Project Report
on
Weed Recognition using CNN

Name: Vandan Chauhan

Student ID: 200465976

Instructor: Dr. Christine Chan

Date: July 31th, 2022

Table of Contents

Introduction

One of the few hurdles in the matters of effective and productive farming is the growth of weeds which spoil the actual outcome of farming as they affect the growth of planted plants. Recognition of weeds is the problem of accurately identifying the species of weeds so that specific measures can be taken for dealing with them. This includes different types of controlling measures for differently evolved species and different types of herbicides on different species of plants which would result in minimum spraying on the other plants of interest and the soil surrounding it. This can be further used to detect the presence of weeds in a particular region. Removal or controlling of weeds is necessary in order to have healthy crops to maximize yield.

Convolutional Neural Network (CNN) has been used as a visual analyzing tool for the past few years and has shown mammoth success. It has become sort of a standard for image processing in the areas of Machine Learning and Deep Learning. CNN is also widely used in image analysis, image segmentation, and video recognition. The credit for the success can be given to CNN for its ability to have a variety of parameters. This has led us to achievements that were not possible in the past using conventional ANNs. Another important thing about CNN is that the images that are to be processed do not need to have spatially dependent features. In other words, images can have different backgrounds, can be in different positions, and yet be processed.

Knowledge and Data Representation

The Aarhus University Signal Processing Group in collaboration with the University of Southern Denmark prepared this dataset. The dataset consists of 4750 images of unique plants belonging to 12 different species, which can be used to train the model. The testing split of the dataset has 794 images that can be used to test the model. There are images that have been taken at several

different stages of the life of a weed. All the images are in annotated RGB format, comprising 10 pixels per mm.

The original version dataset had been recorded at the Aarhus University Flakkebjerg Research Station, in a collaboration between Aarhus University and the University of Southern Denmark. The version used is inherited from the original, is smaller in size, and doesn't contain images of automatically segmented or single segmented plants.

Approach, Techniques, and Algorithms

In the dataset, all the images are in a ratio of 1:1. However, the sizes vary, i.e. some images are 350 x 350, some are 1030 x 1030, etc. Hence, first of all, it is necessary to convert all the images to one size. To keep the processing time to a minimum without affecting the results to a greater extent, the images are resized to 128 x 128.

Preprocessing

Neural Networks expect the values of a dataset to be numerical. However, the dataset has categories in the form of names, like "Fat Hen", "Maize", etc. Hence, these have to be converted to a numerical format. This is done using LabelEncoder().

Image processing is done using the values of image matrices. For this, since we encoded the values, it's now a vector quantity. Hence, it has to be converted to a binary matrix. This is done by using the keras.utils function. The values are converted to a binary matrix.

Now that we have 'arranged' the data as per what we need, we can start putting it through layers. For that, we first split the training data so we can verify the accuracy after the model is trained once. For that, the train_test_split function of the sklearn library is used.

Weed Type (Text)	Weed Type (Numerical)	Weed Type (Text)	Weed Type (Numerical)
Black-grass	0	Scentless Mayweed	6
Charlock	1	Loose Silky-bent	7
Cleavers	2	Maize	8
Common Chickweed	3	Sugar beet	9
Fat Hen	4	Shepherds Purse	10
Common wheat	5	Small-flowered Cranesbill	11

Table 1.1: Encoded Table

The layers of the images have to be passed sequentially. Since we have no layer with multiple inputs, we can use the Sequential model. The Sequential model, which is imported from the TensorFlow library, is used to pass all the layers of the image, sequentially.

CNN Layers

Spatial Convolution

The image has to be identified as a set of weighted coefficients. Images are essentially a series of 2D data in a discrete form that have been combined linearly.

Relu (Rectified Linear Activation Unit) is used for this layer as well. If the output is positive, the function will spit out the output directly, or else it will return 0. In this way, it prevents

exponential growth of the function and reduces noise. For this, the Conv2D function from the Keras layers library is used.

MaxPooling

The input to this layer is downsampled, i.e. only a few entries from all the present inputs are considered from the input. This is done along the spatial dimensions of the image, which are height and width in this case. For each input channel, the maximum value over an input window is chosen.

Layer Flattening

The spatial dimensions of the input to this layer is collapsed into the dimensions of the channel.

Batch Normalization

The outputs from the previous layers are in no way, normalized. Hence we use Batch Normalization, which applies a transformation formula to the input. This maintains the mean output in the vicinity of 0. The standard deviation is maintained at close to 1.

Dense layers

In a Dense layer, the layer is tightly bound to the values of the preceding layer. The properties of the layers of images that have gone through the CNN functions have changed. Hence, this layer makes changes to the dimensionality of the output, compared to the previous layer. Thus, the CNN model can define the connection between the values of the data that was fed to it.

Dropout Layer

Overfitting might occur where the algorithm won’t perform as per the needs. This happens due to the model learning the detail and noise in the training data to an extent that is unnecessary. This has a negative impact on the performance of the model of new data.

The Dropout layer is added for the deduction of this fault. It randomly fixes the values of input units to 0. It follows a frequency of rate for each step, in the training period. This prevents overfitting.

Dense Layers with softmax activation

Softmax is a mathematical function. All the values in a vector are converted to a vector of probabilities. The values of probabilities are corresponding to the relative scale of each value in the vector.

Compiling with arguments

The model needs to be configured for the training. This is done by the compile function.

Training the model

The model was trained using model.fit for 30 epochs, each with a batch size of 100. The number of epochs was chosen to be such since having a lot of those would possibly bring down the accuracy.

The model was again trained and validated, using the split data, the training images that were split into train and test subsets.

Evaluation of the model

The model was then evaluated using model.evaluate() which returned accuracy and loss parameters for the model.

Viewing in a Tabulated form

For being able to see the results in a proper way, the data was tabulated. The results can be accessed just by clicking.

Structural Diagram and Explanation

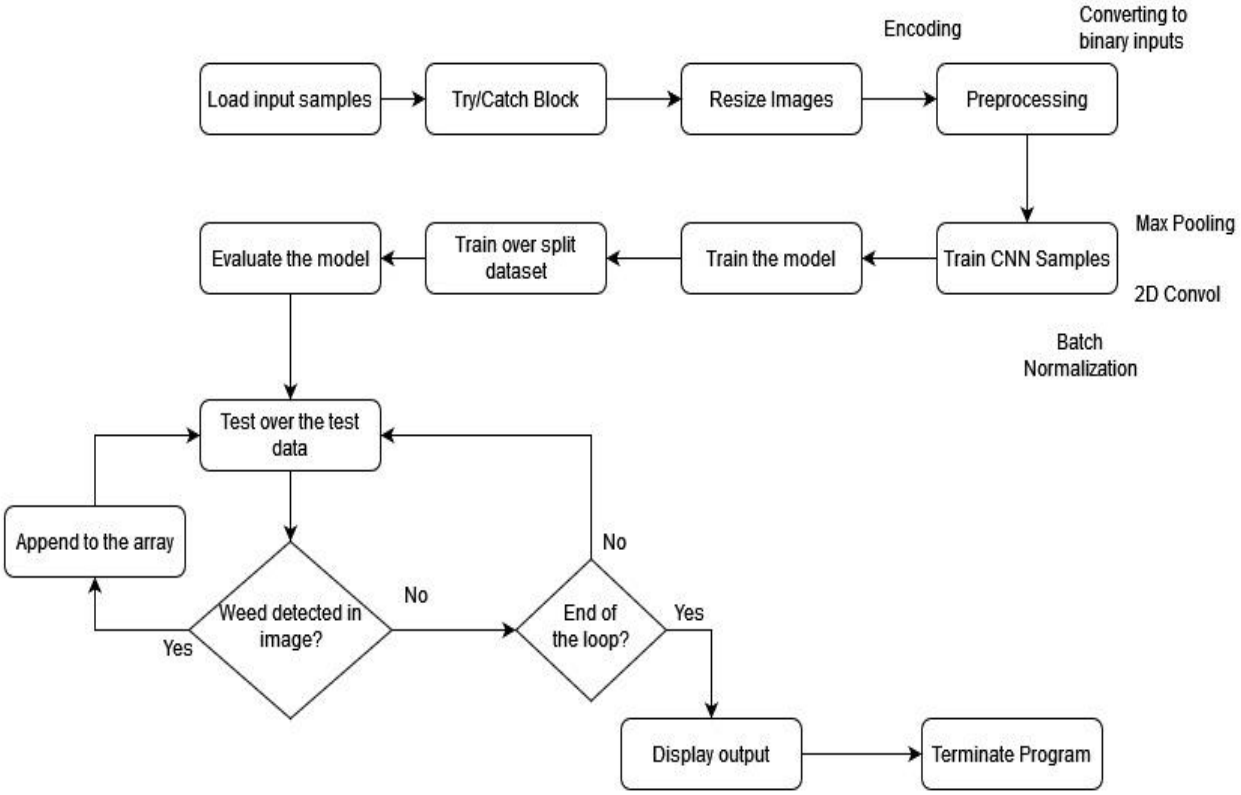

Figure 1.1: Program Flow

- For the project, we first feed the dataset to the program. The images are read from the directories of the respective species.
- We then use a try block, where we resize the images.
- We do some pre-processing on the images, that will allow the algorithm to process them, such as Encoding and Converting to binary inputs, and splitting training data into training and test data.
- We then train the CNN layers using MaxPooling, 2D Convolution, and Densing. We also use Batch Normalization, Flatten layers.

- We then train the model and repeat, but this time using the train-test split as validation data.
- We then evaluate the model and predict the classes of the weeds in the images.
- We repeat this for all images and then terminate the program once we reach the end of the dataset.

User guide

- Dataset: Download the dataset here: https://www.kaggle.com/competitions/plant-seedlings-classification/
- After you download the dataset, extract the folder to your local HDD.
- After extraction, upload the folder to your Google Drive.
- The package contains a Python Notebook.
- Open Google drive on your web browser and upload the notebook to your drive. OR Search for Google Collab on your browser, and open the Python notebook.
- Run the first code snippet by pressing the play button located to the left of it.
- Look up the dataset folder you uploaded to the drive through the pane on the left of your Google Collab webpage.
- Right-click on each folder, i.e. the test and the train folder, copy their paths and paste them into the second snippet of the code which is titled Initialization.
- Select the number of epochs and enter the number of epochs in the snippet titled Training the Model.
- Run each snippet of the code one after one or select Run All from the Navigation Bar.

Sample Sessions

Image processing is done on matrices, hence the list of vectors which is the output from

```
[33] #Converting the list to numpy array
     import numpy as np
     X_train1 = np.array(X_train)
     y_train1 = np.array(y_train)
```

Encoding Labels according to category:

The data of the directory is in text format. It is converted to the numerical format.

```
[4]  #Converting the named categorical labels to one-hot encoded format
     from sklearn.preprocessing import LabelEncoder
     lenc = LabelEncoder()
     y_train2 = lenc.fit_transform(y_train1)
     import tensorflow as tf
     y_train1 = tf.keras.utils.to_categorical(y_train2)
```

We import Layers and Sequential from the TensorFlow library for the layers to be “passed on” to the algorithm sequentially.

```
#Using layers module
from tensorflow.keras import layers

#Using the sequential module to define a model sequentially/in a sequential manner
from tensorflow.keras.models import Sequential
model = Sequential()
```

Here we train the CNN layers by implementing 2D convolutions, Max Pooling, and Batch Normalization. Dropout is used to prevent overfitting.

```
[40] model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(128,128,3)))
     model.add(layers.MaxPool2D(pool_size=(2,2)))
     model.add(layers.Flatten())
     model.add(layers.BatchNormalization())
     model.add(layers.Dense(256,activation='relu'))
     model.add(layers.Dropout(0.4))
     model.add(layers.Dense(12,activation='softmax'))
     model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

The model is trained by using the training dataset. I chose to have 30 epochs and kept the batch size at 100.

```
[44] #Training the model
     model.fit(X_train1,y_train1,epochs=30,batch_size=100)

     48/48 [==============================] - 62s 1s/step - loss: 0.1515 - accuracy: 0.9552
     Epoch 3/30
     48/48 [==============================] - 62s 1s/step - loss: 0.1882 - accuracy: 0.9531
     Epoch 4/30
     48/48 [==============================] - 62s 1s/step - loss: 0.1983 - accuracy: 0.9533
     Epoch 5/30
     48/48 [==============================] - 63s 1s/step - loss: 0.1544 - accuracy: 0.9573
     Epoch 6/30
     48/48 [==============================] - 62s 1s/step - loss: 0.1700 - accuracy: 0.9619
     Epoch 7/30
     48/48 [==============================] - 63s 1s/step - loss: 0.1779 - accuracy: 0.9573
     Epoch 8/30
     48/48 [==============================] - 62s 1s/step - loss: 0.1392 - accuracy: 0.9684
     Epoch 9/30
     48/48 [==============================] - 66s 1s/step - loss: 0.1411 - accuracy: 0.9663
     Epoch 10/30
     48/48 [==============================] - 63s 1s/step - loss: 0.1407 - accuracy: 0.9661
     Epoch 11/30
```

The model was trained again, using the data that was obtained from the train_test splitting.

```
model.fit(x_tr,y_tr,validation_data=(x_vl,y_vl),epochs=30,batch_size=100)

Epoch 1/30
36/36 [==============================] - 51s 1s/step - loss: 0.1325 - accuracy: 0.9725 - val_loss: 0.3321 - val_accuracy: 0.9613
Epoch 2/30
36/36 [==============================] - 51s 1s/step - loss: 0.1836 - accuracy: 0.9674 - val_loss: 0.4799 - val_accuracy: 0.9150
Epoch 3/30
36/36 [==============================] - 51s 1s/step - loss: 0.0933 - accuracy: 0.9745 - val_loss: 0.5788 - val_accuracy: 0.8872
Epoch 4/30
36/36 [==============================] - 51s 1s/step - loss: 0.1039 - accuracy: 0.9756 - val_loss: 0.1125 - val_accuracy: 0.9739
Epoch 5/30
36/36 [==============================] - 51s 1s/step - loss: 0.1068 - accuracy: 0.9789 - val_loss: 0.1591 - val_accuracy: 0.9655
Epoch 6/30
36/36 [==============================] - 51s 1s/step - loss: 0.1258 - accuracy: 0.9725 - val_loss: 0.0820 - val_accuracy: 0.9857
```

Evaluation done on the model showed a 98.86% accuracy.

```
[46] #Evaluating the model
     loss, acc = model.evaluate(X_train1,y_train1)
     print(loss,acc)
```

```
149/149 [==============================] - 19s 129ms/step - loss: 0.0916 - accuracy: 0.9886
0.09156651794910431 0.9886316061019897
```

The prediction was done on the model. This gave out the class that gave out the maximum value at that specific position. To say, if the species found was scentless Mayeweed, it would give out 6 as the output.

```
[47] #we predict the class using the trained model
     y_pred = model.predict(X_test1)
```

```
out = []
for i in y_pred:
  #This helps to find the position at which the value is maximum
  out.append(np.argmax(i))
print(out)
```

```
[10, 6, 8, 3, 6, 3, 0, 6, 6, 3, 8, 10, 6, 8, 6, 6, 11, 8, 6, 3, 3, 10, 0, 9, 11, 7, 9, 3, 8, 2, 3
```

I chose to make a table of the images, so it can be accessed easily. I used pandas for this.

```
[51] import pandas as pd
     #Creating a dataframe & exporting in the csv file format with images & labels columns
     results = {'Labels':out,'Images':a}
     results = pd.DataFrame(results)
```

```
#Checking the dataframe
results.head()
```

	Labels	Images
0	Small-flowered Cranesbill	0021e90e4.png
1	Loose Silky-bent	003d61042.png
2	Scentless Mayweed	007b3da8b.png
3	Common Chickweed	0086a6340.png
4	Loose Silky-bent	00d090cde.png

Sample Listings

```
dummy = cv2.imread('/content/drive/My Drive/vndn_1/plant-seedlings-classification/train/' + i + '/' + j)
dummy = cv2.resize(dummy,(128,128))
```

Image read: Takes the image as an input.

Image resize: Takes the image and resizes it.

```
import numpy as np
X_train1 = np.array(X_train)
y_train1 = np.array(y_train)
```

List to array: Takes the list and converts it to an array.

```
from sklearn.preprocessing import LabelEncoder
lenc = LabelEncoder()
y_train2 = lenc.fit_transform(y_train1)
import tensorflow as tf
y_train1 = tf.keras.utils.to_categorical(y_train2)
```

LabelEncoder: Used the text data as numerical by converting it to numerical.

Transform: Transforms to the format of the argument, i.e. the image set size

Categorisation**:** Takes an array as input and gives a binary matrix as output.

```
from sklearn.model_selection import train_test_split
x_tr,x_vl,y_tr,y_vl = train_test_split(X_train1,y_train1)
```

Splitting train and test: Splits data into train and test subsets.

```
[40] model.add(layers.Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(128,128,3)))
     model.add(layers.MaxPool2D(pool_size=(2,2)))
     model.add(layers.Flatten())
     model.add(layers.BatchNormalization())
     model.add(layers.Dense(256,activation='relu'))
     model.add(layers.Dropout(0.4))
     model.add(layers.Dense(12,activation='softmax'))
     model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

CNN Layers:

Conv2D: Filters- 32, kernel size = (3, 3), ReLU activation, shape of input 128 x 128 with 3 layers.

MaxPool2D: Downsampling along spatial dimensions, in a window size that is defined as pool size (2, 2).

Flatten: Flattens the input. Without any arguments, the output is the same size as the input.

Batch Normalization: Normalizes the output using mean and standard deviation, since there is no parameter.

Densing: Dimensionality of the output - 256, ReLU activation, gives out a tensor with the shape of (batch size, .., dimensionality).

Dropout: Dropout rate - 0.4, the fraction of units to drop to prevent overfitting.

Compile: Optimizer name- adam, a gradient descent algorithm for training. loss -returns a tensor.

Metrics: accuracy - list of metrics to be followed while training and testing, and for outputs.

```
#Training the model
model.fit(X_train1,y_train1,epochs=30,batch_size=100)
```

Train the model for the parameters, for epochs - 30 and batch size - 100.

```
#Evaluating the model
loss, acc = model.evaluate(X_train1,y_train1)
print(loss,acc)
```

Evaluate to predict accuracy, using the training dataset as input.

```
#we predict the class using the trained model
y_pred = model.predict(X_test1)
```

Predicts the class of image using the trained model

```
out = []
for i in y_pred:
  #This helps to find the position at which the value is maximum
  out.append(np.argmax(i))
print(out)
```

Finding out what argument is maximum for a particular image.

```
import pandas as pd
#Creating a dataframe & exporting in
results = {'Labels':out,'Images':a}
results = pd.DataFrame(results)
```

```
#Checking the dataframe
results.head()
```

Display as a table.

```
[ ]  from PIL import Image
     im = Image.open("/content/drive/MyDrive/vndn_1/plant-seedlings-classification/test/0021e90e4.png")
     im.show()
     im2 = Image.open("/content/drive/MyDrive/vndn_1/plant-seedlings-classification/test/03a2ee656.png")
```

```
[ ]  display(im, im2)
```

Display the image.

Discussion

For this project, there were several methods to design and train a model. There were quite a few models which are similar to the one I implemented.

I can say for sure that this model has comparatively better accuracy for almost the same or the same number of epochs, compared to the ones I found during my research. To say in numbers, the model I implemented has a 98.86% accuracy in detecting weeds compared to the one which was similar and had an accuracy of 95.89%. This is due to the overfitting prevention implemented in this model.

Other similar works and methods used

There were an ample amount of similar models. The functions used in them had a few substitutes as well. For instance, a model implemented used the Early stopping function. The use of this would prevent the model from proceeding further if the accuracy improvements aren't significant. Since I already had used the Dropout layer, I decided not to include this. There were a few models that I studied that used real-time applications. Hence, there were a few that calculated the percentage of the soil pixels in the image using a window and predicted the outcomes.

Conclusion

This approach of weed species recognition can be used for the correct identification of weed species. This can be further used for controlling the spread and evolution of the weeds. This approach with this accuracy can be used for farming and general usage, which continue getting influenced by automation. Usage of herbicides should be as little as possible and without harming nearby plants and soil. Weed detection, which is sort of a subsidiary of weed recognition, can be used for that cause. This will not only result in better crop health and farm yields but also help prevent the quality of soil from degrading.

Future Work

Although the detection algorithm using Convolution Neural Network (CNN) was good enough still improvements can be made in the results. The limitation in the above approach faced was sunlight-light intensity. If the variation in the light intensity of the captured images is quite high then the improvement can be done by including the light intensity feature in the detection. A better camera can be used in order to capture much better images with different sensors to overcome the problem that arises due to sunlight variation and shadows.

References

[1] Jiang, Zhang, C., Qiao, Y., Zhang, Z., Zhang, W., & Song, C. (2020). CNN feature based graph convolutional network for weed and crop recognition in smart farming. Computers and Electronics in Agriculture, 174, 105450. https://doi.org/10.1016/j.compag.2020.105450

[2] Yu, Schumann, A. W., Sharpe, S. M., Li, X., & Boyd, N. S. (2020). Detection of Grassy Weeds in Bermudagrass with Deep Convolutional Neural Networks. Weed Science, 68(5), 545–552. https://doi.org/10.1017/wsc.2020.46

[3] Hu, Ma, C., Tian, Z., Shen, G., & Li, L. (2021). Rice Weed detection method on YOLOv4 convolutional neural network. 2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA), 41–45. https://doi.org/10.1109/CAIBDA53561.2021.00016

[4] Dyrmann, M., Jørgensen, ,R.N., & Midtiby, H. S. (2017). RoboWeedSupport - detection of weed locations in leaf occluded cereal crops using a fully convolutional neural network. *Advances in Animal Biosciences, 8*(2), 842-847. doi:https://doi.org/10.1017/S2040470017000206

[5] M. Vaidhehi & C. Malathy (2022) An unique model for weed and paddy detection using regional convolutional neural networks, Acta Agriculturae Scandinavica, Section B — Soil & Plant Science, 72:1, 463-475, DOI: 10.1080/09064710.2021.2011395

[6] Reddy, Rohitharun, S., & Sujana, S. (2022). Weed Detection Using AlexNet Architecture In The Farming Fields. 2022 3rd International Conference for Emerging Technology (INCET), 1–6. https://doi.org/10.1109/INCET54531.2022.9824586