DEFINITION

This project is taken from kaggle competition:

https://www.kaggle.com/c/invasive-species-monitoring/

Project Overview

Tangles of kudzu overwhelm trees in Georgia while cane toads threaten habitats in over a dozen countries worldwide. These are just two invasive species of many which can have damaging effects on the environment, the economy, and even human health. Despite widespread impact, efforts to track the location and spread of invasive species are so costly that they're difficult to undertake at scale.

Currently, ecosystem and plant distribution monitoring depends on expert knowledge. Trained scientists visit designated areas and take note of the species inhabiting them. Using such a highly qualified workforce is expensive, time inefficient, and insufficient since humans cannot cover large areas when sampling.

This problem can be solved using Image classification using neural nets. This is one of the applications of computer vision Domain. This is one of the problems we should consider solving now as its important for the maintenance of ecosystem and it's a burning issue.

Problem Statement

The problem at hand is to analyse the sample images and label them correctly as efficiently as possible, whether a particular sample has dangerous levels of these two invasive species or not. This is a binary classification model with the target label (invasive, present in train_labels.csv), predicting if a sample is dangerous or not. Here each sample is an image of an area of forest evaluated. We can use Image classification to train the neural network model with the already available training data to label the new unseen samples. we can use the accuracy metric of the neural networks to define the accuracy.

Evaluation Metric

We can use the basic accuracy as the evaluation metric. Meaning simply the number of samples it got correctly out of all the samples in test data, calculated in percentages. i.e

Accuracy= (true positives + true negatives)/dataset

size. Since the data set is balanced as is discussed later it suffices to use accuracy as a metric over some thing like F-beta score coz we are not biased about predicting a particular output correctly over the other.

ANALYSIS

Data Exploration

The data set contains pictures taken in a Brazilian national forest. In some of the pictures there is *Hydrangea*, a beautiful invasive species original of Asia. Based on the training pictures and the labels provided, the participant should predict the presence of the invasive species in the testing set of pictures. Each data sample used here is an image which needs to be analysed for excessive growth of invasive plant species. Here each sample image is of size (866, 1154,3) and they all look to be of same size. Each image is RGB channelized . This data set is obtained from the kaggle competition of invasive plant species challenge.

Data Files Description

- **train.7z** - the training set (contains 2295 images).
- **train_labels.csv** - the correct labels for the training set.
- **test.7z** - the testing set (contains 1531 images), ready to be labelled by your algorithm.

Train.7z contains 2295 image samples, where train_label.csv contains the label for that image(image number and label pairs).

```
train_label_df=pd.read_csv("train_labels.csv")
train_label_df.head()
```

	name	invasive
0	1	0
1	2	0
2	3	1
3	4	0
4	5	1

```
train_label_df.invasive.value_counts()
```

```
1    1448
0     847
Name: invasive, dtype: int64
```

as can be seen from above picture we have a single feature and single label here. Here name indicates name of the image which is available in train folder.

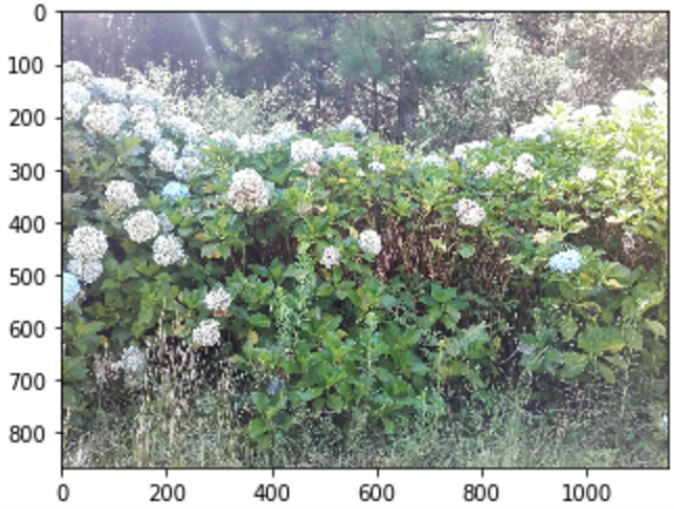
Invasive column represents the label of that particular image. Here we can see the images of different labels are randomly present in the folder. There are a total of 2295 pictures of which 1448 samples have invasive species and 847 samples have non-invasive species.

Link to dataset: https://www.kaggle.com/c/invasive-species-monitoring/data

Exploratory Visualization

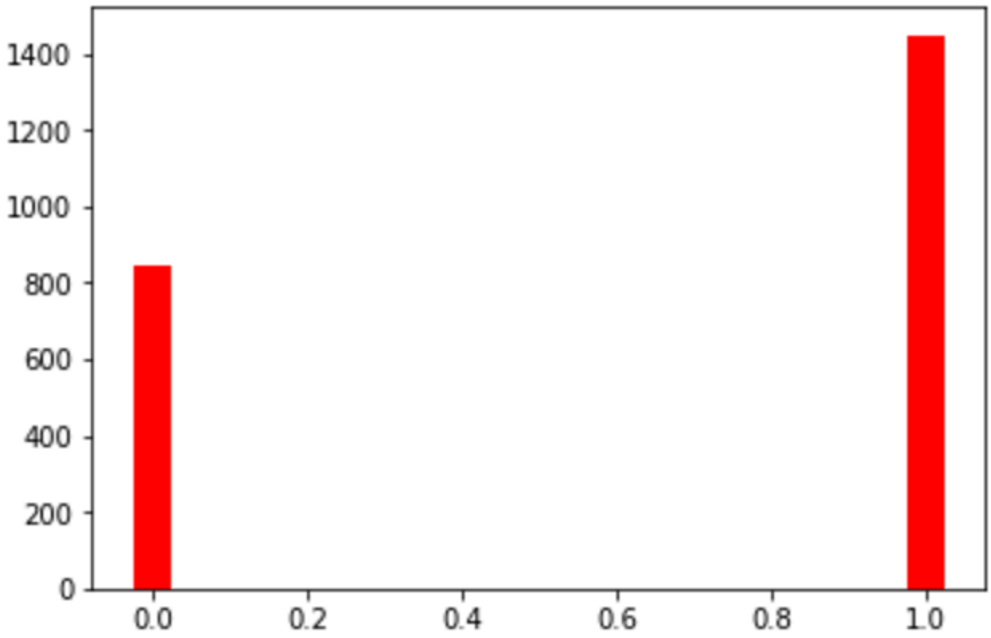
Here we will visualize a single sample from the data available.

```
Populating the interactive namespace from numpy and matplotlib
(866, 1154, 3)
```



so basically here we can see the actual image sample. Here the flowers can be seen everywhere and hence an invasive sample also the size of the image can be seen all the different samples have the same size, but

we will reshape all the images later to a standard size for simplicity.



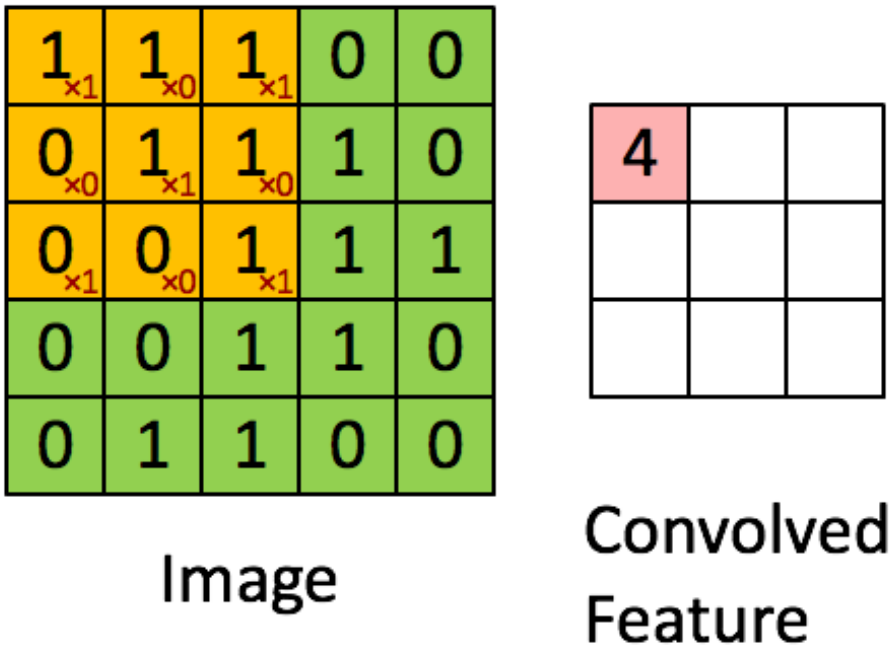
as can be seen from the frequency plot above there are considerable number of samples in each class relatively hence the dataset can be considered as balanced only.

Algorithms and Techniques

we can use deep neural networks for classifying the images. Given the data set is a set of images it would be better to use convolutional neural networks to achieve better accuracy. We can use gradient descent for training the neural networks. we will first split the data into training and testing sets train the model on training set and evaluate the accuracy on the test set finally. A typical convent architecture consists of

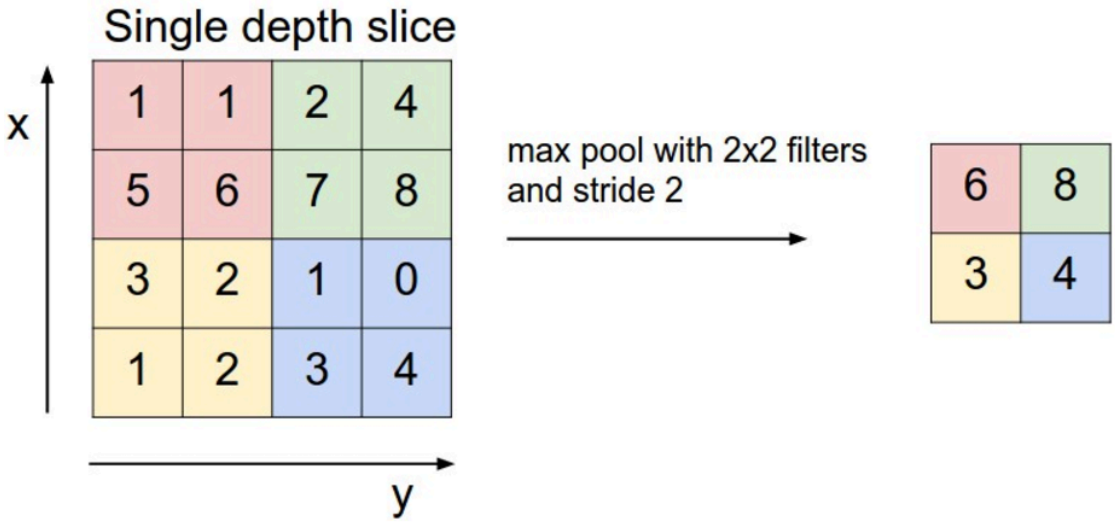
Convolution Layer, Pooling Layer and Fully Connected Layers.

Convolution Layer: conv layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. The output shape of a conv layer usually depends on number of filters used and its taken as one of the inputs to conv layer. Stride is number of pixels by which we move the filter in determining each output value. In this project the filter size used is 3*3 and default stride of 1 is used. Each Conv output size is used in incremented fashion. Below is an image which demonstrates the working of convolutions



as we keep sliding the filter we keep generating the output. with zero padding and stride of 1 the output shape will be the same as input. The activation used is ReLu function which aids with non-linearity and differentiability to the model.

Pooling Layer: It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.There are many variations of pooling techniques available, for this project we are using MAX pooling.The below image demonstrates how max pooling works.



as can be seen from above with a 2*2 filter size, maximum value of each filter segment is selected as the output.

Fully Connected Layer:

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. It's a usual process to use dropouts along with fully connected layer. Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. At each training stage, individual nodes are either dropped out of the net with probability *1-p* or kept with probability *p*, so that a reduced network is left. Since a fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data, drop out is used to prevent overfitting.

Benchmark

We can use a fully connected layer with one hidden layer as a bench mark model. It serves as a basic model

for classification. And we can then try and improve the accuracy using convolutions. A single hidden layer model is neither too naïve nor too complicated which makes a perfect benchmark model.

```
acc = benchMarkModel.evaluate(x_test, y_test)[1]
print("/n")
print('Evaluation accuracy:{0}'.format(round(acc, 4)))
```

```
704/795 [=========================>....] - ETA: 0s/n
Evaluation accuracy:0.7497
```

Have achieved a decent accuracy of approx. ~0.75 accuracy.

METHODOLOGY

Data Preprocessing

A little preprocessing of the data is done

1.First each sample is brought to same shape i.e (64,64,3)

2)the data is split into train and test sets. First 1500 images are taken into train set and remaining into test set. since the image labels are randomly distributed no need for shuffling the data.

Implementation

The model was trained on the preprocessed data as mentioned in the previous section. Here we can see two different models one is the Benchmarkmodel and other is the final model using the convolutions. Few salient points about the implementation:

1.both training and validation images are preprocessed and made ready for training the model.

2.the model is implemented using keras which runs tensorflow in the background.

3.Benchmark model has a single fully connected hidden layer and a final output layer

4.the final model has 3 convolution max pool combos with incremental sizes of 32,64,128 with 3*3 strides in conv2D and 2*2 filter in maxpooling, and then comes two fully connected hidden layers with dropouts at each step for regularization with a final output layer of single dimension.

5.I am using relu activation for internal layers and sigmoid activation for output layers for both models.

6.Used adam for optimisation which is very similar to SGD with a little improvement in efficiency. The loss function used is binary cross entropy.

7.Then the benchmark model is fitted with a single epoch and batch size of 20

8.The final model is fitted with 20 epochs and batch size of 20.

```
In [11]: print(cnn.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 29, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               460900
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               10100
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 564,349
Trainable params: 564,349
Non-trainable params: 0
_________________________________________________________________
None
```

The final model architecture can be seen above. The only challenging part here was to fine tune the parameters to get a good accuracy score.

Refinements

Initially was using high number of layers approx. 10 conv layers and 5 fully connected layers. The model was clearly overfitting. Each conv layer had 128,256,512,1024 output size, combinations of them. Got a really low accuracy of around ~0.63 which was lower than the benchmark accuracy.

The model was not learning after first 4-5 epochs itself.The summary provided in the images below

```
]: model.fit(x_train, y_train, epochs=20, batch_size=20)
   acc = model.evaluate(x_val, y_val)[1]
   print('Evaluation accuracy:{0}'.format(round(acc, 4)))
```

```
Epoch 18/20
1500/1500 [==============================] - 196s - loss: 0.6923 - acc: 0.5380
Epoch 19/20
1500/1500 [==============================] - 196s - loss: 0.6890 - acc: 0.5600
Epoch 20/20
1500/1500 [==============================] - 196s - loss: 0.6893 - acc: 0.5680
795/795 [==============================] - 40s
Evaluation accuracy:0.6239
```

```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_1 (ZeroPaddin (None, 152, 202, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 150, 200, 128)     3584
_________________________________________________________________
zero_padding2d_2 (ZeroPaddin (None, 152, 202, 128)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 150, 200, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 100, 128)      0
_________________________________________________________________
zero_padding2d_3 (ZeroPaddin (None, 77, 102, 128)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 75, 100, 256)      295168
_________________________________________________________________
zero_padding2d_4 (ZeroPaddin (None, 77, 102, 256)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 75, 100, 256)      590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 50, 256)       0
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 39, 52, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 37, 50, 512)       1180160
_________________________________________________________________
zero_padding2d_6 (ZeroPaddin (None, 39, 52, 512)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 37, 50, 512)       2359808
_________________________________________________________________
zero_padding2d_7 (ZeroPaddin (None, 39, 52, 512)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 37, 50, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 18, 25, 512)       0
_________________________________________________________________
zero_padding2d_8 (ZeroPaddin (None, 20, 27, 512)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 18, 25, 1024)      4719616
_________________________________________________________________
zero_padding2d_9 (ZeroPaddin (None, 20, 27, 1024)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 18, 25, 1024)      9438208
_________________________________________________________________
zero_padding2d_10 (ZeroPaddi (None, 20, 27, 1024)      0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 18, 25, 1024)      9438208
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 9, 12, 1024)       0
_________________________________________________________________
flatten_2 (Flatten)          (None, 110592)            0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               28311808
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
=================================================================
Total params: 58,885,249
Trainable params: 58,885,249
Non-trainable params: 0
_________________________________________________________________
None
```

As the number of layers and units reduced with refinement, the final model had achieved a good accuracy and it has learnt fully in the 20 epochs itself as we can see the accuracy becoming kind of stagnant around 18-19 epochs(can be seen in the notebook).The hyper parameters of the final model have been stated in the implementation section. That was the best combination I could get to. The model accuracy achieved with this is ~0.868.

RESULTS

Model Evaluation and Validation

The final architecture and hyper parameters were chosen because they performed best among the tried combinations. The final model which I arrived at consists of 3 conv max pool combinations one flatten layer and three fully connected layers with dropouts except the output layer. The outputs of convolutions are of sizes 32,64,128 respectively. Two fully connected hidden layers of size 100 and a output layer of size 1(coz binary classification). . A summary of this can be seen below.

```
In [11]: print(cnn.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 29, 29, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_3 (Dense)              (None, 100)               460900
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               10100
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 101
=================================================================
Total params: 564,349
Trainable params: 564,349
Non-trainable params: 0
_________________________________________________________________
None
```

Since the final model has achieved a good accuracy of 0.85+ I feel the model and hyper parameters used are appropriate and the parameters are discussed on detail in the implementation section. As can be seen from the model implementation the accuracy has been increasing with each epoch until it became more or less stagnant around 20 epochs and that is where the model would have fully learnt. And the fact that the final training accuracy of 0.97 and the achieved testing accuracy of 0.87 being In the close range adds up for the robustness of the model implying no or very little overfitting happening here. And since the final model is fed with data which has random labels and final accuracy been measured on the unseen data later, I

believe the model is well generalized and robust enough.

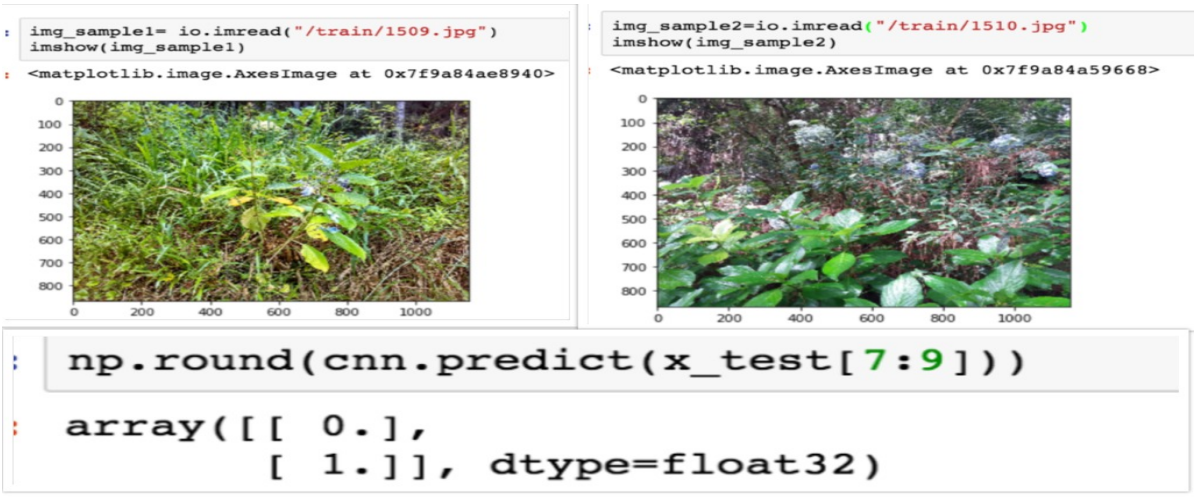
Justification

The final model have achieved a substantial improvement in accuracy over the benchmark model. The final model has achieved an accuracy of 0.86 where as the benchmark has 0.70. With the given set of parameters the model has learnt fully in 20 epochs. With the accuracy achieved this model is significant enough to solve the project.

CONCLUSION

Free_Form Visualization

I



Have taken two random samples from the test data which have a different label just for understanding the model (image numbers and array indices differ coz of forming the test set in a particular fashion can be seen from the code notebook). We can see the model correctly predicting the samples, sample number 1 has no invasive species and sample number two has the invasive species plant (can be seen from the flowers in the image) which is reflected in the prediction. Convolution neural networks learn these patterns in images by themselves. In the first layer they learn the edges and curves and stuff and eventually they start learning objects in the later layers.

Reflection

The problem at hand was a image classification problem. we need to look for invasive species in the image samples available and label the future unseen samples accordingly. Since convolutions are the best for image classification we are using convolutional neural networks (using keras) for classifying the images. The only challenge of the problem is to correctly fine tune the hyper-parameters to arrive at a better accuracy. The final model which we arrived at gives a decent accuracy and satisfactory.

Improvements

The model can be even more fine-tuned for the hyper-parameters leading to a better accuracy. There are more better algorithms available today like pre-trained VGG16 models which give even higher accuracy. This model can be improvised in that area.