

Plant Species Identification

Introduction

Agriculture, forestry, rural medical, and other trading or commercial applications all require rudimentary plant identification. The most encouraging approach to bridging the gap between the communities of botany and computer science is also automating plant identification. The stem, roots, flowers, leaves, and fruits of plants all have geometrical and aspect characteristics that can be used to identify the species. However, the inter-species differences for many of these traits are frequently subtle and unimportant. However, fruits and flowers are not seasonal, but leaves remain attached to plants for a longer period of time, and they also have somewhat distinctive physical characteristics like texture and form that make them useful for species identification.

Dataset Description

Link to **Leafsnap-dataset**:- <http://leafsnap.com/dataset/>

Leafsnap is an electronic field guide for identifying tree species from photos of their leaves.

The dataset covers all 185 tree species from the Northeastern United States.

Images of leaves taken from two different sources:

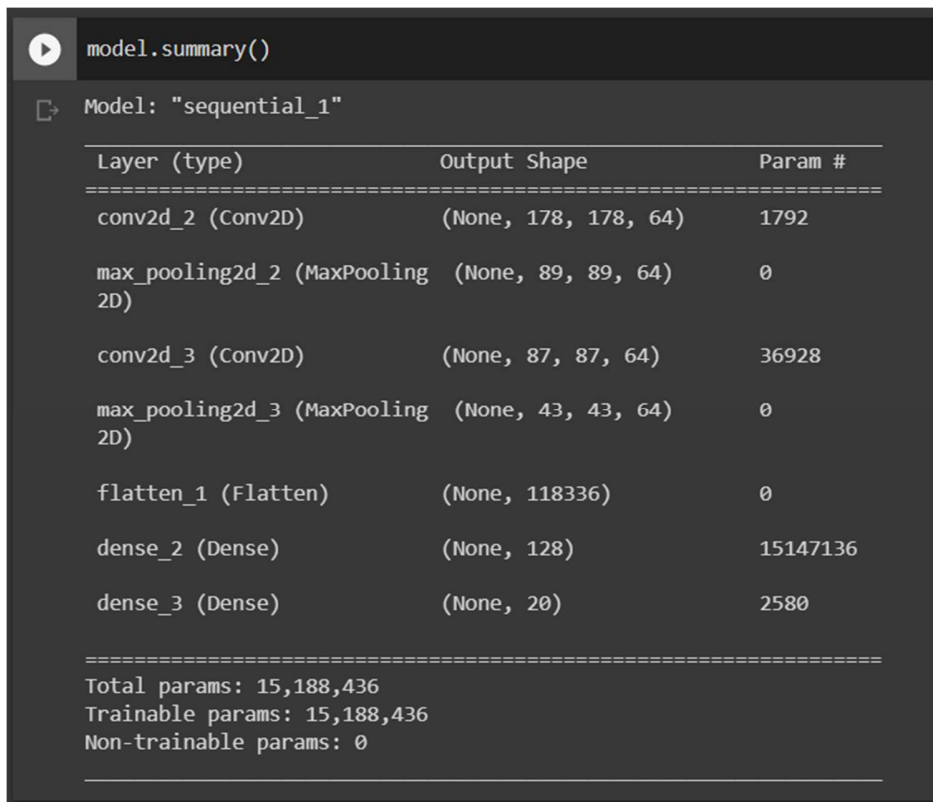
- "Lab" images, consisting of high-quality images taken of pressed leaves.
- "Field" images, consisting of "typical" images taken by mobile devices (iPhones mostly) in outdoor environments.

And for our project we have randomly selected 20 species from the dataset for plant species classification.

Proposed Method

Convolutional Neural Network

A CNN is a particular type of network design for deep learning algorithms that is utilized for tasks like image recognition and pixel data processing.



```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 178, 178, 64)	1792
max_pooling2d_2 (MaxPooling 2D)	(None, 89, 89, 64)	0
conv2d_3 (Conv2D)	(None, 87, 87, 64)	36928
max_pooling2d_3 (MaxPooling 2D)	(None, 43, 43, 64)	0
flatten_1 (Flatten)	(None, 118336)	0
dense_2 (Dense)	(None, 128)	15147136
dense_3 (Dense)	(None, 20)	2580

=====
Total params: 15,188,436
Trainable params: 15,188,436
Non-trainable params: 0
=====

CNN Model Summary

Convolution Layer: Different attributes of the input image are extracted by the convolutional function. Here in our model, we applied 2 convolution model of filter size 64 each.

Pooling Layer: Pooling minimizes the bulk of the feature map. It makes the features strong against distortion and noise. For extracting the max feature from the images, we used MaxPooling layer after each convolution layer.

Activation Function: Linear functions are simple to compute however is limited due to their complexity and that is why non-linear functions are employed. Rectified linear unit (ReLU)

permits for rapid and more productive training by plotting negative numbers to zero and maintaining positive numbers as it is. So, we applied ReLU activation function in our CNN model.

Fully connected Layer: Our fully connected network consists of 2 layers. .i.e., input layer and output layer. The shape of input layer is same as the shape of output generated by the convolution network and the shape of output layer is same as the number of classes available for classification. And the activation function used here is SoftMax. As it helps in multiclassification, SoftMax allocates decimal probabilities to every class in a multi-class problem.

Characteristics	L1		L2	
	Conv	Pool	Conv	Pool
Filter size	3*3	2*2	3*3	2*2
Stride	1	1	1	1
Number of filters	64		64	
Output size	178*178	89*89	87*87	43*43

Proposed CNN Architecture

Transfer Learning Using VGG16 pre trained model

Another approach that is implemented by using a pre-trained network called VGG16 with modifications in the last few layers according to the application. VGG-16 is a convolutional neural network that is 16 layers deep. Transfer learning gives the potential to use the pre-trained networks by tweaking them with application-specific data. The main part of transfer learning is to reuse knowledge obtained in a prior training process, to enhance the learning policy in a new or extra complex mission.

As there was a smaller number of images for each plant species. So, we used data augmentation technique to get more different sets of images. We use data augmentation techniques like RandomRotation, RandomZoom, RandomFlip. By using these techniques, we were able to produce 3 times of our data to feed the VGG16 model.

Below is the representation of VGG16 Model summary.

vgg.summary()

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, None, None, 3)]	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
=====		
Total params: 14,714,688		
Trainable params: 0		
Non-trainable params: 14,714,688		

VGG Model Summary

Below is the representation of Fully connected layer consisting of input and output layer. Shape of input layer is same as the shape of output of VGG and the shape of output is same as the number of classes.

model.summary()

Model: "sequential_3"

Layer (type)	Output Shape	Param #
sequential_2 (Sequential)	(None, 224, 224, 3)	0
rescaling_1 (Rescaling)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, None, None, 512)	14714688
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 205)	5143245
dropout_1 (Dropout)	(None, 205)	0
dense_3 (Dense)	(None, 20)	4120

Total params: 19,862,053
 Trainable params: 5,147,365
 Non-trainable params: 14,714,688

Dense Layer Summary

Dropout is the most well-known and helpful approach in case of overfitting on CNN. The main point of dropout is to randomly drop units and related connections during training. So, similarly to avoid overfitting, we added dropout layer to our model. And at the time of training, it randomly drops 20% of the neuron units.

Model Evaluation

After training the model with our dataset till 11 epochs we find the following observations

- The training accuracy of the model was: 93.77%
- The validation accuracy of the model was: 94.01%

Below figure represents the training performance after each epoch.

```

Epoch 1/11
76/76 [=====] - 30s 169ms/step - loss: 1.4983 - accuracy: 0.5486 - val_loss: 0.6153 - val_accuracy: 0.8286
Epoch 2/11
76/76 [=====] - 10s 129ms/step - loss: 0.6012 - accuracy: 0.8160 - val_loss: 0.3316 - val_accuracy: 0.9218
Epoch 3/11
76/76 [=====] - 10s 130ms/step - loss: 0.4345 - accuracy: 0.8613 - val_loss: 0.2470 - val_accuracy: 0.9268
Epoch 4/11
76/76 [=====] - 10s 131ms/step - loss: 0.3661 - accuracy: 0.8804 - val_loss: 0.2206 - val_accuracy: 0.9268
Epoch 5/11
76/76 [=====] - 10s 133ms/step - loss: 0.2888 - accuracy: 0.9057 - val_loss: 0.1742 - val_accuracy: 0.9468
Epoch 6/11
76/76 [=====] - 10s 135ms/step - loss: 0.2670 - accuracy: 0.9128 - val_loss: 0.1859 - val_accuracy: 0.9501
Epoch 7/11
76/76 [=====] - 10s 135ms/step - loss: 0.2368 - accuracy: 0.9223 - val_loss: 0.1294 - val_accuracy: 0.9584
Epoch 8/11
76/76 [=====] - 10s 136ms/step - loss: 0.2083 - accuracy: 0.9327 - val_loss: 0.1358 - val_accuracy: 0.9601
Epoch 9/11
76/76 [=====] - 10s 138ms/step - loss: 0.1649 - accuracy: 0.9518 - val_loss: 0.1143 - val_accuracy: 0.9667
Epoch 10/11
76/76 [=====] - 11s 139ms/step - loss: 0.1803 - accuracy: 0.9406 - val_loss: 0.1440 - val_accuracy: 0.9418
Epoch 11/11
76/76 [=====] - 11s 142ms/step - loss: 0.1764 - accuracy: 0.9377 - val_loss: 0.1588 - val_accuracy: 0.9401

```

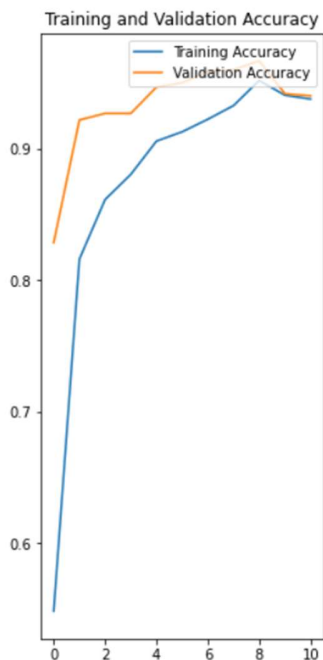
Model Evaluation

Model Validation

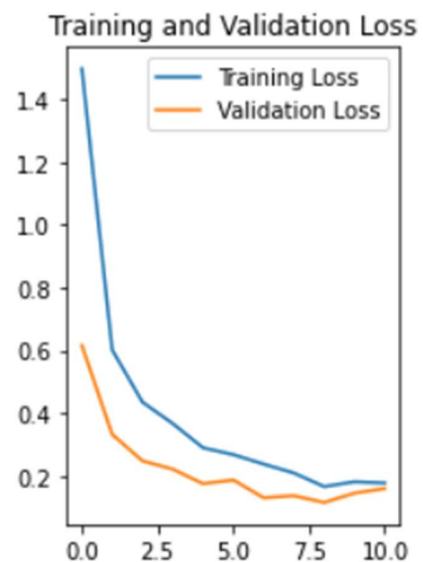
We plotted the training and validation accuracy graph and training and validation loss graph.

Following were the observations we got.

- After each epoch the training and validation accuracy increases.
- After each epoch the training and validation loss decreases.



Training and Validation accuracy



Training and validation Loss

Model Deployment

After deciding on selecting the Convolutional Neural Network for our project, we wanted to implement a way in which our model would be available for everyone to use. For this, we decided to develop a web application and deploy our machine learning model.

For the frontend, we used vanilla HTML to structure the webpage and Vanilla CSS to style it. The UI is kept simple and as easy to understand as possible for the comfort of the user.

For a backend, we decided to use python's flask framework and TensorFlow to load models for making the web application functional.

Finally, we had with us the "TWIGSee" application. A web application that allows users upload the image of a leaf of a plant and then identify the plant's species.



The TWIGSee Application Logo



Our Web Application in action.

Note:

Prerequisite before using the web application: Please download the model from the following link and save it in the "functionality" folder under task 3.

<https://drive.google.com/file/d/1sG0TFTqZrsXKFYqUjwixzRhysJPLQQRf/view?usp=sharing>

Conclusion

Through our project, we have developed a machine learning based application, which allows the user to input images of the leaves of a plant and in return obtain the species of the plant. In large scale, this system can be automated. Moreover, there is even more scope for improvement of the application, by implementing a machine learning model to predict plant species from other features of the plants such as their structures and flowers.

References

1. <https://link.springer.com/article/10.1007/s11831-016-9206-z#:~:text=Building%20accurate%20knowledge%20of%20the,study%20and%20management%20of%20biodiversity>.
2. https://medium.com/@PK_KwanG
3. <http://leafsnap.com/dataset/>
4. https://link.springer.com/chapter/10.1007/978-981-15-9509-7_50
5. <https://towardsdatascience.com/build-your-first-image-classifier-with-convolution-neural-network-cnn-b4e9034ec5cb>