Machine Learning Engineer Nanodegree

Capstone Project

Joel Newswanger

December 13, 2017

Definition

Project overview

One of the major challenges faced by agriculture is weeds. Weeds compete with crop species for nutrients, sunlight, and water, and often out-compete crops for these, due to their greater growth rate. Since they are a drain in the production of crops the presence of weeds greatly decreases the efficiency of agriculture economically and environmentally.

Crops are especially vulnerable to weed encroachment when they are seedlings. This is because once a plant is grown it will shade the ground and take up the water and nutrients, making it harder for other plants to grow in the same area. This means that whichever plant grows larger first will dominate, and as before mentioned, the weeds grow faster than crops. The otherwise empty field full of seedlings is a prime place for weeds to spring up and overtake.

Current methods for dealing with weeds in fields that have crop sprouted in them are full of problems. To be able to spray the weeds with herbicide, which is the conventionally preferred way, the field must either not have any crop in it yet, or be planted with an herbicide resistant crop. Some people have problems with such crops, overuse of pesticides causes environmental problems, and weeds can become herbicide resistant. Manual removal of weeds from amongst crop seedlings is very difficult and can be not great for the soil too.

A proposed solution includes robotic weed exterminators (not a technical term) that roam the fields, finding and killing weeds. These robots would need to use camera images to visually identify and classify weed and crop seedlings, roast the weeds with a laser, and pass harmlessly over the seedlings. The laser works well because heating a plant to a high temperature is a reliable way to kill it to the roots, the water in it conducts the heat throughout the whole plant. Precision sprays of herbicide, and manual extraction are also possibilities for different applications.

The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has recently released a dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages as a Kaggle Competition. The species are 3 common crops and 9 common weeds as seedlings. I will use this data to train and test my multi-class classification CNN.

Problem Statement

These robots will need to be able to use multi-class classification to identify and classify crop and weed seedlings from camera images in order to know which ones to protect and which to unleash their destructive capabilities on. For this the designers look to machine learning and computer vision. Many CNNs have proved very accurate in multi-class image classification problems. Since the training data set available for pictures of seedlings is rather small, a good and compute-time efficient way to make a CNN for it is to use a pre-trained very large CNN like inceptionResNetv2. The final, fully connected layers will be replaced with ones trained on the seedling data, and the upper layers will be fine-tuned to the data.

Metrics

While training, my model will optimize based on the categorical crossentropy attribute of the callback function. To evaluate models' performances the mean f-score will be used. The mean f-score is a good choice because it shows the balance of the recall and the accuracy, to give a more complete look at how the model will perform on unseen data.

Analysis

Data exploration

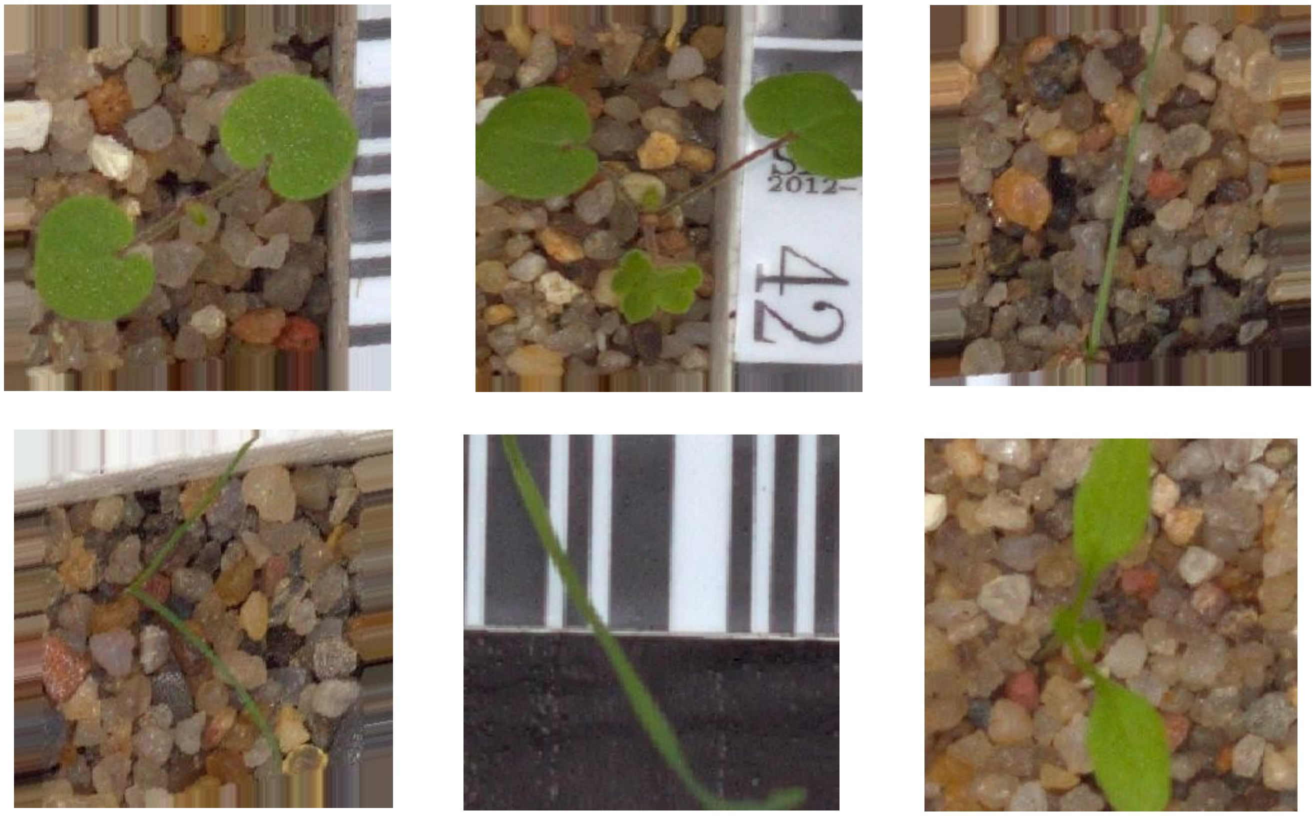
The dataset contains images of approximately 960 unique plants belonging to 12 species at several growth stages. The species are 3 common crops and 9 common weeds, and there are between 221 and 611 images in the training/validation set for each species. 221 images were randomly selected from each species, and loaded as train/validation samples. 80% of the train/validation samples were randomly selected as training data and the other 20% were used as validation data. The original images were of varying dimensions, but were all resized to 299*299, and the pixel values were divided by 255 when the images were loaded as tensors.

Exploratory visualization

Report on the contents of the training dataset

```
There are 263 pictures of Black-grass
There are 390 pictures of Charlock
There are 287 pictures of Cleavers
There are 611 pictures of Common Chickweed
There are 221 pictures of Common wheat
There are 475 pictures of Fat Hen
There are 654 pictures of Loose Silky-bent
There are 221 pictures of Maize
There are 516 pictures of Scentless Mayweed
There are 231 pictures of Shepherds Purse
There are 496 pictures of Small-flowered Cranesbill
There are 385 pictures of Sugar beet
```

Images after augmentation.



The algorithm will see the images at different rotations and levels of zoom, etc. every time it sees them so it will learn important features rather than individual images.

Algorithms and Techniques

A baseline CNN model will be constructed using Conv2D, maxPooling, Flatten, and Dense layers from keras. The final models will be created from the inceptionResNetv2 CNN with pre-trained imagenet weights, and a globalAveragePooling and Dense layer.

These models have been chosen because CNNs like inceptionResNetv2 have demonstrated state of the art accuracy in image classification. Convolutional layers work well by moving a window of chosen size back and forth across the entire 2D array, making a node for each set of pixels. This process returns a mask showing the parts of the image where pixels match the feature that that node represents. This method retains the spatial relationships of the image, and allows patterns to be picked out. As you add more convolutional layers, they can piece the patterns into more and more complex shapes. MaxPooling layers are used to reduce the dimensionality of the data as the number of features increases by convolution. They accomplish this by moving a similar window across the array, and taking the max value from it, so if the window is 2*2 you essentially halve the dimensions of the array. Flatten makes the data 1 dimensional so that it can enter a fully connected layer. Dropout layers help to reduce over fitting by turning off a certain percentage of the nodes in the preceding layer to limit the amount that can be learned. Dense layers have every node connected to every node in the layers before and after them and are the main building block of neural networks.

Because the dataset is rather small, a pre-trained CNN can provide more efficient learning. These CNNs are very deep, with hundreds of convolutional layers, which enables recognition of many features and objects. InceptionResNetv2 was chosen because it has shown the highest accuracy out of all the available pre-trained CNNs in keras, even though it's great depth means more computation time.

In addition, image augmentation, and fine-tuning techniques were used to increase the accuracy of the results. Image augmentation was implemented using keras.preprocessing.image.ImageDataGenerator. Image augmentation makes minor changes to batches of the training data so that rather than just memorizing certain images, the model actually learns features of the images that will help it recognize other images of the same thing. Fine-tuning re-trains the weights of the CNN using a very low learning rate, so that it can learn features more relevant to the current dataset.

Benchmark

The baseline scratch-built CNN was used as a benchmark for the project. This is a good benchmark, because it proves that I am obtaining better results using the pre-trained model than is possible otherwise. The benchmark achieved a mean f-score of 0.801 on test data.

Methodology

Data Preprocessing

221 images were randomly selected from each species, and loaded as train/validation samples. 80% of the train/validation samples were randomly selected as training data and the other 20% were used as validation data. There were 2103 training images and 549 validation images, the test dataset included 794 unlabeled images. All images were resized to 299*299, and the pixel values were divided by 255 when the images were loaded as tensors.

Keras.preprocessing.image.ImageDataGenerator was used to implement image augmentation. Shear, Zoom, and horizontal flip were used to augment the training images, while the validation and test images were unaugmented.

Implementation

The scratch-built baseline CNN was build of three layers of Conv2D and maxPooling, the first with 16 nodes, second with 32, and final with 64, followed by a flatten, a Dense layer with 12 nodes, and a softmax activation. All convolutional layers used relu activation, 3*3 windows, and same padding. The rmsprop optimizer, and categorical crossentropy were used when compiling the model. Model checkpoint was used to save only the weights with the lowest categorical crossentropy. The model was trained for 20 epochs on the data augmentation generator. The original intent was for the baseline model to not use image augmentation, but without image augmentation the model did not exceed 50% accuracy. Once image augmentation was implemented it achieved 75% validation accuracy, which was a better benchmark. This model went smoothly, because Keras provides exceptional ease of use, and good documentation.

Next the bottleneck features were extracted from inceptionResNetv2, and a fully connected model was trained on them. Because of using bottleneck features image augmentation could not be used. I changed this model to use GlobalAveragePooling, because Flatten consistently gave poor < 20% accuracy results, and Pooling easily gave > 80%. After much experimentation with fully connected layers, the best configuration was two Dense layers with 122 and 12 nodes separated by a 50% dropout layer. This model used tanh activation on the first dense layer and a final softmax activation. After training for 25 epochs this model achieved 82% validation accuracy and a mean f-score of 0.84 on the test data.

Another model implementing fine-tuning and able to use image augmentation was created using the inceptionResNetv2 CNN pre-trained on imagenet. This model trained the last 10 convolutional groups while keeping the first 618 layers frozen. It used a globalAveragePooling and two Dense layers with 122 and 12 nodes like the previous model, but excluded the dropout layer. The fully connected layers were trained first with the whole CNN frozen for 5 epochs, then the last 10 convolutional blocks were fine tuned for 12 epochs with a SGD optimizer with a learning rate of .0001. The model trained very slowly, partially because I loaded all the data into memory, which was unnecessary because I was using a data generator which could import batches from directory. This model achieved validation accuracy of 88.6 %, and a mean f-score of 0.8942 on the test data. Another model was attempted with fine-tuning of the last 30 convolutional blocks, but it trained unbearably slowly and failed to produce a better mean f-score.

Refinement

My initial scores with the CNNs were around 15% accuracy until I stopped using the Flatten in favor of GlobalAveragePooling, which raise the scores into the 70s-80s. By experimenting with one to three dense layers I settled on two. I experimented with between 16 and 1200 nodes in the first Dense layer, and found that 122 gave the most information without overfitting the model. Adding a dropout layer helped to avoid overfitting as well. After much experimentation, tanh and relu activations seemed to work pretty much equally well on my models. When fine-tuning I found that trying to fine-tune too many layers just slows the process down without improving results.

Results

Model evaluation and validation

The final model was derived by implementing image augmentation and fine-tuning of a moderate number of convolutional layers, on inceptionResNetv2 pre-trained on imagenet. This model was chosen because it achieved the greatest accuracy, and mean f-score. When tested on the "in the wild" data, none of my models did better than 15% accuracy, so while the model generalizes well to unseen test data that is as clean and controlled as the training data, this model would not be sufficient for actually controlling a weeding robot, without being trained on more realistic data. The model can be trusted to identify laboratory quality photos of seedlings, but not lower quality photos.

Justification

The final model significantly improved upon the benchmark that I set, from 80% to 90%, by utilizing transfer learning and fine-tuning. In the application to weeding robots 10% misidentification could be bad for profits. Luckily, the robot only really needs to know if the plant is a weed or not. If the accuracy of the final model is judged based on whether it chose correctly weed or not weed, allowing confusion between weed species, then it scores 98.2% accuracy. Only 1.8% of the time will the robot allow a weed to live or kill a crop seedling. This is not perfect, but it is actually probably still an improvement over the current situation without the robot helper, so I would contest that the algorithm solves the stated problem, but further work is needed to perfect it.

Conclusion

Free-Form visualization

The output from the Weed/Not Weed accuracy test shows that Sugar beet is the hardest to distinguish from weeds. 7 out of 9 Weed/Not Weed errors were made on Sugar beets.

```
Mis-identified  Black-grass  as  Sugar beet .

Mis-identified  Black-grass  as  Common wheat .

Mis-identified  Cleavers  as  Sugar beet .

Mis-identified  Common wheat  as  Common Chickweed .

Mis-identified  Fat Hen  as  Sugar beet .

Mis-identified  Sugar beet  as  Common Chickweed .

Mis-identified  Sugar beet  as  Fat Hen .

Mis-identified  Sugar beet  as  Fat Hen .

Mis-identified  Sugar beet  as  Common Chickweed .
```

```
Guessed  0.9829867674858223 % correct.
```

A Sugar Beet seedling

A Common Chickweed Seedling

A Fat Hen Seedling

```
   These three species of seedlings look very similar and the final model has
most difficulty telling Sugar Beets from weeds, especially Chickweed and Fat
Hen.
```

Reflection

For this project, I set out to create a CNN that could accurately classify plant seedlings as Weeds or Crops. I treated it as a multi-class classification problem, of finding the species of ever seedling. I created a scratch-built CNN with image augmentation that achieved 80% accuracy, as a baseline. I used bottleneck features from inceptionResNetv2 trained on imagenet to train a fully connected model that achieved 84% accuracy. Finally, I fine-tuned the top ten convolutional groups of inceptionResNetv2, using image augmentation, to create a model that achieved 90% accuracy. I tested the final model against real-world data, and on its performance on Weed/Not weed classification to see how well it met the stated problem.

Some interesting developments were that GlobalAveragePooling2Ds worked better than Flatten on my transfer learning models, that fine-tuning more layers of the CNN did not improve results, and that my models did very poorly with real world data. The hardest part of the project was getting the data imported and preprocessed. I later learned that with flow_from_directory I didn't even have to load all the data into memory like I did. The rest of the project just required a lot of experimentation and computing time.

The final model fits my expectation for the project in all ways, except that it cannot without some further work handle real world data. I had hoped that It would do well on the real world test data. If limited to laboratory data, it can perform satisfactorily at the needed tasks. I think that training it on the sorts of field setting images that it would be working with would lead to better "in the wild" results.

Improvement

I think that 90% accuracy leaves a lot of room for improvement. This could be accomplished by using the bagged average of a number of models. Hopefully the other models wouldn’t struggle with sugar beets as much as the first one and help it make the right choice.