PLANT CLASSIFICATION USING DEEP LEARNING

June 27, 2019

Soma Ram Ganesh
Dandu Sandeep
Vedati Shravan

Contents

0.1 Introduction

Now we are going to train a Deep Learning model to predict a Plant and classify among the 12 plant species respectively Black-grass, Charlock, Cleavers, Common Chickweed, Common wheat, Fat Hen, Loose-Silky-bent, Maize, Scentless Mayweed, Shepherds Purse, Small-flowered Cranesbill, Sugar beet.
Finally we are going to build a User Interface which can be used by a Layman to use it for prediction.
The dataset have been recorded at Aarhus University Flakkebjerg Research station in collaboration between University of Southern Denmark and Aarhus University.

0.2 Requirements

1. Python 3.6
2. Pycharm IDE
3. Flask Framework
4. Python packages respectively Numpy, Pandas, Keras, OpenCV, Tensorflow

0.3 Explanation

This has mainly 2 parts:

Part 1:
We are going to sort all the dataset into two files.
One file contains all the images belonging to the 12 Plant Species.
Another file contains all the labels (plant names) belonging to the images in previous file.
Then we are going to create a Deep Learning Architecture and Initialize a ADAM optimizer.
Then we use ImageDataGenerator for Data Augmentation and initializing the training.
Then training continues till 100 Epochs.
Finally we are going to plot a Classification report on the Trained model and see

that we got a 86% accuracy.

Then we are going to save the weights and architecture of the trained model for future use.

Part 2:

We are going to use the Flask Framework to make a simple WebApp with our Model running on the server which provides a simple UI for the prediction purposes.

0.4 Code Overview

PART 1: Jupyter Notebook

1. Importing all the required Packages:

```
import numpy as np # linear algebra
import pandas as pd # data processing
import os
import cv2
import matplotlib .pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras import backend as K
import random
```

2. Defining the Model Architecture:

```
class SmallVGGNet:
    @staticmethod
    def build(width, height, depth, classes):
        model = Sequential()
        model.add(Conv2D(32, (3,3), activation = 'relu', input_shape = (width, height, 3)))
        model.add(Conv2D(32, (3,3), activation = 'relu'))
        model.add(Conv2D(32, (3,3), activation = 'relu'))
        model.add(MaxPooling2D(pool_size = (2,2)))
        model.add(Dropout(0.3))

        model.add(Conv2D(64, (3,3), activation ='relu'))
        model.add(Conv2D(64, (3,3), activation ='relu'))
        model.add(Conv2D(64, (3,3), activation ='relu'))
        model.add(MaxPooling2D(pool_size = (2,2)))
        model.add(Dropout(0.3))

        model.add(Conv2D(128, (3,3), activation ='relu'))
        model.add(Conv2D(128, (3,3), activation ='relu'))
        model.add(Conv2D(128, (3,3), activation ='relu'))
        model.add(MaxPooling2D(pool_size = (2,2)))
        model.add(Dropout(0.3))


        model.add(Dense(256, activation = "relu"))
        model.add(Flatten())
        model.add(Dropout(0.3))
        model.add(Dense(12, activation = "softmax"))


        return model
```

3. All the Images and their respective Labels are being saved into 2 separate arrays:

```
data = []
labels = []

for path, _, files in os.walk('../input/nonsegmentedv2'):
    imagesPaths =  sorted([image for image in files])
    #print(imagesPaths)
    for imagePath in imagesPaths:
        image = cv2.imread('../input/nonsegmentedv2/'+ path.split('/')[3] + '/' + imagePath)
        image = cv2.resize(image, (96,96))
        data.append(image)
        labels.append(path.split('/')[3])
```

4. Converting arrays into Numpy arrays:

```
data = np.array(data, dtype='float32')
labels = np.array(labels)
```

5. Splitting of Train and Test data in 3:1 ratio:

```
# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data,labels, test_size=0.25, random_state=42)
```

6. One hot encoding for all 12 target classes:

```
## Here We are just doing the one-hot encoding for the labels.
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)
```

7. Setting up the ImageDataGenerator for Data Augmentation:

```
# construct the image generator for data augmentation
# construct the image generator for data augmentation
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
                         height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
                         horizontal_flip=True, fill_mode="nearest")

"""
Image augmentation allows us to construct “additional” training data from our existing training
data by randomly rotating, shifting, shearing, zooming, and flipping.

Data augmentation is often a critical step to:

1) Avoiding overfitting
2) Ensuring your model generalizes well
"""

# initialize our VGG-like Convolutional Neural Network
model = SmallVGGNet.build(width=96, height=96, depth=3,classes=len(lb.classes_))
```

8. Initializing Adam Optimizer and Starting Training:

```
# initialize our initial learning rate, # of epochs to train for,
# and batch size

EPOCHS = 100
BS = 32

# initialize the model and optimizer
print("[INFO] training network...")

model.compile(Adam(lr=0.0001), loss='categorical_crossentropy',
              metrics=['accuracy'])

# train the network
h = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
                                validation_data=(testX, testY),
                                steps_per_epoch=len(trainX) // BS,
                                epochs=EPOCHS, )
```

9. Code for Printing Classification Report and Plots for visualizing Train loss, Val loss, Train acc, Val acc:

```
# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=32)
print(classification_report(testY.argmax(axis=1),
                            predictions.argmax(axis=1), target_names=lb.classes_))

# plot the training loss and accuracy
N = np.arange(0, EPOCHS)
plt.figure()
plt.plot(N, h.history["loss"], label="train_loss")
plt.plot(N, h.history["val_loss"], label="val_loss")
plt.plot(N, h.history["acc"], label="train_acc")
plt.plot(N, h.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy (SmallVGGNet)")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

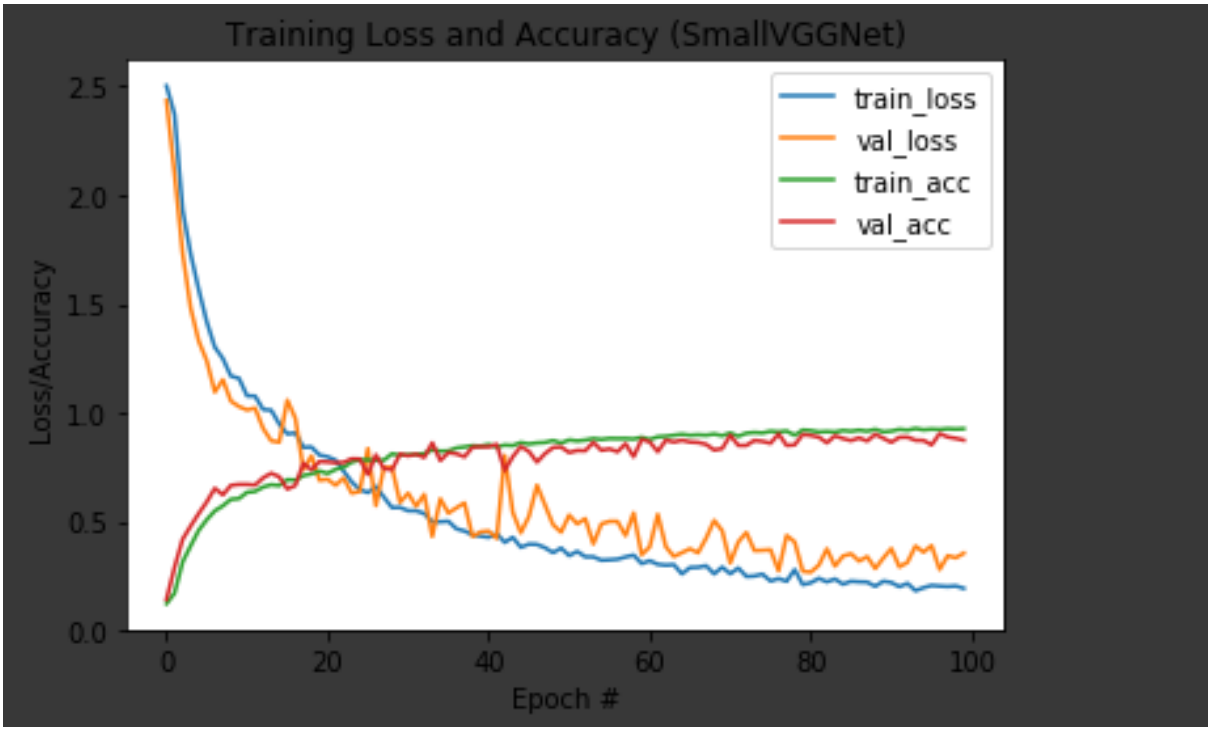
10. Classification Report:

```
[INFO] evaluating network...
                         precision    recall  f1-score   support

             Black-grass       0.57      0.75      0.65        88
                Charlock       0.73      0.96      0.83       114
                Cleavers       0.95      0.54      0.69        78
        Common Chickweed       0.94      0.96      0.95       173
            Common wheat       0.88      0.84      0.86        68
                 Fat Hen       0.96      0.97      0.96       155
        Loose Silky-bent       0.87      0.72      0.79       179
                   Maize       0.87      0.97      0.91        61
       Scentless Mayweed       0.97      0.87      0.92       149
         Shepherd's Purse       0.85      0.95      0.90        56
Small-flowered Cranesbill       0.97      0.98      0.98       158
              Sugar beet       0.91      0.92      0.91       106

               micro avg       0.88      0.88      0.88      1385
               macro avg       0.87      0.87      0.86      1385
            weighted avg       0.89      0.88      0.87      1385
```

11. Plot between Loss and Accuracy:



12. Importing Package for Saving and Loading the Model:

```
[ ]  from keras.models import model_from_json
     import json
```

13. Saving the Model:

```
model_json = model.to_json()
with open("model.json","w") as json_file:
    json_file.write(model_json)
model.save_weights("amodel.h5")
print("Saved model")
```

14. Loading the saved model:

```
#loading saved model

json_file = open('model.json','r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
#load weights into new model
loaded_model.load_weights("amodel.h5")
print("Loaded model")
```

15. Setting Optimizer for Loaded model:

```
#predictiong using loaded model
loaded_model.compile(Adam(lr=0.0001), loss='categorical_crossentropy',
              metrics=['accuracy'])
```

16. Testing the Loaded model for prediction:

```
image = cv2.imread('../input/nonsegmentedv2/Fat Hen/100.png')
image = cv2.resize(image, (96,96))
np_image = np.array(image)
y = np.expand_dims(np_image, axis=0)
pred = loaded_model.predict(y)
print(pred)
pred.shape
```

17: Prediction Output:

```
y_pred_binary = pred.argmax(axis=1)


if y_pred_binary==[0]:
    print("Black-Grass")
elif y_pred_binary==[1]:
    print("Charlock")
elif y_pred_binary==[2]:
    print("Cleavers")
elif y_pred_binary==[3]:
    print("Common Chickweed")
elif y_pred_binary==[4]:
    print("Common Wheat")
elif y_pred_binary==[5]:
    print("Fat Hen")
elif y_pred_binary==[6]:
    print("Loose silky-bent")
elif y_pred_binary==[7]:
    print("Maize")
elif y_pred_binary==[8]:
    print("Scentless Mayweed")
elif y_pred_binary==[9]:
    print("Shepherd's Purse")
elif y_pred_binary==[10]:
    print("Small-flowered Cranesbill")
elif y_pred_binary==[11]:
    print("Sugar beet")
```

PART 2: Making of Flask Web App

1. Importing all the required Packages:

```
import numpy as np
import os
from flask import Flask, request, render_template
from keras.models import model_from_json
from keras.optimizers import Adam
import cv2
```

2. Initializing App and a Directory path for the images which are uploaded:

```
app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))
```

3. Initial routing to upload.html :

```
@app.route("/")
def index():
    return render_template("upload.html")
```

4. Route to prediction api when Image submitted :

```
@app.route("/upload", methods=["POST"])
def upload():
```

5. Loading the model:

```
global loaded_model
global model
# loading saved model
print(" * Loading Keras model...")
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("amodel.h5")
loaded_model.compile(Adam(lr=0.0001), loss='categorical_crossentropy',
                     metrics=['accuracy'])
print("Model Loaded")
```

6. Saving uploaded image to a folder and Predicting the Number Class:

```
target = os.path.join(APP_ROOT, 'images/')
print('Saving Uploaded image into Images Directory')
if not os.path.isdir(target):
    os.mkdir(target)
global destination
for file in request.files.getlist("file"):
    print(file)
    filename = file.filename
    destination = "/".join([target, filename])
    print(destination)
    file.save(destination)
print('Image Saved')
#replacing '\' with '\\' to avoid reading it as a special character
print('Converting Path text to Raw Path Text')
print(destination)
dest = destination.replace("\\", "\\\\")
dest1 = dest.replace("//", "\\\\")
print("done converting")
print(dest1)
print('Reading Image')
image = cv2.imread(dest1)
print('Resizing image to 96x96')
image = cv2.resize(image, (96, 96))
print('Converting image into Numpy array')
np_image = np.array(image)
print('Setting Numpy array dimensions')
y = np.expand_dims(np_image, axis=0)
print('Predicting')
pred = loaded_model.predict(y)
print(pred)
print('It is predicted as:')

y_pred_binary = pred.argmax(axis=1)
print(y_pred_binary)
```

7. Converting binary class into Name class and sending the response:

```
if y_pred_binary == [0]:
    print("Black-Grass")
    y_pred_word = "Black-Grass"
elif y_pred_binary == [1]:
    print("Charlock")
    y_pred_word = "Charlock"
elif y_pred_binary == [2]:
    print("Cleavers")
    y_pred_word = "Cleavers"
elif y_pred_binary == [3]:
    print("Common Chickweed")
    y_pred_word = "Common Chickweed"
elif y_pred_binary == [4]:
    print("Common Wheat")
    y_pred_word = "Common Wheat"
elif y_pred_binary == [5]:
    print("Fat Hen")
    y_pred_word = "Fat Hen"
elif y_pred_binary == [6]:
    print("Loose silky-bent")
    y_pred_word = "Loose silky-bent"
elif y_pred_binary == [7]:
    print("Maize")
    y_pred_word = "Maize"
elif y_pred_binary == [8]:
    print("Scentless Mayweed")
    y_pred_word = "Scentless Mayweed"
elif y_pred_binary == [9]:
    print("Shepherd's Purse")
    y_pred_word = "Shephers's Purse"
elif y_pred_binary == [10]:
    print("Small-flowered Cranesbill")
    y_pred_word = "Small-flowered Cranesbill"
elif y_pred_binary == [11]:
    print("Sugar beet")
    y_pred_word = "Sugar beet"
```

8. Sending response to the results page:

```
    return render_template("result.html",prediction = y_pred_word)
```

9. Running the App:

```
107 if __name__ == '__main__':
108     app.run(debug=True)
```

10. Content of upload.html file:

```
<!DOCTYPE html>
<html lang = "en">
<head>
<title>Plant Pred</title>
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min
     .js"></script>
</head>
<body>

<form id="upload-form" action="{{ url_for('upload') }}" method="POST" enctype
    ="multipart/form-data">


    <input  type="file" name="file" accept="image/*" multiple>

    <input type="submit" value="predict" >
</form>
</body>

</html>
```

11. Content of results.html file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prediction</title>
</head>
<body>
<p style="text-align: center;"><b style="color: #fe2a27;"> It  is a
    {{prediction}} plant.</b></p>
</body>
</html>
```

0.5 Usage

1. Clone the GitHub Repo and extract it.
2. Open terminal.
3. Navigate to the Extracted folder.
4. Install Virtual Environment.
5. Intall packages mentioned in requirements.txt
6. Enter "flask run" command.
7. Navigate to https://localhost:5000/ to use the Prediction App.

0.6 Conclusion

1. Atlast we made and deployed a Plant Prediction WebApp.
2. You can find the Jupyter Notebook Kernel on `https://www.kaggle.com/ramganeshguptha/plant-seedling-classification-among-12-species`.
3. You can find the Code on my Github `https://github.com/Ramganeshsoma/Plant-Prediction-Web-App-using-Deep-Learning`
4. You are free to give a feedback and try leaving a Star on the Repo.