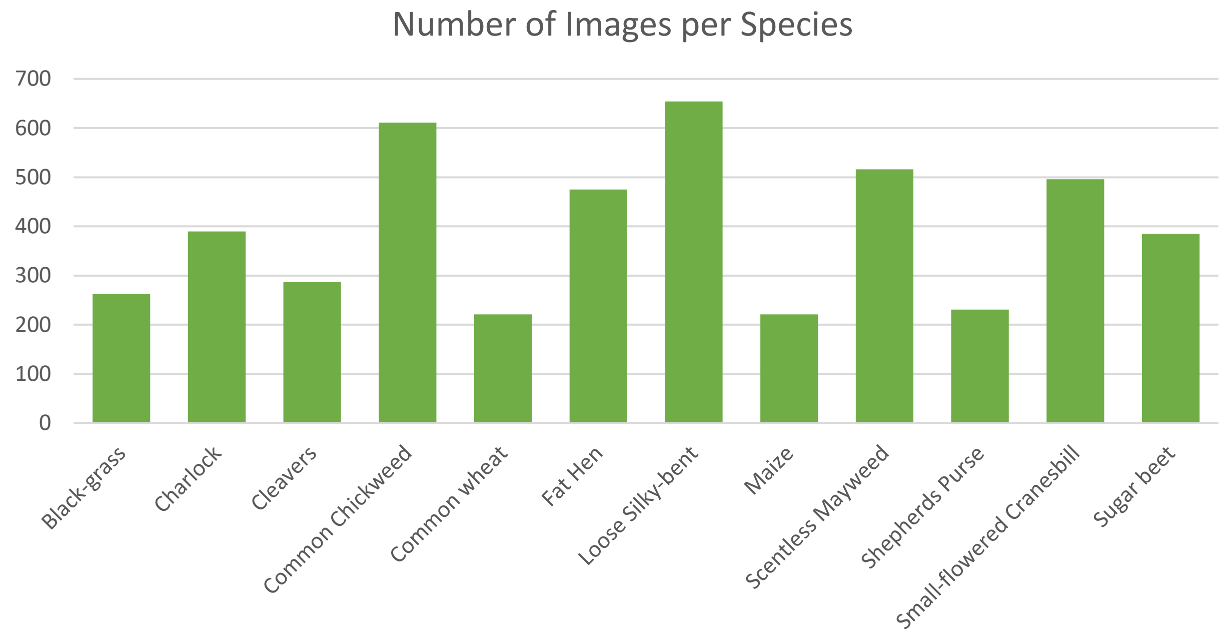

Figure 2: *Number of plant images that belong to each of the 12 species, revealing a non-uniform distribution of data.*

Algorithms and Techniques

A Convolutional Neural Network would be the tool of choice for this project; since the challenge is to classify images of plants, a CNN is well suited for detecting basic features within certain areas of an image and then building on them to learn more complex features, allowing it to accurately identify high-level elements within an image [4]. For this particular project, a relatively simple CNN that consists of 3 convolutional layers interleaved with 3 pooling layers will form the core of the initial, exploratory model. This network would accept a normalized image as an input and would have its outputs passed into a global average pooling layer for dimensionality reduction, followed by a final dense layer that comprises 12 nodes, one for each of the plant species to be predicted.

Benchmark

A basic benchmark model would be a classifier that randomly assigns a plant species to a given image. Such a model would provide an intuitive baseline against which a more complex model could be evaluated; by calculating the micro-averaged F1-score produced by this random classifier, it can be easily compared against the same score produced by other models.

Methodology

Data Preprocessing

A random 70% of images are designated the training set, 20% are designated the validation set, and the remaining 10% are designated the testing set. This gives the classifier a chance to predict data that it had never "seen" (i.e. trained on) before; the results of which would be used to determine the accuracy of the classifier according to the established evaluation metric.

```python
# Obtain plant image paths:
import random
random.seed(12)

import os
import numpy as np
from glob import glob

image_paths = []
for i, dir in enumerate(os.listdir("data")):
    for sub_dir in glob("data\\" + dir + "\\*"):
        image_paths.append([sub_dir, i]) # (path and index)

print('There are %d plant images.' % len(image_paths))
```

Figure 3: *(Python) Obtain plant image paths.*

```
# Split data into training (70%), validation (20%), and testing (10%) sets:
from keras.utils import np_utils
random.shuffle(image_paths)

training_count = int(len(image_paths) * 0.7)
validation_count = int(len(image_paths) * 0.2)

training_paths = np.array([image_path[0] for image_path in
image_paths[:training_count]])
training_targets = np_utils.to_categorical(np.array([image_path[1] for
image_path in image_paths[:training_count]]), 12)
del image_paths[:training_count]

validation_paths = np.array([image_path[0] for image_path in
image_paths[:validation_count]])
validation_targets = np_utils.to_categorical(np.array([image_path[1] for
image_path in image_paths[:validation_count]]), 12)
del image_paths[:validation_count]

testing_paths = np.array([image_path[0] for image_path in image_paths])
testing_targets = np_utils.to_categorical(np.array([image_path[1] for
image_path in image_paths]), 12)

print('There are %d training images, %d validation images, and %d testing
images.' % (len(training_paths), len(validation_paths),
len(testing_paths)))
```

Figure 4: *(Python) Split data into training, validation, and testing sets.*

Each image is resized to 224 x 224 pixels, converted to an array of size 224 x 224 x 3 (accounting for the 3 RGB channels), and rescaled by having each pixel in each channel divided by 255. This normalization allows all images to be fed into the same CNN, which accepts images of fixed size, and scales each colour channel such that it ranges from 0 to 1 rather than 0 to 255.

```
# Load images, resize them to 224 x 224 pixels, and divide each channel
pixel by 255:
from keras.preprocessing import image
from tqdm import tqdm
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

def path_to_tensor(image_path):
    # Load RGB image as PIL.Image.Image type:
    img = image.load_img(image_path, target_size=(224, 224))
    # Convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3):
    x = image.img_to_array(img)
    # Convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return
4D tensor:
    return np.expand_dims(x, axis=0)

def paths_to_tensor(image_paths):
    return np.vstack([path_to_tensor(image_path) for image_path in
tqdm(image_paths)])

training_tensors = paths_to_tensor(training_paths).astype('float32') / 255
validation_tensors = paths_to_tensor(validation_paths).astype('float32') /
255
testing_tensors = paths_to_tensor(testing_paths).astype('float32') / 255
```

Figure 5: *(Python) Load images, resize them to 224 x 224 pixels, and divide each channel pixel by 255.*

Implementation

With the images normalized and the training and testing sets prepared, the training data is fed into a CNN of 3 convolutional layers interleaved with 3 pooling layers, as outlined in the Algorithms and Techniques section. Each convolutional layer will consist of a 2 x 2 kernel, 16-64 filters, and a ReLU activation function, while each pooling layer will be represented by a max pooling function with a 2 x 2 filter.

This narrowing and deepening network is intended to extract only the relevant features of an image while keeping the total number of parameters manageable. The output of the final pooling layer is fed into a global average pooling layer to further condense the network into a 1D array of nodes; this connects to a final dense layer that comprises 12 nodes, one for each of the plant species to be predicted.

```
# Set up the Convolutional Neural Network:
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential

model = Sequential()

model.add(Conv2D(filters=16, kernel_size=2, padding='same',
activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=32, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(GlobalAveragePooling2D())

model.add(Dense(12, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

model.summary()
```

Figure 6: *(Python) Set up a basic Convolutional Neural Network.*

A summary of such a model may be visualized as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0         
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 12)                780       
=================================================================
Total params: 11,324.0
Trainable params: 11,324.0
Non-trainable params: 0.0
_________________________________________________________________
```

To start off, this model is trained on the training set for 5 epochs. From that point on, the number of epochs, layers, and their specific parameters will be tuned with the intent of producing a reasonably accurate plant species classifier.

```python
# Train model:
from keras.callbacks import ModelCheckpoint

checkpointer = 
ModelCheckpoint(filepath='saved_models/weights.best.from_scratch.hdf5', 
verbose=1, save_best_only=True)

model.fit(training_tensors, training_targets, 
          validation_data=(validation_tensors, validation_targets),
          epochs=5, batch_size=20, callbacks=[checkpointer], verbose=2)

print("Finished training model.")
```

Figure 7: *(Python) Train the CNN for 5 epochs.*

Refinement

A benchmark model that randomly classifies plant images would, on average, produce 1 true positive for every 12 true positives and false positives, and the same for every 12 true positives and false negatives. As such, the precision, recall, and micro-averaged F1-score would come out to 1/12, or 0.0833.

The initial CNN produced a micro-averaged F1-score of 0.2716 when tested on the testing set. While this is somewhat higher compared to the benchmark model's expected F1-score of 0.0833, it misclassifies images the majority of the time; tuning the CNN should improve its F1-score.

```
# Load weights:
model.load_weights('saved_models/weights.best.from_scratch.hdf5')

# Get index of predicted plant species for each image in testing set:
plant_species_predictions = [np.argmax(model.predict(np.expand_dims(tensor,
axis=0))) for tensor in testing_tensors]

# Report micro-averaged F1-score:
from sklearn.metrics import f1_score
test_accuracy = f1_score(np.argmax(testing_targets, axis=1),
plant_species_predictions, average='micro')
print('Micro-averaged F1-score: %.4f' % test_accuracy)
```

Figure 8: *(Python) Load the best weights, predict the species of each plant image in the testing set, then calculate the micro-averaged F1-score of the predictions.*

Adding more convolutional and pooling layers to the CNN improved its F1-score, while changing the kernel size of the convolutional layers from 2 x 2 to 3 x 3 did not seem to help much. Training the CNN for more epochs often improved the F1-score, although there were a few cases where the CNN produced dead neurons; the model ended up assigning the same class to all test images. To alleviate this specific issue, the convolutional layers' activation function was changed from ReLU to Leaky ReLU, to propagate a small non-zero gradient when the neuron is inactive instead of completely eliminating the information. This idea was taken further by utilizing a Parametric ReLU instead; a more general Leaky ReLU whose coefficient of leakage is not constant, but made into a parameter that is learned by the CNN.

With those insights, a CNN with 6 convolutional layers using Parametric ReLUs and 6 pooling layers was set up and trained for 40 epochs. The resultant model produced an F1-score of 0.8295, a significant improvement over the initial CNN.

```python
# Set up the Convolutional Neural Network:
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential
from keras.layers.advanced_activations import LeakyReLU, PReLU

model = Sequential()

model.add(Conv2D(filters=16, kernel_size=2, padding='same',
input_shape=(224, 224, 3)))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=32, kernel_size=2, padding='same'))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=64, kernel_size=2, padding='same'))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=128, kernel_size=2, padding='same'))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=256, kernel_size=2, padding='same'))
model.add(PReLU())
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=512, kernel_size=2, padding='same'))
model.add(PReLU())
model.add(GlobalAveragePooling2D())

model.add(Dense(12, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

model.summary()
```

Figure 9: *(Python) Set up a Convolutional Neural Network that uses more layers and Parametric ReLUs.*

Results

Model Evaluation and Validation

The final CNN setup produced the highest F1-score out of the initial and intermediate CNNs when tested on unseen images (the testing set). This CNN consists of 6 convolutional layers, each with a 2 x 2 kernel and double the number of filters of the previous convolutional layer; the convolutional layers have 16, 32, 64, 128, 256, and 512 filters in order. Each convolutional layer uses Parametric ReLU as the activation function, and is connected to a max pooling layer with a pool size of 2, except the last convolutional layer is connected to a global average pooling layer. The final pooling layer is densely connected to an output layer that consists of 12 nodes (one for each plant species to be classified) and uses softmax as the activation function.

A summary of this CNN may be visualized as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208       
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 224, 224, 16)      802816    
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080      
_________________________________________________________________
p_re_lu_2 (PReLU)            (None, 112, 112, 32)      401408    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256      
_________________________________________________________________
p_re_lu_3 (PReLU)            (None, 56, 56, 64)        200704    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 128)       32896     
_________________________________________________________________
p_re_lu_4 (PReLU)            (None, 28, 28, 128)       100352    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 128)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 256)       131328    
_________________________________________________________________
p_re_lu_5 (PReLU)            (None, 14, 14, 256)       50176     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 256)         0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 7, 7, 512)         524800    
_________________________________________________________________
p_re_lu_6 (PReLU)            (None, 7, 7, 512)         25088     
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 12)                6156      
=================================================================
Total params: 2,286,268.0
Trainable params: 2,286,268.0
Non-trainable params: 0.0
_________________________________________________________________
```

Justification

The final CNN model produced a micro-averaged F1-score of 0.8295, a significant improvement over both the initial CNN's F1-score of 0.2716 and the benchmark model's expected F1-score of 0.0833. Such a model misclassifies plant species fewer than once every 5 tries; while it may not be suitable for systems that would fully rely on such a model for plant species identification, it can still be a reasonable tool for use in amateur crop growing projects or as a preliminary plant species identifier in industrial applications, one that would aid (but not supplant) humans in identifying crops.

Conclusion

Free-Form Visualization

The predictions of the final CNN model may be visualized using a confusion matrix, allowing one to easily identify which plant species the model might have struggled with. Figure 10 below shows that most plant species have been correctly identified, save for Black-grass and Loose Silky-bent; the model struggles to differentiate between those two species. While a more complex CNN might help differentiate between those species, it may be more susceptible to overfitting.

	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	15	0	0	0	2	1	19	0	0	0	0	0
Charlock	0	28	1	0	0	0	0	1	1	0	0	1
Cleavers	0	1	22	0	0	1	0	0	1	1	0	0
Common Chickweed	0	0	0	52	0	1	0	0	1	0	1	1
Common wheat	0	0	0	0	17	1	0	0	0	0	0	0
Fat Hen	1	0	0	0	0	39	0	0	0	0	0	1
Loose Silky-bent	7	0	0	1	4	2	49	1	2	0	0	0
Maize	0	0	0	1	0	0	0	29	1	0	0	2
Scentless Mayweed	1	0	0	0	0	0	0	1	46	3	0	1
Shepherds Purse	0	0	0	0	0	0	0	0	3	12	0	0
Small-flowered Cranesbill	0	1	0	1	0	0	0	0	0	1	51	2
Sugar beet	1	0	0	1	1	2	0	1	1	2	0	34

Figure 10: *Confusion matrix that summarizes the prediction results of the final CNN model. Each row represents the number of plants that were predicted to be a specific species.*

Reflection

The goal of this project was to develop a system that could reasonably predict the species of a plant seedling given its image. To approach this task, images of plant seedlings that represent 12 different species were split into training, validation, and testing sets, then preprocessed for use in a Convolutional Neural Network. A basic benchmark model was created to provide an intuitive baseline against which more advanced models may be compared. A basic CNN was then created and trained using the training and validation sets, then tested on the testing set. Using the micro-averaged F1-score as a metric for accuracy, the CNN was gradually improved upon until its F1-score was reasonable compared to the initial CNN's and the benchmark model's.

Particularly interesting was the tendency of the CNN to produce dead neurons once in a while, this was attributed to the use of ReLU as the activation function, which blocks the propagation of any signal if it were negative. Replacing this activation function with a Leaky ReLU or Parametric ReLU seemed to alleviate this issue, as it allowed negative signals to very weakly propagate through neurons.

The main challenge of this project was the amount of time needed to train a CNN or any of its modified versions; with long training times, the ability to quickly iterate upon and improve the architecture of a CNN noticeably suffers, and makes the development of a neural network that much more time-consuming.

The final CNN developed for this project reasonably solves the original problem, the classification of plant seedlings, and can be used to solve image recognition problems in general.

Improvement

In this project, the CNN was manually improved upon by implementing one modification (e.g. number of convolutional layers, kernel size, activation function) at a time. A more advanced CNN may instead make use of layers in parallel, as in the Inception and Xception models, offloading the manual process of CNN tuning to the model itself; instead of having a human decide which modification to attempt next, an advanced CNN would be able to pick and choose the parameters it requires for a specific problem. For example, such a CNN could have 3 convolutional layers in parallel, each with a different kernel size; through the process of training, the CNN may end up using one of the 3 layers more heavily as that may provide the most efficient flow of information through the network. Since the final CNN developed in the project can be made more general using this approach, it's highly likely that the final CNN could be used as a benchmark for an even more advanced CNN.

References

[1] https://www.kaggle.com/c/plant-seedlings-classification

[2] https://www.kaggle.com/wiki/MeanFScore

[3] https://vision.eng.au.dk/plant-seedlings-dataset/

[4] http://deeplearning.net/tutorial/lenet.html