Machine Learning Engineer Nanodegree

Capstone Project

Abin Saju
May 27th, 2019

Project Overview

The project aims to use publicly available data to automatically classify seedlings. The project was proposed on Kaggle, a website which promotes use of publicly available datasets for machine learning projects.

Classification of seedlings is an important process in agriculture. It is important to be able to identify a weed in a batch of cultivated plants to ensure the seedling does not have to compete for the limited resources. Identifying and clearing the weeds ensures high crop yield. The current method for the task involves visual inspection which is an expensive and resource consuming task. The project aims to provide an automated ML solution.

Data for the project is downloaded from https://www.kaggle.com/c/plant-seedlings-classification/data. The dataset contains coloured images of 960 unique pants belonging to 12 species at several growth stages [1]. In total, there are 4750 training data points and 794 testing points. The data is provided publicly by the authors of the paper on plant seedling classification [2].

Problem Statement

The project aims to use publicly available data to automatically classify seedlings. The training dataset contains RGB images with species labels identifying each image as one of twelve species. During training, the performance of the model will be evaluated using a micro-averaged f1-score. Once the model is trained on the dataset, it would be used to predict images in the testing dataset. The predictions will then be verified by submitting the generated file to Kaggle.

Using machine learning to solve the problem means the problem can be easily scaled with a high degree of accuracy. The final objective of the project would be to successfully classify all data points in the testing set.

Metrics

I am using the evaluation metric provided on the Kaggle website. Which is a micro-averaged F1-score.

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

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

F1-score is the harmonic mean of precision and recall

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

Figure 1: micro F1 score [3]

Data Exploration

The training dataset contains 4750 images for 12 different species. The images are split up like shown below:

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

The breakdown of the classes show that images for each class is not consistent. For example, ‘Shepherds Purse’ contains 221 images while ‘Loose Silky-bent’ contains 654 images. To improve the dataset, adding augmented data will be tried.

Exploratory Visualization

The images in the dataset contain images of seedlings in a tray. The seedlings seems to be sprouting in a tray of tiny pebbles. From visual inspection of the data it is clear that the images lack contrast. Below a sample image from each class is shown.

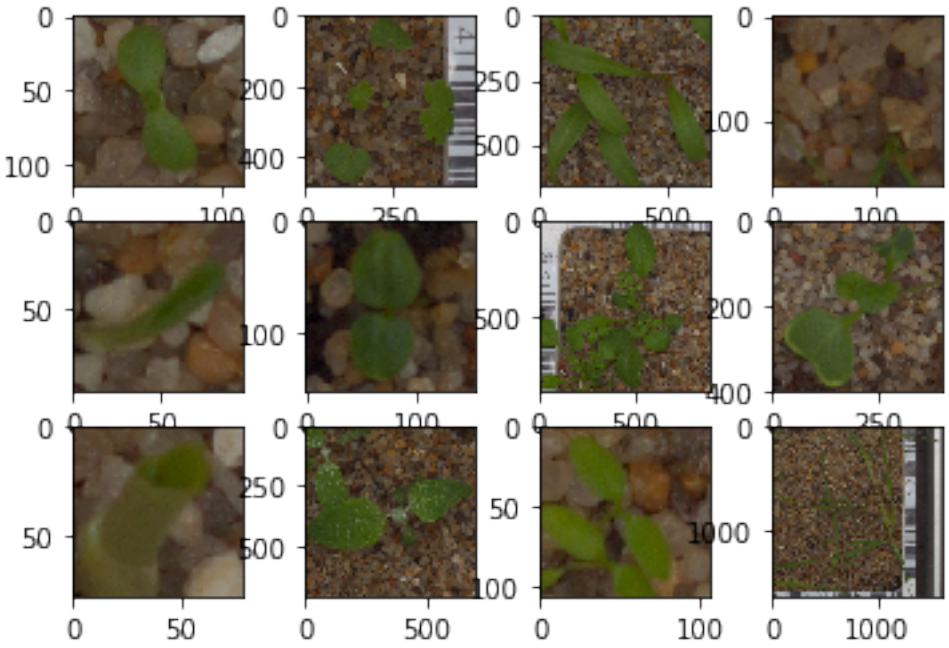


Figure 2: Image from each of the classes

It is clear from the image that data pre processing would be important in this case, to enhance the quality of the images. Looking more at a specific class, it can be said the data is not of the highest quality, as the background varies greatly from image to image and green splotches in the images are sometimes barely visible.

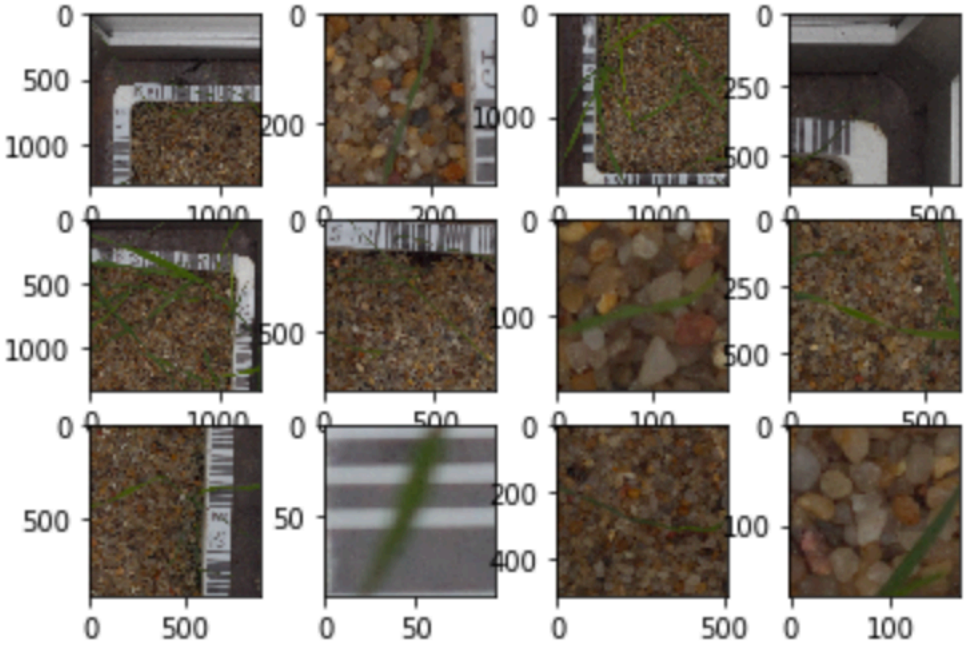


Figure 3: 12 Images from Black-grass class

Algorithms and Techniques

For the problem, Convolutional Neural Networks (CNN's) are used, as this is an image classification task. Initially a simple model would be attempted to rapidly identify issues with the implementation, once a stable model is achieved, it will be swapped with a ResNet model.

ResNet is a CNN architecture which is highly regarded to be one of the best for image classification tasks [4]. To aid the model the keras pre-processing library will be used to augment the data. The input layer of the model will need to be updated to suit images of size 224x224. The below code achieves this by initialising the input tensor parameter with the shape of the inputs. The imagenet weights are also loaded in this step to prevent the need to train the entire model from scratch. Using the weights allows the model to train quicker with minimal impact to performance.

```
input_tensor = Input(shape=(224, 224, 3))
r50_model = ResNet50(input_tensor = input_tensor, weights='imagenet', inc
lude_top=False)
```

Benchmark

The performance of the model will be compared to the ResNet top-1 error from the ResNet paper [5]. The model was trained on a classification task, similar to this project and it would be interesting to compare the performance of the two models.

Data Preprocessing

Looking at the images in the dataset, it is clear that the quality of the images aren't ideal. To enhance the characteristics of the seedling, a data augmentation step was added. It uses opencv2 to enhance the RGB image.

```
def transform_image(img):
    lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

    lab_planes = cv2.split(lab)

    clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(8,8))

    lab_planes[0] = clahe.apply(lab_planes[0])
    lab_planes[1] = clahe.apply(lab_planes[1])
    lab_planes[2] = clahe.apply(lab_planes[2])

    lab = cv2.merge(lab_planes)

    return cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
```

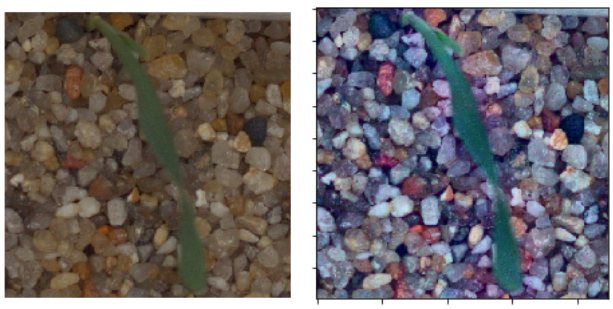


Figure 4: Image of common wheat seedling, before(left) and after (right)

Implementation

A resenet50 model is implemented with a metrics call back for F1 score.

```
from keras.callbacks import Callback
from sklearn.metrics import confusion_matrix, f1_score, precision_score,
recall_score
from sklearn.model_selection import train_test_split

class Metrics(Callback):
    def on_train_begin(self, logs={}):
        self.val_f1s = []
        self.val_recalls = []
        self.val_precisions = []

    def on_epoch_end(self, epoch, logs={}):
        val_predict = (np.asarray(self.model.predict(self.validation_data
[0]))).round()
        val_targ = self.validation_data[1]
        _val_f1 = f1_score(val_targ, val_predict, average = 'micro')
        _val_recall = recall_score(val_targ, val_predict, average = 'micr
o')
        _val_precision = precision_score(val_targ, val_predict, average =
'micro')
        self.val_f1s.append(_val_f1)
        self.val_recalls.append(_val_recall)
        self.val_precisions.append(_val_precision)
        print '— val_f1: %f — val_precision: %f — val_recall %f' %(_val_f
1, _val_precision, _val_recall)
        return

metrics = Metrics()
```

During the setup of the model to save time, not every layer is trained, instead weights from a pre-trained model are loaded and the first 172 layers are frozen. The last couple of layers which contain very specific feature information is retrained to fit the model.

```
from keras.applications.resnet50 import ResNet50
from keras import Model
from keras.layers import Input

#Load in the ResNet model
input_tensor = Input(shape=(224, 224, 3))
r50_model = ResNet50(input_tensor = input_tensor, weights='imagenet', inc
lude_top=False)
x = r50_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='sigmoid')(x)
# Add Dense layer with 12 node's for the 12 classe
predictions = Dense(12, activation='softmax')(x)
# this is the model we will train
r50_model = Model(inputs=r50_model.input, outputs=predictions)

for layer in r50_model.layers[:172]:
   layer.trainable = False
for layer in r50_model.layers[172:]:
   layer.trainable = True
```

```
### TODO: Compile the model.
r50_model.compile(loss='categorical_crossentropy', optimizer='rmsprop', m
etrics=['accuracy'])
```

```
### Train the model.
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.resnet
50.hdf5',
                               verbose=1, save_best_only=True)

history = r50_model.fit(X_train, y_train,
          validation_data=(X_test, y_test),
          epochs=20, batch_size=20, callbacks=[checkpointer, metrics], ve
rbose=1)
```

Refinement

Due to the limited amount of images available, further data augmentation is used for training of the model. This is done using the keras image pre-processing library. This is done with the hope that increased number of augmented data would prevent the model from overfitting.

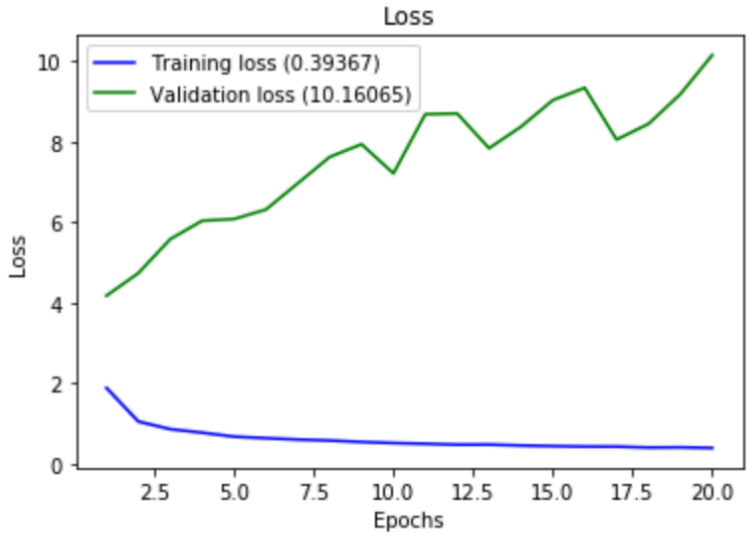
```
from keras.preprocessing import image

datagen =image.ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    validation_split=0.2,
)

datagen.fit(X_train)
```

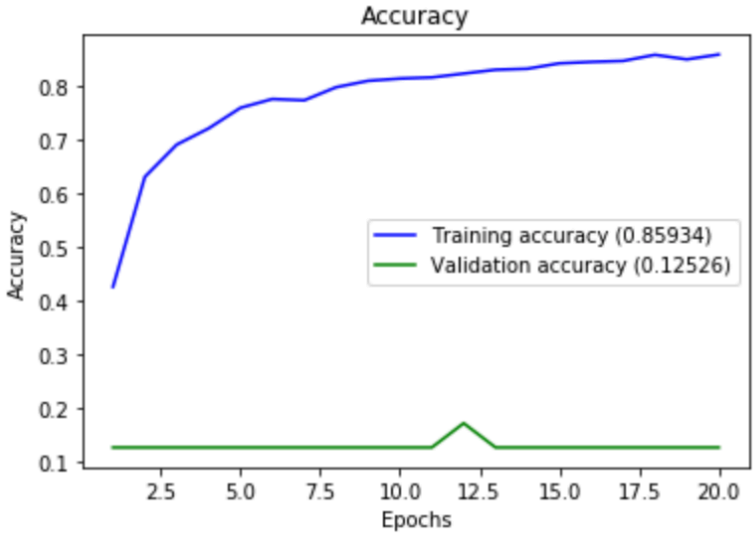
The fit call on model is also updated.

```
history = r50_model.fit_generator(datagen.flow(X_train, y_train), steps_p
er_epoch=len(X_train)/32,
          validation_data=(X_test, y_test),
          epochs=20, callbacks=[checkpointer, metrics], verbose=1)
```

Model Evaluation and Validation

The final model was the combination of the implemented model with the refinement steps. Validation data was added in the 'fit' call so the model could be continuously evaluated as the training progressed.





From the graphs it is clear due to the divergence between the training accuracy and the validation accuracy that the model overfitted the data. This means, though the model will work great on the training data, it wouldn't be as effective on unseen data.

Justification

The top 1 error of the ResNet-50 model on imagenet dataset is 20.74 [5]. Comparing it to the trained model it is ~15.17 on the training set and ~88.48 on the validation set. Though the model outperformed on the training set, this was due to overfitting, its performance on the validation set suggests the model did not learn the model well.

Free-Form Visualization

It is interesting to compare the plots of the simple model and the complex ResNet model. The simple model manages to outperform significantly on the validation data. This could suggest that it could be worth building up a simpler model, than using a complex model with the current dataset.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 27, 27, 64)        0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 12)                780
=================================================================
Total params: 11,324
Trainable params: 11,324
Non-trainable params: 0
```

Figure 5: Summary of the simple model

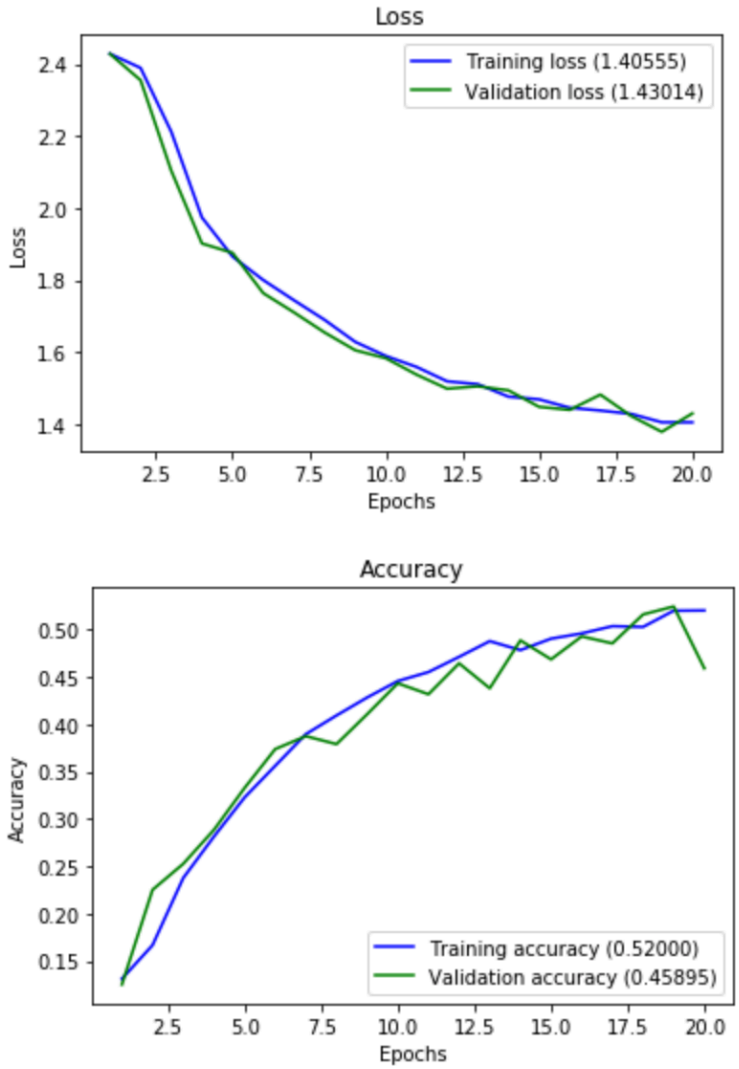


Figure 6: History plot with simple model

Reflection

The project involved using dataset from a Kaggle project and using it to a train a ResNet model, so it can accurately predict the test images. Initially I used a very simple model to train, to iron out any kinks in the solution and accurately report a F1 score. However, I ran into overfitting issues, to resolve this I tried to use some data augmentation using the keras pre processing library and image enhancements using openCV2.

Once above steps were completed, a ResNet model was implemented to fit the training set. However, after numerous tries the model kept overfitting the dataset. Freezing/unfreezing of different layer counts were attempted however they either took too long to train or didn't yield significantly better results. I still think ResNet is the ideal choice as it was trained for a classification task and should be ideal for this project, if I were to go back I would attempt steps laid out in the next section and attempt to train the model again.

Improvement

The trained model severely overfitted the data, preventing it from being a reliable model for unseen data. For further iterations, I would try to increase the size of the dataset, this would increase the number of datapoints per class that the model can use. Another major boost can be expected by improving the quality of the images, as seen from the exploratory visualisation the features in the classes are not very clear in some cases.

In terms of the model, I would attempt to unfreeze more layers and running the training for more epochs, this could possibly help with the overfitting of that data. The plot from the simple model also seems promising, if better data can't be easily attained building on the simple problem could be promising.

Bibliography

[1] M. Dyrmann and P. Christiansen, "Plant Seedlings Dataset," 2014. [Online]. Available: https://vision.eng.au.dk/plant-seedlings-dataset/.

[2] T. M. Giselsson, R. N. Jørgensen, P. K. Jensen, M. . Dyrmann and H. S. Midtiby, "A Public Image Database for Benchmark of Plant Seedling Classification Algorithms," *Biosystems Engineering,* vol. , no. , p. , 2016.

[3] Kaggle Inc, "Plant Seedlings Classification," [Online]. Available: https://www.kaggle.com/c/plant-seedlings-classification/overview/evaluation.

[4] A. . Canziani, A. . Paszke and E. . Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , p. , 2017.

[5] K. . He, X. . Zhang, S. . Ren and J. . Sun, "Deep Residual Learning for Image Recognition," *arXiv: Computer Vision and Pattern Recognition,* vol. , no. , pp. 770-778, 2016.