DEEP LEARNING (CS 6005)

MINI PROJECT ASSIGNMENT 2

COMPUTER VISION WITH TRANSFER LEARNING APPLICATION

BY

NAME: SRIDHAR.S

ROLL NO: 2018103068

BATCH: P

DATE: 12.05.2021

PROBLEM STATEMENT:

We are given a set of dog and cat images. The **task is to build a model to predict the category of an animal: dog or cat** and use Transfer Learning applications to model to enhance its performance.

DATASET DESCRIPTION:

The Asirra (animal species image recognition for restricting access) dataset was introduced in 2013 for a machine learning competition. The dataset includes 25,000 images with equal numbers of labels for cats and dogs.

CAT & DOG Dataset File Formats

The dataset consists of two folders

- Training_set
    - Cats (4000 images)
    - Dogs (4000 images)
- Testing_set
    - Cats (1000 images)
    - Dogs (1000 images)

MODULES:

Module 1: Importing Necessary libraries

The necessary libraries which are used to build a CNN model like Keras, Sequential, Dense, Dropout, Flatten, Conv2D, MaxPooling2D etc are imported

Module 2: Data Preparation

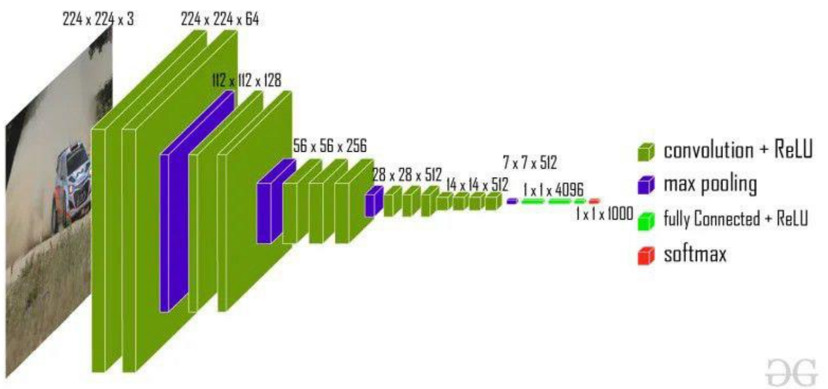
In this module the data is processed the various steps involved in processing dataset include:

- Creating folders for Train, Validation & Test data and sub folders like Cats & Dogs under each folder
- Reshape the images using Image data Generator
- Display a random image after reshaping

Module 3: Importing pretrained model VGG16

VGG16:

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual computer vision competition. Each year, teams compete on two tasks. The first is to detect objects within an image coming from 200 classes, which is called object localization. The second is to classify images, each labeled with one of 1000 categories, which is called image classification. VGG 16 was proposed by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab of Oxford University in 2014 in the paper "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION". The architecture of model is shown below.



This model achieves 92.7% top-5 test accuracy on ImageNet dataset which contains 14 million images belonging to 1000 classes.

Module 4: Pre-Trained Model as Feature Extractor Preprocessor

The features for training, validation and test data are extracted from pretrained model VGG16.

Module 5: Building a Convolutional Neural Network model without Transfer learning Application

Two Convolutional models are developed in this module

- CONVNET 1 without Data Augumentation Parameters
- CONVNET 2 with Data Augumentation Parameters

Various layers in CONVNET 1:

- **Conv2D**: The convolutional (Conv2D) layer. It is like a set of learnable filters. I choosed to set **32 filters** for the **first conv2D** layers and **64 filters** for the **second conv2D** layers, **128 filters** for the **third & fourth conv2D** layers. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image. The features will be "extracted" from the image.

- **MaxPooling2D**: It acts as a **downsampling filter**. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size more the pooling dimension is high, more the downsampling is important. The **images get half sized.**

- **Flatten**: Transforms the format of the images from a 2d-array to a 1d-array

- **Dense:** The **dense layer** is a fully connected **layer**, meaning all the neurons in a **layer** are connected to those in the next **layer**. Two dense layers are used one with 512 neurons and other with 1 neuron

<u>Various layers in CONVNET 2:</u>

- **Conv2D**: The convolutional (Conv2D) layer. It is like a set of learnable filters. I choosed to set **32 filters** for the **first conv2D** layers and **64 filters** for the **second conv2D** layers, **128 filters** for the **third conv2D** layer. Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. Filters can be seen as a transformation of the image. The features will be "extracted" from the image.

- **MaxPooling2D**: It acts as a **downsampling filter**. It looks at the 2 neighboring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting. We have to choose the pooling size more the pooling dimension is high, more the downsampling is important. The **images get half sized.**

- **Flatten**: Transforms the format of the images from a 2d-array to a 1d-array

- **Regularization function**: I used **Dropout** technique to regularize which specifies the percentage of neurons to be droped at each iteration.

- **Dense:** The **dense layer** is a fully connected **layer**, meaning all the neurons in a **layer** are connected to those in the next **layer**. Two dense layers are used one with 512 neurons and other with 1 neuron

Activation function used in both models:

- **Activation function** used:
    - Hidden layer - **Relu**: given a value x, returns max(x, 0).
    - Output layer – **Sigmoid** / **logistic** - The input to the **function** is transformed into a value between 0.0 and 1.0

The model built is compiled is compiled with parameters such as:

- **Optimizer**: **RMSProp** - exponentially weighted average of the squares of past gradients.
- **Loss function**: I used **binary_crossentropy** for classification, each images belongs to one class only

Module 6: Building a Convolutional Neural Network model with Transfer learning Application

Transfer Learning:

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem. In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest.

Three Convolutional models are developed in this module

- CONVNET 3 with features from pretrained model
- CONVNET 4 - Pre-Trained Model as Feature Extractor in Model
    - CONVNET 4a - with frozen layers from pretrained model
    - CONVNET 4b - with Fine tuning of pretrained model

Various layers in CONVNET 3:

- **Dense:** The **dense layer** is a fully connected **layer**, meaning all the neurons in a **layer** are connected to those in the next **layer**. Two dense layers are used one with 256 neurons and other with 1 neuron

- **Regularization function**: I used **Dropout** technique to regularize which specifies the percentage of neurons to be droped at each iteration.

Various layers in CONVNET 4:

We can use some or all of the layers in a pre-trained model as a feature extraction component of a new model directly. This can be achieved by loading the model, then simply adding new layers. This may involve adding new convolutional and pooling layers to expand upon the feature extraction capabilities of the model or adding new fully connected classifier type layers to learn how to interpret the extracted features on a new dataset, or some combination.

- **Layer 1:** Load the **VGG16 model** without the classifier part of the model by specifying the *"*include_top*"* argument to *"*False*"*, and specify the preferred shape of the images in our new dataset as 150 * 150.

- **Flatten**: Transforms the format of the images from a 2d-array to a 1d-array

- **Dense:** The **dense layer** is a fully connected **layer**, meaning all the neurons in a **layer** are connected to those in the next **layer**. Two dense layers are used one with 256 neurons and other with 1 neuron

Activation function used in the models:

- **Relu** – used in first dense layer
- **Sigmoid** – used in second dense layer

The model built is compiled is compiled with parameters such as:

- **Optimizer**: **RMSProp** - exponentially weighted average of the squares of past gradients.
- **Loss function**: I used **binary_crossentropy** for classification, each images belongs to one class only

Module 7: Evaluation

The model is evaluated by

- By plotting accuracy and loss curves
- Displaying accuracy and loss score

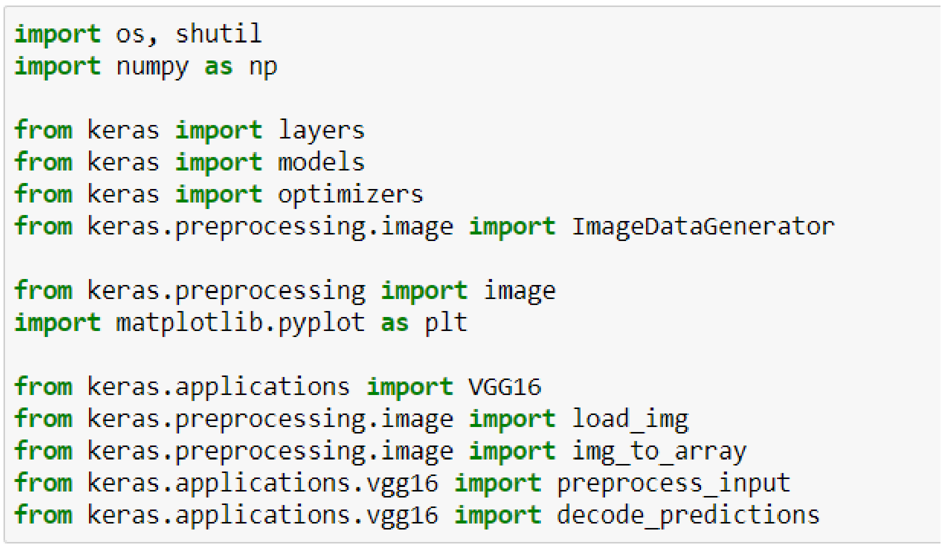
Module 8: Prediction

The VGG16 model is used predict the output for the sample input

PROGRAM SNAPSHOTS:

MODULE 1:

#Importing necessary libraries



```
import os, shutil
import numpy as np

from keras import layers
from keras import models
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator

from keras.preprocessing import image
import matplotlib.pyplot as plt

from keras.applications import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
```

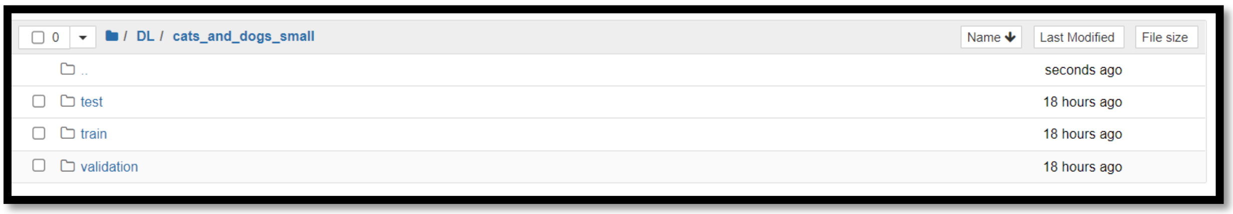
MODULE 2:

#Creating directory cats_and_dogs_small with folders:

- Train
  - Cats
  - Dogs
- Test
  - Cats
  - Dogs
- Validation
  - Cats
  - Dogs

```
original_dataset_dir1 = r"C:\Users\Sridhar\DL\kaggle_original_data\train\cats"
original_dataset_dir2 = r"C:\Users\Sridhar\DL\kaggle_original_data\train\dogs"


base_dir = r"C:\Users\Sridhar\DL\cats_and_dogs_small"
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)

validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)

test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)



train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)

test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```

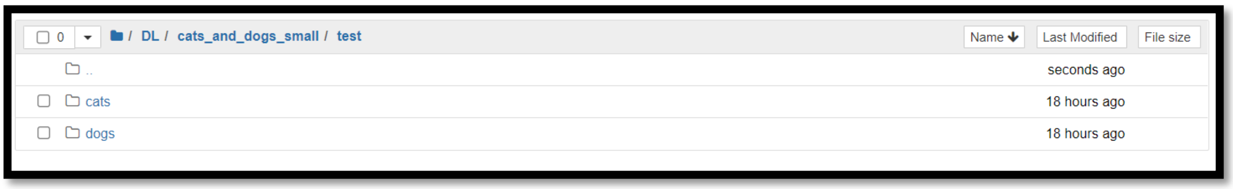
After creating directories



/ DL / cats_and_dogs_small

Name	Last Modified	File size
..	seconds ago	
test	18 hours ago	
train	18 hours ago	
validation	18 hours ago	

Test folder with sub folders:



/ DL / cats_and_dogs_small / test

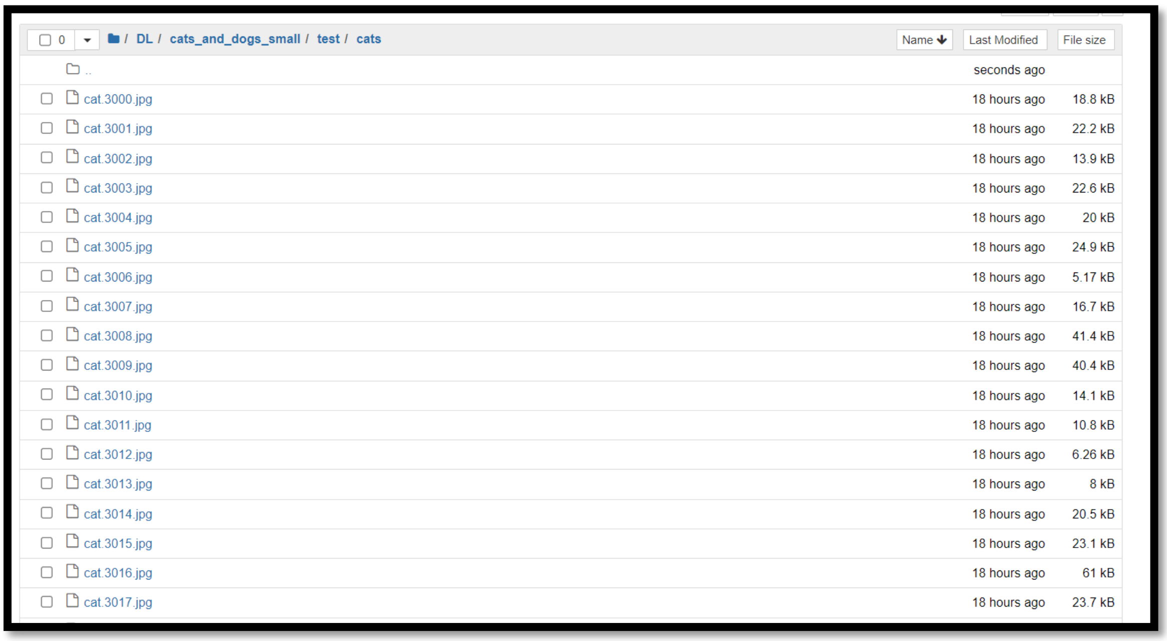
Name	Last Modified	File size
..	seconds ago	
cats	18 hours ago	
dogs	18 hours ago	

- Validation and Train folder will also contain subfolders (Cats & Dogs) like test folder

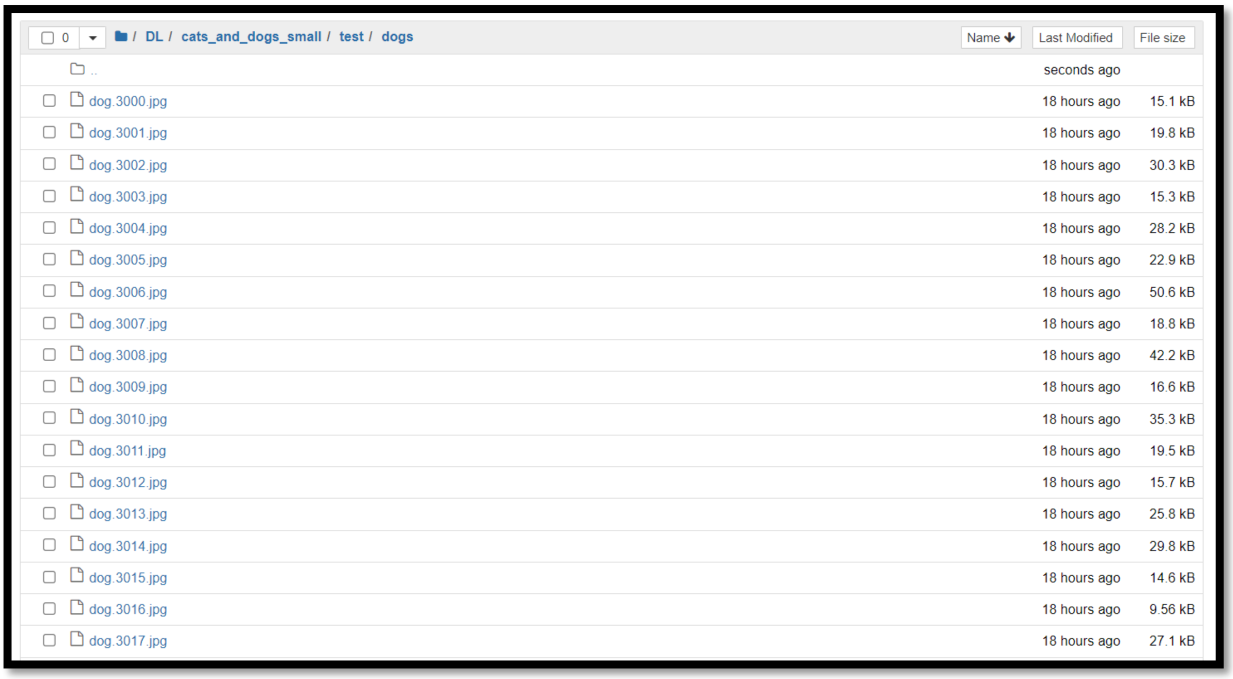
#Loading Images to the subfolders (Cats & Dogs) of Train, Test & Validation folder

```
fnames = ['cat.{}.jpg'.format(i) for i in range(2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir1, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

fnames = ['cat.{}.jpg'.format(i) for i in range(2000, 3000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir1, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)

fnames = ['cat.{}.jpg'.format(i) for i in range(3000, 4000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir1, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

fnames = ['dog.{}.jpg'.format(i) for i in range(2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir2, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

fnames = ['dog.{}.jpg'.format(i) for i in range(2000, 3000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir2, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)

fnames = ['dog.{}.jpg'.format(i) for i in range(3000, 4000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir2, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

Example Images of Cats & Dogs in Test folder


0 / DL / cats_and_dogs_small / test / cats

Name	Last Modified	File size
..	seconds ago	
cat.3000.jpg	18 hours ago	18.8 kB
cat.3001.jpg	18 hours ago	22.2 kB
cat.3002.jpg	18 hours ago	13.9 kB
cat.3003.jpg	18 hours ago	22.6 kB
cat.3004.jpg	18 hours ago	20 kB
cat.3005.jpg	18 hours ago	24.9 kB
cat.3006.jpg	18 hours ago	5.17 kB
cat.3007.jpg	18 hours ago	16.7 kB
cat.3008.jpg	18 hours ago	41.4 kB
cat.3009.jpg	18 hours ago	40.4 kB
cat.3010.jpg	18 hours ago	14.1 kB
cat.3011.jpg	18 hours ago	10.8 kB
cat.3012.jpg	18 hours ago	6.26 kB
cat.3013.jpg	18 hours ago	8 kB
cat.3014.jpg	18 hours ago	20.5 kB
cat.3015.jpg	18 hours ago	23.1 kB
cat.3016.jpg	18 hours ago	61 kB
cat.3017.jpg	18 hours ago	23.7 kB


0 / DL / cats_and_dogs_small / test / dogs

Name	Last Modified	File size
..	seconds ago	
dog.3000.jpg	18 hours ago	15.1 kB
dog.3001.jpg	18 hours ago	19.8 kB
dog.3002.jpg	18 hours ago	30.3 kB
dog.3003.jpg	18 hours ago	15.3 kB
dog.3004.jpg	18 hours ago	28.2 kB
dog.3005.jpg	18 hours ago	22.9 kB
dog.3006.jpg	18 hours ago	50.6 kB
dog.3007.jpg	18 hours ago	18.8 kB
dog.3008.jpg	18 hours ago	42.2 kB
dog.3009.jpg	18 hours ago	16.6 kB
dog.3010.jpg	18 hours ago	35.3 kB
dog.3011.jpg	18 hours ago	19.5 kB
dog.3012.jpg	18 hours ago	15.7 kB
dog.3013.jpg	18 hours ago	25.8 kB
dog.3014.jpg	18 hours ago	29.8 kB
dog.3015.jpg	18 hours ago	14.6 kB
dog.3016.jpg	18 hours ago	9.56 kB
dog.3017.jpg	18 hours ago	27.1 kB

- Similarly images will be stored in subfolders (Cats & Dogs) of Train and Validation folder

#Displaying number of images of cats & dogs in each folder:

```python
print('total training cat images:', len(os.listdir(train_cats_dir)))
```

```
total training cat images: 2000
```

```python
print('total training dog images:', len(os.listdir(train_dogs_dir)))
```

```
total training dog images: 2000
```

```python
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
```

```
total validation cat images: 1000
```

```python
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
```

```
total validation dog images: 1000
```

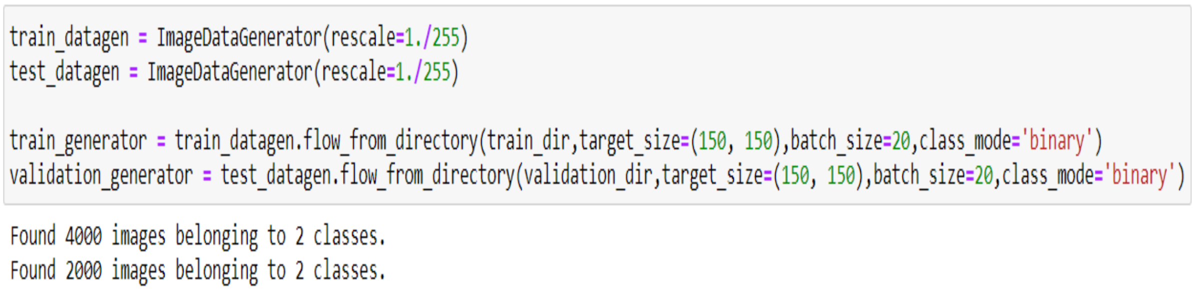
```python
print('total test cat images:', len(os.listdir(test_cats_dir)))
```

```
total test cat images: 1000
```

```python
print('total test dog images:', len(os.listdir(test_dogs_dir)))
```

```
total test dog images: 1000
```

#Reshaping the images without Data Augumentation:

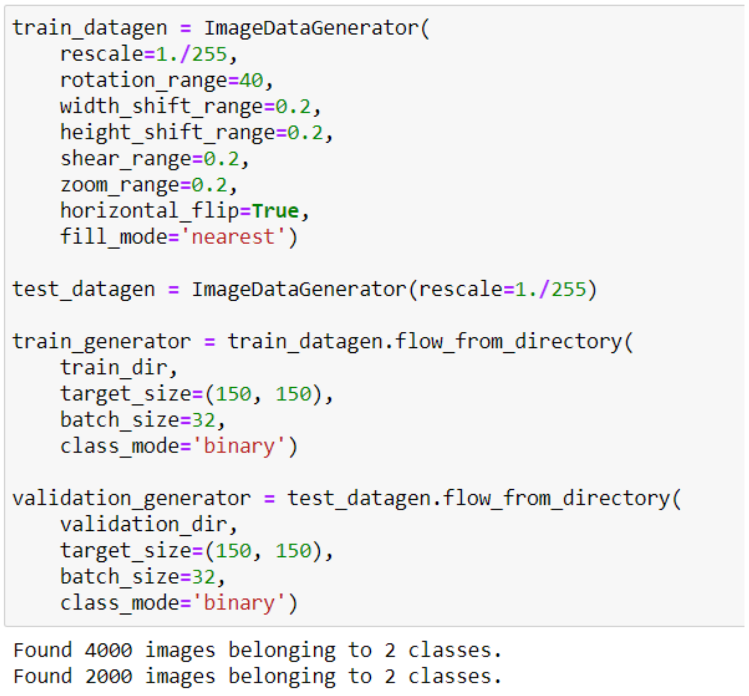


```
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,target_size=(150, 150),batch_size=20,class_mode='binary')
validation_generator = test_datagen.flow_from_directory(validation_dir,target_size=(150, 150),batch_size=20,class_mode='binary')
```

```
Found 4000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

- This is used as the **feature** for **CONVNET 1**

#Reshaping the images with Data Augumentation



```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')
```

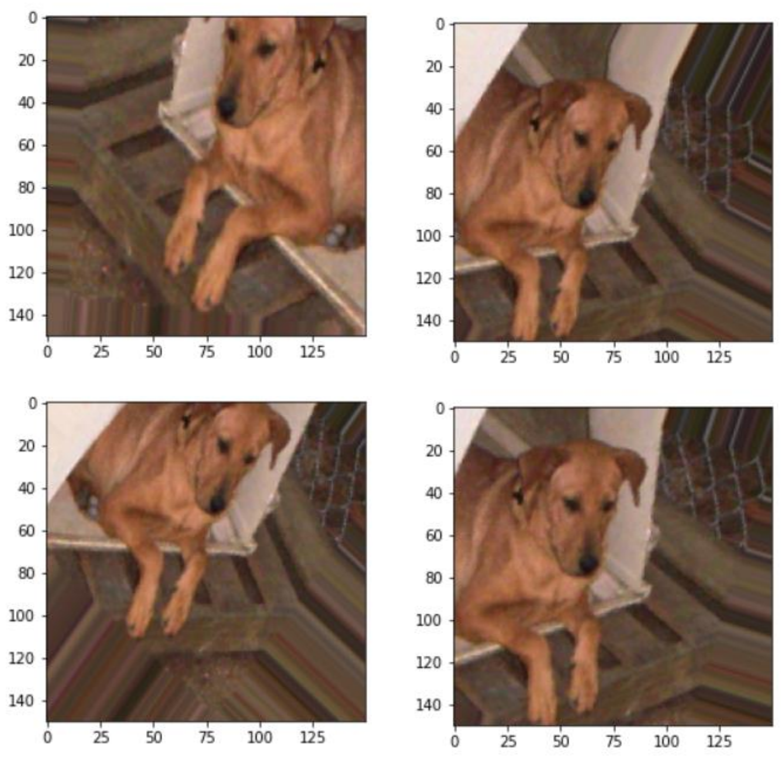
```
Found 4000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

- This is used as **feature** for **CONVNET 2**, **CONVNET 4a** & **CONVNET 4b**

#Displaying Random Image after Data Augumentation

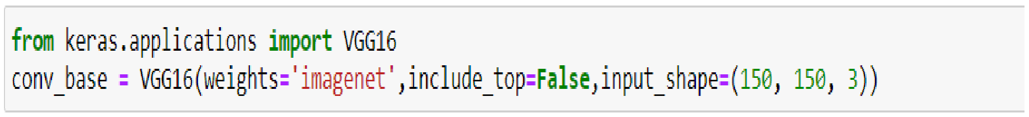
```python
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
```

```python
from keras.preprocessing import image
import os
fnames = [os.path.join(train_dogs_dir, fname) for
    fname in os.listdir(train_dogs_dir)]
img_path = fnames[3]
img = image.load_img(img_path, target_size=(150, 150))

x = image.img_to_array(img)
x = x.reshape((1,) + x.shape)
i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break
plt.show()
```



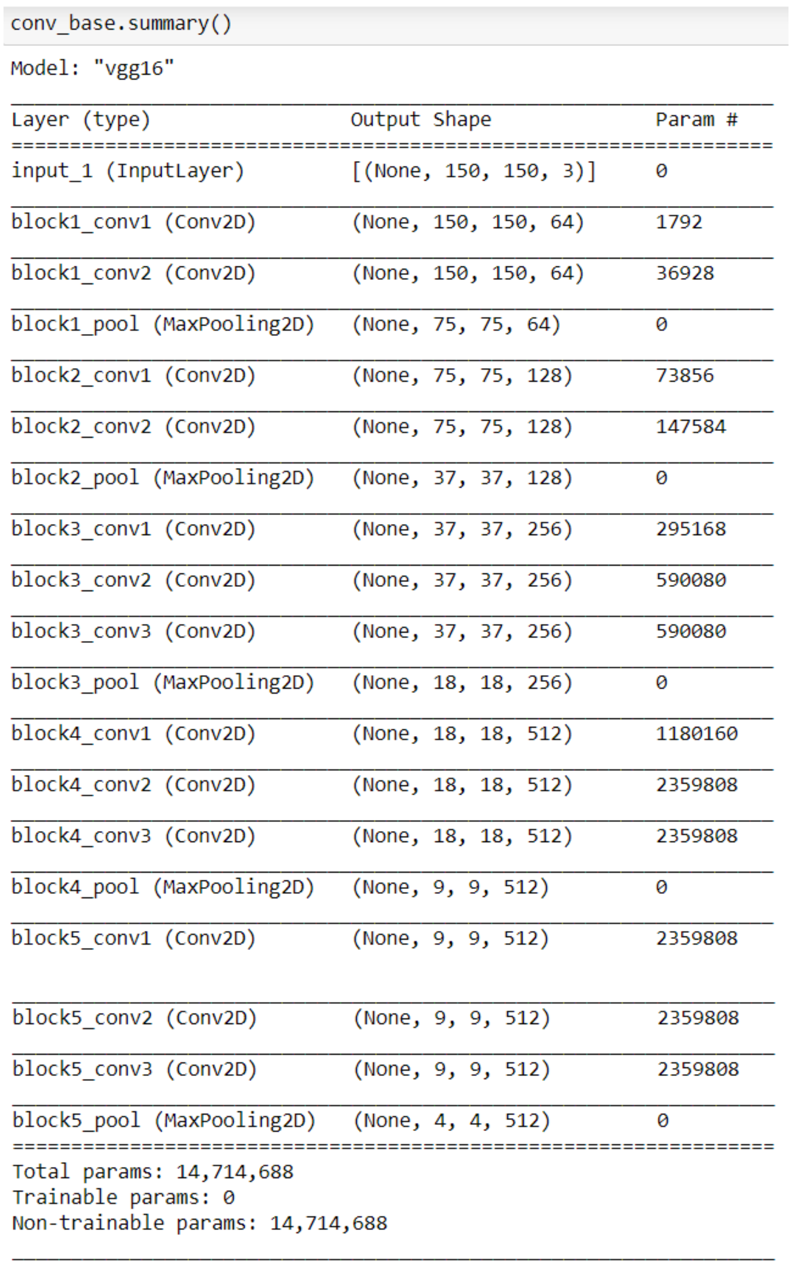
MODULE 3:

#Importing VGG16 Model:



```
from keras.applications import VGG16
conv_base = VGG16(weights='imagenet',include_top=False,input_shape=(150, 150, 3))
```

#conv_base Model Summary:



```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808

_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

MODULE 4:

#Extract features using pretrained model VGG16

```
from keras.preprocessing.image import ImageDataGenerator

base_dir = r"C:\Users\Sridhar\DL\cats_and_dogs_small"

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
```

```
Found 4000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
Found 2000 images belonging to 2 classes.
```

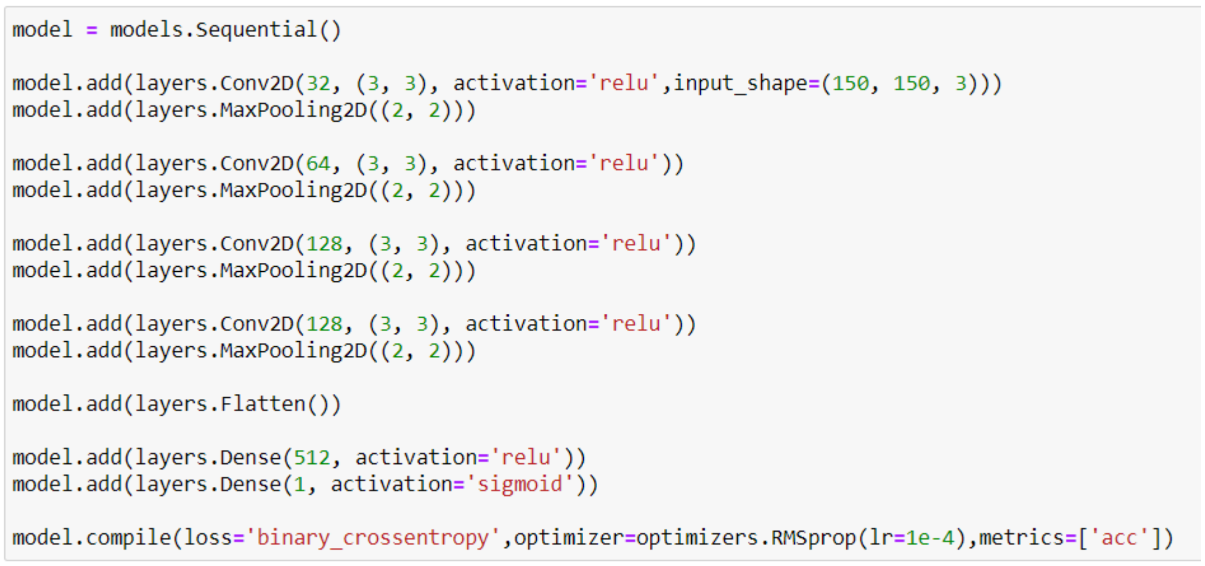
#Reshaping Extracted features

```
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

- This is used as a **feature** for **CONVNET 3**

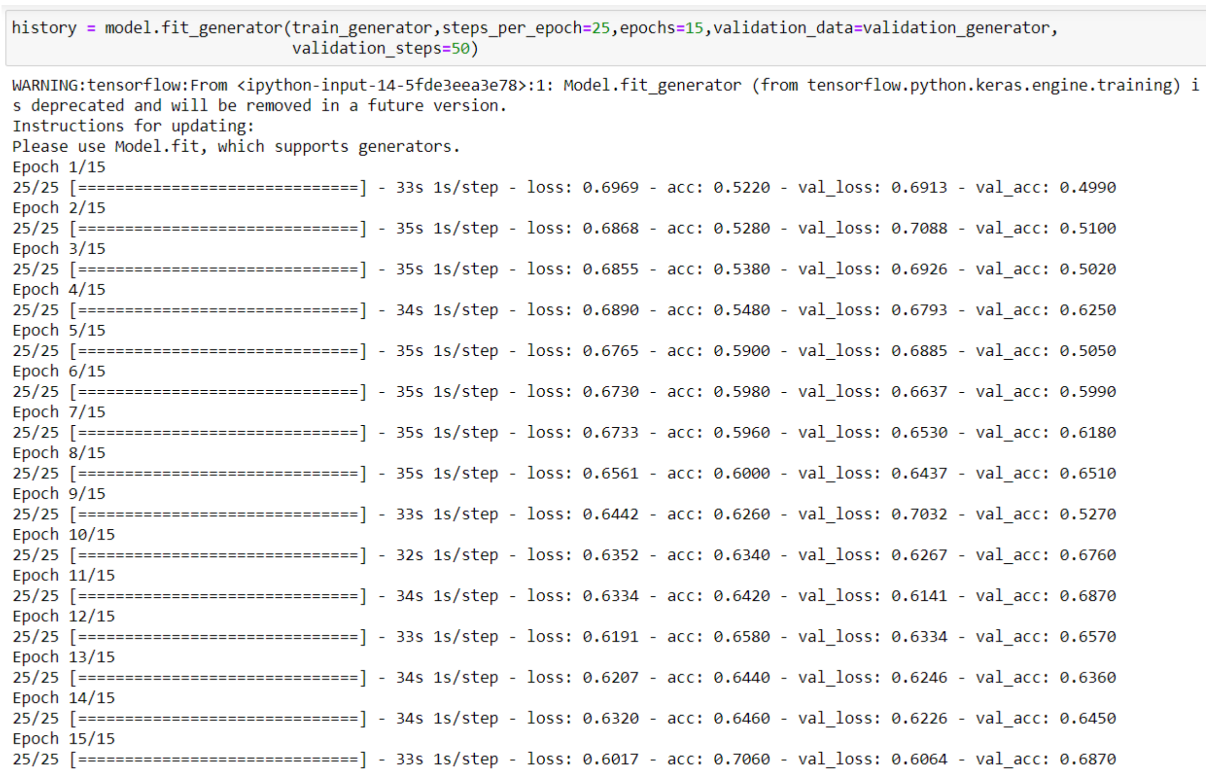
MODULE 5:

#Building CONVNET 1 Model:



```
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer=optimizers.RMSprop(lr=1e-4),metrics=['acc'])
```

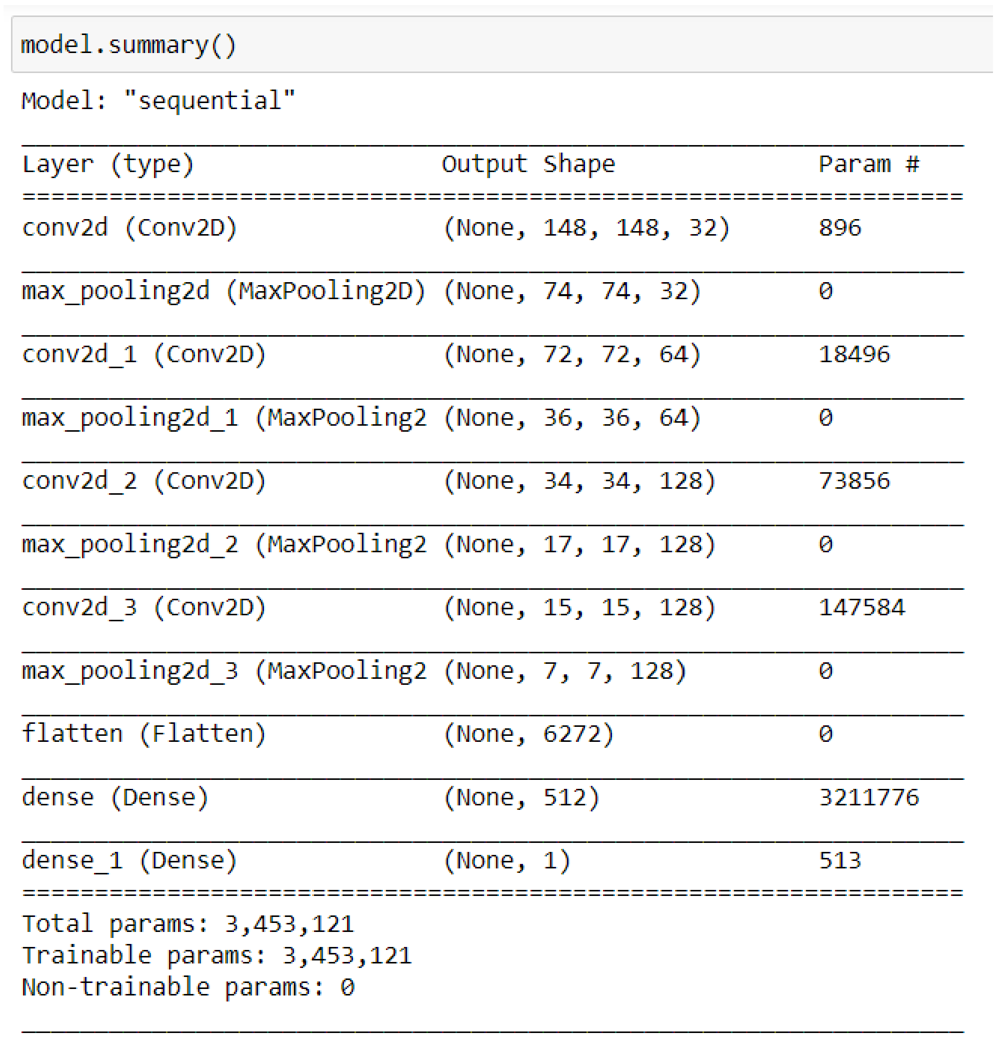
#Fitting CONVNET 1 Model:



```
history = model.fit_generator(train_generator,steps_per_epoch=25,epochs=15,validation_data=validation_generator,
                              validation_steps=50)
```

```
WARNING:tensorflow:From <ipython-input-14-5fde3eea3e78>:1: Model.fit_generator (from tensorflow.python.keras.engine.training) i
s deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/15
25/25 [==============================] - 33s 1s/step - loss: 0.6969 - acc: 0.5220 - val_loss: 0.6913 - val_acc: 0.4990
Epoch 2/15
25/25 [==============================] - 35s 1s/step - loss: 0.6868 - acc: 0.5280 - val_loss: 0.7088 - val_acc: 0.5100
Epoch 3/15
25/25 [==============================] - 35s 1s/step - loss: 0.6855 - acc: 0.5380 - val_loss: 0.6926 - val_acc: 0.5020
Epoch 4/15
25/25 [==============================] - 34s 1s/step - loss: 0.6890 - acc: 0.5480 - val_loss: 0.6793 - val_acc: 0.6250
Epoch 5/15
25/25 [==============================] - 35s 1s/step - loss: 0.6765 - acc: 0.5900 - val_loss: 0.6885 - val_acc: 0.5050
Epoch 6/15
25/25 [==============================] - 35s 1s/step - loss: 0.6730 - acc: 0.5980 - val_loss: 0.6637 - val_acc: 0.5990
Epoch 7/15
25/25 [==============================] - 35s 1s/step - loss: 0.6733 - acc: 0.5960 - val_loss: 0.6530 - val_acc: 0.6180
Epoch 8/15
25/25 [==============================] - 35s 1s/step - loss: 0.6561 - acc: 0.6000 - val_loss: 0.6437 - val_acc: 0.6510
Epoch 9/15
25/25 [==============================] - 33s 1s/step - loss: 0.6442 - acc: 0.6260 - val_loss: 0.7032 - val_acc: 0.5270
Epoch 10/15
25/25 [==============================] - 32s 1s/step - loss: 0.6352 - acc: 0.6340 - val_loss: 0.6267 - val_acc: 0.6760
Epoch 11/15
25/25 [==============================] - 34s 1s/step - loss: 0.6334 - acc: 0.6420 - val_loss: 0.6141 - val_acc: 0.6870
Epoch 12/15
25/25 [==============================] - 33s 1s/step - loss: 0.6191 - acc: 0.6580 - val_loss: 0.6334 - val_acc: 0.6570
Epoch 13/15
25/25 [==============================] - 34s 1s/step - loss: 0.6207 - acc: 0.6440 - val_loss: 0.6246 - val_acc: 0.6360
Epoch 14/15
25/25 [==============================] - 34s 1s/step - loss: 0.6320 - acc: 0.6460 - val_loss: 0.6226 - val_acc: 0.6450
Epoch 15/15
25/25 [==============================] - 33s 1s/step - loss: 0.6017 - acc: 0.7060 - val_loss: 0.6064 - val_acc: 0.6870
```

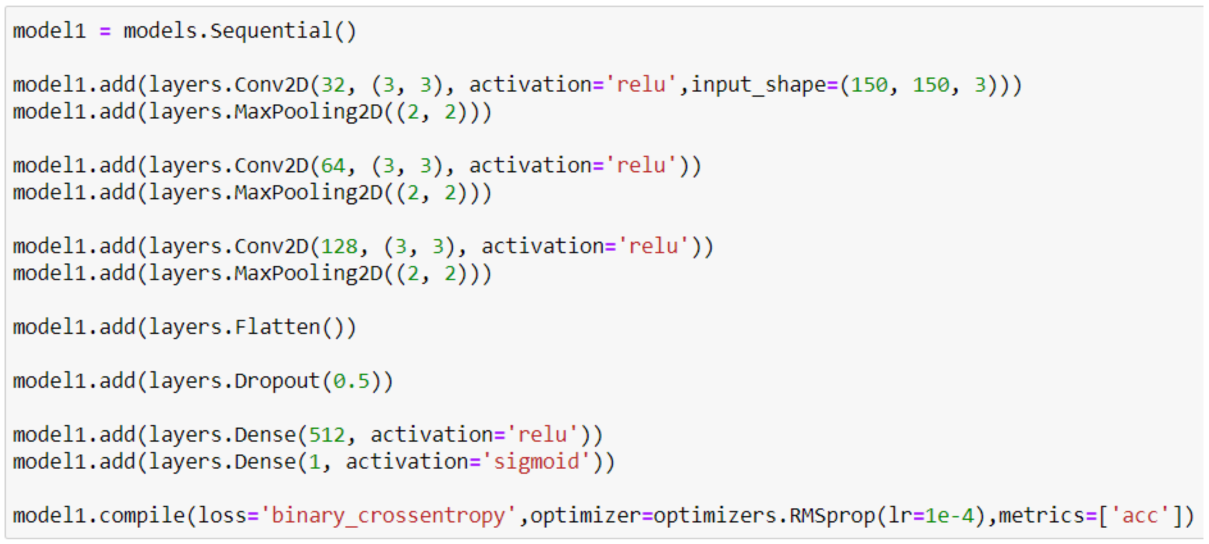
#CONVNET 1 Model summary:



```
model.summary()
```

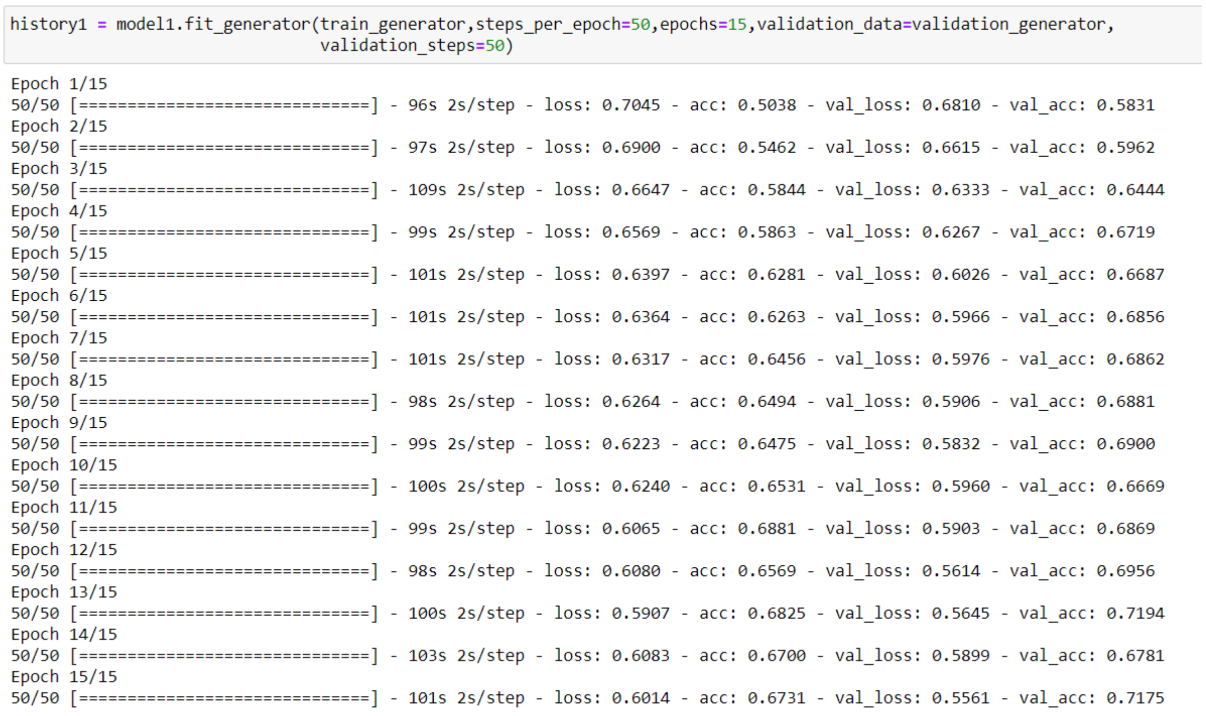
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               3211776   
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

#Building CONVNET 2 Model:



```
model1 = models.Sequential()

model1.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(150, 150, 3)))
model1.add(layers.MaxPooling2D((2, 2)))

model1.add(layers.Conv2D(64, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))

model1.add(layers.Conv2D(128, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))

model1.add(layers.Flatten())

model1.add(layers.Dropout(0.5))

model1.add(layers.Dense(512, activation='relu'))
model1.add(layers.Dense(1, activation='sigmoid'))

model1.compile(loss='binary_crossentropy',optimizer=optimizers.RMSprop(lr=1e-4),metrics=['acc'])
```

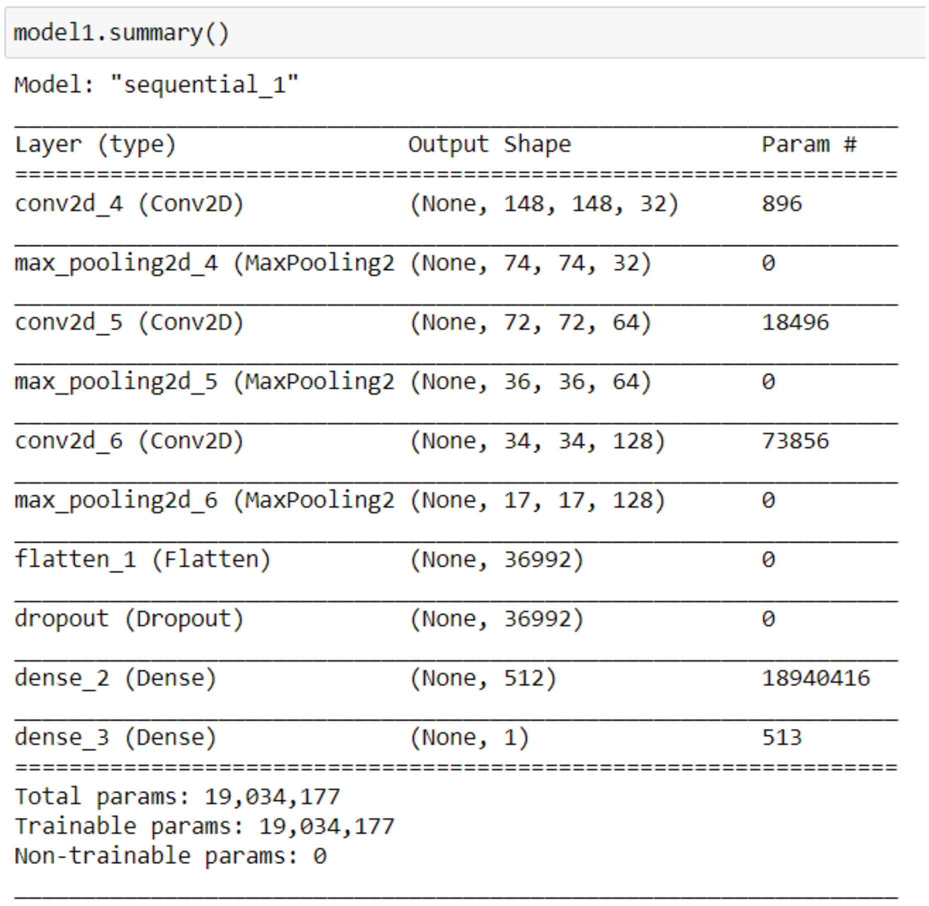
#Fitting CONVNET 2 Model:



```
history1 = model1.fit_generator(train_generator,steps_per_epoch=50,epochs=15,validation_data=validation_generator,
                                validation_steps=50)
```

```
Epoch 1/15
50/50 [==============================] - 96s 2s/step - loss: 0.7045 - acc: 0.5038 - val_loss: 0.6810 - val_acc: 0.5831
Epoch 2/15
50/50 [==============================] - 97s 2s/step - loss: 0.6900 - acc: 0.5462 - val_loss: 0.6615 - val_acc: 0.5962
Epoch 3/15
50/50 [==============================] - 109s 2s/step - loss: 0.6647 - acc: 0.5844 - val_loss: 0.6333 - val_acc: 0.6444
Epoch 4/15
50/50 [==============================] - 99s 2s/step - loss: 0.6569 - acc: 0.5863 - val_loss: 0.6267 - val_acc: 0.6719
Epoch 5/15
50/50 [==============================] - 101s 2s/step - loss: 0.6397 - acc: 0.6281 - val_loss: 0.6026 - val_acc: 0.6687
Epoch 6/15
50/50 [==============================] - 101s 2s/step - loss: 0.6364 - acc: 0.6263 - val_loss: 0.5966 - val_acc: 0.6856
Epoch 7/15
50/50 [==============================] - 101s 2s/step - loss: 0.6317 - acc: 0.6456 - val_loss: 0.5976 - val_acc: 0.6862
Epoch 8/15
50/50 [==============================] - 98s 2s/step - loss: 0.6264 - acc: 0.6494 - val_loss: 0.5906 - val_acc: 0.6881
Epoch 9/15
50/50 [==============================] - 99s 2s/step - loss: 0.6223 - acc: 0.6475 - val_loss: 0.5832 - val_acc: 0.6900
Epoch 10/15
50/50 [==============================] - 100s 2s/step - loss: 0.6240 - acc: 0.6531 - val_loss: 0.5960 - val_acc: 0.6669
Epoch 11/15
50/50 [==============================] - 99s 2s/step - loss: 0.6065 - acc: 0.6881 - val_loss: 0.5903 - val_acc: 0.6869
Epoch 12/15
50/50 [==============================] - 98s 2s/step - loss: 0.6080 - acc: 0.6569 - val_loss: 0.5614 - val_acc: 0.6956
Epoch 13/15
50/50 [==============================] - 100s 2s/step - loss: 0.5907 - acc: 0.6825 - val_loss: 0.5645 - val_acc: 0.7194
Epoch 14/15
50/50 [==============================] - 103s 2s/step - loss: 0.6083 - acc: 0.6700 - val_loss: 0.5899 - val_acc: 0.6781
Epoch 15/15
50/50 [==============================] - 101s 2s/step - loss: 0.6014 - acc: 0.6731 - val_loss: 0.5561 - val_acc: 0.7175
```

#CONVNET 2 Model summary:

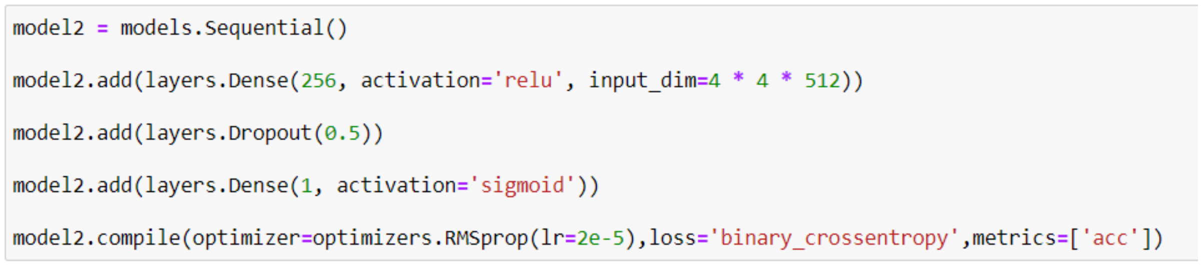


```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 36992)             0         
_________________________________________________________________
dropout (Dropout)            (None, 36992)             0         
_________________________________________________________________
dense_2 (Dense)              (None, 512)               18940416  
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513       
=================================================================
Total params: 19,034,177
Trainable params: 19,034,177
Non-trainable params: 0
_________________________________________________________________
```

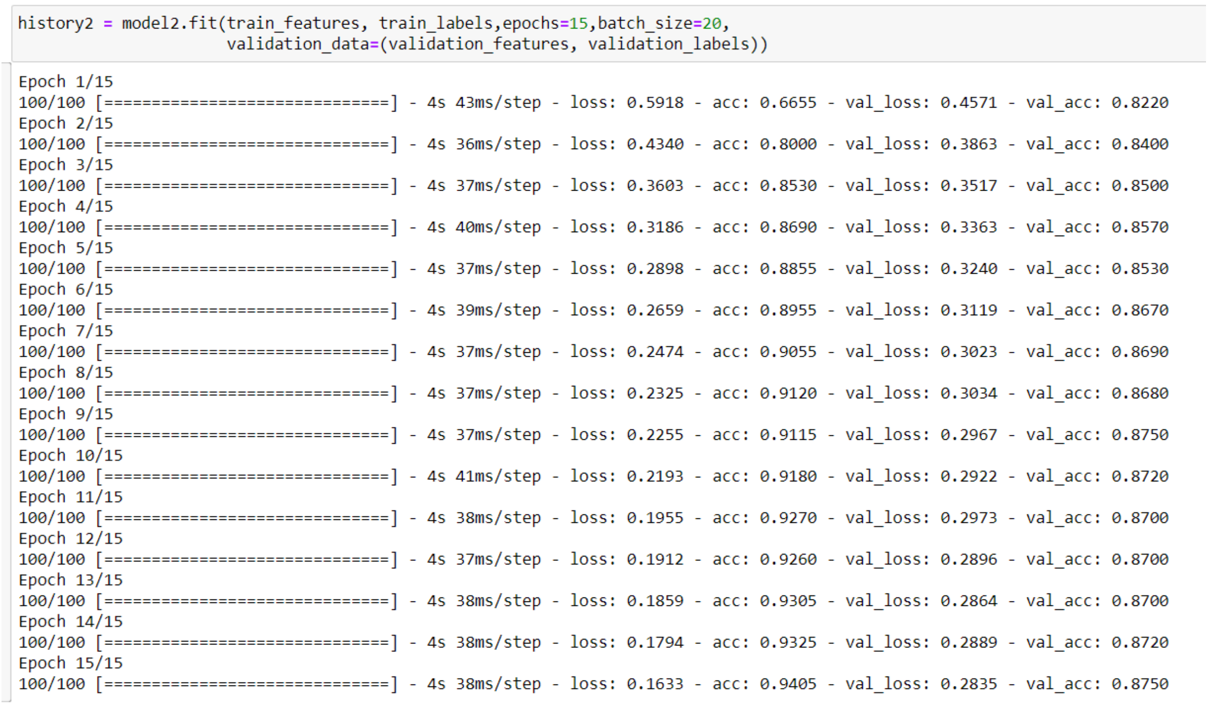
MODULE 6:

#Building CONVNET 3 Model:



```
model2 = models.Sequential()

model2.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))

model2.add(layers.Dropout(0.5))

model2.add(layers.Dense(1, activation='sigmoid'))

model2.compile(optimizer=optimizers.RMSprop(lr=2e-5),loss='binary_crossentropy',metrics=['acc'])
```

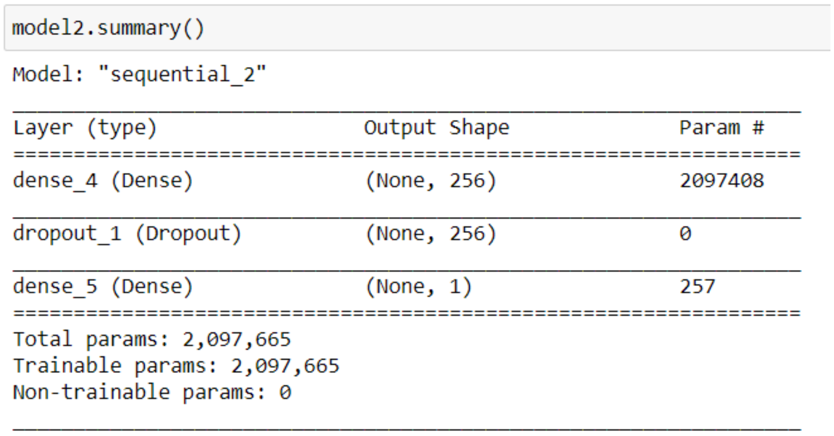
#Fitting CONVNET 3 Model:



```
history2 = model2.fit(train_features, train_labels,epochs=15,batch_size=20,
                      validation_data=(validation_features, validation_labels))
```

```
Epoch 1/15
100/100 [==============================] - 4s 43ms/step - loss: 0.5918 - acc: 0.6655 - val_loss: 0.4571 - val_acc: 0.8220
Epoch 2/15
100/100 [==============================] - 4s 36ms/step - loss: 0.4340 - acc: 0.8000 - val_loss: 0.3863 - val_acc: 0.8400
Epoch 3/15
100/100 [==============================] - 4s 37ms/step - loss: 0.3603 - acc: 0.8530 - val_loss: 0.3517 - val_acc: 0.8500
Epoch 4/15
100/100 [==============================] - 4s 40ms/step - loss: 0.3186 - acc: 0.8690 - val_loss: 0.3363 - val_acc: 0.8570
Epoch 5/15
100/100 [==============================] - 4s 37ms/step - loss: 0.2898 - acc: 0.8855 - val_loss: 0.3240 - val_acc: 0.8530
Epoch 6/15
100/100 [==============================] - 4s 39ms/step - loss: 0.2659 - acc: 0.8955 - val_loss: 0.3119 - val_acc: 0.8670
Epoch 7/15
100/100 [==============================] - 4s 37ms/step - loss: 0.2474 - acc: 0.9055 - val_loss: 0.3023 - val_acc: 0.8690
Epoch 8/15
100/100 [==============================] - 4s 37ms/step - loss: 0.2325 - acc: 0.9120 - val_loss: 0.3034 - val_acc: 0.8680
Epoch 9/15
100/100 [==============================] - 4s 37ms/step - loss: 0.2255 - acc: 0.9115 - val_loss: 0.2967 - val_acc: 0.8750
Epoch 10/15
100/100 [==============================] - 4s 41ms/step - loss: 0.2193 - acc: 0.9180 - val_loss: 0.2922 - val_acc: 0.8720
Epoch 11/15
100/100 [==============================] - 4s 38ms/step - loss: 0.1955 - acc: 0.9270 - val_loss: 0.2973 - val_acc: 0.8700
Epoch 12/15
100/100 [==============================] - 4s 37ms/step - loss: 0.1912 - acc: 0.9260 - val_loss: 0.2896 - val_acc: 0.8700
Epoch 13/15
100/100 [==============================] - 4s 38ms/step - loss: 0.1859 - acc: 0.9305 - val_loss: 0.2864 - val_acc: 0.8700
Epoch 14/15
100/100 [==============================] - 4s 38ms/step - loss: 0.1794 - acc: 0.9325 - val_loss: 0.2889 - val_acc: 0.8720
Epoch 15/15
100/100 [==============================] - 4s 38ms/step - loss: 0.1633 - acc: 0.9405 - val_loss: 0.2835 - val_acc: 0.8750
```

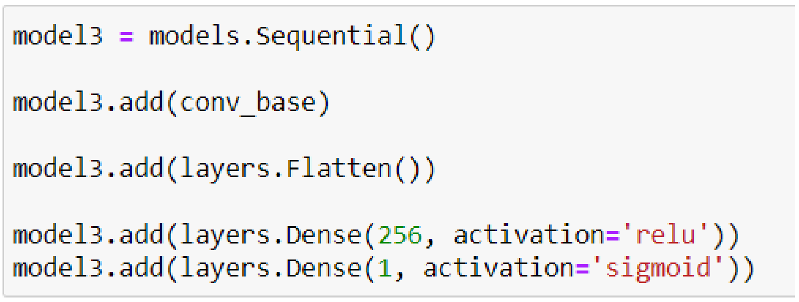
#CONVNET 3 Model summary:



```
model2.summary()
```

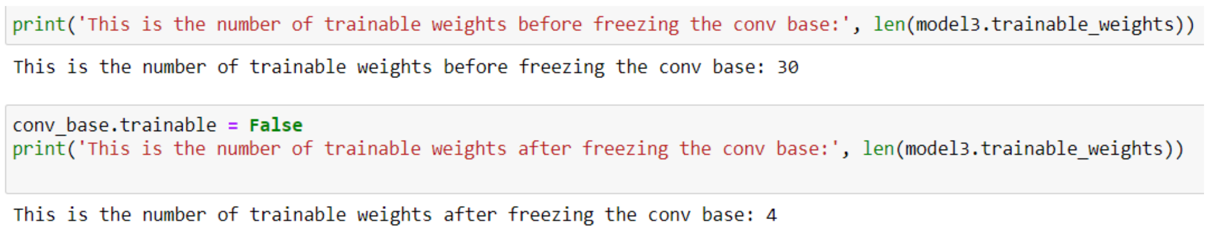
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 256)               2097408
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 257
=================================================================
Total params: 2,097,665
Trainable params: 2,097,665
Non-trainable params: 0
_________________________________________________________________
```

#Building CONVNET 4 Model:



```
model3 = models.Sequential()

model3.add(conv_base)

model3.add(layers.Flatten())

model3.add(layers.Dense(256, activation='relu'))
model3.add(layers.Dense(1, activation='sigmoid'))
```

# CONVNET 4a - with frozen layers from pretrained model



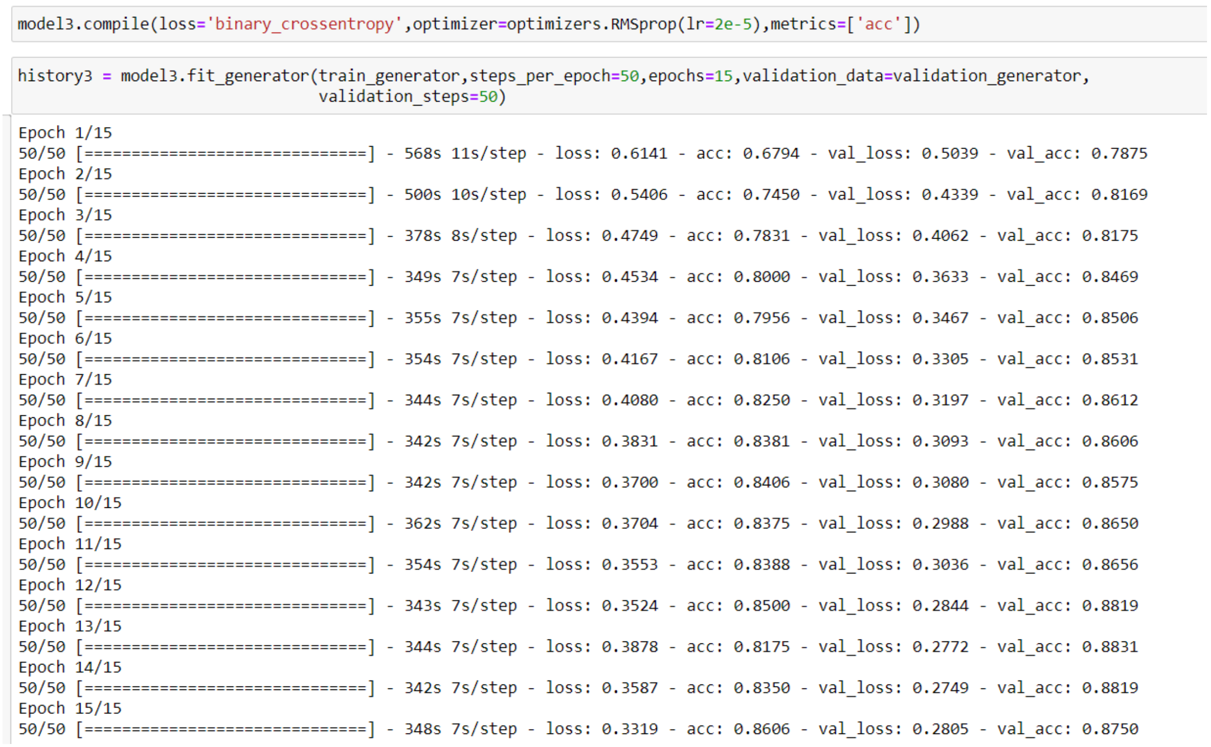
```
print('This is the number of trainable weights before freezing the conv base:', len(model3.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 30
```

```
conv_base.trainable = False
print('This is the number of trainable weights after freezing the conv base:', len(model3.trainable_weights))
```

```
This is the number of trainable weights after freezing the conv base: 4
```

#Fitting CONVNET 4a Model:

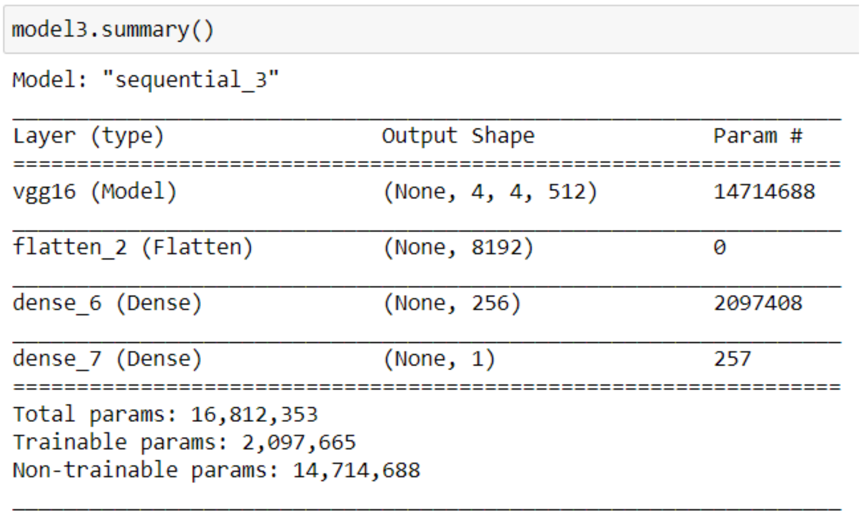


```
model3.compile(loss='binary_crossentropy',optimizer=optimizers.RMSprop(lr=2e-5),metrics=['acc'])
```

```
history3 = model3.fit_generator(train_generator,steps_per_epoch=50,epochs=15,validation_data=validation_generator,
                                validation_steps=50)
```

```
Epoch 1/15
50/50 [==============================] - 568s 11s/step - loss: 0.6141 - acc: 0.6794 - val_loss: 0.5039 - val_acc: 0.7875
Epoch 2/15
50/50 [==============================] - 500s 10s/step - loss: 0.5406 - acc: 0.7450 - val_loss: 0.4339 - val_acc: 0.8169
Epoch 3/15
50/50 [==============================] - 378s 8s/step - loss: 0.4749 - acc: 0.7831 - val_loss: 0.4062 - val_acc: 0.8175
Epoch 4/15
50/50 [==============================] - 349s 7s/step - loss: 0.4534 - acc: 0.8000 - val_loss: 0.3633 - val_acc: 0.8469
Epoch 5/15
50/50 [==============================] - 355s 7s/step - loss: 0.4394 - acc: 0.7956 - val_loss: 0.3467 - val_acc: 0.8506
Epoch 6/15
50/50 [==============================] - 354s 7s/step - loss: 0.4167 - acc: 0.8106 - val_loss: 0.3305 - val_acc: 0.8531
Epoch 7/15
50/50 [==============================] - 344s 7s/step - loss: 0.4080 - acc: 0.8250 - val_loss: 0.3197 - val_acc: 0.8612
Epoch 8/15
50/50 [==============================] - 342s 7s/step - loss: 0.3831 - acc: 0.8381 - val_loss: 0.3093 - val_acc: 0.8606
Epoch 9/15
50/50 [==============================] - 342s 7s/step - loss: 0.3700 - acc: 0.8406 - val_loss: 0.3080 - val_acc: 0.8575
Epoch 10/15
50/50 [==============================] - 362s 7s/step - loss: 0.3704 - acc: 0.8375 - val_loss: 0.2988 - val_acc: 0.8650
Epoch 11/15
50/50 [==============================] - 354s 7s/step - loss: 0.3553 - acc: 0.8388 - val_loss: 0.3036 - val_acc: 0.8656
Epoch 12/15
50/50 [==============================] - 343s 7s/step - loss: 0.3524 - acc: 0.8500 - val_loss: 0.2844 - val_acc: 0.8819
Epoch 13/15
50/50 [==============================] - 344s 7s/step - loss: 0.3878 - acc: 0.8175 - val_loss: 0.2772 - val_acc: 0.8831
Epoch 14/15
50/50 [==============================] - 342s 7s/step - loss: 0.3587 - acc: 0.8350 - val_loss: 0.2749 - val_acc: 0.8819
Epoch 15/15
50/50 [==============================] - 348s 7s/step - loss: 0.3319 - acc: 0.8606 - val_loss: 0.2805 - val_acc: 0.8750
```

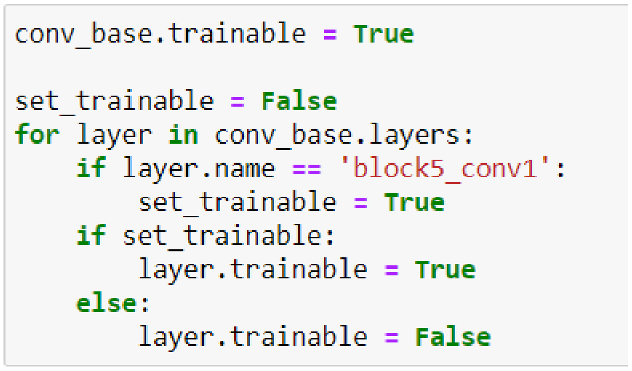
#CONVNET 4a Model summary:



```
model3.summary()
```

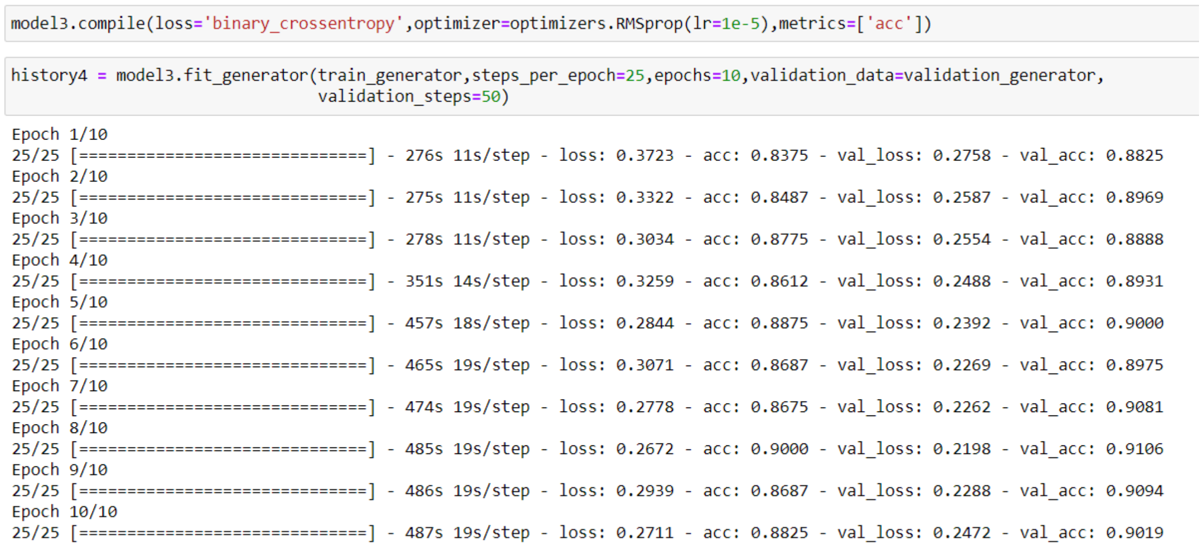
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688  
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_6 (Dense)              (None, 256)               2097408   
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 257       
=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
_________________________________________________________________
```

#CONVNET 4b - with Fine tuning of pretrained model



```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

#Fitting CONVNET 4b Model:

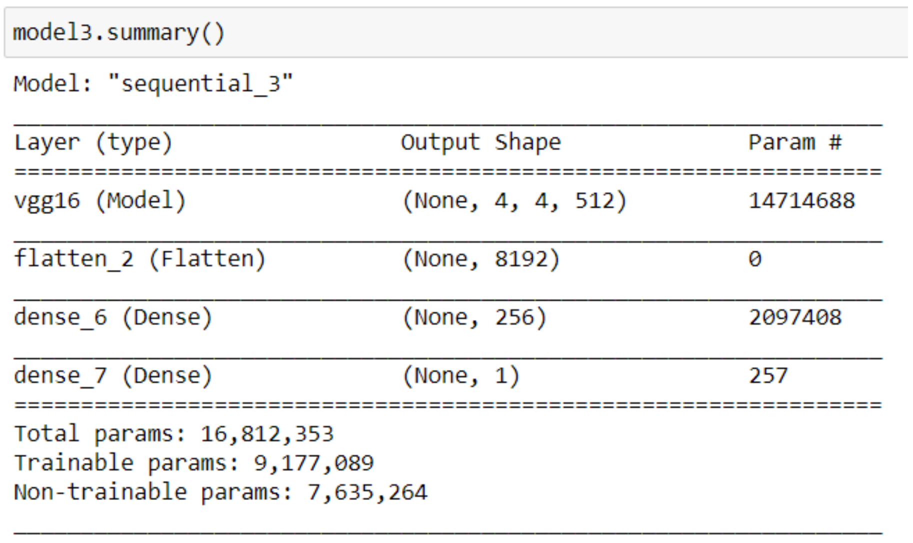


```
model3.compile(loss='binary_crossentropy',optimizer=optimizers.RMSprop(lr=1e-5),metrics=['acc'])
```

```
history4 = model3.fit_generator(train_generator,steps_per_epoch=25,epochs=10,validation_data=validation_generator,
                                validation_steps=50)
```

```
Epoch 1/10
25/25 [==============================] - 276s 11s/step - loss: 0.3723 - acc: 0.8375 - val_loss: 0.2758 - val_acc: 0.8825
Epoch 2/10
25/25 [==============================] - 275s 11s/step - loss: 0.3322 - acc: 0.8487 - val_loss: 0.2587 - val_acc: 0.8969
Epoch 3/10
25/25 [==============================] - 278s 11s/step - loss: 0.3034 - acc: 0.8775 - val_loss: 0.2554 - val_acc: 0.8888
Epoch 4/10
25/25 [==============================] - 351s 14s/step - loss: 0.3259 - acc: 0.8612 - val_loss: 0.2488 - val_acc: 0.8931
Epoch 5/10
25/25 [==============================] - 457s 18s/step - loss: 0.2844 - acc: 0.8875 - val_loss: 0.2392 - val_acc: 0.9000
Epoch 6/10
25/25 [==============================] - 465s 19s/step - loss: 0.3071 - acc: 0.8687 - val_loss: 0.2269 - val_acc: 0.8975
Epoch 7/10
25/25 [==============================] - 474s 19s/step - loss: 0.2778 - acc: 0.8675 - val_loss: 0.2262 - val_acc: 0.9081
Epoch 8/10
25/25 [==============================] - 485s 19s/step - loss: 0.2672 - acc: 0.9000 - val_loss: 0.2198 - val_acc: 0.9106
Epoch 9/10
25/25 [==============================] - 486s 19s/step - loss: 0.2939 - acc: 0.8687 - val_loss: 0.2288 - val_acc: 0.9094
Epoch 10/10
25/25 [==============================] - 487s 19s/step - loss: 0.2711 - acc: 0.8825 - val_loss: 0.2472 - val_acc: 0.9019
```

#CONVNET 4b Model summary:

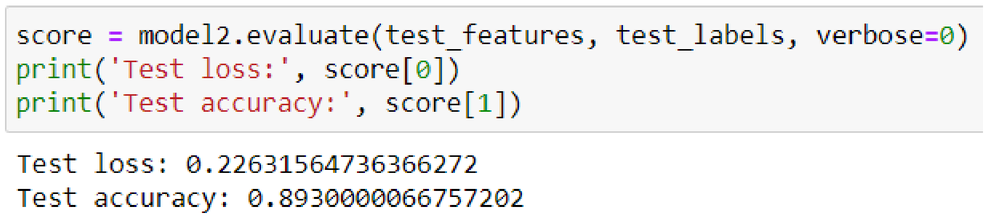


```
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 9,177,089
Non-trainable params: 7,635,264
_________________________________________________________________
```

MODULE 7:

#Printing Test Accuracy & Loss score:

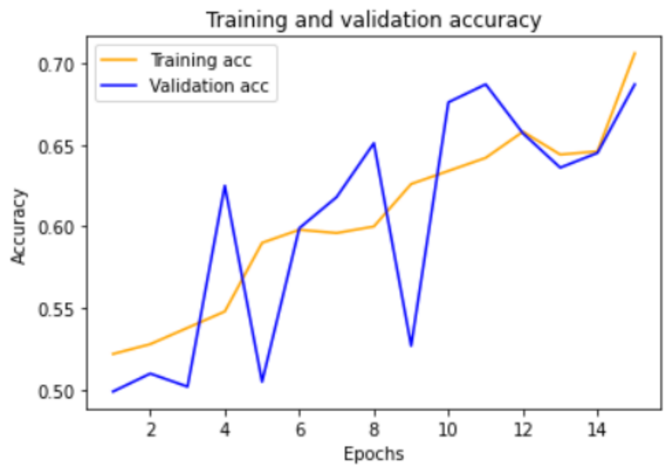


```
score = model2.evaluate(test_features, test_labels, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

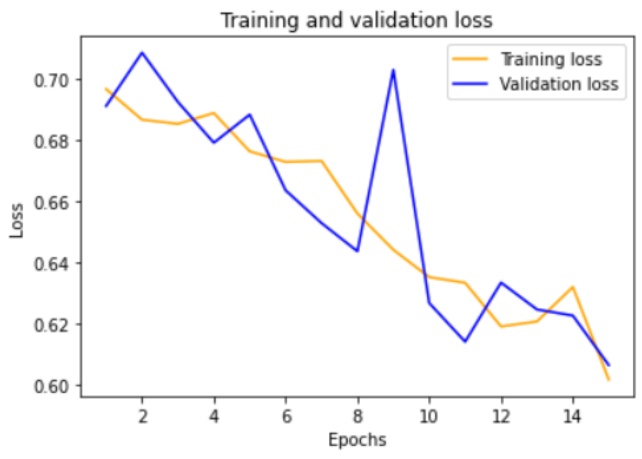
```
Test loss: 0.22631564736366272
Test accuracy: 0.8930000066757202
```

#Accuracy and Loss curve for CONVNET 1 Model:

```
acc = history.history['acc']
val_acc = history.history['val_acc']

epochs = range(1, 16)

plt.plot(epochs, acc, 'orange', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```

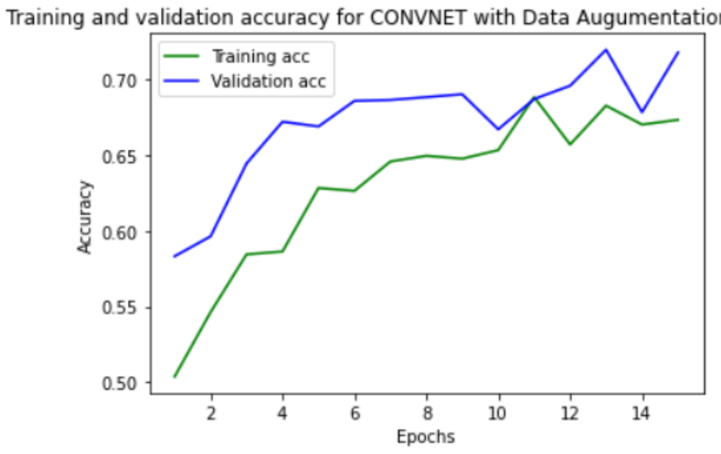


```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, 16)

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

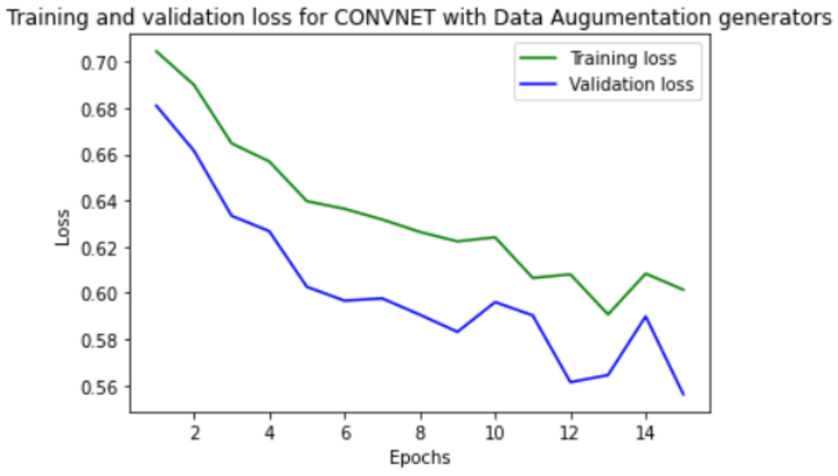


#Accuracy and Loss curve for CONVNET 2 Model:

```
acc = history1.history['acc']
val_acc = history1.history['val_acc']

epochs = range(1, 16)

plt.plot(epochs, acc, 'green', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy for CONVNET with Data Augumentation generators')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```

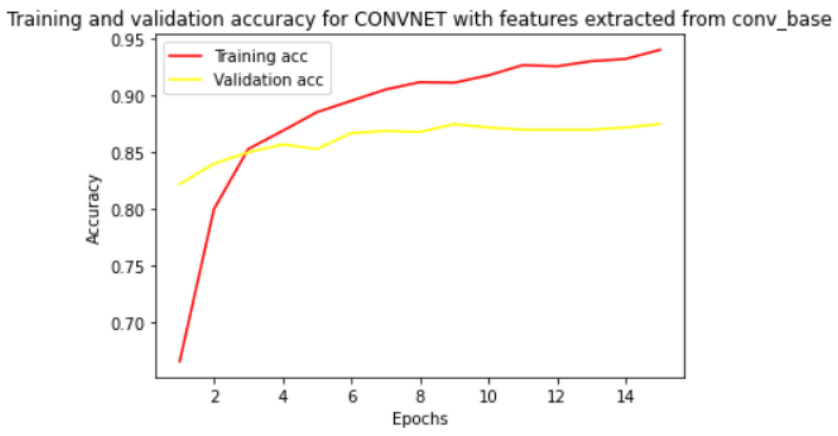


```
loss = history1.history['loss']
val_loss = history1.history['val_loss']

epochs = range(1, 16)

plt.plot(epochs, loss, 'green', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss for CONVNET with Data Augumentation generators')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

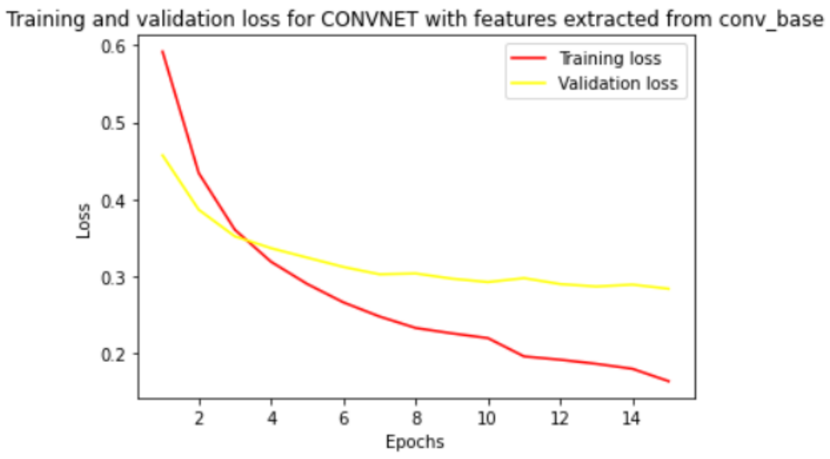


#Accuracy and Loss curve for CONVNET 3 Model:

```
acc = history2.history['acc']
val_acc = history2.history['val_acc']

epochs = range(1, 16)

plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'yellow', label='Validation acc')
plt.title('Training and validation accuracy for CONVNET with features extracted from conv_base')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```

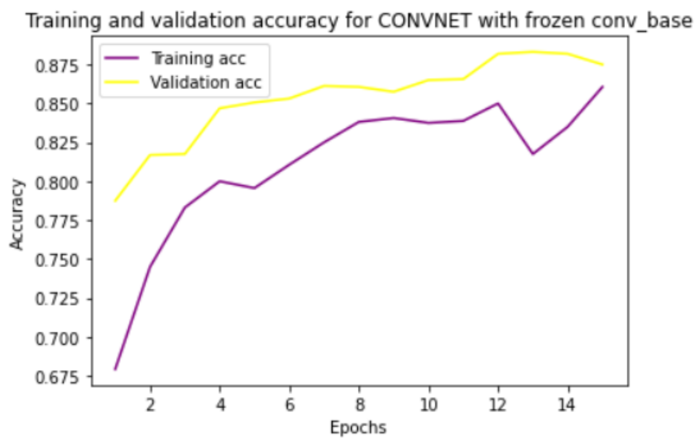


```
loss = history2.history['loss']
val_loss = history2.history['val_loss']

epochs = range(1, 16)

plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'yellow', label='Validation loss')
plt.title('Training and validation loss for CONVNET with features extracted from conv_base')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

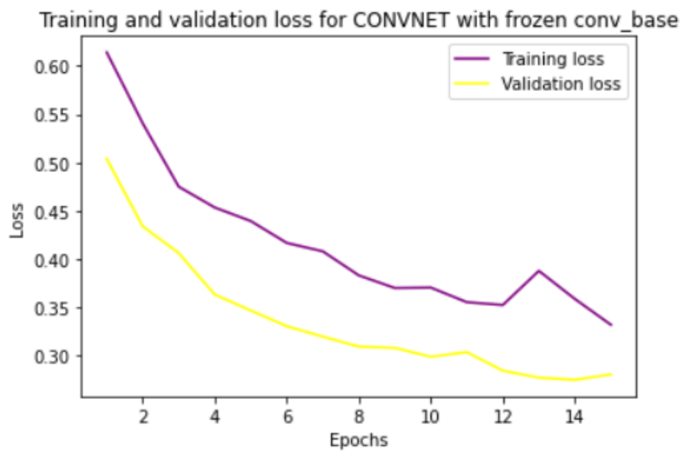


#Accuracy and Loss curve for CONVNET 4a Model:

```
acc = history3.history['acc']
val_acc = history3.history['val_acc']

epochs = range(1, 16)

plt.plot(epochs, acc, 'purple', label='Training acc')
plt.plot(epochs, val_acc, 'yellow', label='Validation acc')
plt.title('Training and validation accuracy for CONVNET with frozen conv_base')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```

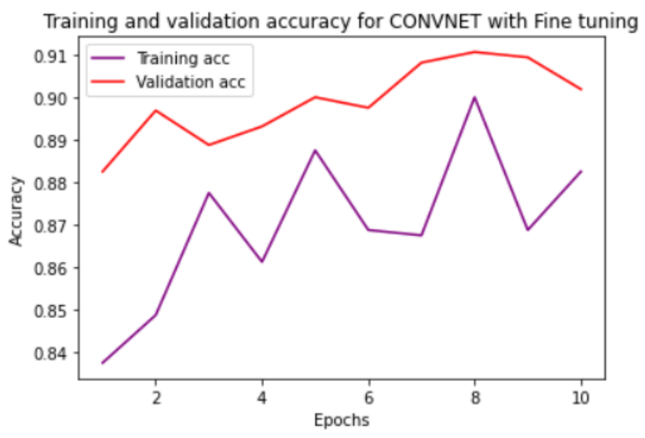


```
loss = history3.history['loss']
val_loss = history3.history['val_loss']

epochs = range(1, 16)

plt.plot(epochs, loss, 'purple', label='Training loss')
plt.plot(epochs, val_loss, 'yellow', label='Validation loss')
plt.title('Training and validation loss for CONVNET with frozen conv_base')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

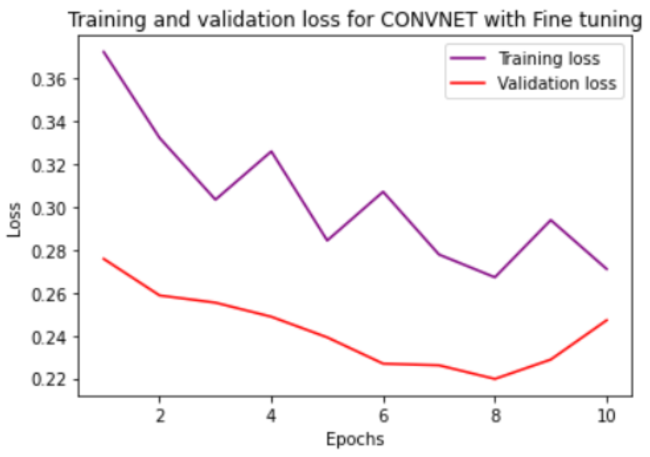


#Accuracy and Loss curve for CONVNET 4b Model:

```
acc = history4.history['acc']
val_acc = history4.history['val_acc']

epochs = range(1, 11)

plt.plot(epochs, acc, 'purple', label='Training acc')
plt.plot(epochs, val_acc, 'red', label='Validation acc')
plt.title('Training and validation accuracy for CONVNET with Fine tuning')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```



```
loss = history4.history['loss']
val_loss = history4.history['val_loss']

epochs = range(1, 11)

plt.plot(epochs, loss, 'purple', label='Training loss')
plt.plot(epochs, val_loss, 'red', label='Validation loss')
plt.title('Training and validation loss for CONVNET with Fine tuning')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



MODULE 8:

Input 1:

```
image = load_img('dog.jpg', target_size=(224, 224))
image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
image = preprocess_input(image)

yhat =  conv_base.predict(image)
label = decode_predictions(yhat)
label = label[0][0]
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
Doberman (36.76%)
```

Input 2:

```
image = load_img('cat.jpg', target_size=(224, 224))
image = img_to_array(image)
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
image = preprocess_input(image)

yhat =  conv_base.predict(image)
label = decode_predictions(yhat)
label = label[0][0]
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
Egyptian_cat (29.80%)
```

- When the input is passed to the conv_base model it predicts the target label and also specifies its breed i.e. which type of cat or dog

RESULT:

Thus the four different CNN models is trained with different set of features using Transfer learning application as well as without using transfer learning application and their performance is viewed by their accuracy and loss curves.

CONCLUSION:

CNN models are best for classifying image dataset with good accuracy. CNN Models with transfer learning applications will enhance the performance of the model. CNN Model with transfer learning application is more flexible than normal CNN. Computational power for classifying image dataset is much less when compared to other learning models because it uses advanced feature extraction technique which uses filters, kernel, pooling etc to extract features from image

REFERENCE:

- https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
- Deep Learning with Python by FRANÇOIS CHOLLET, Manning Publications Co.