# Machine Learning Engineer Nanodegree
## Capstone Project

Sarthak Sahu
19 July 2018

## Invasive Species Monitoring: Identify images of invasive hydrangea (Kaggle Competition)

# I. Definition

## Domain Background

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.

Because scientists cannot sample a large quantity of areas, some machine learning algorithms are used in order to predict the presence or absence of invasive species in areas that have not been sampled. The accuracy of this approach is far from optimal, but still contributes to approaches to solving ecological problems.

Techniques from computer vision alongside other current technologies like aerial imaging can make invasive species monitoring cheaper, faster, and more reliable.

Currently the only techniques used for identifying the invasive hydrangea are through trained scientists only. But using deep learning for identifying plants and plant diseases have been done before. In the **Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks** by Bin Liu et al. This paper proposes an accurate identifying approach for apple leaf diseases based on deep convolutional neural networks. This paper uses AlexNet for identifying Apple Diseases.[1]

Another paper **Deep learning models for plant disease detection and diagnosis** by K.P Ferentinos discusses using CNN for plant disease detection.[2]

[1] *www.mdpi.com/2073-8994/10/1/11/pdf*
[2] *https://www.researchgate.net/publication/322941653_Deep_learning_models_for_plant_disease_detection_and_diagnosis*

# Problem Statement

We are required to develop algorithms to more accurately identify whether images of forests and foliage contain invasive hydrangea or not. The problem requires training a model to do the classification. The problem is a standard image classification problem wherein we will be given labelled images to train our model.

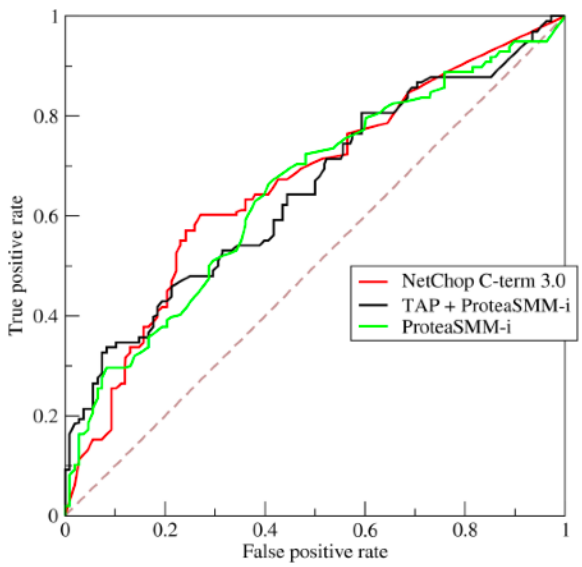
We will use deep learning to identify invasive hydrangea in the images. More specifically, we will use Convolutional Neural Networks as this is a proven technique to classify images.

# Evaluation Metrics

Submissions are evaluated on Area under ROC curve between the predicted probability and the observed target. The ROC curve is a graphical plot that illustrates the performance of any binary classifier system as its discrimination threshold is varied. When we have a binary classifier we get output either 0 or 1. We give a threshold around which we give classification into one of the classes. The performance of the classifier varies as we change the threshold.

So, to plot the ROC curve we plot true positive rate (y axis) vs false positive rate (x axis) at various threshold settings. As we vary the thresholds, we get a curve. The area under the curve gives an idea of how good the classifier is.

True positive rate = Correctly Classified Positives/(Correctly Classified as Positives+ Falsely Classified as Negatives)
False positive rate = Incorrectly Classified as Positives/(Incorrectly Classified as Positives+ Correctly classified as Negatives )



In our case the ROC curve will be drawn between predicted probability of invasive species vs actual observation.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

Since, the problem is meant for a Kaggle competition this provides a good means to generate rankings among the predictions of various competitors.

# II. 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. Training pictures have labels which should be used to train model and testing set contains pictures on which prediction is to be made.

File descriptions

- 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 labeled by your algorithm.
- sample_submission.csv - a sample submission file in the correct format.

Data fields

- name - name of the sample picture file (numbers)
- invasive - probability of the picture containing an invasive species. A probability of 1 means the species is present.

Image Resolution: 1154 x 866

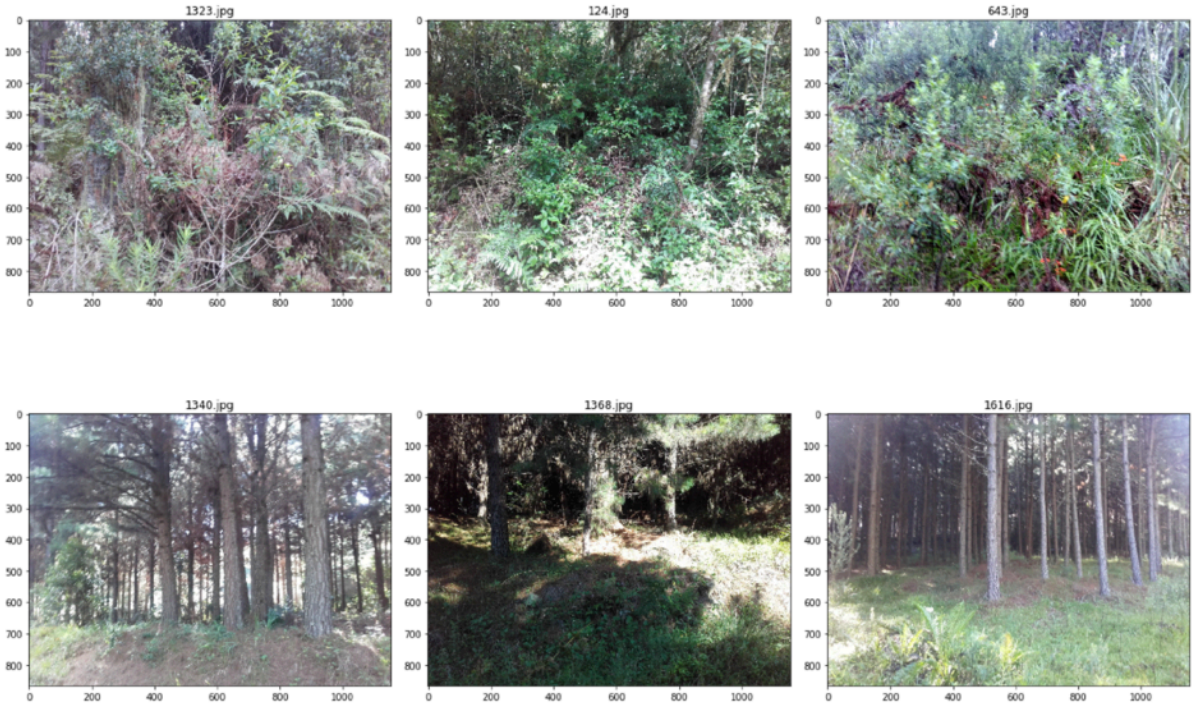
Link to dataset:- https://www.kaggle.com/c/invasive-species-monitoring/data

There are total 847 non invasive species images and 1448 invasive species images respectively in the dataset.

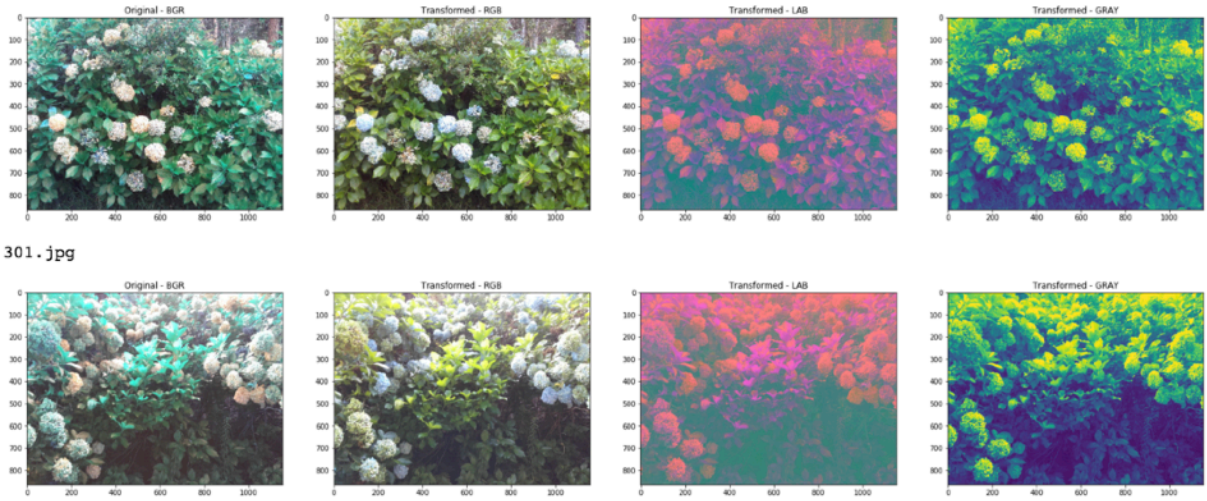
# Exploratory Visualisation

1. Images containing Invasive species

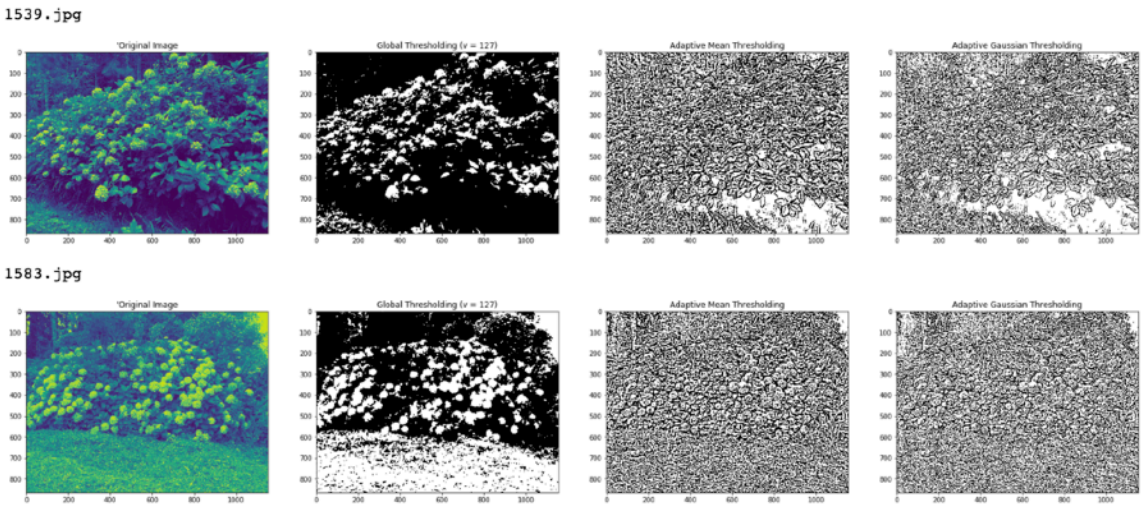

2. Images containing non invasive species



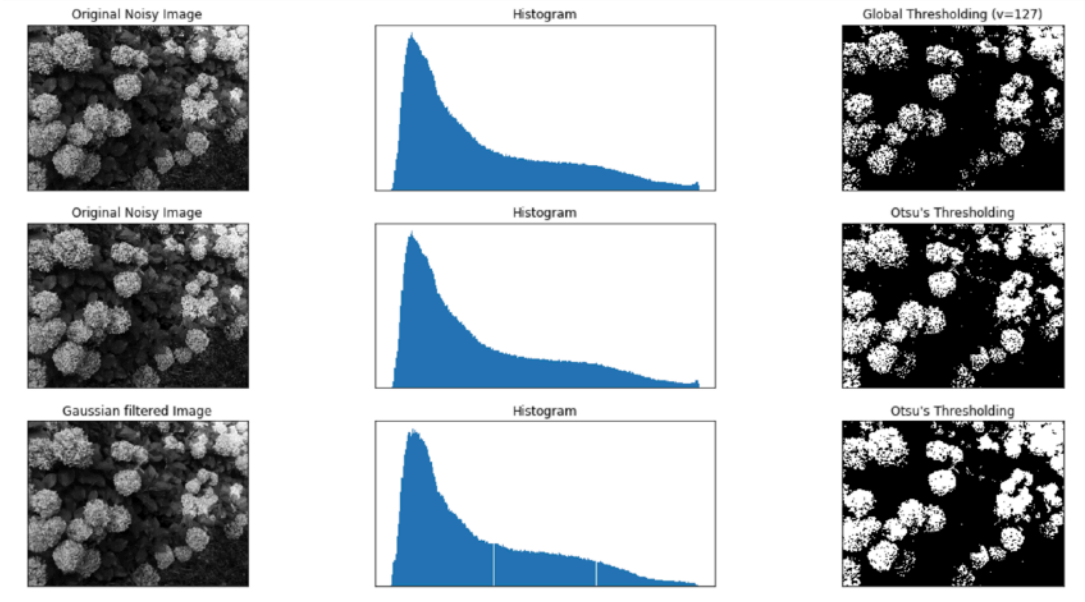
3. Invasive Species with colour transformations




4. Applying thresholding to images containing invasive species




5. Applying Otsu's Thresholding to images containing invasive species



I applied these colour transformations and thresholding techniques to bring out the details in the images containing the invasive hydrangea.

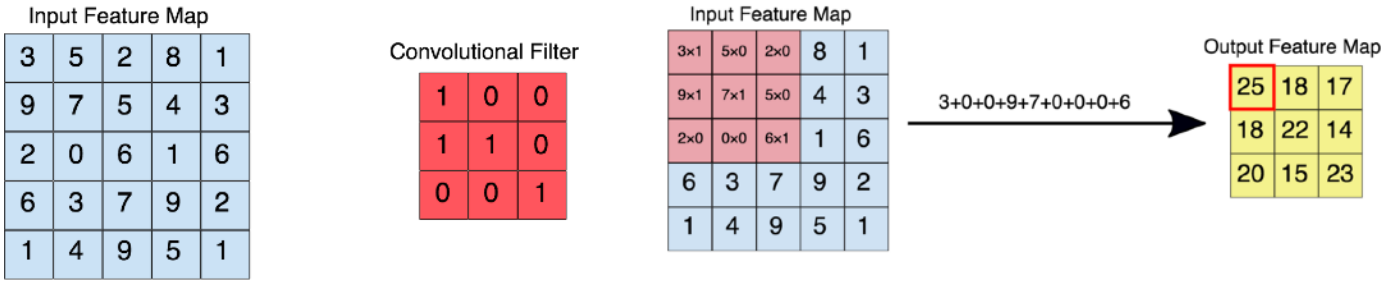
# Algorithms and Techniques

A solution to the problem is training a deep neural network model with the training images. The deep neural network can be Artificial Neural Networks or Convolutional Neural Networks. CNNs are better at classifying images so, it is likely that the solution proposed will have a CNN to perform classification. Perhaps, transfer learning can be used later on to improve the performance of the model. In transfer learning we use pre-trained model weights on similar kind of problem to solve our problem. One example is inception V3 model which can be used for transfer learning.

**Convolutional Neural Networks:**

CNN receives an input feature map, it is basically an image in the form of 3D Array, first and second dimensions are height and width and the third dimension is the colour channel. There are three colour channels Red, Green and Blue. CNN performs a stack of operations namely Convolution, Relu and Pooling.

1. Convolution: Convolution applies filters to the feature map with the help of 3x3 or 5x5 (typically) tiles also called as kernel. The tiles slide over the grid horizontally and vertically. The tiles perform pair wise multiplication with the overlapping region in the feature map and then sums them all. During training the CNN learns the optimal values for filter matrices that enable it extract meaningful features.

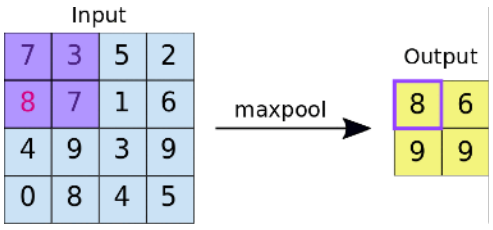



2. Relu: Relu is an activation function applied after each convolution operation. Relu function is-

$$F(x) = max(0, x)$$

it returns x for all values of x > 0 otherwise it returns 0.

3. Pooling: After ReLU comes a pooling step, in which the CNN downsamples the convolved feature (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information. A common algorithm used for this process is called max pooling. Max pooling also operates in a similar way to convolution. In this step we take the max value from the tile. It takes two parameters, size and stride.




We also have **Fully connected layers**. Their job is to learn from the features extracted by convolution and max pooling. In the output layer with either use softmax activation (for multi class

classification) or we use a Sigmoid Activation (for binary classification). Since our problem is a binary classification problem we will use sigmoid activation. Both of these activation give a value between 0 and 1.

## Benchmark

A benchmark model is a simple 2 convolutional layer CNN with max pooling and one fully connected layer. Such a model is a good starting point for image classification tasks. I will make a submission using the model on Kaggle and take the score and rank received as benchmark. I will then try to improve by applying successive better approaches.

# III. Methodology

## Data Preprocessing

The preprocessing steps needed were that data was to be downloaded and stored in folders in such a way that Keras' 'flow_from_directory' could be used easily. Images can be fed to a neural network using image generators directly from the directory. So I used some python scripts to arrange the data and generate validation dataset. The validation dataset I made was balanced, i.e, 200 images each of invasive and non-invasive.

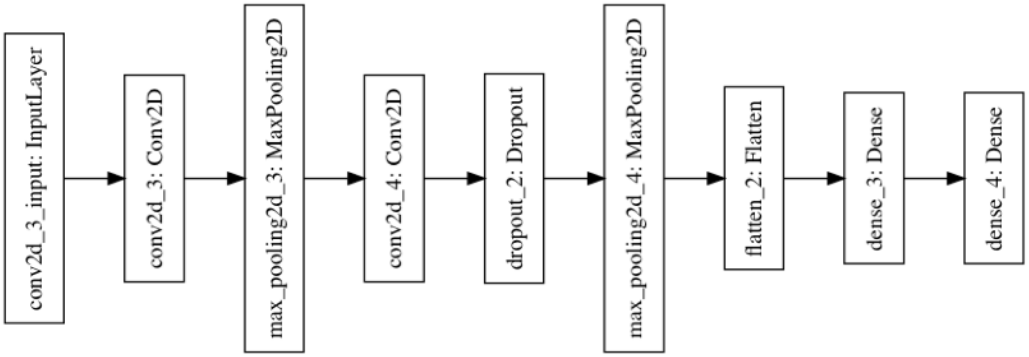
```
import os
import pandas as pd
import shutil

TRAIN_DIR = 'input/train'
INVASIVE_DIR = 'input/train/invasive/'
NON_INVASIVE_DIR = 'input/train/non-invasive/'

os.mkdir(INVASIVE_DIR)
os.mkdir(NON_INVASIVE_DIR)

df = pd.read_csv('input/train_labels.csv')

for img in os.listdir(TRAIN_DIR):
    PATH = os.path.join(TRAIN_DIR, img)
    if os.path.isdir(PATH):
        continue
    label = df.invasive[int(img.split('.')[0])-1]
    if label == 0:
        shutil.copy2(PATH, NON_INVASIVE_DIR)
    elif label == 1:
        shutil.copy2(PATH, INVASIVE_DIR)
```

```
import random

VALIDATION_INVASIVE = 'input/validation/invasive/'
os.mkdir('input/validation/')
os.mkdir(VALIDATION_INVASIVE)

random.seed(2108)
images = random.sample(os.listdir(INVASIVE_DIR),200)

try:
    for img in images:
        PATH = os.path.join(INVASIVE_DIR, img)
        shutil.copy2(PATH, VALIDATION_INVASIVE)
except e:
    print(e)
finally:
    for img in images:
        PATH = os.path.join(INVASIVE_DIR, img)
        os.remove(PATH)
```

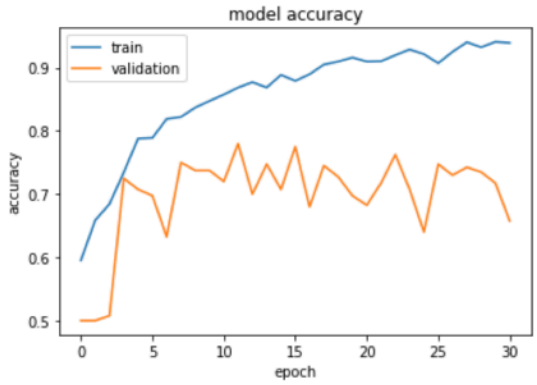
```
VALIDATION_NON_INVASIVE = 'input/validation/non-invasive/'
os.mkdir(VALIDATION_NON_INVASIVE)
random.seed(2108)
images = random.sample(os.listdir(NON_INVASIVE_DIR),200)

try:
    for img in images:
        PATH = os.path.join(NON_INVASIVE_DIR, img)
        shutil.copy2(PATH, VALIDATION_NON_INVASIVE)
except e:
    print(e)
finally:
    for img in images:
        PATH = os.path.join(NON_INVASIVE_DIR, img)
        os.remove(PATH)
```

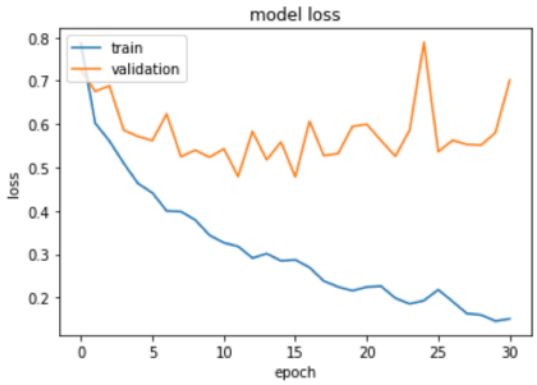
## Implementation

I used Keras with tensor flow back-end for implementing the CNN for predicting the invasive hydrangea in the images. Implementation of neural networks in Keras is very simple. Firstly I started out with a very simple CNN with the architecture given below :




I have coded the basic neural network in the basic_cnn.ipynb jupyter notebook. I trained the model till early stopping. And model stopped training after about 30 epochs.




So, clearly the model was not improving as validation loss and accuracy did not improve much.

Steps to implement the basic model:

1. First we need to feed image data to our model, we can do that using NumPy Arrays. But feeding the entire data, especially when image sizes are large is not possible if system memory is not very high. So, we need to use image data generators. Image data generators are available in Keras, so I used it. For using flow from directory we have already arranged the data in the previous step in folders.

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255)

validation_datagen = ImageDataGenerator(rescale = 1./255)


training_set = train_datagen.flow_from_directory(TRAIN_DIR,
                                                 target_size = (IMG_SIZE, IMG_SIZE),
                                                 batch_size = BATCH_SIZE,
                                                 classes = ['non-invasive', 'invasive'],
                                                 class_mode = 'binary')

validation_set = validation_datagen.flow_from_directory(VALIDATION_DIR,
                                                 target_size = (IMG_SIZE, IMG_SIZE),
                                                 batch_size = BATCH_SIZE,
                                                 classes = ['non-invasive', 'invasive'],
                                                 class_mode = 'binary',
                                                 shuffle=False)
```

2.Then we need to define the architecture of our model, we can do that using the keras.layers module.

```
classifier = Sequential()

classifier.add(Conv2D(64, (3, 3), input_shape = (IMG_SIZE, IMG_SIZE, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier.add(Dropout(DROP_PROB))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

classifier.add(Flatten())

classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

3. We then defined some callbacks like EarlyStopping and Checkpoint. Early stopping stops the training of the model when it stops improving and Checkpoint saves the model and its weights whenever it improves. We monitored the validation loss, whenever validation loss decreased we saved the model. When it stopped decreasing for 15 steps we stopped training the model.

4. Train the model using fit_generator.

```
history = classifier.fit_generator(training_set, nb_epoch = 100, validation_data = validation_set, callbacks=callba
8/8 [==============================] - 73s 9s/step - loss: 0.4023 - acc: 0.8185 - val_loss: 0.5250 - val_acc: 0.7500

Epoch 00008: val_loss improved from 0.56218 to 0.52502, saving model to model-benchmark.h5
Epoch 9/100
8/8 [==============================] - 70s 9s/step - loss: 0.3768 - acc: 0.8368 - val_loss: 0.5401 - val_acc: 0.7375

Epoch 00009: val_loss did not improve from 0.52502
Epoch 10/100
8/8 [==============================] - 72s 9s/step - loss: 0.3420 - acc: 0.8492 - val_loss: 0.5236 - val_acc: 0.7375

Epoch 00010: val_loss improved from 0.52502 to 0.52361, saving model to model-benchmark.h5
Epoch 11/100
8/8 [==============================] - 71s 9s/step - loss: 0.3317 - acc: 0.8541 - val_loss: 0.5433 - val_acc: 0.7200

Epoch 00011: val_loss did not improve from 0.52361
Epoch 12/100
8/8 [==============================] - 74s 9s/step - loss: 0.3235 - acc: 0.8644 - val_loss: 0.4791 - val_acc: 0.7800

Epoch 00012: val_loss improved from 0.52361 to 0.47910, saving model to model-benchmark.h5
```

5. The last step is load the best model that is saved by checkpoint. Then we load all the test images into a Numpy array and we the give predictions.

```
import cv2
import os
import pandas as pd
from tqdm import tqdm

df = pd.read_csv('input/train_labels.csv')

def create_test_data():
    testing_data = []
    for img in tqdm(os.listdir(TEST_DIR)):
        name = img.split('.')[0]
        path = os.path.join(TEST_DIR, img)
        img = cv2.imread(path)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        testing_data.append(np.array([name, np.array(img/255)], dtype=object))
    np.save('test_data-{}.npy'.format(IMG_SIZE), testing_data, allow_pickle=True)
    return testing_data
```

```
testing_data = None
if "test_data-{}.npy".format(IMG_SIZE) in os.listdir():
    testing_data = np.load("test_data-{}.npy".format(IMG_SIZE))
else:
    testing_data = create_test_data()
```

```
testing_x = np.array([i[1] for i in testing_data]).reshape(-1,IMG_SIZE,IMG_SIZE,3)
testing_name = np.array([i[0] for i in testing_data])
```

```
prediction_prob = list(classifier.predict(testing_x).transpose())[0]
prediction_classes = classifier.predict_classes(testing_x).transpose()[0]
```

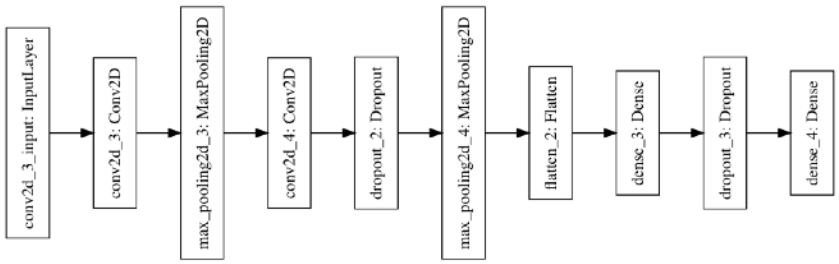
6. Then we do some post prediction analysis to check the performance of our model as done in the notebook with the help of visualisations.

The same steps are followed when we do refinement over the initial process.

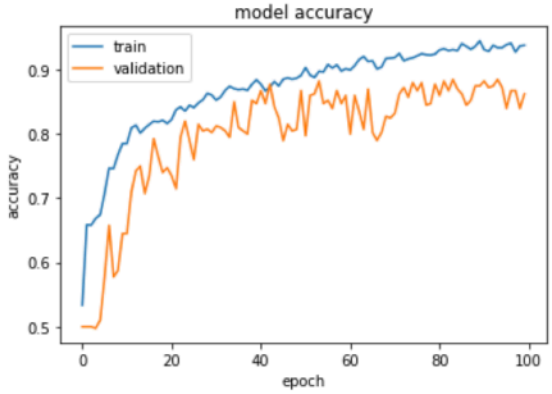
The main challenges with step is feeding the data to the classifier. For that we have already done image folder arrangement and data will be fed using image data generator. Another challenge is the getting test images normalised and getting them in form of an array for the predict method. If one is very familiar in working with Numpy arrays then it will be quite easy otherwise for someone like me, will have to go through documentation frequently. Other steps like tuning the model, modifying the architecture is quite easy as Keras makes these things simple to use with one liners.

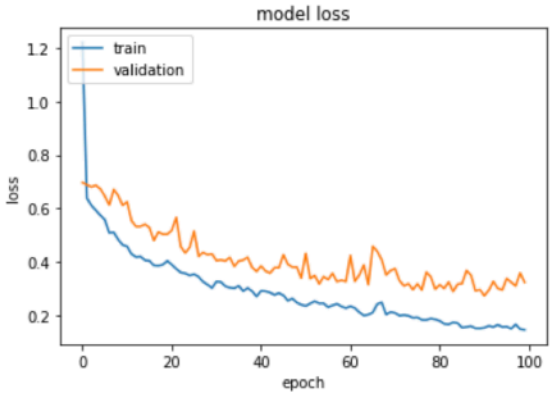
## Refinement

After making the basic model and since it did not give much improvement, I tried to improve the model by adding one more deep layer and added dropout to it to reduce overfitting.

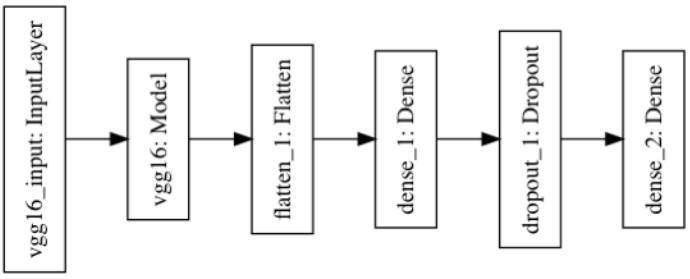



This model is also coded in the basic_cnn.ipynb Jupiter notebook. I trained this model till early stopping, but this model kept on improving. This did not stop early. But seeing the visualisation, it was evident that the model had stopped improving after 60 epochs and it did not make sense to train it further.

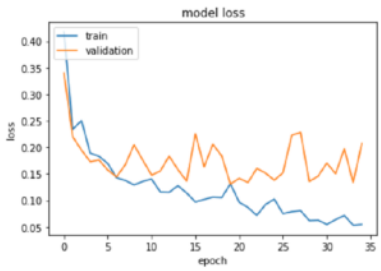



After this, I decided to fine tune a VGG-16 pre-trained model. In the earlier iterations I set all the layers non-trainable. Then in the later iterations, I decided to make last 4 layers as trainable. I added an extra dense layer before a sigmoid output layer. The weights I used in the model were the image-net weights. This way the model was already good at identifying complex geometries. It now only had to adapt to learning invasive hydrangea in the images.







This model too stopped improving after 35 epochs and it was stopped early. This model gave the best result as I expected it to in the beginning. I had also used a larger image size in this model, 150x150 unlike the previous models where I used 64x64 image size. I decided to do this because in the data exploration I found out that model might face difficulty in finding the invasive hydrangea where the hydrangea were small in size in the image. Increasing the size a bit preserves more information. I have implemented this model in a separate notebook names improved_cnn.ipynb.

# IV. Results

## Model Evaluation and Validation

I have evaluated the models based on the metrics discussed.
Area under ROC score as given by submitting the csv file on kaggle gave me the following scores:

1. Basic CNN - 0.86871
2. Basic CNN 2 - 0.86126
3. Improved CNN with VGG16 - 0.97293

The validation accuracies of each model:

1. Basic CNN - 0.78
2. Basic CNN 2- 0.88
3. Improved CNN with VGG16 - 0.95

No of errors by each model on the validation set:

1. Basic CNN - 90/400
2. Basic CNN 2- 47/400
3. Improved CNN with VGG16 - 22/400

The final model has a validation accuracy of about 95% and training accuracy of about 95% too, as this was the point when it had the lowest loss value. This clearly shows that model doesn't overfit and generalises well to data beyond the training set. This is further shown by the fact that there are only 22 errors out of 400 in the validation set prediction.

## Justification

The results obtained by the last model are quite good. It would place me in top 250 in the Kaggle leaderboard for the contest. It showed improvement over the benchmark model so, this can be considered a good solution. I expected the CNN2 to perform better than the CNN1, but it seems that just accuracy is not a good measure. The probabilities given by it for predictions are causing it to get a poor area under ROC score. So, varying the threshold for measuring area under roc, results in poor score for the models.

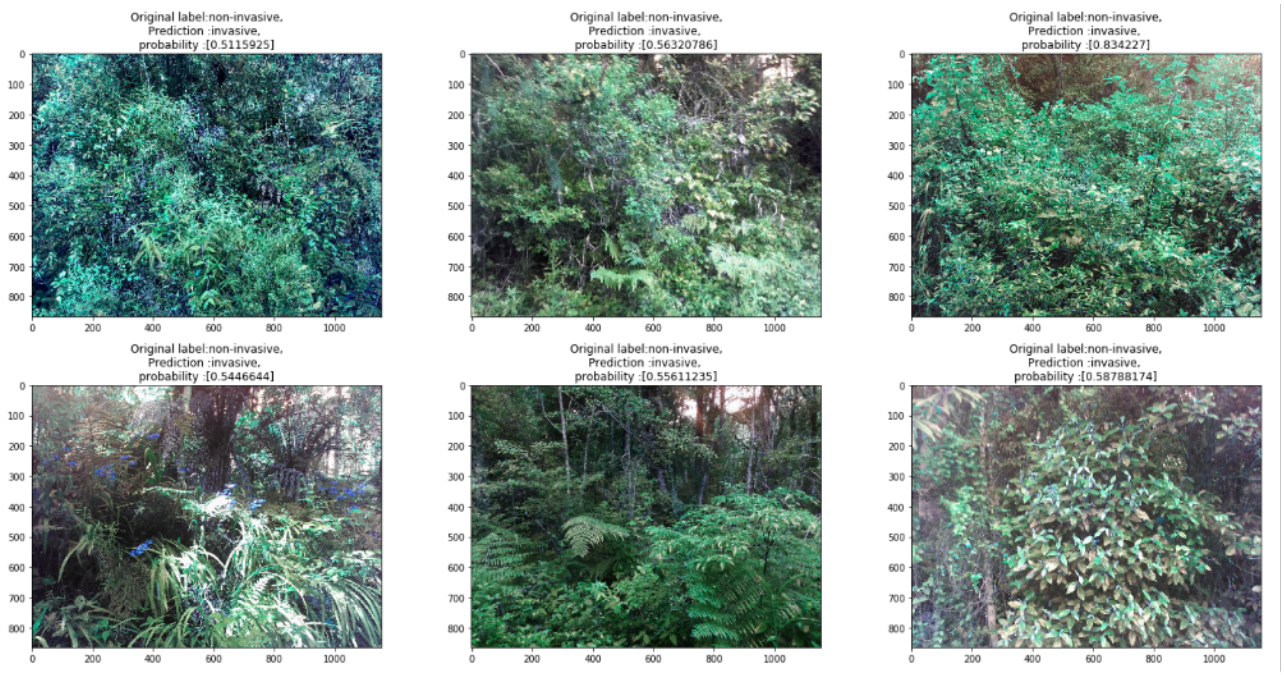
The transfer learning technique works here quite well because the VGG-16 model was trained for 1000 classes and it had already learned to identify many shapes. By training last few layers and adding a dense layer it learned to classify the invasive and non-invasive species. The VGG-16 model is more deeper compared to our basic model so it was expected to perform well. Also one more advantage of using pre-trained weights reduced the training time and gives result within a reasonable amount of time on a not so powerful computer.

# V. Conclusion

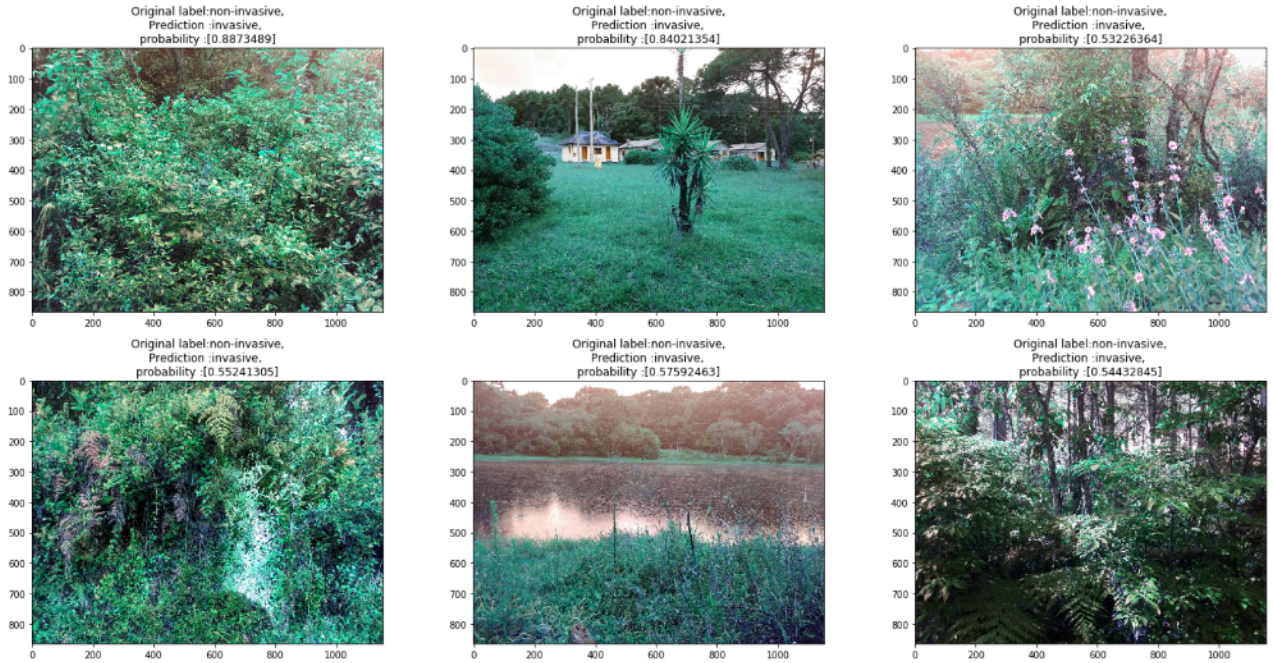
## Free-Form Visualisation

In the notebooks basic_cnn and improved_cnn, I have visualised the erroneous predictions being made by the models. Below are some samples of the wrong predictions of the models.

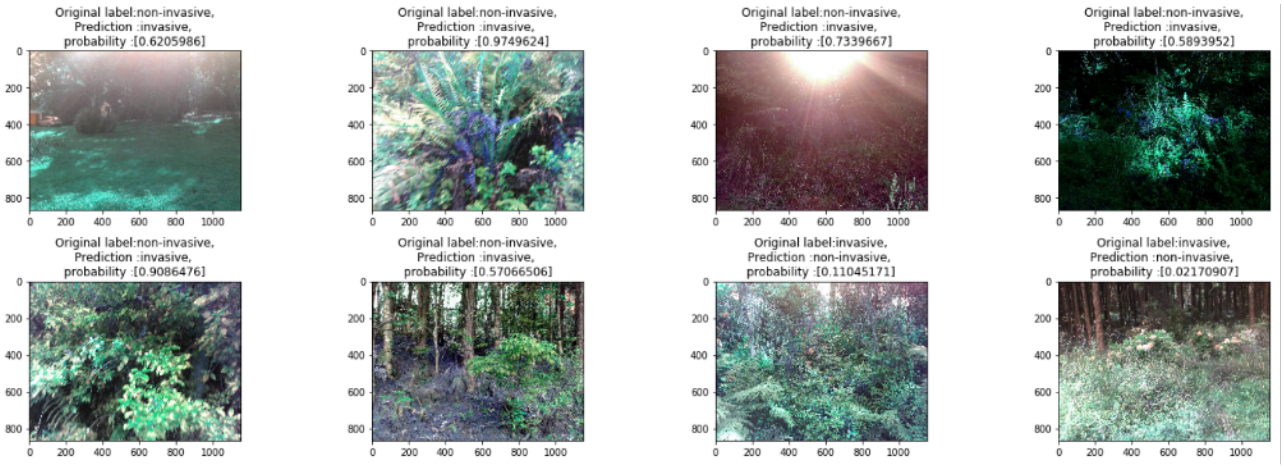
1. Wrong predictions by the basic CNN




2. Wrong predictions by basic CNN 2



3. Wrong Predictions by improved CNN (VGG-16)




# Reflection

This project was a unique experience for me. The projects that I did before had well setup notebooks and visualisations where I had to fill in code. In this project I had to do everything on my own. This was really challenging and I really enjoyed the process. I was overjoyed when I saw that I got a score of 0.97 on the final model after struggling for days to improve the score from around 0.85.

I started working on the project in a Kaggle kernel. Then I realised that it was not very productive to work in the kernel. I moved on to working in google colab notebooks. Earlier I was loading all of the image files in a Numpy array and that would give resource exhausted error on larger image sizes. So, I decided to use image generators. Now arranging the data and getting it to feed to CNN in this way was new for me. So, it was difficult for me, and it took a lot of time. Finally, I downloaded the data on to my own system to improve the productivity.

One more difficult aspect was tuning the parameters. The models take time to train so, tuning the model was a time taking process. I tried different optimisers, ‘Adam’, ‘RMSprop’, ‘sgd'.

I had not made visualisations on my own before. I was basically using the functions in helper files provided by Udacity to visualise. So, doing the visualisations on my own was very rewarding experience and deciding what to look for was as much a challenge as it was fun.

Step by step recap of the actual productive work:

1. Download the dataset and arrange them into folders.
2. Analyse the dataset. Check what invasive hydrangea actually looks like by making some plots and applying the filters.
3. Training a basic CNN.
4. Trying to improve the basic CNN by changing the architecture and hyper-parameters.
5. Make visualisations of the predictions in the step.
6. Using transfer learning. Downloading VGG16 weights and adding our own dense layer to classify the images.
7. Making the last 4 layers as trainable. Tuning some hyper-parameters.
8. Make visualisations of the predictions in the step.
9. Prepare the report.

## Improvement

There is definitely scope for improvement as my rank would be among top 250 according to score I got. I used image size of 150x150 in the model with which I made the final submission. If I would use images of larger size and train the model for longer I would get better results. Training a few more layers of VGG16 might be a good option too. Perhaps some other deeper models like Inception V3, Resnet-50 could have performed better. Using the advanced techniques and better preprocessing, I can get to around top 50.

## References


1. https://www.kaggle.com/c/invasive-species-monitoring
2. https://www.quora.com/Whats-ROC-curve
3. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
4. Transfer learning - Andrew Ng lecture - https://www.youtube.com/watch?v=yofjFQddwHE
5. https://www.learnopencv.com/keras-tutorial-fine-tuning-using-pre-trained-models/
6. developers.google.com