Invasive Species Monitoring: Identify images of invasive hydrangea

Team members: Jiawei Dong 0592393

Problem definition and motivation:

The ecosystem is formed through long-term evolution. After hundred-thousands of years of competition, exclusion, adaptation, and mutual benefit, the species in the system have formed a close relationship that is now interdependent and restrictive. After an alien species invades, it may be excluded from the system because it cannot adapt to the new environment; or it may not be able to compete or restrict its creatures in the new environment. Change or destroy the local ecological environment and severely damage biodiversity.

The invasive hydrangea, as an invasive species, required costly efforts to track the location and spread of them that they're difficult to undertake at scale. So, we need machine learning algorithms to predict the probability of the absence of invasive hydrangea combined with aerial imaging and other techniques to solve the invasive problem in a cheaper way.

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

Solution---The steps of three pre-train model

ALL three pre-train model are transfer learning of ImageNet

Transfer learning is the idea of accumulating knowledge in a model trained for a specific task. For example, the knowledge accumulated by a model in the task of identifying flowers in an image can be transferred to another model to help Predicting a different but related task (such as identifying cats and dogs).

When doing transfer learning, we all default that different tasks have relevance, but how to define relevance and how to mathematically describe the strength of relevance between tasks are subjective decisions that are biased towards humans. We often use ImageNet as a fine-tuning pre-trained model because the large data set of ImageNet itself ensures that the learned network has a high generalization.

Introduction of three pre-train model:

VGG16:

VGG is an abbreviation of Visual Geometry Group Network; the number 16 represents 16 total layers in VGG structure, which includes 13 convolutional layers and 3 fully-connected layers.

An improvement of VGG16 compared to AlexNet is to use several consecutive 3x3 convolution kernels instead of the larger convolution kernels in AlexNet (11x11, 7x7, 5x5). Using a small convolution kernel is better than using a large convolution kernel because multiple non-linear layers can increase the network depth to ensure more complex learning Mode, and the cost is relatively small (fewer parameters).

In VGG, three 3x3 convolution kernels are used instead of 7x7 convolution kernels, and two 3x3 convolution kernels are used instead of 5 * 5 convolution kernels. By doing so, under the same perceptual field, the network becomes more in-depth; thus, the effect of the neural network improved. For example, the superposition of three 3x3 convolution kernels with a step size of 1 can be regarded as a receptive field of size 7 (in fact, it means that three 3x3 continuous convolutions are equivalent to a 7x7 convolution). The total parameter is 3x (9xC ^ 2). In a single 7x7 convolution kernel, the total parameter is 49xC ^ 2, where C refers to the number of input and output channels. 27xC ^ 2 is less than 49xC ^ 2, which means that the parameters are reduced, and the 3x3 convolution kernel is better for maintaining the image properties.

Inception V3:

The principle of building the inception v3 model:
Avoid the representational bottleneck, which is a large-scale compression of feature maps caused by pooling.

The increase in features will speed up the training process.The increasing number of mutually independent features will cause the input decomposed more thoroughly, and the correlation between the decomposed sub-features is low. The internal correlation between sub-features is high. It is more accessible to convergence with groups of strong correlation feature.

Similar to VGG, the inception v3 model also decompose the 5 * 5 7 * 7 convolution kernel into multiple 3 * 3 convolution kernels. Besides, it used asymmetric convolution kernel, which decomposes n * n convolution kernel into n * 1 n * 1 convolution kernel.

ResNet:

ResNet solves the problem of the difficult training of deep CNN models. In 2014, VGG had only 19 layers, and in 2015, ResNet has as many as 152 layers. Besides the depth, residual learning makes the depth of the network work. For a stacked-layer structure, when the input is 'x', the learned feature is recorded as 'H(x)', now we hope that it can learn the residual 'F(x)= H(x) -x' so that the original learning feature is 'F(x) +x'.

Residual learning is easier than direct feature learning. When the residual is 0, the stacking layer only does identity mapping at this time. At least the network performance will not decrease; in fact, the residual will not be 0, which will also make the stacking layer learn new features based on the input features.

The ResNet network consults to the VGG19 network, and add the residual unit through the short circuit mechanism. The change is mainly reflected in ResNet directly using stride = 2 convolution for downsampling, and replacing the fully connected layer with the global average pool layer. An important design principle of ResNet is that when the size of the feature map is reduced by half, the number of feature maps is doubled, which maintains the complexity of the network layer.

Implementation of three pre-train model:

Before implementation any model:
1:importing all necessary libraries.
2:Read all images and labels data and split training set and validation set.
3:Normalized all images data to help computer read imformation better
Code:

```
x_train /= 255
x_valid /= 255
```

4:Random shift, rotate, horizontal flip the training images. The original images are all taken at a similar angle. This step can help model extract more important features and reduce overfitting. Only the training images are needed to perform these steps.
Code:

```
train_datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True)
train_datagen.fit(x_train)
```

VGG16 implementation:

Code and model structure are shown in the google colab notebook.

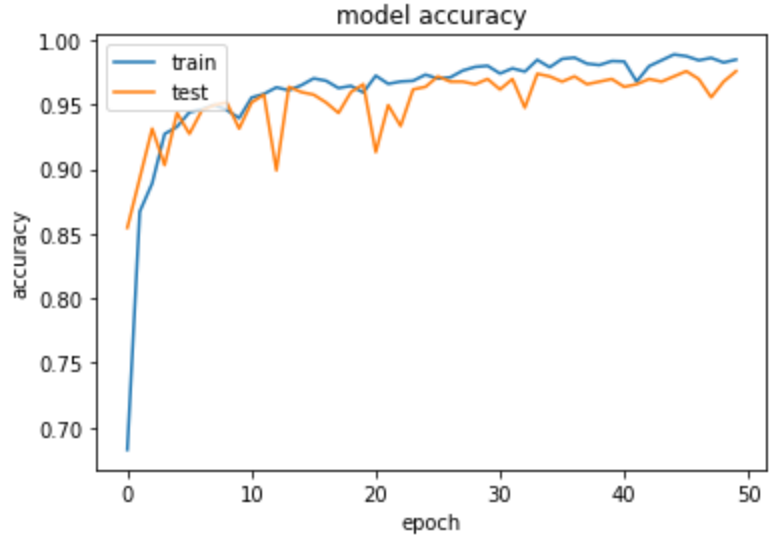
The downloading pre-train model is from: https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5. There is total 16 layers and 25m parameters.
In the compile of the model: the activation is sigmoid because the prediction results are 0 and 1. After the pre-train model, I add a flatten layer and two dense layers to obtain the results. SGD optimizers are applied with 0.0001 learning rate and 0.9 momentum which accelerates SGD in the relevant direction and dampens oscillations.

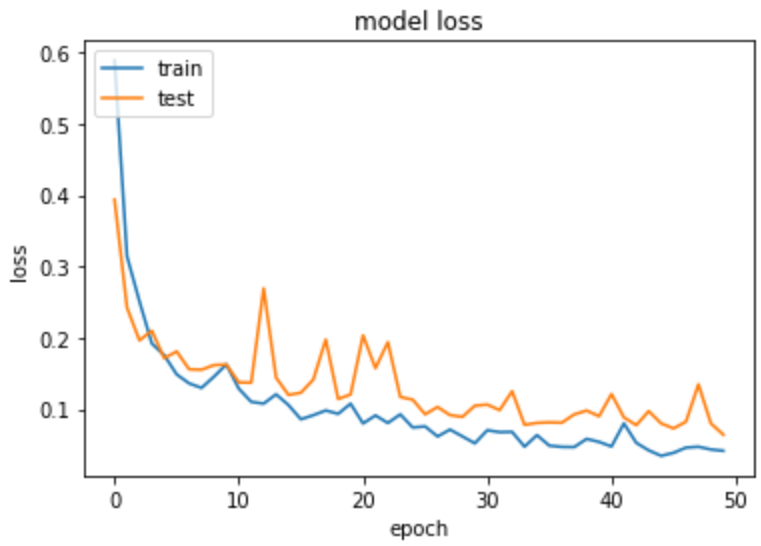
Result of VGG16:

The total running time is around 17 mins.

There is 0.99 for the training accuracy and 0.9758 for the validation accuray

```
Epoch 50/50
56/56 [==============================] - 49s 872ms/step - loss: 0.0237 -
acc: 0.9916 - val_loss: 0.1011 - val_acc: 0.9758
```





Inception V3 implementation:

The downloading pre-train model is from:
https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
In the inception V3 model, I use Adam optimizer with 0.001 learning rate, which can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum.

```
optimizer = optimizers.adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, decay=0.0)
```

For the callbacks, I use `ReduceLROnPlateau`. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

```
reduce_lr = ReduceLROnPlateau(monitor='val_acc',
                              patience=5,
                              verbose=1,
                              factor=0.1,
                              cooldown=10,
                              min_lr=0.00001)
```

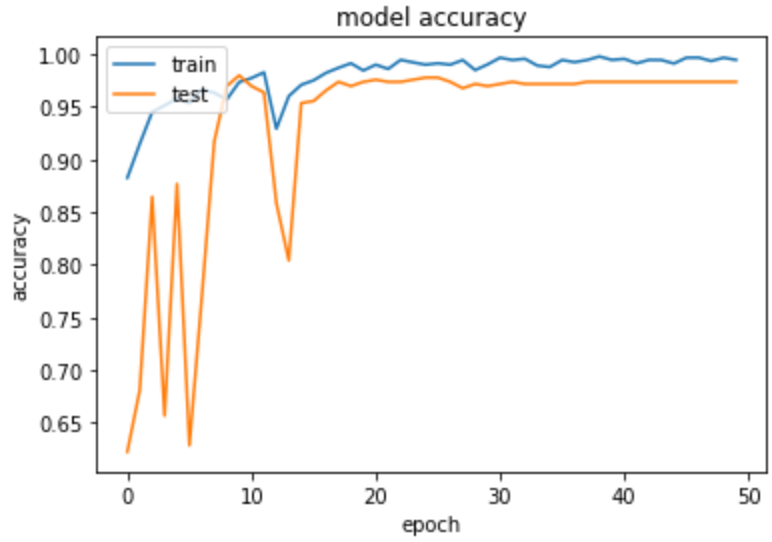
Result of Inception V3:

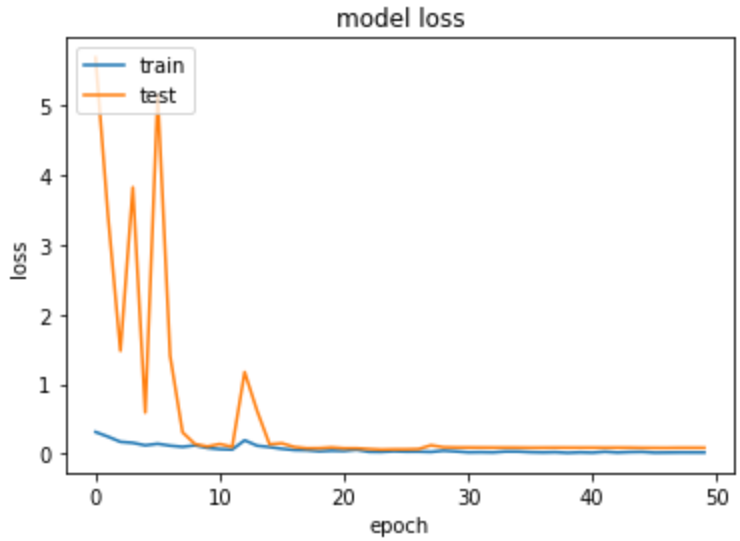
The total running time is around 17 mins.

There is 0.9946 for the training accuracy and 0.9737 for the validation accuray

```
Epoch 50/50
```

```
- 19s - loss: 0.0165 - acc: 0.9946 - val_loss: 0.0838 - val_acc: 0.9737
```





As you can see in the loss curve and accuracy curve, in the earlier epochs, there is bouncing and that is the point the ReduceLROnPlateau callback reduce the learning rate.

ResNet 50 implementation:

The downloading pre-train model is from:
https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models
In ResNet 50 model, I used SGD optimizer with 0.0001 learning rate and 0.9 momentum. I use the binary_corssentropy loss function.

Because of ResNet 50 has need a high performance GPU, by the GPU provide by google colab, I have to freeze some layers to process in a shorter time.

```
for layer in model_resnet.layers[:175]:
    layer.trainable = False
for layer in model_resnet.layers[175:]:
    layer.trainable = True
```

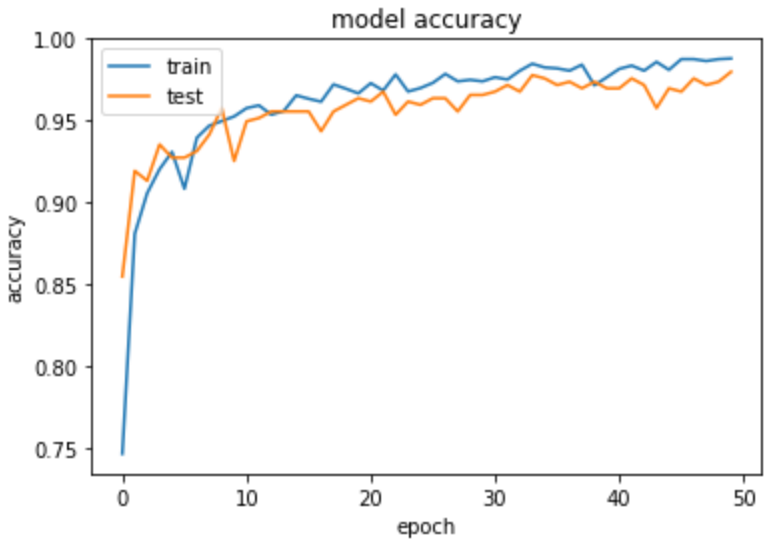
Even I did that, each epoch still take around 20mins to train. The accuracy of the 5 epoch is:0.9066, and val_accuracy is `0.91371715189279185`.

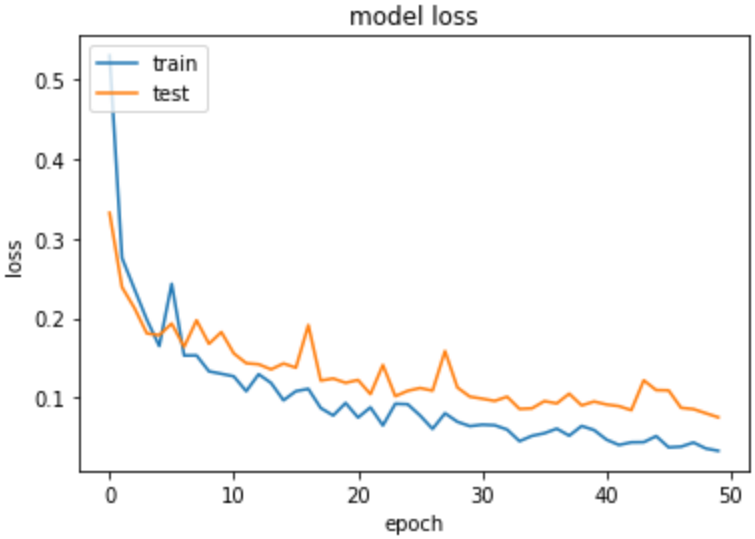
```
Epoch 5/5
1800/1800 [==============================] - 1178s 655ms/step - loss:
0.2164 - acc: 0.9066
```

Validate code:

```
preds = model_resnet.evaluate(x_valid, y_valid)
print ("Loss = " + str(preds[0]))
print ("Test Accuracy = " + str(preds[1]))
```

So, I use the same python code in anaconda to use my own GPU to compute the training process with more epoch and save the model in `content/ResNet50_weights.hdf5`.



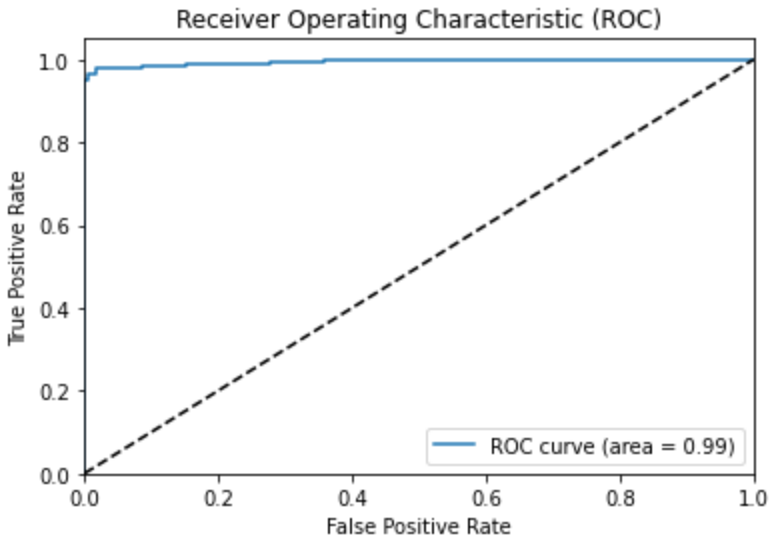


The Accuracy curve and loss curve are generated by anaconda with 50 epoch traning.

Ensembled system:

This is a binary graphic classification problem, the results are true or false. So, a majority voting can be applied to combine three pre-train model.

The idea is, set the true as 1 and false as 0. If there are two or three models predicted true for the testing image, the prediction output will be true. This means, the final output is the majority of the prediction. The predict output is a probability, and according to the



ROC to convert the probability for each model to the true all false, I use the equation (prediction>0.95).

Code:

```
#using majority voting to combine three models
def prediction_es(img):
    pred_v3=model_v3.predict(np.array( [img,] ) )>0.97
    pred_xgg=model_v3.predict(np.array( [img,] ) )>0.97
    pred_resnet=model_resnet(np.array( [img,] ) )>0.97
    pred_final=int(pred_v3)+int(pred_xgg)+int(pred_resnet)>=2
    return pred_final
```

For the accuracy of the ensemble model:

```
#Analysis the accuracy of ensemble system
pred_combine = np.array([])
#the length of y_valid is 495
for i in range(len(y_valid)):
  pred_combine = np.append(pred_combine,int(prediction_es(x_valid[i])))

accuracy=np.sum(pred_combine==np.array(y_valid))/len(y_valid)

print("The Accuracy of ensembled model is: "+ str(accuracy))
```

```
The Accuracy of ensembled model is: 0.9854771784232366
```

Which have a improvement of each model.

Video Output:

Same as last time, I made two videos, one is with a camera and one is an uploaded image. Because I run three models at the same time, there is an execution delay, so in the camera detection video I make a seperate camera screen to have a fluence video(same as last time). Because the camtasia trial expired, I used another trial software, and that software is not allowed to modify anything so I keep it original.

As I explained in the last report, this project is to detect whether there is invasive hydrangea in the forest picture, so the output can only tell if there is invasive hydrangea in the camera or not. The dataset doesn't have any object of invasive hydrangea itself but only the forest picture with or without it. Thus I can't perform any object detection with a rectangle. I don't have dataset to do so.

Comparison of three pre-train model and keras model:

VGG16:

Compare to my keras model. The structure of VGGNet is very simple. The entire network uses the same size of convolution kernel size (3x3) and maximum pooling size (2x2).

The combination of several small filter (3x3) convolutional layers is better than one large filter (5x5 or 7x7) convolutional layer:

It is verified that the performance can be improved by continuously deepening the network structure.

VGG consumes more computing resources and uses more parameters resulting in more memory usage. The increase of parameters is not caused by the factorization of 7x7 to 3x3.Most of the parameters are from the first fully connected layer and VGG has three fully connected layers.

Inception V3:

There are too many parameters in VGG, mainly the fully connected layer, and the calculation is not efficient; the main reason is the sparse structure. Inception V3 tries to reduce the number of fully connected layers (reducing parameters) and uses a dense structure that improving computational efficiency. Inception V3 used asymmetric convolution kernel, which decomposes n * n convolution kernel into n * 1 n * 1 convolution kernel to reduce total calculations. Inception V3 has a balanced width and depth.

The defects of Inception V3: the structure is complex, and it is more difficult to modify. If directly increase the width and depth, then Inception's primary advantages (improving computing efficiency, reducing parameters) will decrease.

ResNet 50:
Deeper networks will be accompanied by gradient disappearance/explosion problems, thus hindering the convergence of the network. This phenomenon of deepening the network depth but decreasing network performance is a degradation problem. ResNet was born to solve this degradation problem.

Deeper networks will be accompanied by gradient disappearance/explosion problems, thus hindering the convergence of the network. The author calls this phenomenon of deepening the network depth but decreasing network performance as a degradation problem. ResNet was born to solve this degradation problem.

In ResNet, a pooling replaced the fully connected layer. Pooling does not require parameters, compared to the fully connected layer; a large number of parameters can be cut off. By doing so, it can save computing resources, and prevent the model from overfitting.

Global average pooling is used instead of max pooling. The experimental results of some of the papers I have reviewed show that the effect of average pooling is slightly better than that of maximum pooling, but the effect of maximum pooling is not much worse. In my model, I choose global average pooling.