

Plant Seedling Classification

Capstone: Machine Learning Engineer Nanodegree

Manvindra Singh

January 31st, 2018

Project Report

1. Definition

Project Overview: Can you differentiate a weed from a crop seedling? Many weed seedlings can be mistaken for crop seedling as they look alike, it becomes hard to differentiate weed from crop. They can be distinguished when they become mature plant but till then soil quality and crop yield will be impacted. For example, Carrot shows relatively slow development in early growth stages. Hence, competing weeds may overtake the crop plant and limit its access to resources such as sunlight, moisture, and nutrient. Thus, weeds may cause major yield losses, if uncontrolled.

Comparing few seedlings:



Seedling Wild Turnip



Seedling Oil Seed Rape



Seedling Charlock

Wild Turnip can be mistaken for Charlock, which has more grass-green true leaves with many hairs. With Oil-seed Rape, which has almost smooth true leaves. Also, with Wild Radish, which has more grass-green true leaves with many hairs. In all crops in which this plant is found it is an injurious weed, which causes losses.

All the above weed looks alike to Shepherd's-purse a member of the same mustard family. Shepherd's purse has been used as a medicinal herb often recommended as a treatment for both internal and external bleeding.



Recently planted Shepherd's purse Plant

The ability to distinguish effectively can mean better crop yields and better stewardship of the environment. The goal of this project is to classify seedling images into its species. This problem is based on Kaggle Competition called "Plant Seedlings Classification". (<https://www.kaggle.com/c/plant-seedlings-classification>).

The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has released this dataset containing plants belonging to 12 species at several growth stages. I obtained the dataset through Kaggle which contains 4750 images in training set and 794 in test set. Number of images varies across species and the size of images varies from 49 X 49 to 3457 X 3991 but most images have aspect ratio of 1.0

Problem Statement

As said earlier, many seedlings of crops look alike another crops. Its difficult to distinguish them with naked eye even for experienced farmers. Each seedling can have different growth rate, some may become larger than other seedling in short time. Seedling from a family may resemble other plant seedling in different growth stage. Maybe 4 months old seedling of a plant resemble 2-month-old seedling of another plant. Attempt here would be to classify seedlings images into its plant species.

Below are few images from seedling for each species in dataset



Maize



Common wheat



Sugar beet



Scentless Mayweed



Chickweed



Shepherd's Purse



Cleavers



Charlock



Fat Hen



Cranesbill



Black-grass



Loose Silky-bent

Some of the plant seedlings are similar in appearance. Shepherd's purse is used as herbal medicine whereas it looks alike charlock is a common weed of cornfields.

Goal is to classify images of seedlings to its species. This make it a computer vision challenge. Deep learning techniques have been very popular and has outperformed traditional approaches in model performance, feature extractions. They have been used to win one biggest image classification competition -ImageNet. Convolutional neural networks(CNN) unlike other deep learning architect are designed to handle spatial information in images very well. I will build a simple CNN using 3 convolution layers each followed by max polling layers while also using different parameters settings. This CNN architecture will be made deepen by including more filters and will be made wider by adding more layers. Then I will also compare these models with models from ImageNet Challenge (Ex ResNet, VGG16, Xception) using transfer learning technique. After these I will determine the best model using mean F-score.

Metrics

Micro-Averaged F1-score

F-score is weighted average of precision and recall given by following equations

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

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

Here

k is class

TP: True Positive, case where correct species predicted

FP: False Positive, where seedling incorrectly classified to a species

FN: False Negative, cases where seedling is not classified to its species

Once we have these, Mean F score calculated as

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

Since we have uneven class distribution and cost of False positive and False negative are very different, we should look at both Precision and Recall.

2. Analysis

Data Exploration

The Aarhus University Signal Processing group, in collaboration with University of Southern Denmark, has released this dataset containing images of approximately 960 unique plants belonging to 12 species at several growth stages. Total number of training images are 4,750 and Test images are 794 without labels used only for submitting result to Kaggle leaderboard. I will use 4,750 training images to train, validate and test the model.

List of species along with number of images given in training set is in the following table

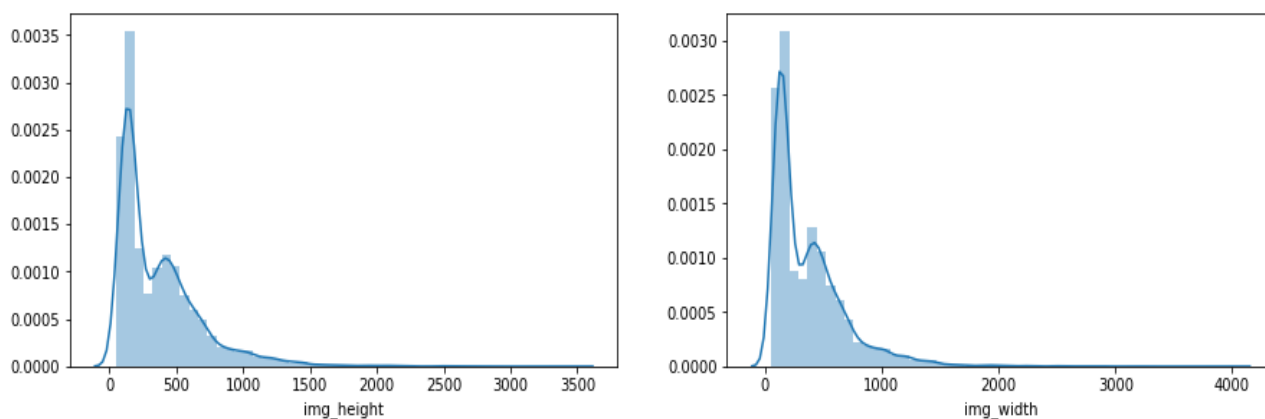
Species	Number of Images
Sugar beet	385
Loose Silky-bent	654
Scentless Mayweed	516
Maize	221
Shepherds Purse	231
Cleavers	287
Charlock	390
Small-flowered Cranesbill	496
Fat Hen	475
Common Chickweed	611
Black-grass	263
Common wheat	221
Total	4,750

Number of images in each category varies from 221 to 654. We can balance the dataset by taking only 221 images in each dataset but here we will not make full use of dataset. Or we can augment data and make each category have 654 images. This way all classes will have same number of training images.

Exploring image size.

Fig 1: Distribution Plot of Image Height and Image Width.

Smallest image 49 x 49



Smallest images in the data is 49*49 and largest goes up to 3500px.

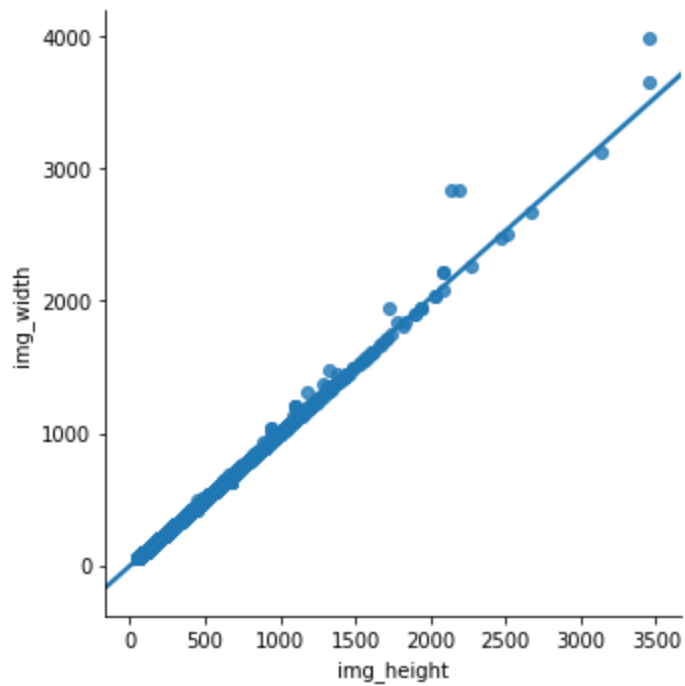
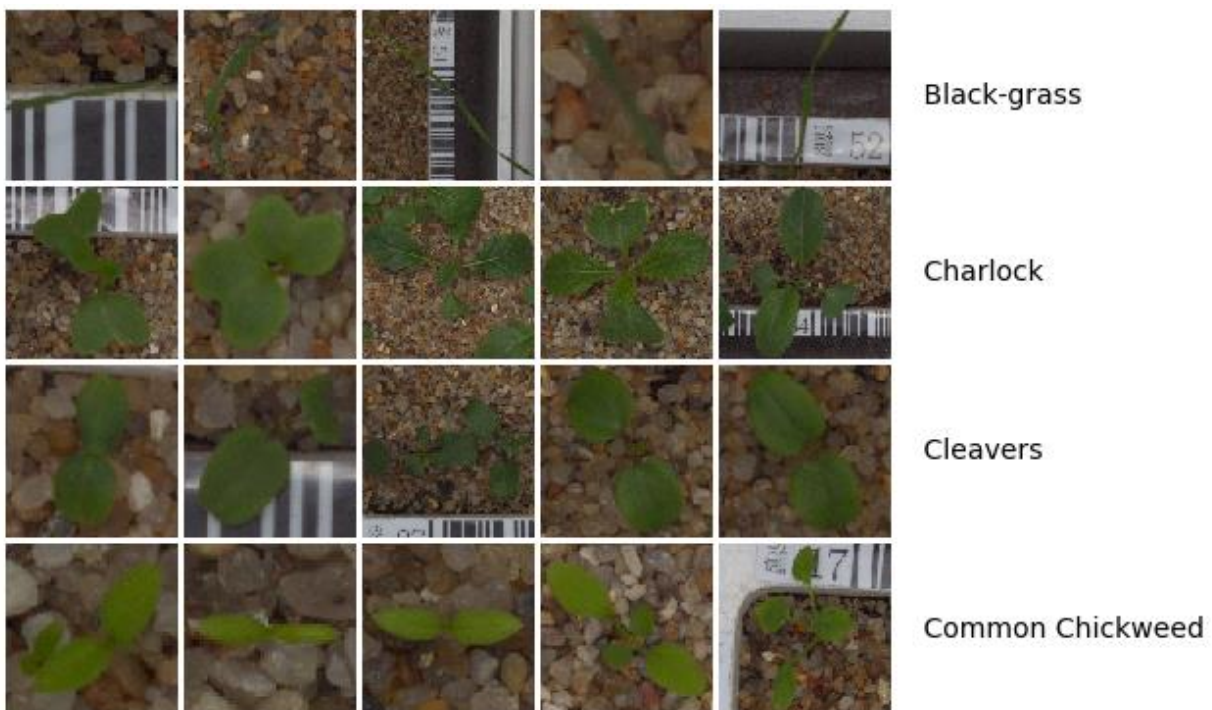


Fig 2: Aspect Ratio of Images

Expect than 4-5 images, rest have aspect ratio 1:1. We need to rescale the image for neural network training. As seen from distplot above, many images are around 100px-150 px. Smallest image is 49px X 49px. I am rescaling them such that I shouldn't scale smaller images to much large size and loss details. Therefore, I choose 128px X 128px as size to rescale images for training.

Exploratory Visualization

Fig 3: 5 Samples images form each species





Input images are Color Images (RGB) containing image of seedling along with the box, soil, stones and scale markings. Black Grass and loose silky-bent seedling have so narrow leaves that majority of image is stone and box. All the plant leaves are green in

color and background doesn't have anything green that matches the seedling color. This will help us in removing the background box, sand, stone etc.

Algorithms and Techniques

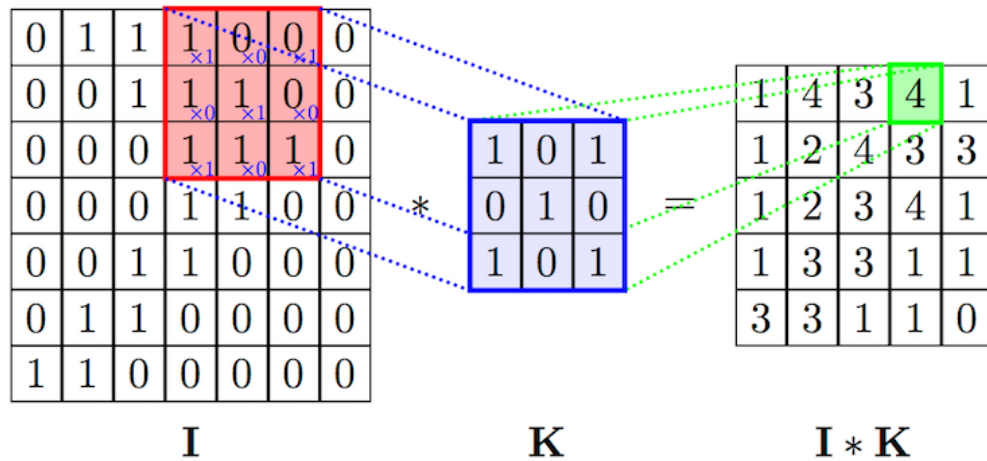
We are dealing with image classification, so this is generally treated as computer vision challenge. It's easy for Human to classify images, for machine it can be uphill task. Neural Network are designed to mimic human brain learning process. Deep learning techniques have been very popular and has outperformed traditional approaches in model performance, feature extractions. I will be using Convolutional neural networks(CNN) to train the model. CNN unlike other deep learning architect are designed to handle spatial information in images very well. In neural networks, the input is a vector, in CNN the input is a multi-channelled image (Here it is 3 channel). CNN have been used to win biggest image classification competition -ImageNet which is like Olympics for computer vision expert

CNN consists of several layers. These layers can be of three types

1. **Convolutional(Conv):** CNN has got its name from this layer. Primary function of this to extract features from image. Convolutional layers consist of rectangular grid of neurons. Each neuron takes input from rectangular section of previous layers.

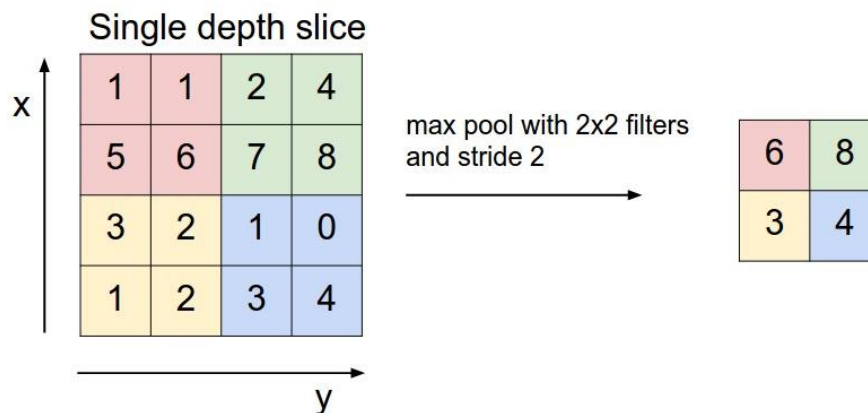
To build Conv Layer:

1. Image is converted into pixel values. (3 Matrix for each channel)
2. Make a filter that is also a matrix which will convolve(slide) over the image
3. We slide the filter over image and compute element wise multiplication and add the multiplication output to get the Convolved feature which will be also be a matrix



2. **Pooling Layer(Pool):** Each filter will result in feature map; many feature maps will increase the number of parameter. To reduce it, we have pooling layer. Pooling layer take small rectangular blocks from the convolutional layer and subsample it to product single output from that rectangular block. We can take maximum, average or linear combination of values in the block.

Fig 5: Max Pooling Layer (<http://cs231n.github.io/convolutional-networks/>)



Global Max pooling layers also exists in some network, its ordinary max pooling layer with pool size (rectangular blocks) equal the size of the input. In above diagram, max Global Max pooling layers will result in number 8 as output rather than 2 by 2 matrix.

3. **Fully Connected Layer(FC):** "Fully Connected" implies every neuron in the previous layer is connected to every neuron on next layer. It is the traditional multi-layer perceptron. Output from Conv-Pool layer above represent high-level features of the input image. FC layer will be used to use these features to classify input image into various species of plants. Fully connected layers are not spatially located anymore so there won't be any more Conv-Pool layers after it.

Operations that are applied on layers and forms part of the network

1. Dropout: Dropout layer are used to prevent network from overfitting. Dropout alternately randomly disable neurons in training.
2. Activation: Apply nonlinear operation to output of Conv or FC layers.
 - a. ReLU: Stands for Rectified Linear Unit. It is a nonlinear element wise operation and replace all negative pixel value in the feature map with 0.
Output = $\max(0, \text{input})$
 - b. Softmax: Calculates probabilities of each target class over all possible target class. Sum of all probabilities will be equal to 1. Ratio of exponential of that input value to sum of exponential of all inputs value is the output of softmax.
3. Optimizers: Optimization function minimizes error function which is mathematical functional dependent on internal parameter (Weight and bias) here. There internal parameter are learned and updated in the direction of optimal solution, here it is minimizing the loss by network training process. I will try different optimizers and compare the result.

Benchmark

No benchmark is mentioned in Kaggle competition page. But we can still consider better than random chance of $1/12=0.08$ as benchmark. But that is too less for any practical use. So, I have developed a simple CNN model with Three convolutional layers with ReLU activation each following with max pooling layers. I achieved mean **F1 score of 0.59** and **Accuracy of 0.61**. This accuracy is better than random chance of $1/12$. I will use this basic CNN model with Mean F1 score of 0.59 as benchmark model.

```
bench_model = Sequential()
bench_model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu', input_shape=(128, 128, 3)))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Dropout(0.4))
bench_model.add(GlobalAveragePooling2D())
bench_model.add(Dense(12, activation='softmax'))
bench_model.summary()
```

I will improve the model by

1. Data Augmentation
2. Making CNN denser: add more layers

3. Making CNN wider: add more filters
4. Using different optimization function

Then I will also use transfer learning by using pretrained model like ResNet50, VGG16, InceptionV3 and Xception.

Methodology

Data Preprocessing

As discussed earlier images are RGB channel and have different sizes. Also, I discussed that I will choose 128px X 128px as image size. I am using Tensorflow in backend of Keras, which require a 4D array as input. I will convert the image into Numpy array of dimension (Number of images, 3, 128, 128) also called 4D Tensor, suitable for supplying to a Keras CNN. Pixel values in 4D Tensor are convert to float32 format and normalized by diving by 255. Following function are used for this purpose.

```
def path_to_tensor(img_path):
    # Loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(128, 128))
    # convert PIL.Image.Image type to 3D tensor with shape (128, 128 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 128, 128, 3) and return 4D tensor
    return np.expand_dims(x, axis=0).astype('float32')/255

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

For loading dataset, load_dataset function is defined with take path of the dataset as input and return files path (path of image), labels of species and numerical value of label

```
#function to load datasets.
def load_dataset(path):
    data = load_files(path)
    X_files = np.array(data['filenames'])
    y_target= np.array(data['target'])
    labels= np.array(data['target_names'])
    # targets = np_utils.to_categorical(np.array(data['target']), 12)
    return X_files,y_target,labels
```

Then I did Stratified sampling to split data into Training, Validation and test set since number of training example in classes are unbalanced.

```

# Stratified Sampling: Split data in 80% training and 20% for validation and test.
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)

for train_index, test_index in sss.split(train_tensors, train_labels):
    X_train, X_temp = train_tensors[train_index], train_tensors[test_index]
    y_train, y_temp = train_labels[train_index], train_labels[test_index]

# Split 20% data in validation and Test set
sss2 = StratifiedShuffleSplit(n_splits=1, test_size=0.5, random_state=42)
for train_index, test_index in sss2.split(X_temp, y_temp):
    X_valid, X_test = X_temp[train_index], X_temp[test_index]
    y_valid, y_test = y_temp[train_index], y_temp[test_index]

print(X_train.shape, "-", y_train.shape)
print(X_valid.shape, "-", y_valid.shape)
print(X_test.shape, "-", y_test.shape)

```

Then after sampling, I carried out One Hot encoding of each train, validation, test set labels.

```

#One hot encoding of labels.
y_train_label = np_utils.to_categorical(y_train, 12)
y_valid_label = np_utils.to_categorical(y_valid, 12)
y_test_label = np_utils.to_categorical(y_test, 12)

```

Implementation

I created following function to make code modular

1. Train: This function is used to train the model. It receives 4 argument, Model variable, name of model, epochs, Training variables(X,y).

```

"""
Train model function: Take model, epoch, Training Tensors and Labels as argument. Fit the model to training data.
"""

def train(model, model_name, epochs=10, x=X_train, y=y_train_label):
    def get_callbacks(filepath):
        es = EarlyStopping('val_loss', patience=5, mode="min")
        msave = ModelCheckpoint(filepath, save_best_only=True)
        return [es, msave]

    file_path = "./weights/weights.from_%s.hdf5" % model_name
    callbacks = get_callbacks(filepath=file_path)
    checkpointer = ModelCheckpoint(filepath=file_path, verbose=1, save_best_only=True)
    history = model.fit(x, y,
                        validation_data=(X_valid, y_valid_label),
                        epochs=epochs, batch_size=32, callbacks=callbacks, verbose=1)
    return history

```


A callback function is defined within this function which return ModelCheckpoint and Earlystopping object.

ModelCheckpoint: We take snapshot of the training to avoid losing training data in case of system failure. In Deep learning, we save the weights of the model. It allows us to define where to checkpoint the model weights, where to save the weight and under what circumstances.

Earlystopping: Stop training if validation loss increases after set of epoch defined by patience parameter.

2. Predict Function:

Use for predicted labels in test data. It takes model as argument and predict labels on test data. It returns F1 score, Accuracy score and Confusion matrix.

```
"""
Prediction on test set made from training data
"""
def predict(model,model_name):
    file_path = "./weights/weights.from_%s.hdf5" % model_name
    model.load_weights(file_path)
    predictions = [np.argmax(model.predict(np.expand_dims(feature, axis=0))) for feature in X_test]
    test_list = y_test_label.argmax(axis=1)
    f1score=f1_score(test_list, predictions, average='weighted')
    acc = accuracy_score(test_list,predictions)
    print('Test F1 Score: %.5f' % f1score)
    print('Test Accuracy Score: %.5f' % acc)

    #compute confusion matrix
    cnf_matrix = confusion_matrix(test_list, predictions)

    # Plot non-normalized confusion matrix
    plt.figure(figsize=(20,10))
    plot_confusion_matrix(cnf_matrix,classes=labels)
    plt.show()
```

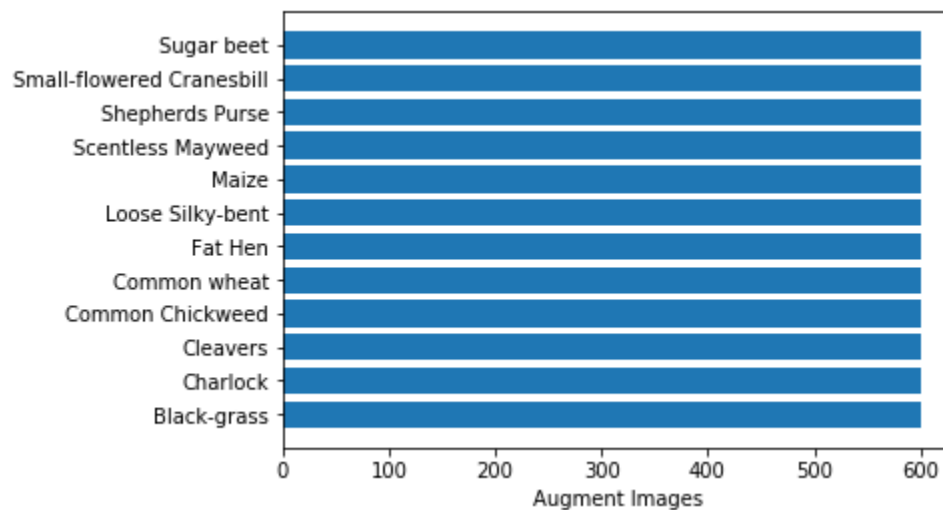
I started training model with basic CNN consisting of CONV-POOL-CONV architect.

```
bench_model = Sequential()
bench_model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu', input_shape=(128, 128, 3)))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
bench_model.add(MaxPooling2D(pool_size=2))
bench_model.add(Dropout(0.4))
bench_model.add(GlobalAveragePooling2D())
bench_model.add(Dense(12, activation='softmax'))
bench_model.summary()
```

I got F1 score of 0.59. I used this as a benchmark.

Refinement

Since data is unbalanced which may result to model preferring classes which large number of training data than other. I balanced the data by augmenting images using Keras ImageDataGenerator for which I followed this article [Building powerful image classification models using very little data](#) by Francois Chollet. Each class after augmentation had 600 training images. Since class Maize and common wheat had only 221 examples, I choose 600 so that each image can augmented just ~3 times. Second reason is more training example may lead to more computation time.



Then I ran the same CNN architect of benchmark model to augmented dataset.

I achieved **F1 Score 0.60812** and **Accuracy of 0.61684** on test data. This is very small improvement over the benchmark because only very few classes data were really augmented. Loose silky bent class was not augmented at all but has highest example correctly classified in test set.

Then I changed optimizer to Adam for which F1 score and Accuracy are reported below.

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Benchmark with Augmentation, Optimizer: RMSPROP	0.61	0.61
Benchmark with Augmentation Optimizer: Adam	0.55	0.55

Score doesn't improve with Adam Optimizer. Next, I added more fully connected layer to make the model dense calling it as "DenseFC1" with optimizer as RMSprop. I added dense layer with 512 nodes followed by layers of 256 nodes both using ReLU activation function.

```
model2 = Sequential()
model2.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu', input_shape=(128, 128, 3)))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model2.add(MaxPooling2D(pool_size=2))
model2.add(Dropout(0.4))
model2.add(GlobalAveragePooling2D())
model2.add(Dense(512, activation='relu'))
model2.add(Dropout(0.4))
model2.add(Dense(256, activation='relu'))
model2.add(Dropout(0.4))
model2.add(Dense(12, activation='softmax'))
model2.summary()
```

DenseFC1 achieved **F1 Score 0.64** and **Accuracy of 0.65** on test data which is again a small improvement over the previous score. Then I changed the optimizer to Adam which doesn't show much improvement.

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
DenseFC1 with RMSProp	0.64	0.65
DenseFC1 with Adam	0.65	0.65

Since I was not getting that much improvement in score, I realized that I may need to change the number of filter and add more convolutional layers. So, I have used the following architecture which consists of 4 CONV layers each followed by Max Pool layers then followed by 2 Fully connected layers. I have also change number of filters starting from 64 to 128 to 128 and then to 64. Also change the filter size to (3,3) as opposed to earlier (2,2).

```

model4=Sequential()
#Conv Layer 1
model4.add(Conv2D(64, kernel_size=(3, 3),activation='relu',padding='same', input_shape=(128, 128, 3)))
model4.add(MaxPooling2D(pool_size=(3, 3)))
model4.add(Dropout(0.2))

#Conv Layer 2
model4.add(Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu' ))
model4.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model4.add(Dropout(0.2))

#Conv Layer 3
model4.add(Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu'))
model4.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model4.add(Dropout(0.2))

#Conv Layer 4
model4.add(Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))
model4.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model4.add(GlobalAveragePooling2D())

#Dense Layers
model4.add(Dense(512, activation='relu'))
model4.add(Dropout(0.2))

#Dense Layer 2
model4.add(Dense(256, activation='relu'))
model4.add(Dropout(0.2))

model4.add(Dense(12, activation='softmax'))
model4.summary()
# Dense Layer added
model4.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['accuracy',fmeasure])
model4_history=train(model4,model_name="Wide2",x = X_train_aug_tensor, y = y_train_aug_label, epochs=20)

```

Calling this as "Wide2" as it contains more filters. This is similar to a model that I have used in another Kaggle competition.

For another iteration calling it as "Wide2Dense", I made a little change in "Wide2" Model by adding more nodes to Fully Connected layers. 1024 in first layers and 512 in second layers. Both model score are represented below.

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Wide2	0.82	0.81
Wide2Dense	0.81	0.81

This has huge improvement in score. Then I thought what If I choose the same filter size in Wide2 of (2,2) as in my benchmark and compare them with new architect.

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Wide2(kernel size 3)	0.82	0.81
Wide2 with kernel size 2	0.63	0.64

This indicate the kernel of size 3 is better choice. Then I made iteration to more "Wide2" model as follow

1. Change Optimizer to Adam: named as "Wide2Adam"
2. Decrease number of filters and increase node in Fully Connected layers: named as "Wide2_FNN"
3. Add additional Convolutional layers before first two max pooling layers. Named as "Wide2_Extra_Conv" (snapshot below)

```
#Change architect by introducing additional Conv layer before max pooling.
model9=Sequential()
#Conv Layer 1
model9.add(Conv2D(64, kernel_size=(3, 3),activation='relu',padding='same', input_shape=(128, 128, 3)))
model9.add(Conv2D(64, kernel_size=(3, 3),activation='relu',padding='same'))
model9.add(MaxPooling2D(pool_size=(3, 3)))
model9.add(Dropout(0.2))

#Conv Layer 2
model9.add(Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu' ))
model9.add(Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu' ))
model9.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model9.add(Dropout(0.2))
```

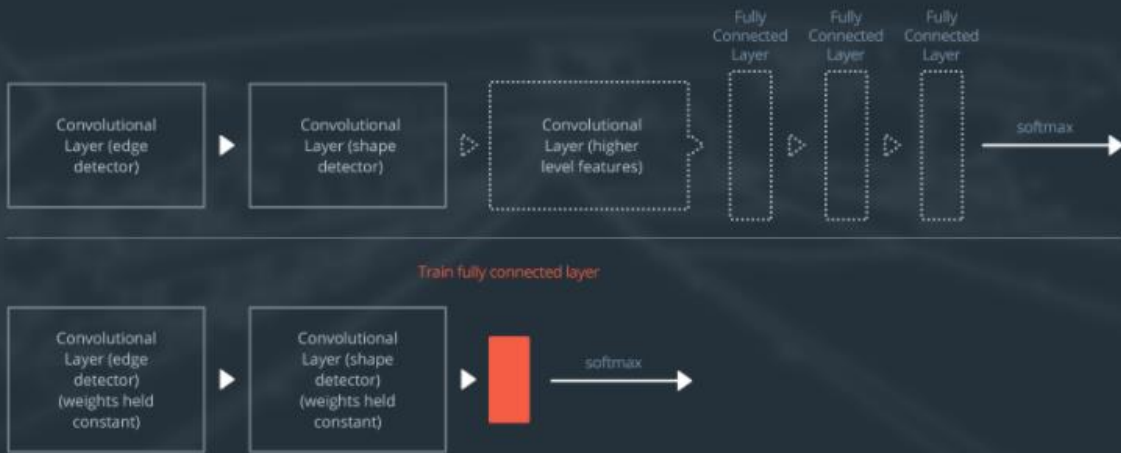
Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Wide2	0.82	0.81
Wide2 with Adam Optimizer	0.77	0.76
Wide2_FNN	0.79	0.78
Wide2_Extra_Conv	0.84	0.84

Multiple Convolutional layer before a max pooling layers has resulted in highest score, but each epoch took double the amount of time when compared with Wide2.

I have got decent amount of score after 9 Refinements. So I went forward with Transfer learning approach using model weights from ImageNet Winning models.

I used Xception and InceptionV3 model weight to carry out transfer learning approach. Since this dataset is small and different from dataset used in ImageNet competition, we should freeze the weights on convolutional layers near to the beginning of the network. I went ahead and tried both freezing only Convolutional layers and freezing layers in the beginning of the network.

Case: Small Data Set, Different Data



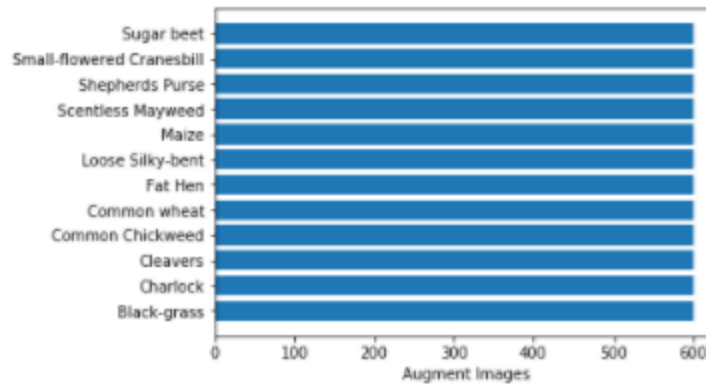
Transfer learning using Xception

As discussed, I freeze weight of all the convolutional layers and removed fully connected layers from the network. Then applied Global pooling average layers followed by fully connect layers of 128 nodes. Got very bad score as expected.

F1 score: 0.36

Accuracy: 0.36

For second part of transfer learning which is to freeze weights of few convolutional layer in beginning of network, I used InceptionV3 CNN Model weights and fine tune the network. Since InceptionV3 require image size to be greater than 224px, I have created new set of training, validation and test set with images of size 224px by 224 px. As before I have augmented the data in each class upto 600 images.



InceptionV3 has in total 312 layers. I have freeze the weights of layers upto 235. Since we are fine tuning we need to lower the learning rate of optimizer, so loss may not diverge. Finetuning is typically carried out with SGD optimizer rather than adaptive learning rate Optimizer like RMSProp.

```
# Fine tuning Inception
'''
'''
layer_to_freeze = 235

# InceptionV3: Transfer Learning
model_InceptionV3 = InceptionV3(weights = "imagenet", include_top=False, input_shape = (224, 224, 3))

#freeze Layer weight
for layer in model_InceptionV3.layers[:layer_to_freeze]:
    layer.trainable = False

# add FC Layers
x = model_InceptionV3.output
x = GlobalAveragePooling2D()(x)
predictions = Dense(12, activation='softmax')(x)
model_InceptionV3 = Model(inputs=model_InceptionV3.input, outputs=predictions)
model_InceptionV3.compile(optimizer=optimizers.SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy', fmeasure])
```

I added GlobalAveragePooling2D layers followed by FullyConnected layer with 12 nodes to classify image. Softmax function is used to return probabilities of example being in that class. I ran the model for 50 epochs. This was the most time-consuming model among all the previous iterations. I had to stop the model in between and ran for 5 epochs without compile the model again. I got the following score

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Transfer learning Xception	0.36	0.36
Fine tuning InceptionV3	0.76	0.75

With more number of augment images and more epochs, InceptionV3 with fine tuning could have performed much better. But due to limitation of computation resource,

couldn't train the model longer enough. At last, I tried with more number of augmented images. 1000 images each class compared to 600 previously and used best performing model both "Wide2" and "Wide2_Extra_Conv" now calling them Wide2_Aug1K and "Wide2_Extra_Conv_Aug1K" respectively.

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Wide 2	0.82	0.81
Wide2_Extra_Conv	0.84	0.84
Wide2_Aug1K	0.85	0.85
Wide2_Extra_Conv_Aug1K	0.86	0.85

Wide2_Extra_Conv_Aug1K took double the time as of Wide2_Aug1K so I choose Wide2_Aug1K as final model.

Results

Model "Wide2_Aug1K" performed best among all the other 12 models. Wide2_Aug1K model has the following architect

```
model5_Aug1k=Sequential()  
#Conv Layer 1  
model5_Aug1k.add(Conv2D(64, kernel_size=(3, 3),activation='relu',padding='same', input_shape=(128, 128, 3)))  
model5_Aug1k.add(MaxPooling2D(pool_size=(3, 3)))  
model5_Aug1k.add(Dropout(0.2))  
  
#Conv Layer 2  
model5_Aug1k.add(Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu' ))  
model5_Aug1k.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))  
model5_Aug1k.add(Dropout(0.2))  
  
#Conv Layer 3  
model5_Aug1k.add(Conv2D(128, kernel_size=(3, 3),padding='same', activation='relu'))  
model5_Aug1k.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))  
model5_Aug1k.add(Dropout(0.2))  
  
#Conv Layer 4  
model5_Aug1k.add(Conv2D(64, kernel_size=(3, 3),padding='same', activation='relu'))  
model5_Aug1k.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))  
model5_Aug1k.add(GlobalAveragePooling2D())  
  
#Dense Layers  
model5_Aug1k.add(Dense(1024, activation='relu'))  
model5_Aug1k.add(Dropout(0.2))  
  
#Dense Layer 2  
model5_Aug1k.add(Dense(512, activation='relu'))  
model5_Aug1k.add(Dropout(0.2))  
  
model5_Aug1k.add(Dense(12, activation='softmax'))  
model5_Aug1k.summary()  
# Dense Layer added  
model5_Aug1k.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['accuracy',fmeasure])  
model5_Aug1k_history=train(model5_Aug1k,model_name="wide2_Aug1K",x = X_train_aug_tensor, y = y_train_aug_label, epochs=20)
```


Model Evaluation and Validation



Model achieved much better score than the benchmark model. It achieved F1 Score of 0.84 and Accuracy of 0.84 on test data. Best model in Training had F1 Score of 0.89 and Accuracy of 0.81. This means model has not overfit to training data and show its robustness. Image Augmentation for classes with led the model generalize training dataset thus avoided overfitting. Higher size of filter (3 by 3) as compared to benchmark (2 by 2) has resulted in better score.

Justification

Model	F1 Score	Accuracy
Benchmark	0.59	0.61
Wide2_Aug1K	0.85	0.85

F1 score has increased more than 20 points from the benchmark which is a significant improvement minding the fact that computational resource was limited. Also benchmark model did had much better F1 score and Accuracy when compared with random chance.

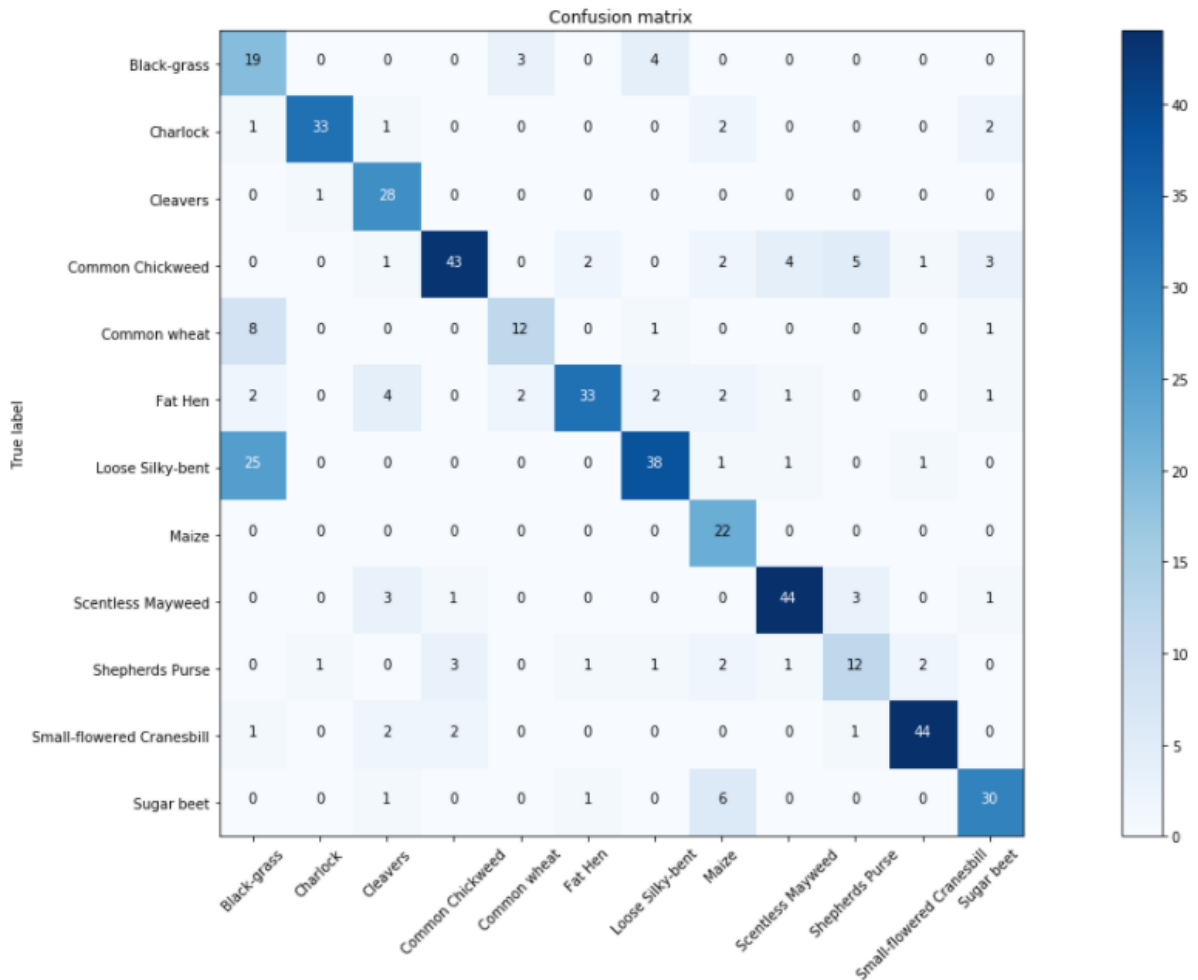
I have also predicted on the test set provided by Kaggle and has received almost same F1 score of 0.85 as in test set made from training data provided by Kaggle. So again, model doesn't overfit on test data and is robust. This was my first submission.

416	new	manvindra singh		0.84760	1	now
Your Best Entry 						
Your submission scored 0.84760, which is not an improvement of your best score. Keep trying!						

Conclusion

Free-Form Visualization

I also computed confusion matrix to dig deeper into each class and understand which class has most wrong classified examples.



This confusion matrix contains test set data only. Y axis is correct label of examples and X axis is label predicted by model. Loose Silky-Bent class has the highest True Negative (sum of value across row except diagonals) which is classified as Black-grass most of the time. As seen from following images, these classes are difficult to classify by naked eye.



Black Grass



Loose Silky Bent

Reflection

I did try a lot of iteration (~ 15 models) which does take lot of time. I ran them on AWS machine which I can use only for few hours due to budget constraint. I didn't jump to different architecture that may have no thought process behind, rather I did small changes to understand how each iteration effect the answers. Because I believed that I couldn't learn by just trying hundreds of models but can through understanding how certain parameter changes solution.

I tried different filter size, different optimizers, Image augmentation to balance class, image augmentation to remove overfitting, changed architect from CONV-POOL-CONV-POOL to CONC-CONV-POOL-CONV-CONV-POOL, tried model with low number of filter in initial CONV layer to model with large number of Filters in initial layers, Make model deep by changing number of neurons in fully connected layers. Finally, I tried Transfer learning and Fine Tuning of ImageNet Model InceptionV3.

I found out that small number of filters in initial layers couldn't capture the High-level features in the images. I hoped that making the network dense will help in increasing accuracy but that didn't happen too. I understood more filters along with increased filter size helped in increasing the accuracy than changing the architecture, using Transfer learning, Making network dense.

Improvement

I got highest misclassification among Black Grass and Loose Silky Bent species plant. These looks very similar as they have very thin leaves. They cover less area in images. I could try some computer vision technique to isolate leaves from background stone and box in the image.

Due to budget constraint, I couldn't try iteration of freezing different layers of InceptionV3 or Try different classifier after CNN layers or Augment data to more number of images.

Final Remarks

I feel satisfied and confident after finishing this project. Learnt AWS on the way with this project. I didn't think that my simple CNN benchmark model can give more than 50% accuracy on multiple class classification problem. I am Impressed by CNN Algorithm and its easy of writing code for that in Keras.