Deep learning for Plant Seedlings Classification

Project Overview

Weed control, which directly affect the efficiency of agriculture, has been researched for several decades. Keeping crop free of weeds is critical in good farm management. Farmers should plan their farm activities well so that they do weeding at the right time and in the right way. For this purpose, to differentiate weed from crop seedling is the essential step. Different seedlings have their unique features on their colors, shape, and outlines. Multiple technologies have been adopted including GPS mapping, ground-based vision identification, remote sensors and so on.

This project focuses on to identify and classify the species of a seedling from an image(as shown on Figure1) using machine learning based computer vision. Fortunately Aarhus University and University of Southern Denmark recently released a dataset[1] containing images of approximately 960 unique plants belonging to 12 species at several growth stages. those labeled data provides the researchers an opportunity to experiment with different image recognition techniques. BesIdes, Kaggle hosts a competition[2] based on the proposed dataset and provides participants a place to communicate and compare different solutions.

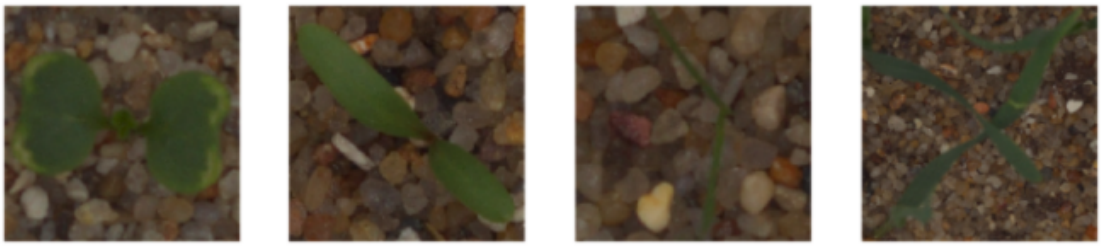


Figure 1: Example images taken from the database

Problem Statement

The topic of this project is to use provided dataset training weed recognition algorithms. The most critical part is to select and adjust the architecture of the model together with its hyper-paramaters to push the limits of the accuracy. To get inspiration from previous research results, transform learning becomes a good start point. The dataset from AU & SDU consists of different seeding images and labeled classification will be used for training and testing separately. The performance will be evaluated on MeanFScore, which at Kaggle is a micro-averaged F1-score.

Considering the good records of Conventional Neural Network at visual recognition challenge ILSVRC[3], A CNN based model is picked as a benchmark. It's architecture presents at figure 2 Benchmark Model. An expected workflow for this project will be

- Data prepare (Download and analyze the data)
- Pre-processing (Image resize, data normalization and data augmentation)
- Implement CNN model
- Iteration of:
  - Model training and validation (Use tensorbord for training visualization)
  - Parameter tuning and architecture adjustment
- Model test

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 48, 48, 3)	0
conv2d_1 (Conv2D)	(None, 46, 46, 16)	448
batch_normalization_1 (Batch	(None, 46, 46, 16)	64
activation_1 (Activation)	(None, 46, 46, 16)	0
conv2d_2 (Conv2D)	(None, 44, 44, 16)	2320
batch_normalization_2 (Batch	(None, 44, 44, 16)	64
activation_2 (Activation)	(None, 44, 44, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 22, 22, 16)	0
conv2d_3 (Conv2D)	(None, 20, 20, 32)	4640
batch_normalization_3 (Batch	(None, 20, 20, 32)	128
activation_3 (Activation)	(None, 20, 20, 32)	0
conv2d_4 (Conv2D)	(None, 18, 18, 32)	9248
batch_normalization_4 (Batch	(None, 18, 18, 32)	128
activation_4 (Activation)	(None, 18, 18, 32)	0
global_max_pooling2d_1 (Glob	(None, 32)	0
dense_1 (Dense)	(None, 64)	2112
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 32)	2080
dropout_2 (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 12)	396

Figure 2: Benchmark Model

Metrics

Solution is evaluated on MeanFScore (micro-averaged F1-score). Given the topic of 12 species classification, The competition host would prefer to both considering the recall (How many seedlings which are added to the correct species category, out of all seedlings of this species) and precision (How many seedlings in the classification output(the predicted species is correct, out of all the ones that added to this category). Besides, the traditional performance measures like accuracy, precision, recall etc may not be very useful when dealing with highly skewed data. because the numbers you may get may not be true representative of the actual performance. For e.g. you have 990 class A and 10 class B and if you classify everything as class A, you still get 99% accuracy which is wrong here[11]. Figure 4 shows a skew distribution.

Given positive/negative rates for each class k, the resulting score is computed this way:

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

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

F1-score is the harmonic mean of precision and recall

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

Data Exploration

Datasets are divided by a training set and a test set of images of plant seedlings at various stages of grown. It comprises annotated RGB images with a physical resolution of roughly 10 pixels per mm. Each image has a filename that is its unique id. The dataset comprises 12 plant species. Samples of each species from the database[1]:

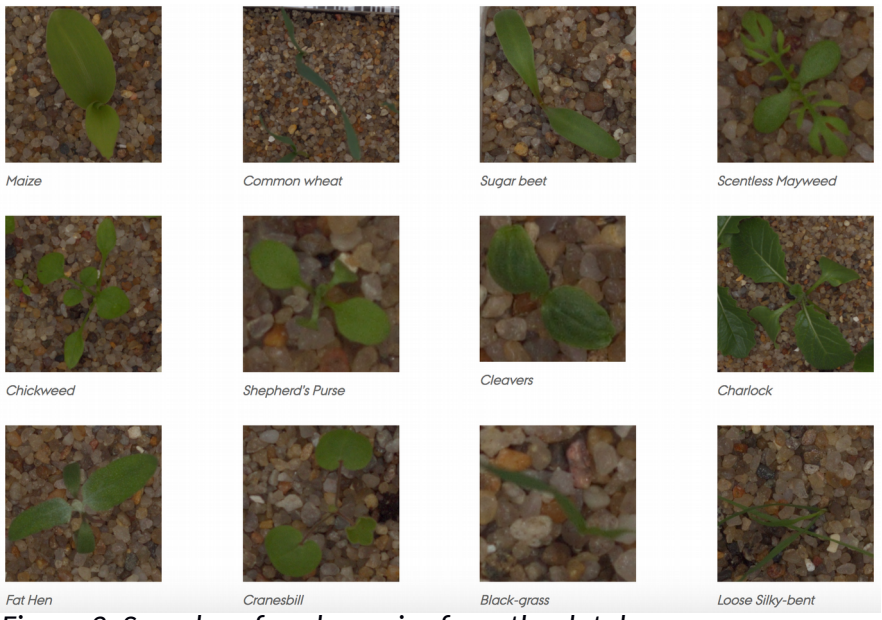


Figure 3: Samples of each species from the database

File descriptions
Train dataset: Images are all in .png format with RGB color space. Dimension is various from image to image. Images are stored in different folders with the name of its seedling category.
./train

- /Black-grass (263 images)
- /Charlock (390 images)
- /Cleavers (287 images)
- /Common Chickweed (611 images)
- /Common wheat (221 images)
- /Fat Hen (475 images)
- /Loose Silky-bent (654 images)
- /Maize (221 images)
- /Scentless Mayweed (516 images)
- /Shepherds Purse (231 images)
- /Small-flowered Cranesbill (496 images)
- /Sugar beet (385 images)

Test dataset: All .png images are put in one folder.

./test
795 images

train.csv : The training set, with plant species organized by folder
test.csv : The test set, need to predict the species of each image

sample_submission.csv : A sample submission file in the correct format

Considering it is a small dataset, all examples of the dataset and data augmentation will be used at this project to support training, validating and testing. Besides we also note there are some images are blurry I attached an example(Black-grass) here:

Exploratory Visualization

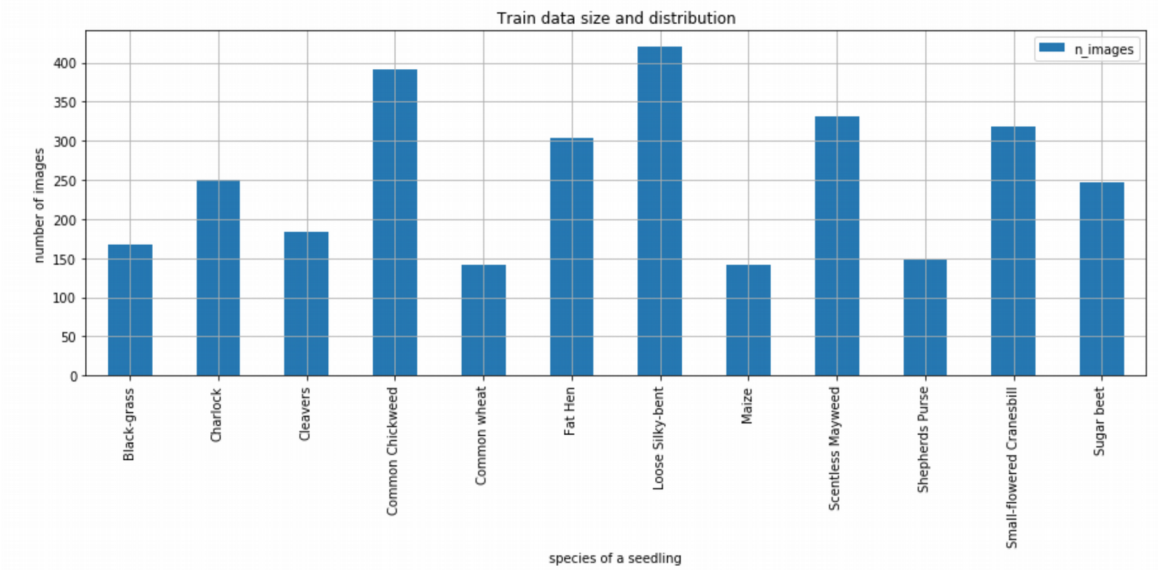


Figure 4: Visualization of sample size and distribution

In this section, the investigation result of the training data is presented. Figure 4 shows the visualization of sample size and distribution. Given the distribution, different seedling species provide different amount of sample images. Loose silky-bent has the largest sample amount which is around 450 pics. Common wheat and Maize have very small amount of images (less than 150 pics). The overall small amount of training dataset and its huge difference on distribution indicate the need of using data augmentation. Figure 5 shows the information of training images by output 20 sample images. The size of images various from image to image. No clear relation between the image size and its seedling category. Figure 6 presents the distribution of height variable. The majority of images are smaller then 1000 pixels.

Given the information of Figure 5, The height/width ratio of the sample images are all 1:1. To check if it is a common feature to all training data. A visualization had been implemented by calculating the ratio of height/width for all images. Figure7 presents the distribution of the ratio for all training images. So the majority of images have the 1:1 ratio. But there are still a few of images has different size ratio.

	class	height	width
9a8531ba0.png	Fat Hen	178	178
88ea1ed2a.png	Fat Hen	706	706
2befc73e1.png	Scentless Mayweed	563	563
07bed57ea.png	Scentless Mayweed	534	534
6850ccd12.png	Loose Silky-bent	153	153
a092428cd.png	Small-flowered Cranesbill	578	578
5139b104e.png	Common Chickweed	163	163
fd3e62689.png	Charlock	737	737
0b26e2d09.png	Small-flowered Cranesbill	163	163
8e69ea8a0.png	Scentless Mayweed	587	587
0f47f8a00.png	Loose Silky-bent	124	124
85b23f3e6.png	Cleavers	382	382
c3b34e88c.png	Loose Silky-bent	407	407
a300eb8b2.png	Common wheat	79	79
18fe42109.png	Common wheat	893	893
eace614b4.png	Fat Hen	342	342
997ffd9ae.png	Loose Silky-bent	186	186
82561432a.png	Common Chickweed	378	378
27d08b6f9.png	Common Chickweed	214	214
5e10b3707.png	Sugar beet	633	633

Figure 5: Image information

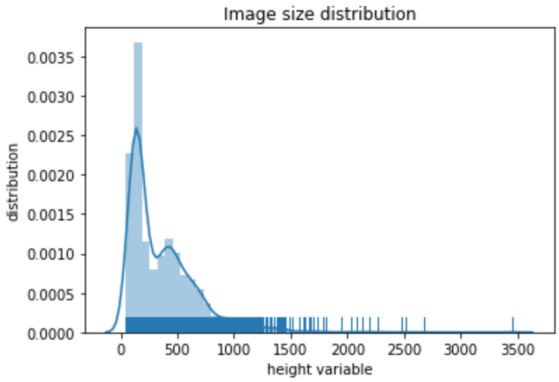


Figure 6: Distribution of height variable

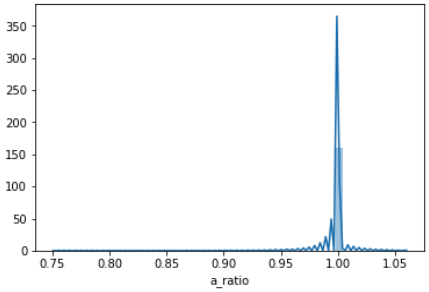


Figure 7: High/Width ratio of training images

Algorithms and Techniques

By evaluating the winner architectures of visual recognition challenge ILSVRC[3] on ImageNet dataset, Convolutional Neural Network is widely adopted at top performance algorithms. Given its good records, CNN is chosen on the proposed project. CNN is very similar to ordinary Neural Networks which is made of neurons. Each neuron calculate the output based on the received inputs, weights, biases and activation function. The difference is that CNN explicitly assume the inputs are images. These makes the forward function more efficient and reduce the parameters number of the model. In general, CNN model consists of a sequence of layers, which are transform one volume of activations to

the next. Three types of layers are commonly used to build the CNN model. There are convolutional layer, pooling layer and full connected layer[9].

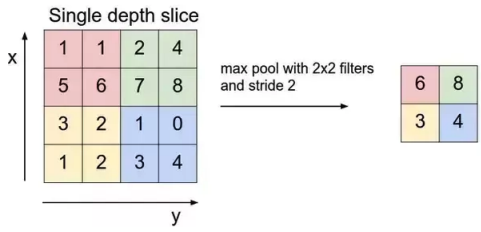
Convolutional layer:
Convolutional layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Pooling Layer:
Pooling layer will perform a downsampling operation along the spatial dimensions (width, height).

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

Attached is an example of Maxpooing (output he max value of each stride in the range of kernel size):



Dropout layer:
Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. More specifically, At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Dropout is a technique used to improve over-fit on neural networks.

In this case, the input of CNN are pixel values of image. For example, the benchmark model uses 48x48 pixels images with RGB color channels. Convolutional layer will calculate the output of neurons that are connected to the regions of the input. Users can define the number of filters, kernel size, stride length, activation function, padding, etc for each convolutional layer. Pooling layer will apply downsampling operation to a convert the input dimension to a pre-defined dimension. Full connected layer will perform as a classifier by compute the scores based on the inputs from previous volume.

Besides the above traditional types of layers for CNN architecture, Normalization layer will also be adopted in this case. The purpose of this layer is to normalize the activations of the previous layer at each batch. Using normalization layers between convolutional layers and nonlinearities, such as ReLU layers, can speed up training of convolutional neural networks and reduce the sensitivity to network initialization.

Benchmark

A CNN model is picked as a benchmark. Its architecture presents at Figure 2. The benchmark model is built by Keras using TensorFlow backend. The input size is 48x48 pixels. Batch size is 64. Train the benchmark model with the train datasets.

The training progress is shown on Figure 8 and Figure 9 logged by tensorboard. Horizontal axis represents step. The accuracy on training is 0.549. The loss on training is 1.347. The accuracy on validation is 0.6821 (Figure 10). The loss on validation is 1.044 (figure 11). Test the model with the test datasets. The **MeanFScore** of the benchmark model is **0.67632.**

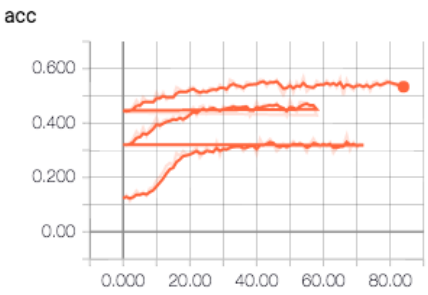


Figure 9: Accuracy on training

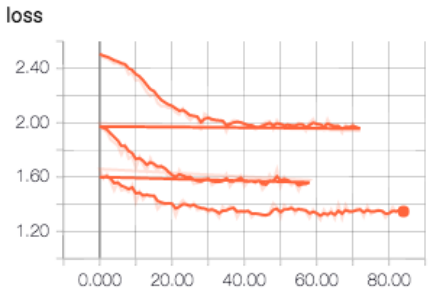


Figure 8: Loss on training

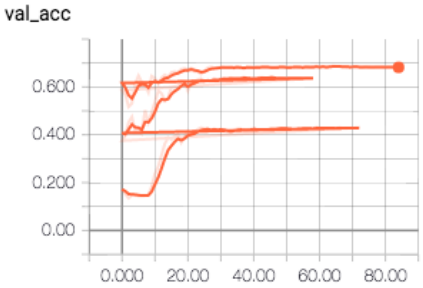


Figure 11: Accuracy on validation

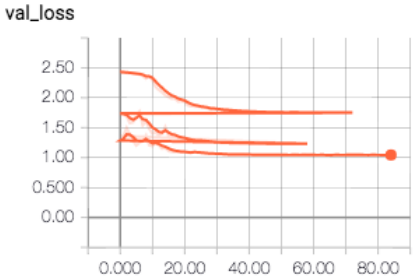


Figure 10: Loss on validation

Methodology

Data Preprocessing

As mentioned at section data exploration, the training datasets needs to be processed before the training step. A serial of image processing has been implemented in this case.
They are

- Image resize, to provide the input of Model with same dimension. (started from 48 x 48 pixels, then changed to 128 x128 pixels which shown better performance)
- Data augmentation, to increase the size of training datasets and to enhance the diversity. (**keras.preprocessing.image.ImageDataGenerator** was used to generate images for training. Those images were generated based on original training dataset and image adjustments which includes rotation, shift, zoom and flip.)
- Shuffle, to avoid providing training data from category to category in sequence.
- Data normalization, to map the RGB value from 0~255 to the range of 0~1, which is a sensitive range for activation function.
- One-hot encode, to encode the seedling categories. (**ImageDataGenerator.flow_from_directory** was used to convert the seedling categories to 2D one-hot encoded labels)

Implementation

The CNN model is implemented by keras. Keras.layers was used to create input layer, conv2d layer, GlobalMaxPooling layer, batch normalization layer, and dropout layer. Loss was defined as categorical_crossentropy. Adam was adopted as the optimizer. Based on the architecture of the benchmark model(as shown on Figure 2), Keras.model was used to

integrate all layers and the optimizer. Total params of the benchmark model is 21,628, of which trainable params is 21,436 and Non-trainable params is 192.

Model.fit_generator was used to train the CNN model with the **batch size** defined as 64 and **epochs** defined as 200. keras.callbacks.LearningRateScheduler was used to adjust the learning rate along the process of training (epoch index). The **learning rate** was set to 0.001*0.9^x, shown as Figure 11. **Dropout rate** was set to 0.5. Activation function was set to **ReLU.** Optimizer was set to **Adam**. More details of the benchmark model implementation can be found at the file: Benchmark_CNN_keras_model_V1-2.ipynb.

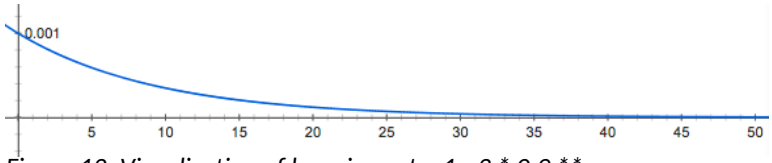


*Figure 12: Visualization of learning rate: 1e-3 * 0.9 ** x*

The challenging part I met during the coding was to predict the test dataset with a static order that the Kaggle can recognize. I was changed the method from predict_generator() to predict(), the method used at Benchmark_CNN_keras_model_V1-2.ipynb. The correct mapping is important. Because I need to upload the predicted result to the web to see the score.

Refinement

In this section, the process of improvement will be presented. The improvements are mainly derived from architecture changes and parameter adjustments. As mentioned in the section Benchmark model, the original score is **0.67632.** After iteration of adjustments, the best test result shows a score at **0.95969.**

Refine 1.
Parameter adjustment: Change input dimension from 48x48 to 128x128. Change batch size from 64 to 400.
Architecture change: increase the filter numbers and increase the depth of the Model.
The training process was logged by tensorboard.
The name of this model is “Model_dim128_deep”. The test result is **0.81360**

Refine 2.
Architecture change: Inspired by NVIDIA CNN model[4], Using the strides = 2(subsampling) to replace the pooling layers between the conversational layer. BatchNormalization layer was proved in this case to improve the performance by comparing the model with and without BN layers. The new architecture is presented as Figure 13.
The name of this model is “Nvidia_bn”. The test result is **0.92821.**

Refine 3.
Architecture change: Inspired by [5],[6] and the ResNet-50. Global Average Pooling layer was used to perform the dimensionality reduction and minimize the overfitting. A combination of GAP and one densely connected layer with a softmax activation function was adopted to replace the FC layers to predict the seedling categories, as shown on Figure 14.
Parameter adjustment: Kernel size of all conversational layer is set to 3x3.
The name of this model is “Nvidia_bn_gap_33“. The test result is **0.93541**

Refine 4.
Parameter adjustment: Increase the batch size to 800. (The test result is **0.95088**)
Architecture change: Increase the depth of the CNN, as shown on Figure 15.
The name of this model is “Nvidia_bn_gap_33_batch800_deep”. The test result is **0.95969 (final solution)**

Figure 16 shows the accuracy on validation for each of the above models. Figure 17 shows the corresponding accuracy during training process. Please note, the test results were described in MeanFScore calculated by Kaggle. However Figure 16, Figure 17 can only shows the result in accuracy, because the project was implemented on Keras 2.0, which

removed the “fbeta_score”, “precision”, and “recall” metric function[10]. So we chose to use “accuracy”, while data augmentation was made at data pre-processing.

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 64, 64, 24)	1824
batch_normalization_5 (Batch	(None, 64, 64, 24)	96
activation_5 (Activation)	(None, 64, 64, 24)	0
conv2d_6 (Conv2D)	(None, 32, 32, 36)	21636
batch_normalization_6 (Batch	(None, 32, 32, 36)	144
activation_6 (Activation)	(None, 32, 32, 36)	0
conv2d_7 (Conv2D)	(None, 16, 16, 48)	43248
batch_normalization_7 (Batch	(None, 16, 16, 48)	192
activation_7 (Activation)	(None, 16, 16, 48)	0
conv2d_8 (Conv2D)	(None, 8, 8, 64)	27712
batch_normalization_8 (Batch	(None, 8, 8, 64)	256
activation_8 (Activation)	(None, 8, 8, 64)	0
conv2d_9 (Conv2D)	(None, 4, 4, 64)	36928
batch_normalization_9 (Batch	(None, 4, 4, 64)	256
flatten_1 (Flatten)	(None, 1024)	0
activation_9 (Activation)	(None, 1024)	0
dense_4 (Dense)	(None, 100)	102500
activation_10 (Activation)	(None, 100)	0
dense_5 (Dense)	(None, 50)	5050
activation_11 (Activation)	(None, 50)	0
dense_6 (Dense)	(None, 12)	612

Total params: 240,454
Trainable params: 239,982
Non-trainable params: 472

Figure 13: BatchNormalization layer added Nvidia based CNN Model

Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 128, 128, 24)	672
batch_normalization_16 (Batc	(None, 128, 128, 24)	96
activation_18 (Activation)	(None, 128, 128, 24)	0
conv2d_17 (Conv2D)	(None, 64, 64, 24)	5208
batch_normalization_17 (Batc	(None, 64, 64, 24)	96
activation_19 (Activation)	(None, 64, 64, 24)	0
conv2d_18 (Conv2D)	(None, 64, 64, 36)	7812
batch_normalization_18 (Batc	(None, 64, 64, 36)	144
activation_20 (Activation)	(None, 64, 64, 36)	0
conv2d_19 (Conv2D)	(None, 32, 32, 36)	11700
batch_normalization_19 (Batc	(None, 32, 32, 36)	144
activation_21 (Activation)	(None, 32, 32, 36)	0
conv2d_20 (Conv2D)	(None, 32, 32, 48)	15600
batch_normalization_20 (Batc	(None, 32, 32, 48)	192
activation_22 (Activation)	(None, 32, 32, 48)	0
conv2d_21 (Conv2D)	(None, 16, 16, 48)	20784
batch_normalization_21 (Batc	(None, 16, 16, 48)	192
activation_23 (Activation)	(None, 16, 16, 48)	0
conv2d_22 (Conv2D)	(None, 8, 8, 64)	27712
batch_normalization_22 (Batc	(None, 8, 8, 64)	256
activation_24 (Activation)	(None, 8, 8, 64)	0
conv2d_23 (Conv2D)	(None, 4, 4, 64)	36928
batch_normalization_23 (Batc	(None, 4, 4, 64)	256
activation_25 (Activation)	(None, 4, 4, 64)	0
conv2d_24 (Conv2D)	(None, 2, 2, 64)	36928
batch_normalization_24 (Batc	(None, 2, 2, 64)	256
activation_26 (Activation)	(None, 2, 2, 64)	0
global_average_pooling2d_2 (	(None, 64)	0
dense_8 (Dense)	(None, 12)	780

Total params: 165,756
Trainable params: 164,940
Non-trainable params: 816

Figure 14: Model architecture (Nvidia_bn_gap_33_batch800_deep)

activation_16 (Activation)	(None, 4, 4, 64)	0
conv2d_15 (Conv2D)	(None, 2, 2, 64)	36928
batch_normalization_15 (Batc	(None, 2, 2, 64)	256
activation_17 (Activation)	(None, 2, 2, 64)	0
global_average_pooling2d_1 (	(None, 64)	0
dense_7 (Dense)	(None, 12)	780

Figure 15: GAP layer is followed by one densely connected layer

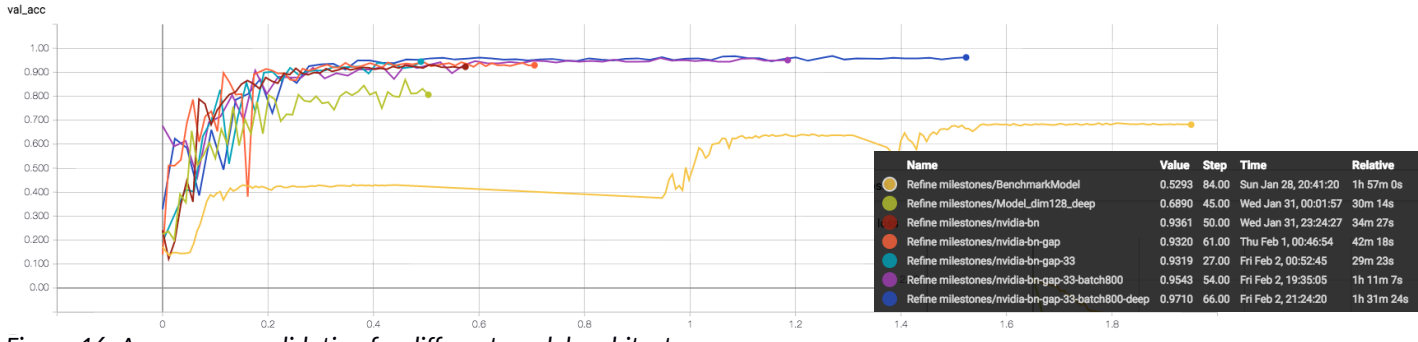


Figure 16: Accuracy on validation for different model architecture

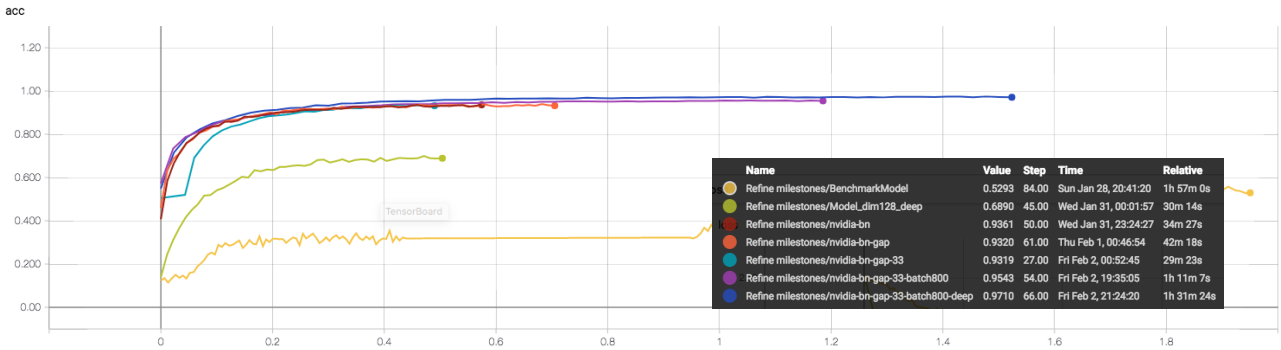


Figure 17: Accuracy on training for different model architectures

Results

Model Evaluation and Validation

The improvement of the model was implemented step by step, as shown on the following workflow:

- Data prepare
- Pre-processing
- Iteration of:
  - Model training and validation
  - Parameter tuning
- Model test

The target parameter or new architecture was adjusted by comparing with the performance of "non-changed" reference model implemented at the last iteration, as presented at Figure 17. The model has been tested on a new test dataset which has 795 images. The overfitting was well controlled. The following list shows the test result and training result along different model architectures:

	Test result	Training result
Model_dim128_deep	0.8136	0.6890
Nvidia_bn	0.9282	0.9310
Nvidia_bn_gap_33	0.9354	0.9319
Nvidia_bn_gap_33_batch800_deep	0.9597	0.9710

Training data was pre-processed to increase the size of dataset and data diversity. To increase the robust of the model, the training data(images) was randomly changed on its orientation and size.

The dimension of input images to the model is (128,128, 3). Inside the model there are 6 conv2d layers setting the subsample(stride size) = (2,2). This design reduces the dimension of the volume to (2,2,64) before entering to the Global Average Pooling(GAP) layer. GAP layer was put to even reduce the dimension to (1,1,64) and minimize the overfitting. Subsampling on those six conv2d layers also help to minimize the overfitting. BatchNormalization is critical in this case. The model without the BN layer could only reach an accuracy lower than 0.2. It help the model in two ways: faster learning and higher accuracy. Besides, I also noticed that a large batch size benefits the model overall performance.

To evaluate the robustness of the model in a much challenging situation, another image dataset provided by AU Signal Processing Group called "ImagesFromTheWild"[7] was used. Given this dataset, the final model was tested on weed images 'from the wild' with no artificial lighting. Unfortunately the accuracy on this test only

shows 0.177083. So I would suppose the final model is currently very sensitive on the background and illumination, which were not trained enough because of the limitation of training dataset. As a third reference, the validation dataset, which was split from the training dataset, was also used to evaluate the final model. The accuracy on validation dataset(756 images, 12 categories) is 0.960317.

ImagesFromTheWild has 96 images(.tiff) within 8 species. There are:

- Charlock 12 images
- Cleavers 12 images
- Common Chickweed 12 images
- Common wheat 12 images
- Fat Hen 12 images
- Scentless Mayweed 12 images
- Shepherds Purse 12 images
- Small-flowered Cranesbill 12 images

Figure 18 and Figure 19 are two examples in dataset ImagesFromTheWild:

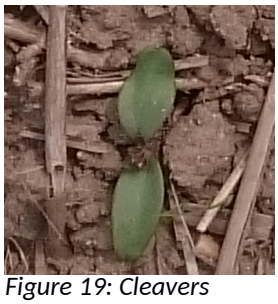


Figure 19: Cleavers

Figure 18: Common Wheat

Justification

A same test datasets was used to evaluate the benchmark model and the proposed final solution. Given the test result, The final solution reached the score at 0.95969 which outperformed the benchmark model with the score at 0.67632. The final solution was selected to use a deeper neural network with 165,756 parameters. However the benchmark model has a compact size with only 21628 parameters. Thus the forward processing speed of final solution is lower then the benchmark model. To use it in reality cases, real-time performance is critical, so more work need to be done to reduce the size of model while maintain the high accuracy in terms of classification. The limitation of training dataset needs also to be taken into consideration. For example the background of seedling images, they are all quite similar or close, the robustness of the solution in terms of dealing with different image background(soil or mixing with some other plants) still needs to be improved.

Conclusion

Free-Form Visualization

To verify the relation between the size and category, swarm plot is implemented as shown on Figure 20. As the plot presented, 12 species has been marked in different colors. Each point means a image. Images in each species are spreading at variety of sizes. The majority of images has the size in a range of 0~1000 pixels. The difference on the image size and its distribution of size at different category indicates the need of image-resize before the training.

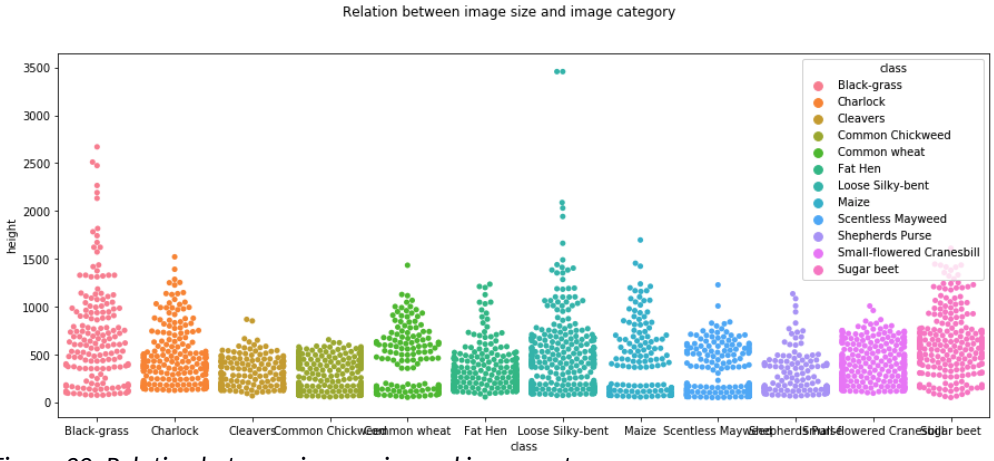
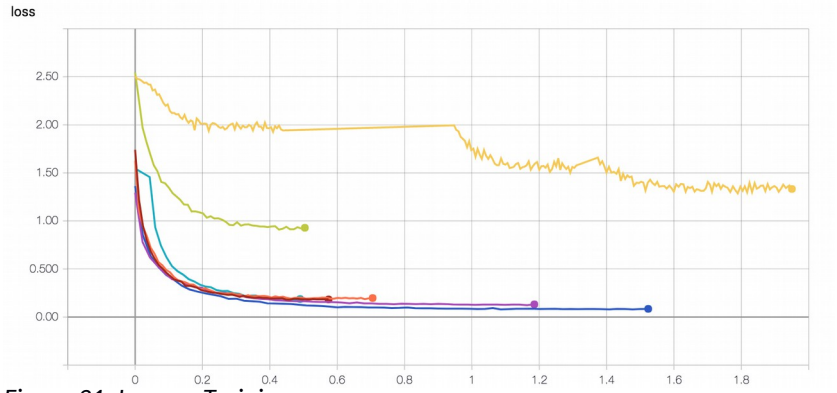
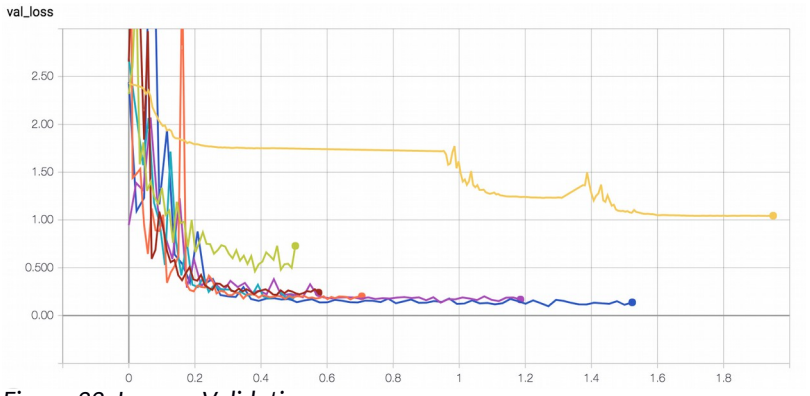


Figure 20: Relation between image size and image category



Name	Value	Step	Time	Relative
Refine milestones/BenchmarkModel	1.333	84.00	Sun Jan 28, 20:41:20	1h 57m 0s
Refine milestones/Model_dim128_deep	0.9290	45.00	Wed Jan 31, 00:01:57	30m 14s
Refine milestones/nvidia-bn	0.1830	50.00	Wed Jan 31, 23:24:27	34m 27s
Refine milestones/nvidia-bn-gap	0.1979	61.00	Thu Feb 1, 00:46:54	42m 18s
Refine milestones/nvidia-bn-gap-33	0.1874	27.00	Fri Feb 2, 00:52:45	29m 23s
Refine milestones/nvidia-bn-gap-33-batch800	0.1321	54.00	Fri Feb 2, 19:35:05	1h 11m 7s
Refine milestones/nvidia-bn-gap-33-batch800-deep	0.08524	66.00	Fri Feb 2, 21:24:20	1h 31m 24s

Figure 21: Loss on Training



Name	Value	Step	Time	Relative
Refine milestones/BenchmarkModel	1.045	84.00	Sun Jan 28, 20:41:20	1h 57m 0s
Refine milestones/Model_dim128_deep	0.7291	45.00	Wed Jan 31, 00:01:57	30m 14s
Refine milestones/nvidia-bn	0.2397	50.00	Wed Jan 31, 23:24:27	34m 27s
Refine milestones/nvidia-bn-gap	0.2012	61.00	Thu Feb 1, 00:46:54	42m 18s
Refine milestones/nvidia-bn-gap-33	0.1957	27.00	Fri Feb 2, 00:52:45	29m 23s
Refine milestones/nvidia-bn-gap-33-batch800	0.1680	54.00	Fri Feb 2, 19:35:05	1h 11m 7s
Refine milestones/nvidia-bn-gap-33-batch800-deep	0.1383	66.00	Fri Feb 2, 21:24:20	1h 31m 24s

Figure 22: Loss on Validation

Loss value implies how well or poorly a certain model behaves. The loss function's value is calculated on training and validation. It was recorded by tensorboard to present how well the model is doing for these two sets. Figure 21 shows the change of loss during the training process. Figure 22 shows the loss on validation. Different models are marked by different colors. The final model (nvidia-bn-gap-33-batch800-deep) got the lowest loss, which indicates its best classification ability among previous models.

Reflection

The initial challenge for me was to choose a benchmark model. There are many outperforming models like Xception and ResNet. However some of the architecture design presented in those model are beyond my understanding. It may provide a good performance, but leave very limited margin to improve or to adjust the architecture. At last I choose to implement a typical CNN model as a benchmark to use it to start.

Training dataset is small. Data preprocessing focuses on data augmentation and data normalization.

The most challenging is on parameter tuning. Given the feedbacks from Kaggle community[8], I started from increasing the parameter of CNN input dimension. Then I added more filters to the model. They all presented positive improvements. I got inspired from NVIDIA CNN architecture and use subsampling(stride size) to reduce the dimension of the volume instead of using MaxPooling lay. Inspired by ResNet to use GlobalAveragePooling and Fully connected layer with softmax activation to do classification. Then I tried to decrease the kernel size and increase the batch size to improve the Model to a better stage. At last I increased the depth of the model as shown at Figure14.

The third challenge is to evaluate the model. Because the test datasets was provided by Kaggle. To be fair on the competition, the labels of those data were not published. On the other hand I had used all training datasets for either training or validation. So, for a while I could not find suitable data for the model evaluation step. Fortunately, the original data provider-AU Data Processing Group published a stand-along data set called ImagesFromTheWild. By using those data to test the final model, the limitation of the model has been exposed. I can see the Model still need to be improved, although it got a good score at Kaggle competition.

Improvement

I was trying to adjust the Learning Rate. But the test result didn't reach the expectation. More investigation needs to be done to clarify the relation between the learning rate and the performance. Tuning parameters cost many time, it would be interesting and efficient if there are automatic methods to find the optimal combination of those parameters. The most critical improvement needs to be done is its robustness. To improve this, more datasets with different background, especially those in real environment with different illumination, need to be integrated. Besides we should always consider the size of the model and the precessing speed while moving forward on its performance.

Reference

[1] A Public Image Database for Benchmark of Plant Seedling Classification Algorithms
https://arxiv.org/abs/1711.05458
[2] Plant Seedlings Classification - Determine the species of a seedling from an image
https://www.kaggle.com/c/plant-seedlings-classification
[3] ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
http://www.image-net.org/challenges/LSVRC/
[4] End to End Learning for Self-Driving Cars
https://arxiv.org/pdf/1604.07316.pdf
[5] Network In Network
https://arxiv.org/pdf/1312.4400.pdf
[6] Global Average Pooling Layers for Object Localization
https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
[7] Image from the wild
https://vision.eng.au.dk/plant-seedlings-dataset/#images-from-the-wild
[8] Discussion-Plant Seedlings Classification
https://www.kaggle.com/c/plant-seedlings-classification/discussion
[9] CS231n Convolutional Neural Networks for Visual Recognition
http://cs231n.github.io/convolutional-networks/
[10] Keras 2.0 release notes
https://github.com/keras-team/keras/wiki/Keras-2.0-release-notes
[11]What metric do I use for learning curves with skewed classes?
https://www.quora.com/What-metric-do-I-use-for-learning-curves-with-skewed-classes