CS6005 DEEP LEARNING

MINI PROJECT 2 – 11/05/2021

Computer Vision with Transfer Learning Application

Name: Reuel Samuel Sam
Registration Number: 2018103053
Batch: P

Abstract – Transfer learning is the process of creating new AI models by fine-tuning previously trained neural networks. Instead of training their neural network from scratch, developers can download a pre-trained, open-source deep learning model and fine tune it for their own purpose. Image classification is an active research area and has been studied in popular applications such as driverless vehicles and emergency robots. Plenty of models have been pre-trained to classify numerous images. In this project, VGG16 model is adopted to classify between images from popular film franchise 'Aliens vs. Predators'

I. PROBLEM STATEMENT

This project tackles to classify images taken from popular movie franchise *'Aliens vs. Predators'* with the application of transfer learning. Here, Oxford's VGG16 model has been adopted as the base model to be fine-tuned and trained upon to classify the images in the dataset. Transfer Learning is popularly used when there appears to be a lack of training samples to train a model from scratch. For this purpose, a dataset was taken with relatively small amounts of training and testing samples with – 494 training images and 200 testing images with each image having 250x250 pixel resolution.

P.T.O

II. DATASET DETAILS

Alien vs. Predator dataset was developed to act as a replacement of Cat vs. Dog dataset for the purposes of transfer learning. This data set is relatively small, as expected of a transfer learning dataset. It has 494 training images with 247 images labeled under ‘Alien’ and 247 images under ‘Predator’. Furthermore, 200 images have been provided as validation set with 100 images labeled under ‘Alien’ and 100 images labeled under ‘Predator’. This dataset was originally used to compare the Keras and PyTorch regarding their performance when it comes to transfer learning. [1]

This is a binary classification dataset with only 2 target classes:

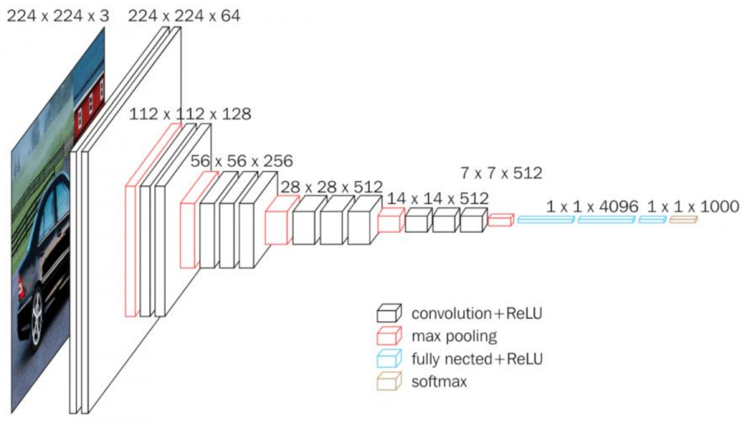
- ⇨ Aliens
- ⇨ Predators

III. PRE-TRAINED MODEL

VGG16 is a 15 layered convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. [2]

The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.

VGG16 was trained for weeks and was using NVIDIA Titan Black GPU’s. The architecture diagram for VGG16 is given below:



IV. MODULES

Python comes with a rich library of modules that make coding rather convenient. This is no different when it comes to Deep Learning. Python has plenty of Deep Learning Libraries that can be made use of. Following are the modules that have been imported and applied in this project

- ⇨ *TensorFlow:* Open Source Machine Learning Platform
- ⇨ *Keras:* Open Source Library that provides python interface for artificial neural networks
- ⇨ *Matplotlib:* Graphical Library (Not necessarily required)
- ⇨ *Seaborn:* Graphical Library
- ⇨ *Pillow:* Image Library

Apart from Python modules, the code can be broken down into modules that run independently:

- ⇨ *Loading of Pre-Trained Model:* Oxford's VGG16 model has been loaded in from Keras library. The trainable layers have been frozen and set as false so as to not train over those layers. The weights imported are the ones from VGG16's trained model for the classification of ImageNet dataset.
- ⇨ *Loading of Data:* The dataset has been loaded from Kaggle's rich collection of machine learning datasets. The Structure of the image dataset is standard with two folders containing training and validation jpg images.
- ⇨ *Preprocessing of Data:* Instead of using NumPy library to perform manual preprocessing, this project makes use of ImageDataGenerator that loads images as it is required instead of loading the images in prior and holding it in memory. The images are sheared, zoomed and horizontally flipped. The training data is then split 85% as training data with 15% of data being set aside for validation. The images found in the validation folder are then taken as testing data. A batch size of 16 is defined so as to train in batches of 16 and the training and validation test is shuffled as well. Normalization is also carried out on the image pixel values.
- ⇨ *Model Addition:* To the existing pre-trained VGG16 model, a Global Average Pooling Layer is first added to reduce the size of the output from the previous layer and then flatten it. In addition to that, 2 fully connected dense layers (128 and 64) with dropouts and batch normalization is added. These dense layers use ReLU activation function while the dropout layer is provided with 0.3 parameter value. Finally, a 2 Neuron Dense layer with sigmoid activation is added at the end for final classification.

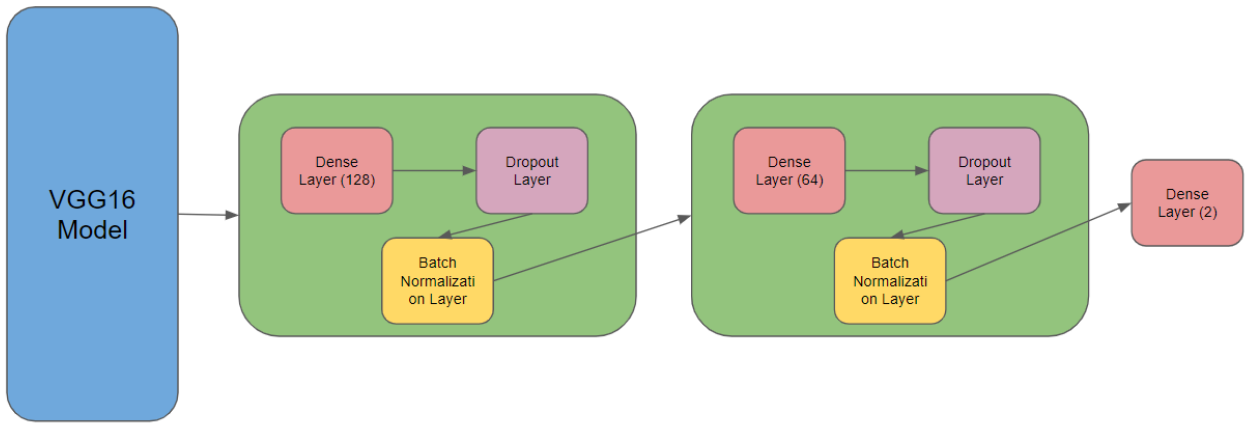
- ⇨ *Hyperparameters:*
    - Epochs: 35 (There was no significant improvement past 10-11 epochs)
    - Optimizer: Adam
    - Learning Rate: 0.001
    - Batch Size: 16
    - Loss Function: Sparse Categorical Cross Entropy

V. MODEL SUMMARY

The Model Architecture adopted to address the problem at hand builds on VGG16's CNN Structure. The different layers added are:

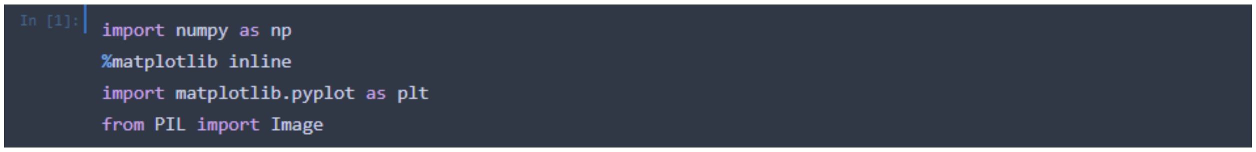
- ⇨ *Dense Layer*
    - Three Dense Layers are used
        - Layer 1: 128 units, ReLU Activation
        - Layer 2: 64 units, ReLU Activation
        - Layer 3: 2 units, Sigmoid Activation
- ⇨ *Dropout Layer*
    - Random inputs are set to 0 during training while the inputs are scaled up. This is to avoid the problem of over-fitting
    - Two dropout layers are used with 0.3 parameter value
- ⇨ *Batch Normalization*
    - Standardizes the input at each layer for a mini batch
    - Rescaling of data to have a mean of 0 and standard deviation of 1
    - Two batch normalization layers are used at the end of the dropout layers

The Model Architecture is shown below:

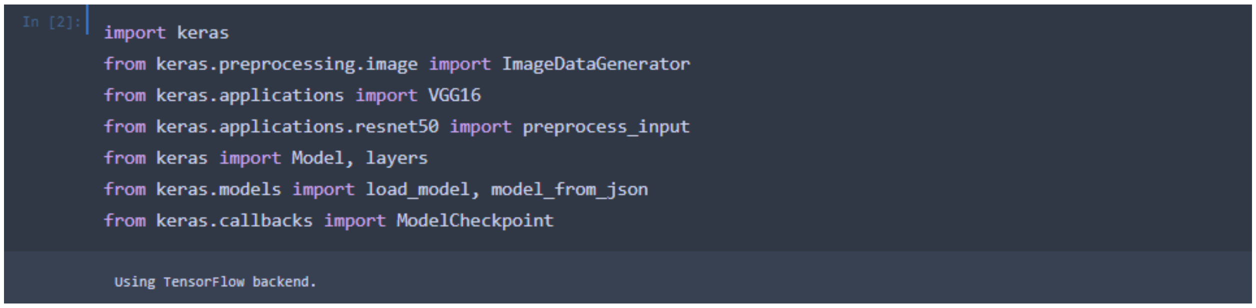


VI. CODING SNAPSHOTS

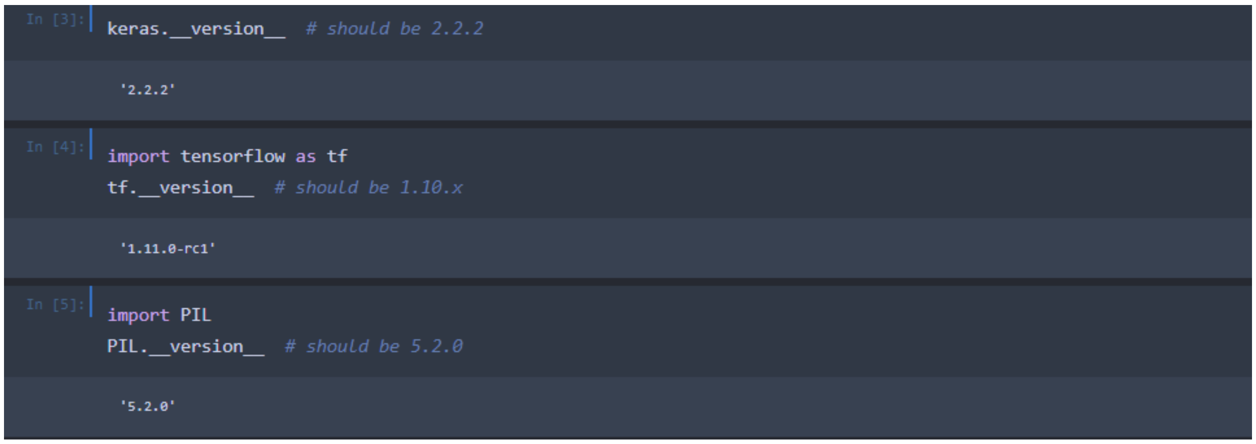
1) Imports



```
In [1]: import numpy as np
        %matplotlib inline
        import matplotlib.pyplot as plt
        from PIL import Image
```

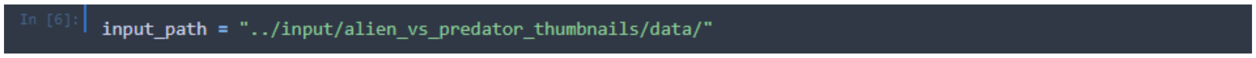


```
In [2]: import keras
        from keras.preprocessing.image import ImageDataGenerator
        from keras.applications import VGG16
        from keras.applications.resnet50 import preprocess_input
        from keras import Model, layers
        from keras.models import load_model, model_from_json
        from keras.callbacks import ModelCheckpoint

        Using TensorFlow backend.
```



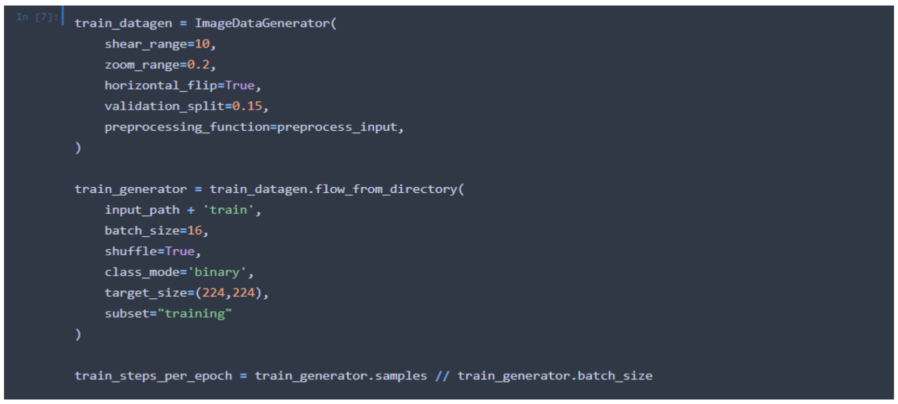
```
In [3]: keras.__version__  # should be 2.2.2

        '2.2.2'

In [4]: import tensorflow as tf
        tf.__version__  # should be 1.10.x

        '1.11.0-rc1'

In [5]: import PIL
        PIL.__version__  # should be 5.2.0

        '5.2.0'
```

2) Loading of data set

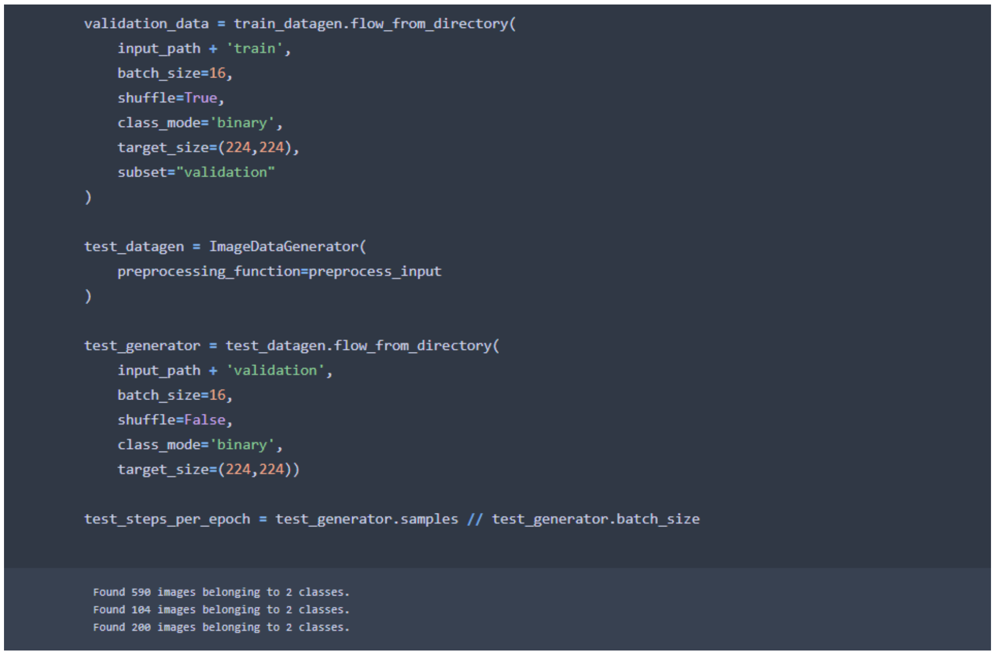


```
In [6]: input_path = "../input/alien_vs_predator_thumbnails/data/"
```

3) Data Generators and Pre Processing



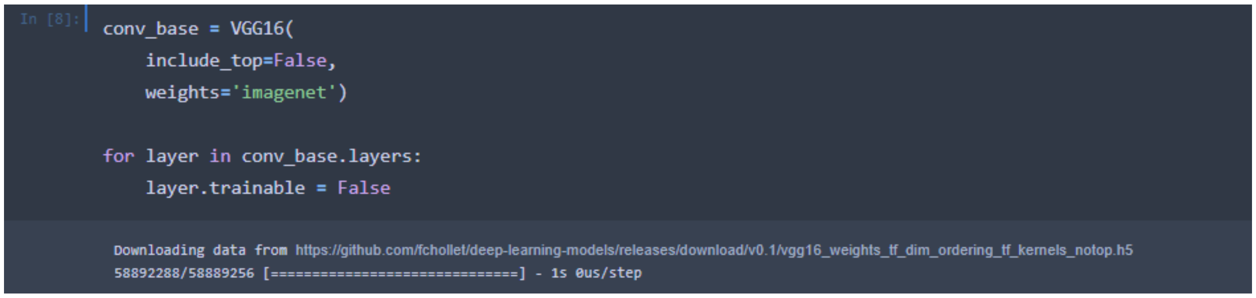
```
In [7]: train_datagen = ImageDataGenerator(
            shear_range=10,
            zoom_range=0.2,
            horizontal_flip=True,
            validation_split=0.15,
            preprocessing_function=preprocess_input,
        )

        train_generator = train_datagen.flow_from_directory(
            input_path + 'train',
            batch_size=16,
            shuffle=True,
            class_mode='binary',
            target_size=(224,224),
            subset="training"
        )

        train_steps_per_epoch = train_generator.samples // train_generator.batch_size
```



```
validation_data = train_datagen.flow_from_directory(
    input_path + 'train',
    batch_size=16,
    shuffle=True,
    class_mode='binary',
    target_size=(224,224),
    subset="validation"
)

test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input
)

test_generator = test_datagen.flow_from_directory(
    input_path + 'validation',
    batch_size=16,
    shuffle=False,
    class_mode='binary',
    target_size=(224,224))

test_steps_per_epoch = test_generator.samples // test_generator.batch_size
```

```
Found 590 images belonging to 2 classes.
Found 104 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
```

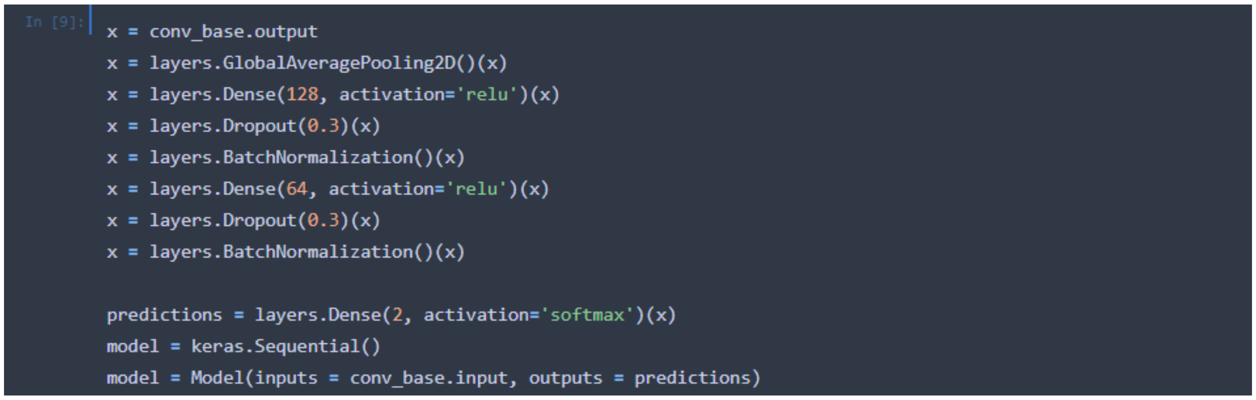
4) *Loading VGG16 Model and setting trainable layers as false*



```
In [8]: conv_base = VGG16(
            include_top=False,
            weights='imagenet')

        for layer in conv_base.layers:
            layer.trainable = False
```

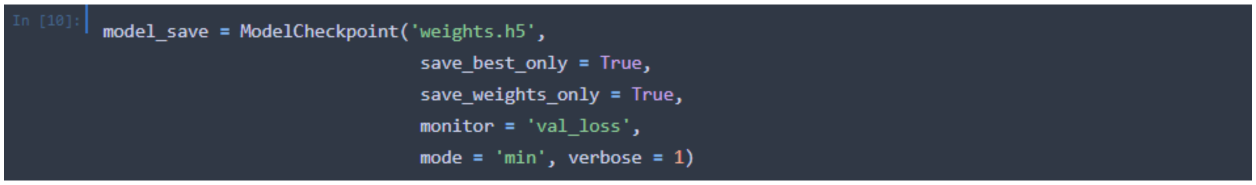
```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
```

5) *Model Addition*



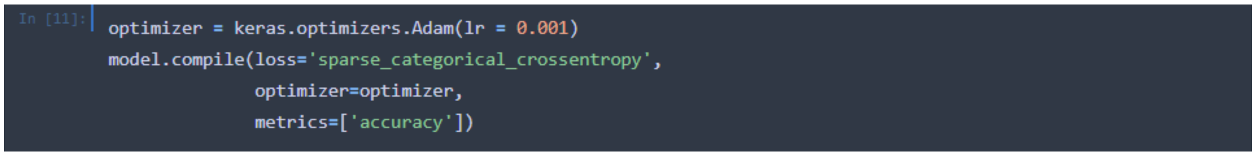
```
In [9]: x = conv_base.output
        x = layers.GlobalAveragePooling2D()(x)
        x = layers.Dense(128, activation='relu')(x)
        x = layers.Dropout(0.3)(x)
        x = layers.BatchNormalization()(x)
        x = layers.Dense(64, activation='relu')(x)
        x = layers.Dropout(0.3)(x)
        x = layers.BatchNormalization()(x)

        predictions = layers.Dense(2, activation='softmax')(x)
        model = keras.Sequential()
        model = Model(inputs = conv_base.input, outputs = predictions)
```

6) Setting up of Checkpoint to choose best model from epochs



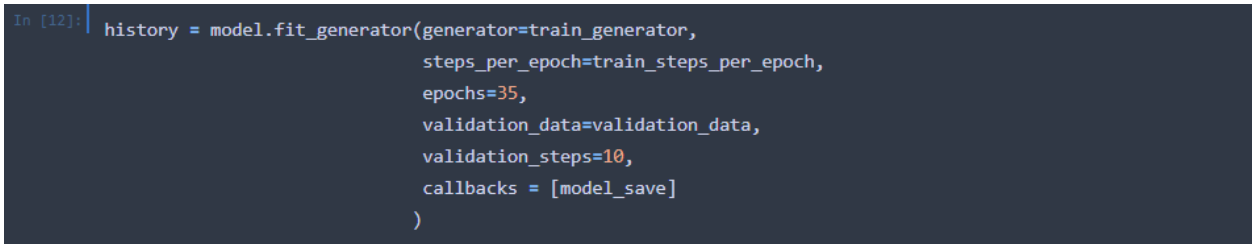
```
In [10]: model_save = ModelCheckpoint('weights.h5',
                                      save_best_only = True,
                                      save_weights_only = True,
                                      monitor = 'val_loss',
                                      mode = 'min', verbose = 1)
```

7) Optimizer and model compilation



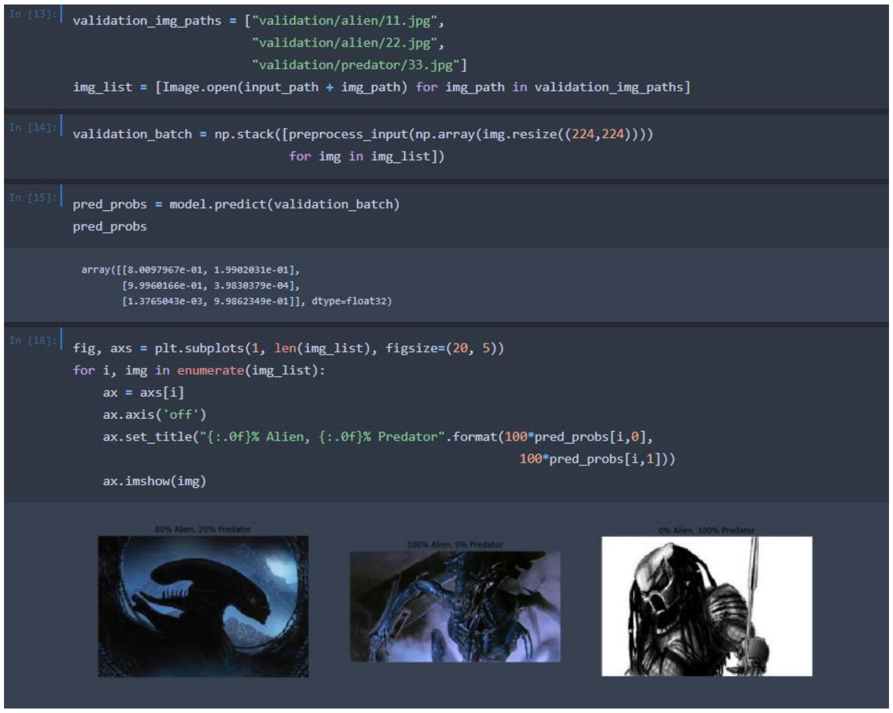
```
In [11]: optimizer = keras.optimizers.Adam(lr = 0.001)
         model.compile(loss='sparse_categorical_crossentropy',
                       optimizer=optimizer,
                       metrics=['accuracy'])
```

8) Training



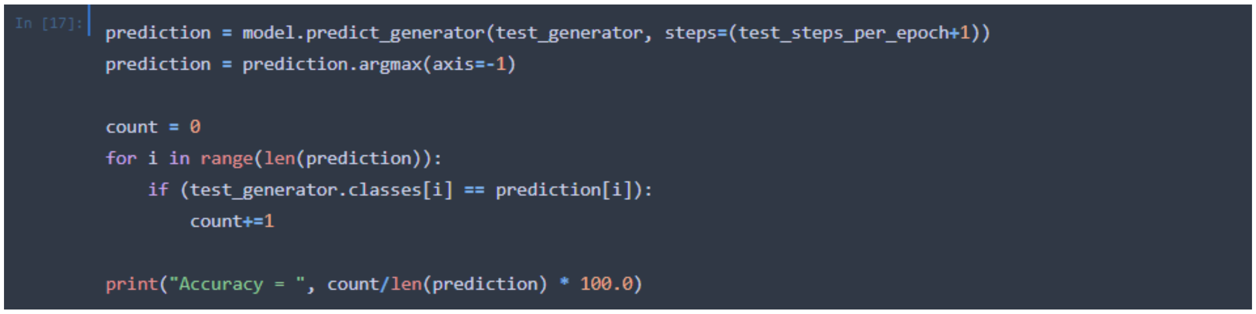
```
In [12]: history = model.fit_generator(generator=train_generator,
                                       steps_per_epoch=train_steps_per_epoch,
                                       epochs=35,
                                       validation_data=validation_data,
                                       validation_steps=10,
                                       callbacks = [model_save]
                                      )
```

9) Displaying Images from dataset



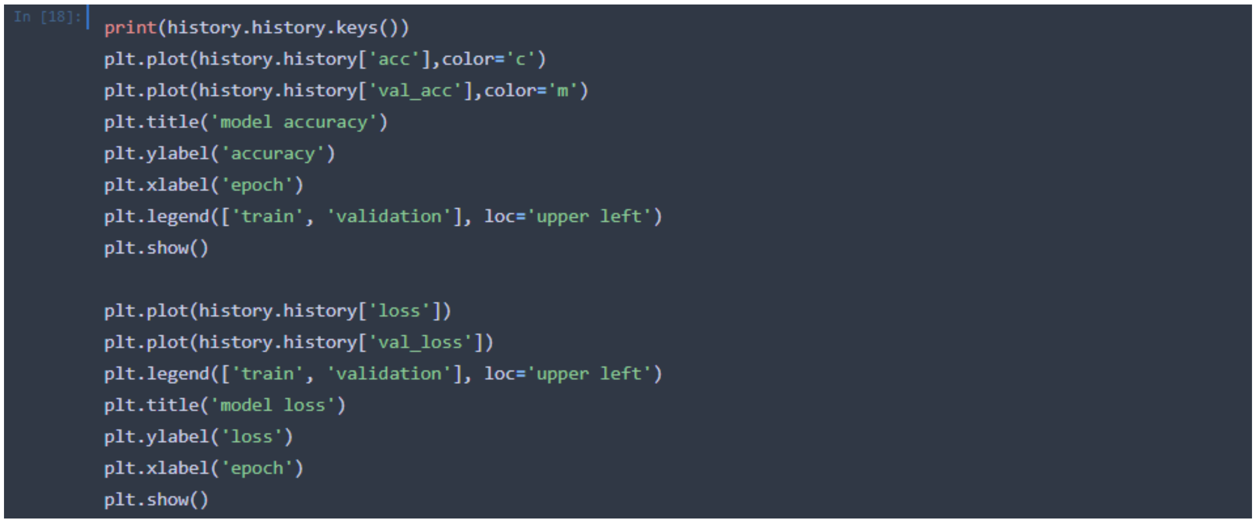
```
In [13]: validation_img_paths = ["validation/alien/11.jpg",
                                 "validation/alien/22.jpg",
                                 "validation/predator/33.jpg"]
         img_list = [Image.open(input_path + img_path) for img_path in validation_img_paths]

In [14]: validation_batch = np.stack([preprocess_input(np.array(img.resize((224,224))))
                                      for img in img_list])

In [15]: pred_probs = model.predict(validation_batch)
         pred_probs

          array([[8.0097967e-01, 1.9902031e-01],
                 [9.9960166e-01, 3.9830379e-04],
                 [1.3765043e-03, 9.9862349e-01]], dtype=float32)

In [16]: fig, axs = plt.subplots(1, len(img_list), figsize=(20, 5))
         for i, img in enumerate(img_list):
             ax = axs[i]
             ax.axis('off')
             ax.set_title("{:.0f}% Alien, {:.0f}% Predator".format(100*pred_probs[i,0],
                                                                    100*pred_probs[i,1]))
             ax.imshow(img)
```

10) Evaluation



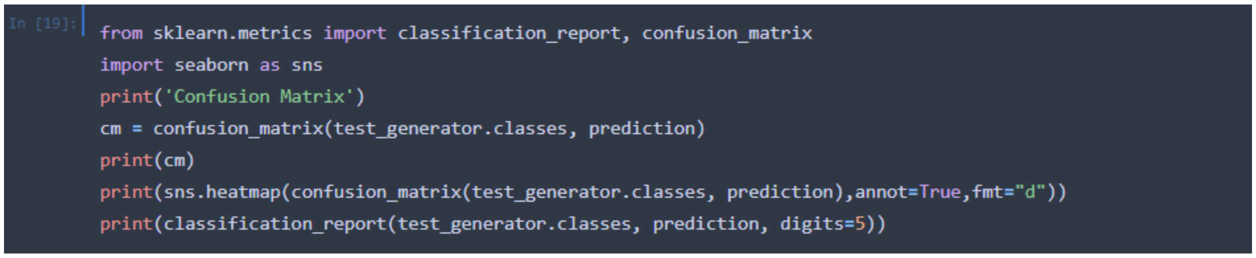
```
In [17]:
prediction = model.predict_generator(test_generator, steps=(test_steps_per_epoch+1))
prediction = prediction.argmax(axis=-1)

count = 0
for i in range(len(prediction)):
    if (test_generator.classes[i] == prediction[i]):
        count+=1

print("Accuracy = ", count/len(prediction) * 100.0)
```

11) Graph



```
In [18]:
print(history.history.keys())
plt.plot(history.history['acc'],color='c')
plt.plot(history.history['val_acc'],color='m')
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train', 'validation'], loc='upper left')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```

12) Confusion Matrix

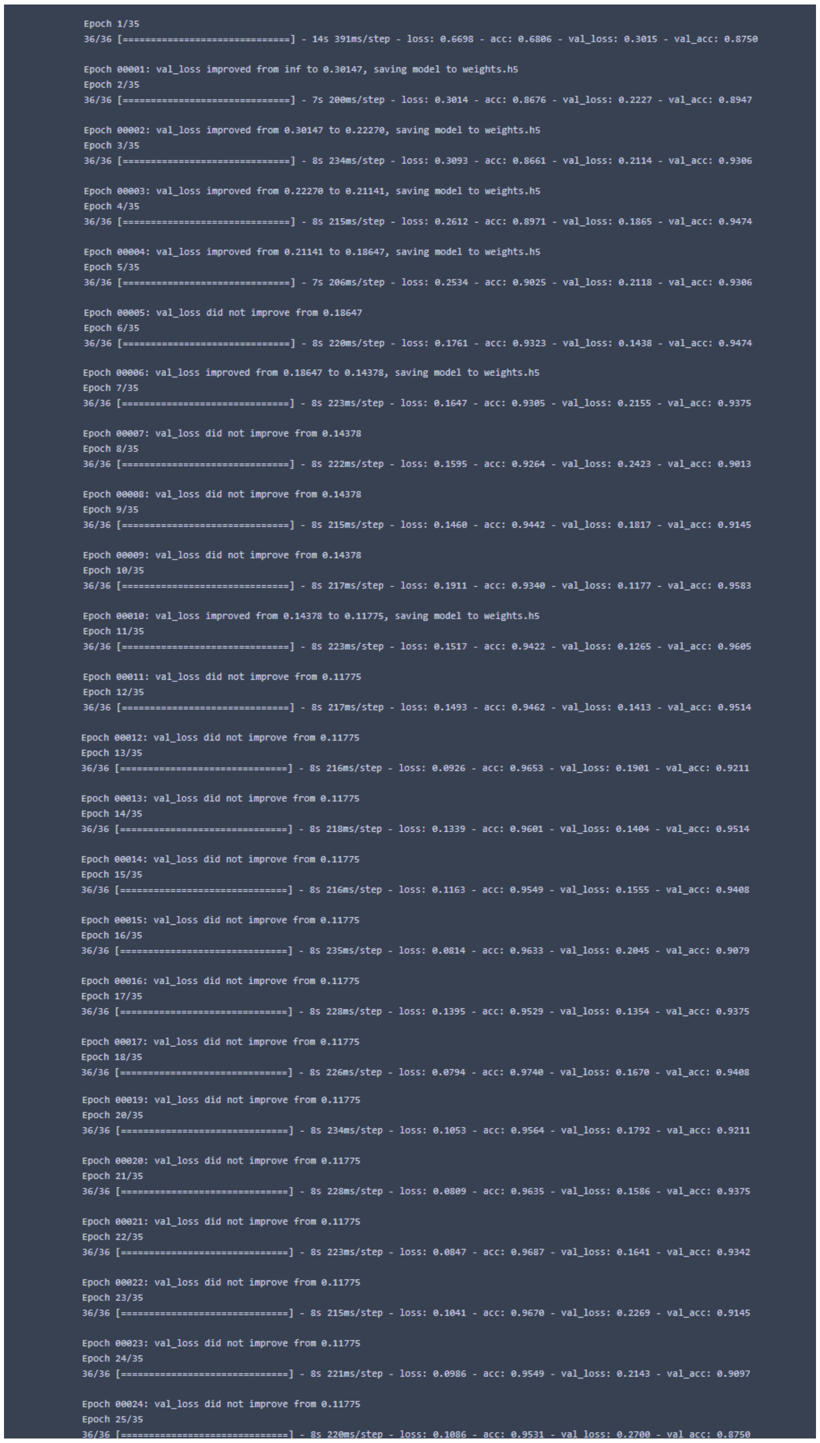


```
In [19]:
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
print('Confusion Matrix')
cm = confusion_matrix(test_generator.classes, prediction)
print(cm)
print(sns.heatmap(confusion_matrix(test_generator.classes, prediction),annot=True,fmt="d"))
print(classification_report(test_generator.classes, prediction, digits=5))
```

VII. RESULTS

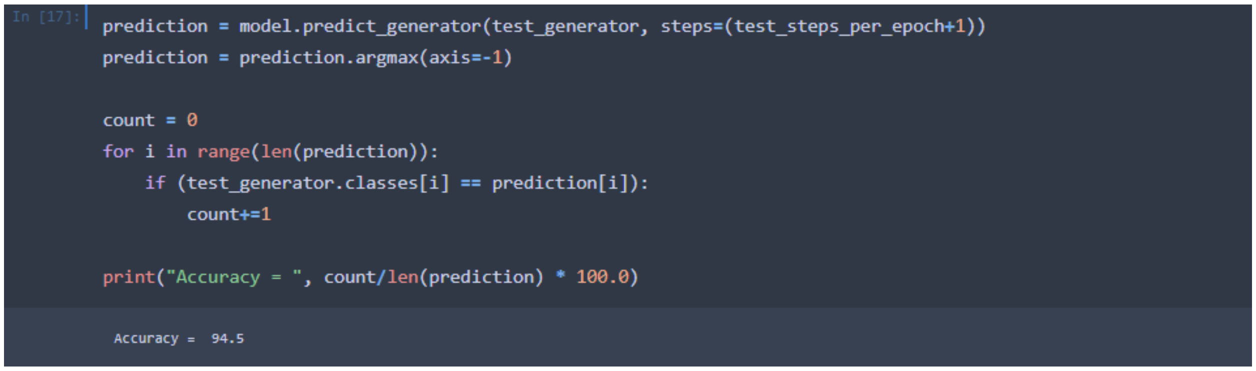
The different outputs corresponding to the driver code has been shown below:

1) *Training*



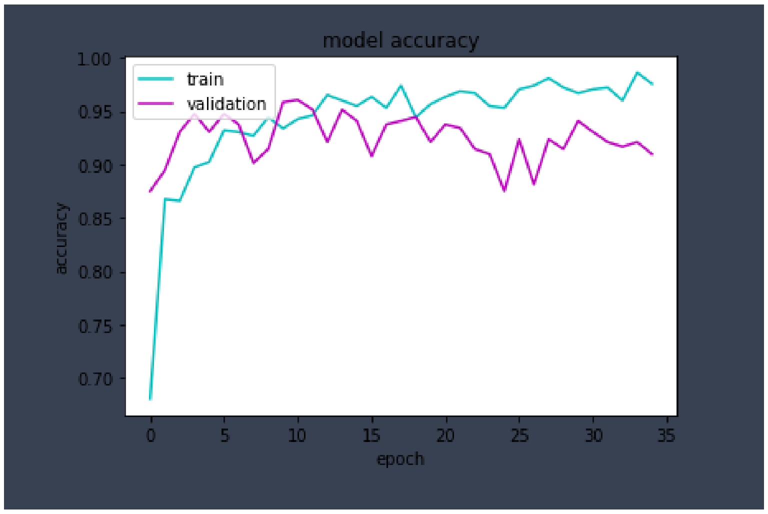
```
Epoch 1/35
36/36 [==============================] - 14s 391ms/step - loss: 0.6698 - acc: 0.6806 - val_loss: 0.3015 - val_acc: 0.8750

Epoch 00001: val_loss improved from inf to 0.30147, saving model to weights.h5
Epoch 2/35
36/36 [==============================] - 7s 200ms/step - loss: 0.3014 - acc: 0.8676 - val_loss: 0.2227 - val_acc: 0.8947

Epoch 00002: val_loss improved from 0.30147 to 0.22270, saving model to weights.h5
Epoch 3/35
36/36 [==============================] - 8s 234ms/step - loss: 0.3093 - acc: 0.8661 - val_loss: 0.2114 - val_acc: 0.9306

Epoch 00003: val_loss improved from 0.22270 to 0.21141, saving model to weights.h5
Epoch 4/35
36/36 [==============================] - 8s 215ms/step - loss: 0.2612 - acc: 0.8971 - val_loss: 0.1865 - val_acc: 0.9474

Epoch 00004: val_loss improved from 0.21141 to 0.18647, saving model to weights.h5
Epoch 5/35
36/36 [==============================] - 7s 206ms/step - loss: 0.2534 - acc: 0.9025 - val_loss: 0.2118 - val_acc: 0.9306

Epoch 00005: val_loss did not improve from 0.18647
Epoch 6/35
36/36 [==============================] - 8s 220ms/step - loss: 0.1761 - acc: 0.9323 - val_loss: 0.1438 - val_acc: 0.9474

Epoch 00006: val_loss improved from 0.18647 to 0.14378, saving model to weights.h5
Epoch 7/35
36/36 [==============================] - 8s 223ms/step - loss: 0.1647 - acc: 0.9305 - val_loss: 0.2155 - val_acc: 0.9375

Epoch 00007: val_loss did not improve from 0.14378
Epoch 8/35
36/36 [==============================] - 8s 222ms/step - loss: 0.1595 - acc: 0.9264 - val_loss: 0.2423 - val_acc: 0.9013

Epoch 00008: val_loss did not improve from 0.14378
Epoch 9/35
36/36 [==============================] - 8s 215ms/step - loss: 0.1460 - acc: 0.9442 - val_loss: 0.1817 - val_acc: 0.9145

Epoch 00009: val_loss did not improve from 0.14378
Epoch 10/35
36/36 [==============================] - 8s 217ms/step - loss: 0.1911 - acc: 0.9340 - val_loss: 0.1177 - val_acc: 0.9583

Epoch 00010: val_loss improved from 0.14378 to 0.11775, saving model to weights.h5
Epoch 11/35
36/36 [==============================] - 8s 223ms/step - loss: 0.1517 - acc: 0.9422 - val_loss: 0.1265 - val_acc: 0.9605

Epoch 00011: val_loss did not improve from 0.11775
Epoch 12/35
36/36 [==============================] - 8s 217ms/step - loss: 0.1493 - acc: 0.9462 - val_loss: 0.1413 - val_acc: 0.9514

Epoch 00012: val_loss did not improve from 0.11775
Epoch 13/35
36/36 [==============================] - 8s 216ms/step - loss: 0.0926 - acc: 0.9653 - val_loss: 0.1901 - val_acc: 0.9211

Epoch 00013: val_loss did not improve from 0.11775
Epoch 14/35
36/36 [==============================] - 8s 218ms/step - loss: 0.1339 - acc: 0.9601 - val_loss: 0.1404 - val_acc: 0.9514

Epoch 00014: val_loss did not improve from 0.11775
Epoch 15/35
36/36 [==============================] - 8s 216ms/step - loss: 0.1163 - acc: 0.9549 - val_loss: 0.1555 - val_acc: 0.9408

Epoch 00015: val_loss did not improve from 0.11775
Epoch 16/35
36/36 [==============================] - 8s 235ms/step - loss: 0.0814 - acc: 0.9633 - val_loss: 0.2045 - val_acc: 0.9079

Epoch 00016: val_loss did not improve from 0.11775
Epoch 17/35
36/36 [==============================] - 8s 228ms/step - loss: 0.1395 - acc: 0.9529 - val_loss: 0.1354 - val_acc: 0.9375

Epoch 00017: val_loss did not improve from 0.11775
Epoch 18/35
36/36 [==============================] - 8s 226ms/step - loss: 0.0794 - acc: 0.9740 - val_loss: 0.1670 - val_acc: 0.9408

Epoch 00019: val_loss did not improve from 0.11775
Epoch 20/35
36/36 [==============================] - 8s 234ms/step - loss: 0.1053 - acc: 0.9564 - val_loss: 0.1792 - val_acc: 0.9211

Epoch 00020: val_loss did not improve from 0.11775
Epoch 21/35
36/36 [==============================] - 8s 228ms/step - loss: 0.0809 - acc: 0.9635 - val_loss: 0.1586 - val_acc: 0.9375

Epoch 00021: val_loss did not improve from 0.11775
Epoch 22/35
36/36 [==============================] - 8s 223ms/step - loss: 0.0847 - acc: 0.9687 - val_loss: 0.1641 - val_acc: 0.9342

Epoch 00022: val_loss did not improve from 0.11775
Epoch 23/35
36/36 [==============================] - 8s 215ms/step - loss: 0.1041 - acc: 0.9670 - val_loss: 0.2269 - val_acc: 0.9145

Epoch 00023: val_loss did not improve from 0.11775
Epoch 24/35
36/36 [==============================] - 8s 221ms/step - loss: 0.0986 - acc: 0.9549 - val_loss: 0.2143 - val_acc: 0.9097

Epoch 00024: val_loss did not improve from 0.11775
Epoch 25/35
36/36 [==============================] - 8s 220ms/step - loss: 0.1086 - acc: 0.9531 - val_loss: 0.2700 - val_acc: 0.8750
```

2) *Accuracy and evaluation*

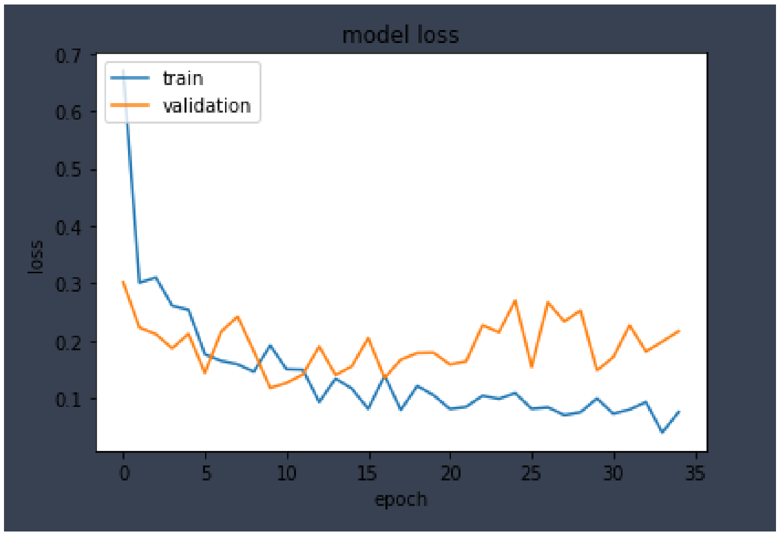


```
In [17]: prediction = model.predict_generator(test_generator, steps=(test_steps_per_epoch+1))
         prediction = prediction.argmax(axis=-1)

         count = 0
         for i in range(len(prediction)):
             if (test_generator.classes[i] == prediction[i]):
                 count+=1

         print("Accuracy = ", count/len(prediction) * 100.0)

          Accuracy =  94.5
```

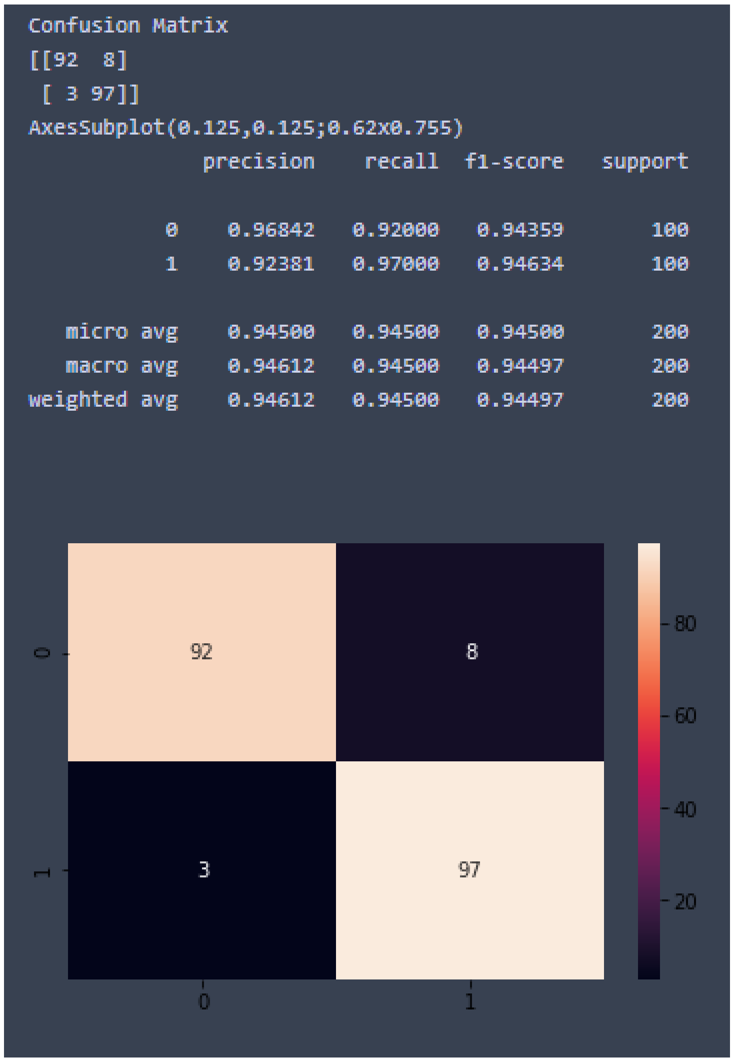
3) *Model Accuracy Graph*



4) *Model Loss Graph*



5) Confusion Matrix



```
Confusion Matrix
[[92  8]
 [ 3 97]]
AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0    0.96842   0.92000   0.94359       100
           1    0.92381   0.97000   0.94634       100

   micro avg    0.94500   0.94500   0.94500       200
   macro avg    0.94612   0.94500   0.94497       200
weighted avg    0.94612   0.94500   0.94497       200
```

VIII. CONCLUSIONS

As seen from the output screenshots, the model performs relatively well with this dataset. It managed to score a Test Accuracy of 93.4% in the 10th epoch along with a validation accuracy of 95.83% in the same 10th epoch. The callback function monitored the validation loss and the model at the 10th epoch was saved as it has the minimum validation loss (of 0.11775).

As a result, the basics of Transfer Learning and its implementation have been understood via this experimental project.

IX. REFERENCES

[1] https://www.kaggle.com/pmigdal/alien-vs-predator-images

[2] https://neurohive.io/en/popular-networks/vgg16/