Classification of Plant Seedlings using CNN

Raghavendra Prasad Savada

Nov 15, 2018

I. Definition

Project Overview

Weeds are one of the most significant causes of crop loss. For example, in India 45% of the crop loss is due to weeds[1]. Reduction in yield and quality due to weed infestations, and cost of controlling weeds have a tremendous economic impact on crop production. One report estimated that corn yield loss due to weeds was in the range of 8 to 18% across US and 3 to 12% across Canada[2]. Therefore, weed management (restricting the weed population to a minimum threshold) is an important part of crop production. Accurate identification and classification of weed species is first and foremost step in the effective weed management.

In recent years, precision agriculture is gaining popularity and has a potential to revolutionize the agriculture sector to meet the food demands of growing world population. One of the main goals of the precision agriculture is to apply the right dose of agricultural inputs at the right spot at the right time by making use of advances in information technology and robotics. With regards to weed management, in precision agriculture, it is possible to identify weeds and apply appropriate herbicide in right dose. Or identified weeds can be completely eliminated[3]. In either case, an accurate system of weed identification and classification is critical.

Recent advances in image recognition can aid accurate identification and classification of weed species in a cost-effective manner. In this project, I have built a Convolutional Neural Network (CNN) to identify crop and weed species at seedling stage. I used a plant seedling data set, released by the Aarhus University Signal Processing group in collaboration with University of Southern Denmark[4]. The dataset has been downloaded from the link https://vision.eng.au.dk/plant-seedlings-dataset/ (file name; V1: Nonsegmented single plants (1.7GB)).

Problem Statement

Weeds should be controlled as early as possible after crop germination to avoid the adverse effects on crops resulting from competition between weeds and crops for space, sunlight and nutrition. Hence it is important to control the weeds in the seedling stage itself by correctly identifying them. However, there are multiple issues that make this problem really challenging. At seedling stage there is a high resemblance between many plant species so that distinguishing crop species from weed species is very difficult and labor intensive, and requires high expertise (Figure 1).

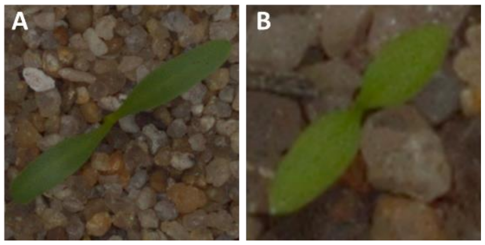

Figure 1. Highly resembling Sugar Beet, a crop (A) and Common Chickweed, a weed (B)[4].

Correctly identifying weed species is equally difficult because of the resemblance between different species (minimal inter-class variation; Figure 2).

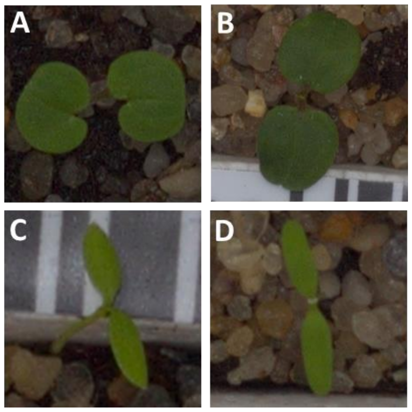

Figure 2. Highly resembling weed species Small-flowered Cranesbill (A) and Cleavers seedlings (B). Similarly, Common chickweed (C) and Fat hen (D) have high resemblance[4].

Correctly identifying weed species is critical as different weed species requires different management practices. For instance, herbicide (chemicals that kill weeds) that works on monocotyledons (e.g., grasses) may not work on dicotyledons (e.g., broad-leafed plants like pea; Figure 3). Therefore, accurate identification and classification of weed species is of paramount importance in effective weed management.

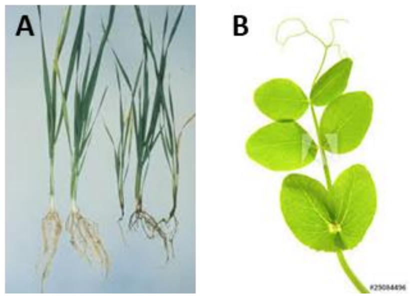

Figure 3. Monocotyledons[5]; wheat seedling (A) Vs Dicotyledons[6]; pea seedling (B).

Furthermore, within each species, there is a significant variation in appearance of seedlings that are at different growth stages (high intra-class variation; Figure 4). It should be noted that at the time of weed management, there may be seedlings of the same weed species that are at different growth stages.

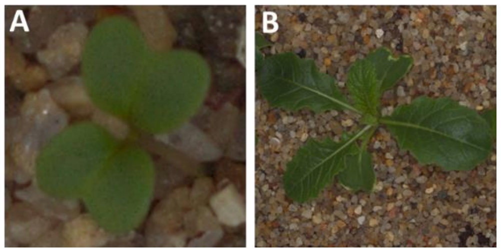

Figure 4. Variation within the same weed species of Charlock that are at different growth stage[4].

I have trained a CNN using the dataset mentioned above, which has images of nine weed species and three crop species (further details in the following sections). CNN are very effective in recognition of images containing complex patterns because they exploit feature locality, in that, they find patterns in different parts of the image. They do it at different granularities, therefore are able to model hierarchically higher level features[7]. The trained CNN should be able to identify the species of the seedling in the image of one of the 12 species included in the dataset. This model is useful in geographical regions where the three crop species included in this dataset (Sugar beet, Maize and Common wheat) are grown and have the infestations of nine weed species included in the dataset. This model can be used to classify the seedlings in the images taken by farmers, or images taken by equipment used in precision agriculture.

Controlling the weeds 100% is not economically viable, because cost of weed control outweighs the yield loss due to weeds. Usually farmers control the weeds to the extent of 80% to 85%. So if this model could achieve the accuracy of over of 85%, it can provide a practical solution to the weed classification problem.

Evaluation Metrics

This is a fairly well-balanced image dataset. Different classes (species) are well represented in the dataset and almost uniformly distributed. Please see Table 1 below for the distribution of class labels. Although, this is not perfectly balanced dataset, it is not highly skewed either. Therefore, I have used intuitive accuracy score (ratio of correctly predicted observations to the total observations) as the evaluation metric to measure performance of the models.

II. Analysis

Data Exploration

The plant seedling data set released by the Aarhus University Signal Processing group[4] contains 5,544 images of 960 unique plants belonging to nine weed species and three crop species at several growth stages over a period of 20 days starting a few days after emergence of seedlings. A dSLR camera (Canon 600D), placed approximately 110 - 115 cm above the soil surface was used with a fixed 50mm lens for recording RGB images. Images have a native resolution of 5184 x 3456 px (approximately 10 pixels per millimeter).

Among the 12 species, nine are weed species (Black-grass, Charlock, Cleavers, Common Chickweed, Fat Hen, Loose Silky-bent, Scentless Mayweed, Shepherd's Purse, Small-flowered Cranesbill) and the remaining 3 are crop species (Sugar beet, Maize and Common wheat). The distribution of class labels is given in the Table 1.

Indices	Species	Class Count
0	Black-grass	310
1	Charlock	452
2	Cleavers	335
3	Common Chickweed	713
4	Common wheat	253
5	Fat Hen	538
6	Loose Silky-bent	766
7	Maize	257
8	Scentless Mayweed	607
9	Shepherds Purse	274
10	Small-flowered Cranesbill	576
11	Sugar beet	463

Table 1. Distribution of class labels in the dataset

Exploratory Visualization

An example image is shown for each of the 12 classes in Figure 5. Minimal inter-class and high intra-class variation problem is discussed above; please see Figure 1 (resemblance between some crop species and weed species), Figure 2 (resemblance among some weed species), and Figure 4 (intra-class variation). Please note that images in Figure 1, 2, and 4 are all from this dataset.

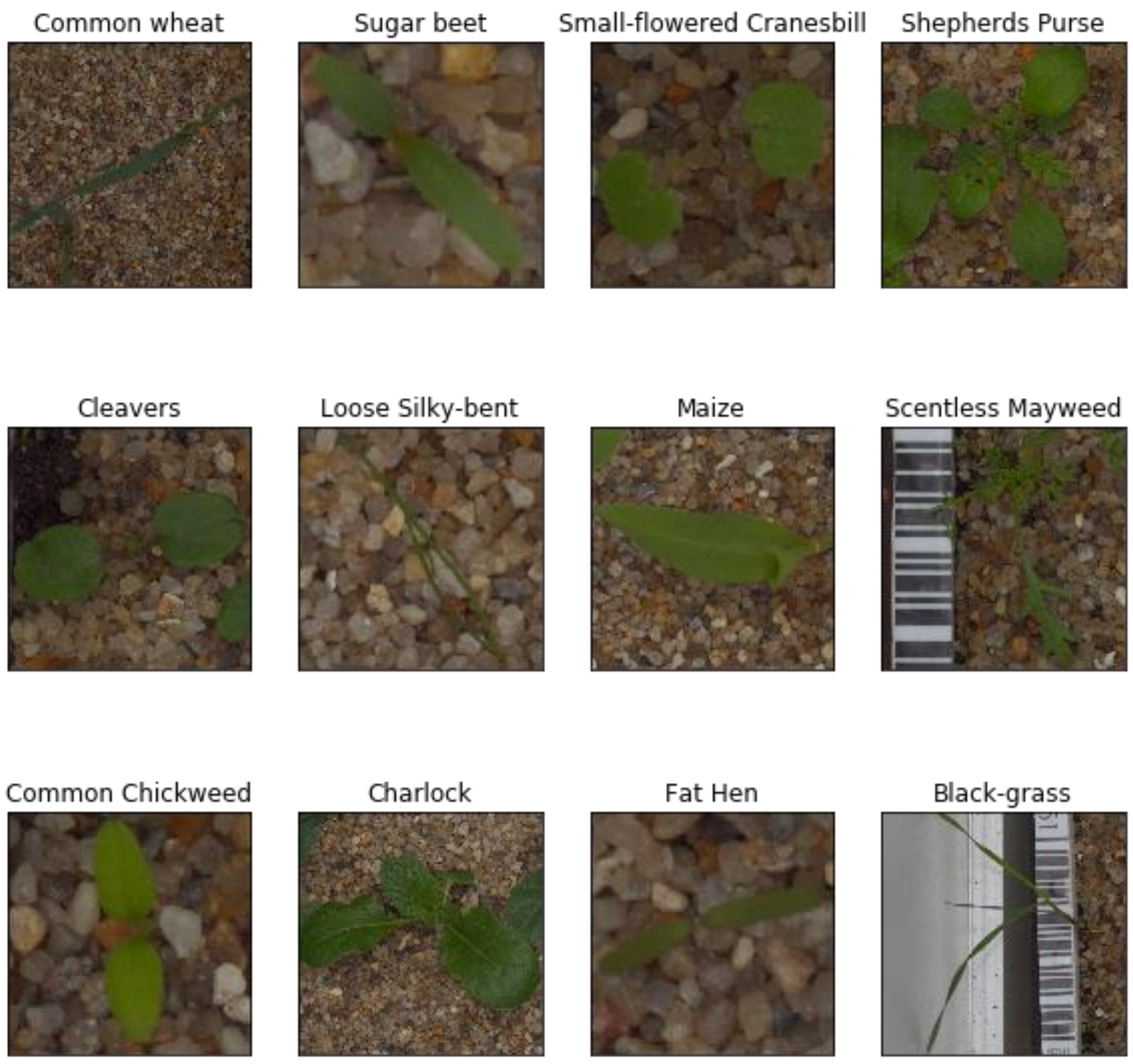

Figure 5. Example images of each of the 12 classes in the dataset. Among these species, Maize, Common Wheat, and Sugar Beat are crop species and the rest are weed species[4].

Algorithms and Techniques

To address the problem of classification of seedlings in images, I have built a CNN using Keras and tensorflow libraries.

Unlike a regular neural network, the layers of a CNN have neurons arranged in 3 dimensions (Figure 6): width, height, depth**.** Depth here refers to the third dimension of an activation volume. This helps to preserve spatial information, which is an important aspect for things like images. Unlike fully connected layers in regular neural network, in CNN, the neurons in a layer will only be connected to a small region of the previous layer. Then pooling layers are used to perform a down sampling operation along the spatial dimensions (width, height) which helps to reduce dimensionality thereby reducing overfitting[8, 9].

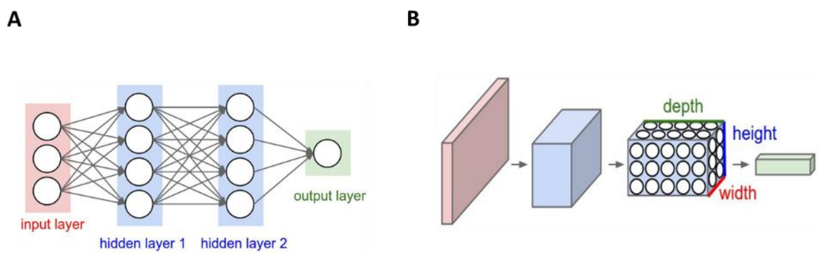

Figure 6. Difference between a regular Neural Network (A) and a CNN (B). Every layer of a CNN transforms the 3D input to a 3D output of neuron activations. In the image above, in part B, the red input layer represents the image, so width, height, depth represent Red, Green, Blue channels (image taken from reference 9).

I have used transfer learning technique, which involves taking a pre-trained neural network and adapting the network to a new, different data set[8]. I have used some of the pre-trained models like Inception V3, Xception, and ResNet50 models in Keras. Inception V3 is a multi-level feature extractor, which computes *1×1*, *3×3*, and *5×5* convolutions within the same module of the network. The output of these filters are then stacked along the channel dimension before being fed into the next layer[10]. Xception is an extension of the Inception architecture which replaces the standard Inception modules with depth wise separable convolutions[10]. They factorize the convolution layers, which results in far less parameters[11]. ResNet-50 model addresses the issue of vanishing gradient. Basically, in this deep CNN architecture, connections are added that skipped layers so the gradient signal has a shorter route to travel. Practice is far ahead of the theory in deep learning, so experimenting with many different architectures is critical to get good results. So I have chosen these three different architectures. VGG was not my choice because my previous experience suggests that its performance is far lower compared to ResNet50 in classifying image dataset with a problem of minimal inter-class variation and high intra-class variation.

This kind of transfer learning is possible because in any of the convolutional network, many of the initial layers mostly learn general patterns in images. Only filters in the final layers are more specific to the dataset used in training these models. So we can use weights from the initial layers of pre-trained architectures and build upon it to train CNNs on new dataset in image recognition problems. These pretrained architectures greatly improves the model performance for the reasons like[8];

- these models have been trained using ImageNet, a huge sample dataset with over 10 million images containing an object from one of 1000 categories.
- hyperparameters were rigorously tuned
- these architectures are result of careful experimentation with countless architectures, and are result of many years of expertise and months of dedicated work
- involved elaborate training that took weeks to train on state-of-the-art GPUs.

Benchmark Model

In the literature, I did not find any published model related to this problem that can be used as bench mark model. So I have trained and tested a vanilla CNN model on the same dataset using the following simple architecture.

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

Then I have examined how the model built using transfer learning compared to this vanilla CNN.

III. Methodology

Data Preprocessing

The dataset in a zip file has been downloaded from the link https://vision.eng.au.dk/plant-seedlings-dataset/ (file name; V1: Nonsegmented single plants (1.7GB)). Image files were imported using sklearn load_files function.

When using TensorFlow as backend, Keras CNNs require a 4D array (also referred to as a 4D tensor) as input with the shape (n, r, c, ch) where, n is total number of images and r, c and ch are rows, columns, and channels for each image, respectively. Since images in this dataset are colored images, each image has three channels. The custom function path_to_tensor in my jupyter notebook takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In my plain vanilla CNN, tensors are further rescaled by dividing each pixel by 255. In transfer learning, tensors are pre-processed using the preprocess_input function associated with each of the Keras transfer learning applications. Briefly, in the preprocess_input function, first, the RGB image is converted to BGR by reordering the channels. Then there is an additional normalization step, wherein, the mean pixel (calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image.

Implementation

The 5,544 labelled images were split into training, validation and test set at a ratio of 70%:15%:15%. Although, I would like to have more images in validation and test set, as it would have limit the images available for training, I used only 15% of the images each for validation and testing. The images were preprocessed as explained in the *Data Preprocessing* section. First, I trained a vanilla CNN that can be used as a benchmark model. I used the following architecture.

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

I used one locally connected convolutional layer, followed by a MaxPooling layer. Pooling layers help to reduce dimensionality thereby helping to minimize the problem of overfitting. They also help speeding up training. I set the padding to 'same' so that I do not lose information from the regions at the edge of the image. I used 'relu' activation function in Conv2D. ReLU (Rectified Linear Unit) speed up training. The gradient computation is very simple (either 0 or 1 depending on the sign of x). The computational step of a ReLU is easy as it does not involve exponentials, multiplication or division operations[12]. The array is then flattened (flatten layer) to a vector in the next layer of the CNN. It is followed by a final fully connected dense layer designed to further elucidate the content of the image. This layer has 12 nodes; one node for each seedling species in the dataset, and has a softmax activation function, so that it returns 12 probabilities (probability of image containing each of the 12 species). Then I compiled the model and fit it. Finally, I used this model to predict seedling species in the images in the test set. This model yielded a test set accuracy of 66.30%.

Then I used transfer learning to build my final CNN. I started off with InceptionV3 pre-trained architecture in Keras, which has been adapted to my dataset. First, images were preprocessed using the preprocess_input function associated with Keras InceptionV3 application. To what extent we can use pre-trained weights from this architecture in our new classification task depends on the size of the new data set, and the similarity of the new data set to the ImageNet data set. Since the seedling dataset is fairly big and ImageNet dataset also contained some plant images, I have used all the weights from all the layers of InceptionV3 except the last fully connected layer. Images have been passed through this pre-trained network, which is used as a fixed feature extractor, and last convolutional output (bottleneck features) has been used as input to the new network with a few layers. The weights of these new layers have been trained, which is able to learn patterns specific to the seedling dataset. The first of these layers is a convolutional layer with 64 filters and a kernel size of 2. I set the padding to ‘same’, and used relu activation function for the reasons explained above. The convolutional layer is followed by a global average pooling layer and a fully connected dense layer. Dense layer contains one node for each seedling species and is equipped with a softmax. I used Global Average Pooling layer because it provides extreme type of dimensionality reduction, thereby reducing training time and overfitting.

I compiled the above model using ‘categorical_crossentropy’ as loss function (as this is a multi-class classification) and ‘adam’ as optimizer. I used adam because it is computationally efficient and has little memory requirements[13]. I fit the model by passing extracted bottleneck features. I set save_best_only parameter to True and train the model for 25 epochs. During each epoch, validation set was used to estimate accuracy and find out whether validation loss improved. Training accuracy was also reported for each epoch. Although training accuracy continuously increased during 25 epochs, validation loss did not improve after 7 epochs. So training this model over 7 epochs may result in overfitting. However, it should be noted that I set the save_best_only parameter to True so that only weights form the best model can be loaded for downstream predictions.

Models that were built upon Xception and ResNet50 were also implemented in the same way as InceptionV3 as explained above. The steps of the implementation is outlined in the Figure 7.

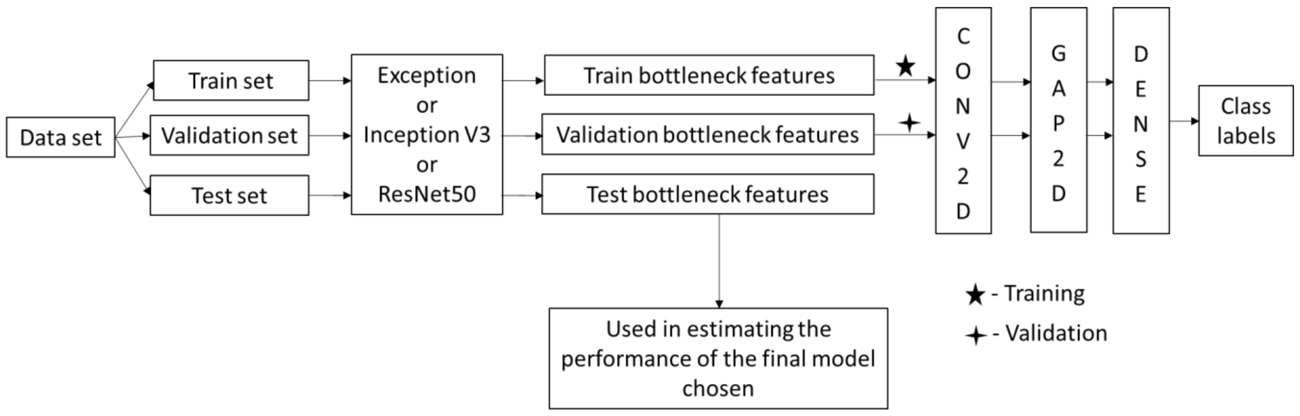
Data set
Train set
Validation set
Test set
Exception or Inception V3 or ResNet50
Train bottleneck features
Validation bottleneck features
Test bottleneck features
CONV2D
GAP2D
DENSE
Class labels
Used in estimating the performance of the final model chosen
★ - Training
✦ - Validation

Figure 7. Outline of the steps that I followed in training CNN.

Refinement

With the model built upon pre-trained InceptionV3 architecture, the validation accuracy of the best model was 77.98%. So I used two other pre-trained architectures available in Keras – Xception and ResNet50 – to see if performance can be improved. With Xception, accuracy on validation set was 82.43%. With ResNet50 the validation accuracy was 89.41%, which was the highest among the three architecture tested here. Considering the best performance of the model built upon ResNet50, I chose this as the final model for further investigation.

The final ResNet50 model has the following architecture and parameters.

```
resnet50_model = Sequential()
resnet50_model.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu', input_shape=train_resnet50.shape[1:]))
resnet50_model.add(GlobalAveragePooling2D())
resnet50_model.add(Dense(12, activation='softmax'))

loss='categorical_crossentropy'
optimizer='adam'
epochs=25
batch_size=20
```

Before arriving at this architecture and parameters, I tried a few other architectures and parameters, which are not shown in the notebook. A few examples of architectures and parameter I tried are given.

```
-------------------------------------------------------------------------
resnet50_2_model = Sequential()
resnet50_2_model.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu', input_shape=train_resnet50.shape[1:]))
```

```
resnet50_2_model.add(MaxPooling2D(pool_size=2))
resnet50_2_model.add(Dropout(0.3))
resnet50_2_model.add(Flatten())
resnet50_2_model.add(Dense(12, activation='softmax'))
resnet50_2_model.summary()

epochs=25
batch_size=32
loss = 'categorical_crossentropy'
optimizer = 'adam'

-------------------------------------------------------------------------
The same architecture as above is tried with filters = 128
-------------------------------------------------------------------------
The same architecture as above is tries with optimizer = ‘rmsprop
-------------------------------------------------------------------------
resnet50_3_model = Sequential()
Resnet50_3_model.add(GlobalAveragePooling2D(input_shape=train_Resnet50.sha
pe[1:]))
Resnet50_model.add(Dense(12, activation='softmax'))
loss='categorical_crossentropy'
optimizer='rmsprop'
epochs=25
batch_size=20

-------------------------------------------------------------------------
The same architecture as above is tried with batch size = 32
-------------------------------------------------------------------------
The same architecture as above is tried with optimizer = 'adam'
-------------------------------------------------------------------------
```

Note: I tried quite a few other combinations that I did not list above.

IV. Results

Model Evaluation and Validation

I chose the model built upon ResNet50 as my final model (please see the *Refinement* section above). While model training, I set the save_best_only parameter to True. So weights from only the best model is saved and loaded to be used in model predictions. The hold out test set was used to test the performance of the final model. As the test set is unseen by the model during training, the performance of the model on test set should give a fair indication of the model’s generalization ability. The final model predicted the test set with an accuracy of 92.06%.

I also estimated how the model performed on three crop species. It is important that model predicts crop species with good accuracy. Otherwise, you will end up destroying crop plants mistaking them for weed. My model recalled three crop species maize, sugar beat and common wheat with an accuracy of 91.18%, 91.04% and 92.50%, respectively.

I tested the final model by using it to predict some images of the seedlings downloaded from the internet to see how well model generalized to unseen data. I could find the appropriate images for seven different species. Among these seven species, my model could correctly predict six (Charlock, Fat Hen, Loose Silky-bent, Small-flowered Cranesbill, Sugar beet, Maize). It failed to predict the species Black-grass (Figure 8); my model predicted it as Loose Silky-bent, which is also a grass species. It should be noted that grass species like wheat, black-grass and loose silky-bent are really hard to identify even for trained eyes. So sometimes even farmers end up uprooting crop plants mistaking them for weeds, or leaving weed plants mistaking them for crop plants. Although, I would like to test more images, I could not find images of the seedling at appropriate stages on the internet.

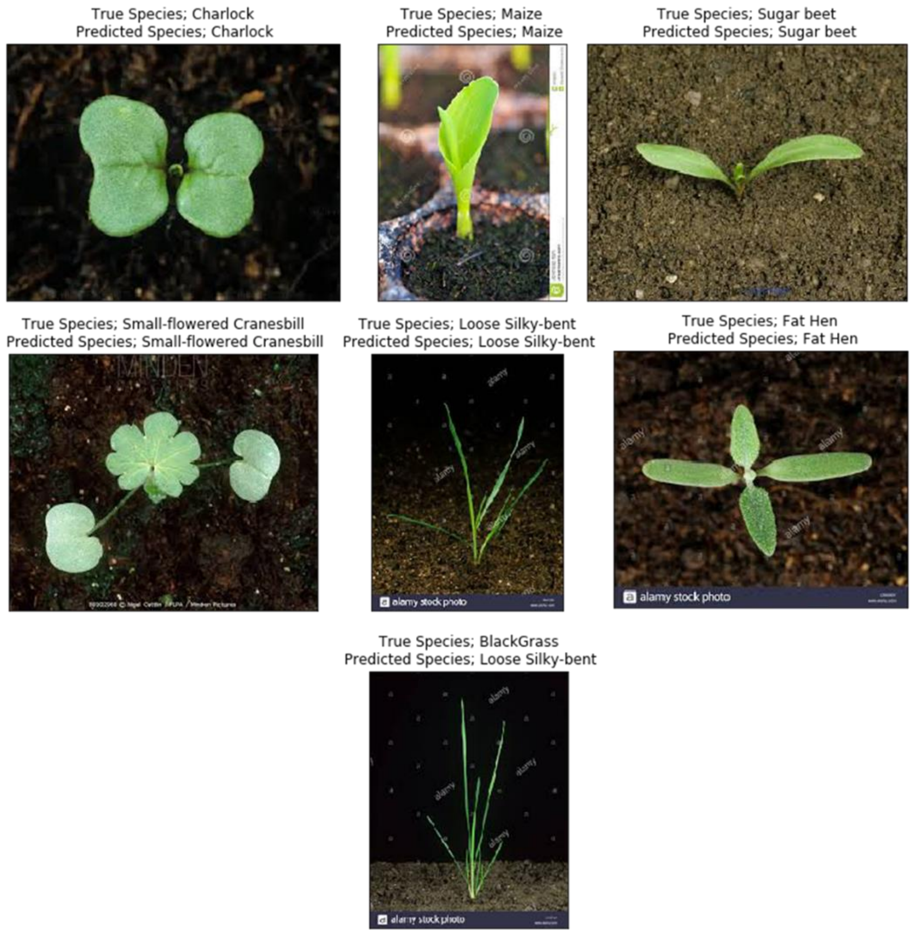
True Species; Charlock
Predicted Species; Charlock
True Species; Maize
Predicted Species; Maize
True Species; Sugar beet
Predicted Species; Sugar beet
True Species; Small-flowered Cranesbill
Predicted Species; Small-flowered Cranesbill
True Species; Loose Silky-bent
Predicted Species; Loose Silky-bent
True Species; Fat Hen
Predicted Species; Fat Hen
True Species; BlackGrass
Predicted Species; Loose Silky-bent

Figure 8. The final model predictions of the seedling images downloaded from the internet.

Justification

When I created a CNN from scratch as a benchmark model, the test accuracy was 66.30%. The test accuracy was significantly increased to 92.06% with the final model that I built on ResNet50 model. This level of accuracy is definitely of practical significance. It must be noted that, even with conventional weed management practices, farmers are not able to eliminate weeds completely. Controlling the weeds 100% is not economically viable, because cost of weed control outweighs the yield loss due to weeds. Usually on the farm, weeds are controlled to the extent of 80% to 85%. So if this model helps to identify weed and crop plants with an accuracy of 92%, it can provide a practical solution to the weed classification problem.

V. Conclusion

The problem of identifying the species of weeds and crops at seedling stage is of great significance in the field of Agriculture, specifically in weed management. It is relevant for both conventional and precision agriculture. This problem is quite challenging because of the frequent occurrence of the minimal inter-class variation (high resemblance between different species at seedling state) and high intraclass variation (seedlings belonging to the same species looks quite different at different growth stages). By using a plant seedling dataset released by a research group in Denmark[4], I trained a CNN that predicts plant seedling species in an image with an accuracy of around 92%. For training the CNN, I used a technique called transfer learning, where I adapted highly efficient pre-trained models to my dataset.

I used three different architectures. I started off with an architecture called InceptionV3, followed by Xception and finally, ResNet50. Among these, the model built on ResNet50 yielded the highest validation accuracy, and could predict images in test set with an accuracy of 92%. This model could also correctly predict species in the six of the seven seedling images downloaded from the internet (not in the dataset). This model can be used to classify the seedlings in the images taken by farmers. For instance, we can develop a mobile app where farmer's take images of the seedlings and feed them to this app, which helps to identify the weed species. This will help them to design effective weed management strategies. This model can also be used to classify images taken by equipment used in precision agriculture.

The model's prediction accuracy of 92% is definitely of practical significance (please see the discussion in the *Justification* section). Considering high resemblance between many species and high variation within species, I thought it would be really challenging to come up with a reasonable model. But the final model did a fairly good job. What made the difference is transfer learning, which improved the model performance to a great extent. This kind of modeling could be used in other problems of Agriculture very effectively. For instance, farmers find it really difficult to correctly identify certain diseases and pests. We can train CNN to classify plant diseases and pests.

One of the great challenges I faced in this project was limited computational power of my laptop. To run my entire notebook, it would take around 12 hours. During this 12 hours, my laptop would freeze many times that I could not use it for any other purpose. The most computationally demanding tasks in this project was preprocessing of images and extracting bottleneck features. So ultimately I used Amazon Web Services (AWS). On an EC2 instance of AWS, my notebook took only 3 hours to run.

Improvement

The CNN model I built could achieve an accuracy of 92% on the test set. While it is of practical significance, it is desirable to further improve the prediction ability of this model. The more accurately you identify the weeds, the more effective your weed management will be. So any increase in the accuracy of this model will help to reduce the yield loss proportionately.

We may try to further improve the performance of this model by;

1. Further tuning hyper-parameters like batch size, optimizers, metrics, and loss function
2. Using other kind of pooling layer (e.g. GlobalMaxPooling2D)
3. Adding a few more fully connected dense layers.
4. expanding training and validation set by image augmentation, which may help model generalize to unseen data better. It should be noted that in real world situations, images may not be taken in the exactly same way as images taken for this dataset. Augmentation helps algorithm to learn invariant representation of an image, so that size (scale invariance), position (translation invariance) or angle (rotation invariance) do not have much effect on prediction accuracy. For instance, to achieve rotation invariance, to the training set we can add some images created by randomly rotating training images.

References

1. http://www.agriinfo.in/default.aspx?page=topic&superid=1&topicid=2173
2. Chandler JM, Hamill AS, Thomas AG (1984) Crop losses due to weeds in Canada and the United States. WSSA special publication, Champaign,
3. Weeding Out; https://www.geospatialworld.net/article/weeding-out/
4. Giselsson TM, Jørgensen RN, Jensen PK, Dyrmann M, Midtiby HS (2017) A Public Image Database for Benchmark of Plant Seedling Classification Algorithms. CoRR abs/1711.05458
5. https://pnwhandbooks.org/plantdisease/host-disease/wheat-triticum-aestivum-fusarium-root-crown-foot-rot-crown-rot-foot-rot-seedling-blight-dryland-foot
6. https://stock.adobe.com/ee/images/pea-seedling-isolated-on-white/29084496
7. https://datascience.stackexchange.com/questions/15903/why-do-convolutional-neural-networks-work
8. Udacity Lessons; Machine Learning Engineer Nanodegree - Part 5. Deep Learning - Lesson 4. Convolutional Neural Networks (https://classroom.udacity.com/nanodegrees/nd009t/parts/0ac87c1d-350a-417b-93c8-392dbf9cb8c2/modules/f5e5f204-ae46-415c-bd9b-a160eecf044f/lessons/52fc79a7-13ff-4065-b3c6-8203ec9ef60c/concepts/603cdafb-4378-4f13-898a-e7efe44464ff)
9. http://cs231n.github.io/convolutional-networks/#conv
10. https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
11. https://www.quora.com/Why-does-a-depth-wise-separable-convolution-model-like-Keras-Xception-perform-exceptionally-well-compared-to-GoogleNet-Inception-or-any-other-TL-models
12. https://stats.stackexchange.com/questions/226923/why-do-we-use-relu-in-neural-networks-and-how-do-we-use-it
13. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/