CS6005-DEEP LEARNING MINI PROJECT-COMPUTER VISION DOCUMENT

Computer Vision
Dogs and Cats Dataset using transfer learning

-Syed Adnan Moin.S

2018103074

Problem Statement:

Neural networks have achieved appreciable performance at image classification, they have been characterized by feature engineering, a tedious process that results in poor generalization to test data.

If we use the simple CNN architecture that we use in the basic datasets such as MNIST, we will get a low validation accuracy of around 60% which is never desirable. I present an approach which works to achieve improved performances without feature engineering.

Learnable filters and pooling layers are used to extract underlying image features. Dropout, regularization along with variation in convolution strategies are applied to reduce overfitting while ensuring increased accuracies in validation and testing. Better test accuracy with reduced overfitting is achieved with a deeper network

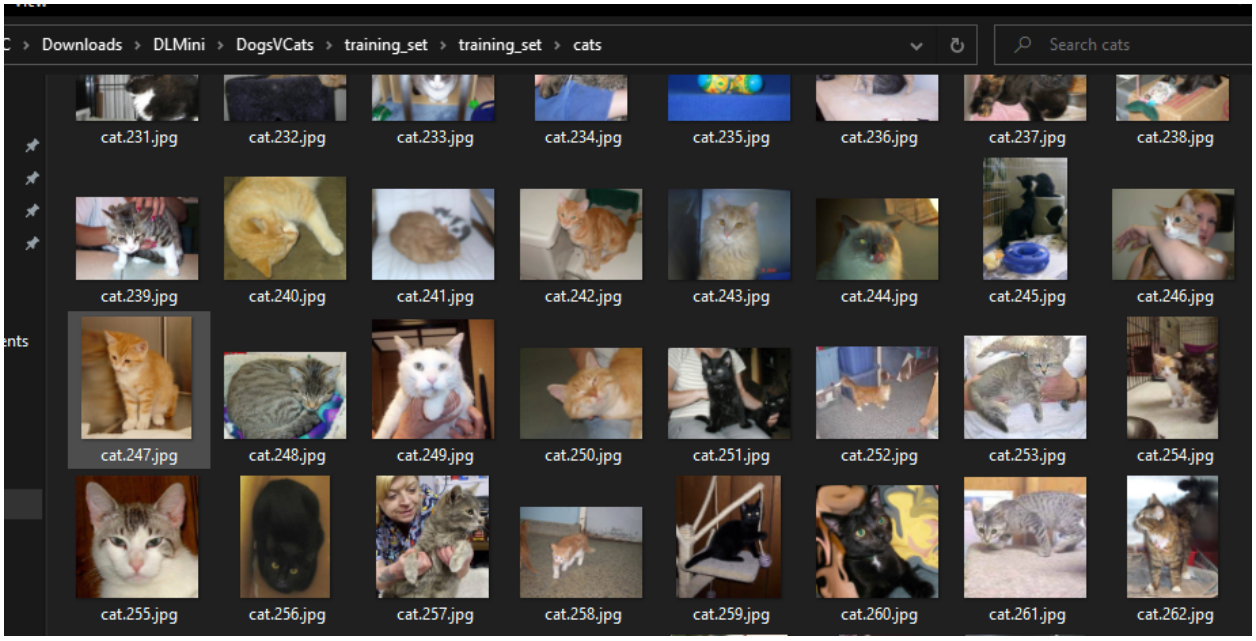
Transfer learning is the reuse of a pre-trained model on a new problem. It can train deep neural networks with comparatively little data. This is very useful in the data science field since most real-world problems typically do not have millions of labeled data points to train such complex models

Dataset Description:

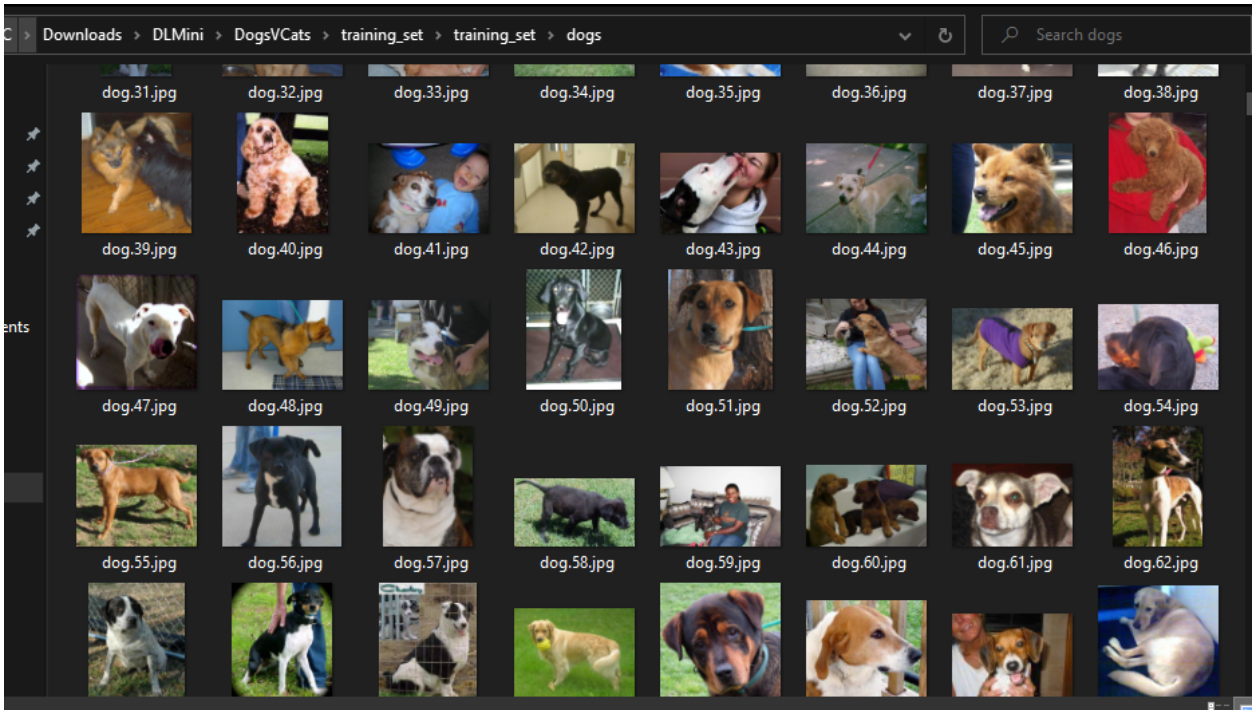
Link: https://www.tfcertification.com/pages/deep-learning

The dataset used for image classification is the Dogs vs. Cats dataset. It is retrieved from Deep Learning A-Z™: Download Practice Datasets, a deep learning course on udemy. This dataset contains 10,000 32x32 colour images in 2 different classes. The 2 different classes are dogs and cats. There are 5000 images of dogs and cats each. The training set consists of 4000 images and the testing set consists of 1000 images.

Below is a snapshot of the cats dataset



Below is a snapshot of the dogs dataset



Module Description:

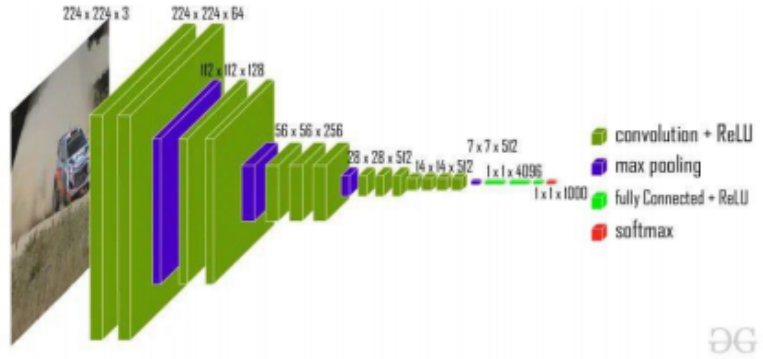
- **Module 1: Data PreProcessing:**
    - The dataset used is the Dogs and Cats which is already downloaded. The dataset is loaded using the directory path to the train and test parts being the x_train, y_train and x_test, y_test.
    - Training images i.e from x_train are plotted using the pyplot package in the matplotlib library.These images are plotted to describe the layout of the figure. The rows and columns of the frame which is for plotting is given in the form of arguments, and the third argument is for providing the index of the current plot.
    - In this module the data is processed using various data preprocessing techniques namely:
        - **Reshaping:**
            - The row vector for an image has the exact same number of elements if you calculate 32*32*3 == 3072. In order to reshape the row vector into (width x height x num_channel) form, there are two steps required. The first step is to use reshape function,

and the second step is to use the transpose function in numpy.

- reshape transforms an array to a new shape without changing its data. Here, the phrase without changing its data is an important part since we don't want to affect the data.

- Splitting images into directories.

- **Module 2: Importing pretrained model 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 the model is shown below.



- **Module 3: Pre-Trained Model as Feature Extractor Preprocessor**
    - The features for training, validation and test data are extracted from pretrained model VGG16.
- **Module 4: Building a CNN without Transfer learning Application**
    - Two Convolutional models are developed in this module
        - CONVNET 1 without Data Augmentation Parameters
        - CONVNET 2 with Data Augmentation Parameters
    - **Layers in CONVNET 1**
        i. Conv2D (Convolution)
            1. The **convolutional layer** is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume.

        ii. MaxPooling2D (Pooling)
            1. its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. It operates on each feature map independently.

        iii. Flatten

1. Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector

iv. Dense

1. 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

- **Layers in CONVNET 2**

v. Conv2D (Convolution)

1. The **convolutional layer** is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume.

vi. MaxPooling2D (Pooling)

1. its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. It operates on each feature map independently.

vii. Flatten

1. Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector

viii. Regularization function

1. I used Dropout technique to regularize which specifies the percentage of neurons to be dropped at each iteration.

ix. Dense

1. 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:**
  - 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
- **Parameters**
  - 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 5: Building a CNN model without Transfer learning Application**
  - 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
  - **Layers in CONVNET 3**
    - Regularization function

        - I used Dropout technique to regularize which specifies the percentage of neurons to be dropped 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
- **Layers in CONVNET 4**
    - 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.
    - 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
    - Flatten
        - Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector
- **Activation Function:**

        - 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
    - **Parameters**
        - 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: Performance Metrics**
    - The model is evaluated by
        - By plotting accuracy and loss curves
        - Displaying accuracy and loss score
- **Module 7: Making a prediction**
    - The VGG16 model is used predict the output for the sample input

Code Snapshots:

Importing the 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 1

#Creating directory cats_and_dogs_small with folders

```
original_dataset_dir1 = r"C:\Users\SyedAdnan\DLMini\train\cats"
original_dataset_dir2 = r"C:\Users\SyedAdnan\DLMini\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)
```

#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)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

#Reshaping the images without Data Augmentation

```
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.
```

#Reshaping the images with Data Augmentation

```
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.
```

Module 2

#Importing VGG16 Model

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

#CNN Model Summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
```

activation_4 (Activation)	(None, 8, 8, 128)	0
batch_normalization_4 (Batch	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
activation_5 (Activation)	(None, 8, 8, 128)	0
batch_normalization_5 (Batch	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20490

Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896

Module 3

#Extract features using pretrained model VGG16

```
: from keras.preprocessing.image import ImageDataGenerator

base_dir = r"C:\Users\SyedAdnan\DLMini\catsNdogs"

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))
```

Module 4

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'))
```

Fitting CONVNET 1 Model:

```
history = model.fit_generator(train_generator,steps_per_epoch=25,epochs=15,validation_data=validation_generator,validat
```

```
WARNING:tensorflow:From <ipython-input-14-5fde3eea3e78>:1: Model.fit_generator (from tensorflow.python.keras.engine.t
raining) is 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: 0.499
0
Epoch 2/15
25/25 [==============================] - 35s 1s/step - loss: 0.6868 - acc: 0.5280 - val_loss: 0.7088 - val_acc: 0.510
0
Epoch 3/15
25/25 [==============================] - 35s 1s/step - loss: 0.6855 - acc: 0.5380 - val_loss: 0.6926 - val_acc: 0.502
0
Epoch 4/15
25/25 [==============================] - 34s 1s/step - loss: 0.6890 - acc: 0.5480 - val_loss: 0.6793 - val_acc: 0.625
0
Epoch 5/15
25/25 [==============================] - 35s 1s/step - loss: 0.6765 - acc: 0.5900 - val_loss: 0.6885 - val_acc: 0.505
0
Epoch 6/15
25/25 [==============================] - 35s 1s/step - loss: 0.6730 - acc: 0.5980 - val_loss: 0.6637 - val_acc: 0.599
0
Epoch 7/15
25/25 [==============================] - 35s 1s/step - loss: 0.6733 - acc: 0.5960 - val_loss: 0.6530 - val_acc: 0.618
0
Epoch 8/15
25/25 [==============================] - 35s 1s/step - loss: 0.6561 - acc: 0.6000 - val_loss: 0.6437 - val_acc: 0.651
0
Epoch 9/15
25/25 [==============================] - 33s 1s/step - loss: 0.6442 - acc: 0.6260 - val_loss: 0.7032 - val_acc: 0.527
0
Epoch 10/15
25/25 [==============================] - 32s 1s/step - loss: 0.6352 - acc: 0.6340 - val_loss: 0.6267 - val_acc: 0.676
0
Epoch 11/15
25/25 [==============================] - 34s 1s/step - loss: 0.6334 - acc: 0.6420 - val_loss: 0.6141 - val_acc: 0.687
0
Epoch 12/15
25/25 [==============================] - 33s 1s/step - loss: 0.6191 - acc: 0.6580 - val_loss: 0.6334 - val_acc: 0.657
0
Epoch 13/15
25/25 [==============================] - 34s 1s/step - loss: 0.6207 - acc: 0.6440 - val_loss: 0.6246 - val_acc: 0.636
0
Epoch 14/15
25/25 [==============================] - 34s 1s/step - loss: 0.6320 - acc: 0.6460 - val_loss: 0.6226 - val_acc: 0.645
0
Epoch 15/15
25/25 [==============================] - 33s 1s/step - loss: 0.6017 - acc: 0.7060 - val_loss: 0.6064 - val_acc: 0.687
0
```

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,vali
```

```
Epoch 1/15
50/50 [==============================] - 96s 2s/step - loss: 0.7045 - acc: 0.5038 - val_loss: 0.6810 - val_acc: 0.583
1
Epoch 2/15
50/50 [==============================] - 97s 2s/step - loss: 0.6900 - acc: 0.5462 - val_loss: 0.6615 - val_acc: 0.596
2
Epoch 3/15
50/50 [==============================] - 109s 2s/step - loss: 0.6647 - acc: 0.5844 - val_loss: 0.6333 - val_acc: 0.64
44
Epoch 4/15
50/50 [==============================] - 99s 2s/step - loss: 0.6569 - acc: 0.5863 - val_loss: 0.6267 - val_acc: 0.671
9
Epoch 5/15
50/50 [==============================] - 101s 2s/step - loss: 0.6397 - acc: 0.6281 - val_loss: 0.6026 - val_acc: 0.66
87
Epoch 6/15
50/50 [==============================] - 101s 2s/step - loss: 0.6364 - acc: 0.6263 - val_loss: 0.5966 - val_acc: 0.68
56
Epoch 7/15
50/50 [==============================] - 101s 2s/step - loss: 0.6317 - acc: 0.6456 - val_loss: 0.5976 - val_acc: 0.68
62
Epoch 8/15
50/50 [==============================] - 98s 2s/step - loss: 0.6264 - acc: 0.6494 - val_loss: 0.5906 - val_acc: 0.688
1
Epoch 9/15
50/50 [==============================] - 99s 2s/step - loss: 0.6223 - acc: 0.6475 - val_loss: 0.5832 - val_acc: 0.690
0
Epoch 10/15
50/50 [==============================] - 100s 2s/step - loss: 0.6240 - acc: 0.6531 - val_loss: 0.5960 - val_acc: 0.66
69
Epoch 11/15
50/50 [==============================] - 99s 2s/step - loss: 0.6065 - acc: 0.6881 - val_loss: 0.5903 - val_acc: 0.686
9
```

```
-
Epoch 12/15
50/50 [==============================] - 98s 2s/step - loss: 0.6080 - acc: 0.6569 - val_loss: 0.5614 - val_acc: 0.695
6
Epoch 13/15
50/50 [==============================] - 100s 2s/step - loss: 0.5907 - acc: 0.6825 - val_loss: 0.5645 - val_acc: 0.71
94
Epoch 14/15
50/50 [==============================] - 103s 2s/step - loss: 0.6083 - acc: 0.6700 - val_loss: 0.5899 - val_acc: 0.67
81
Epoch 15/15
50/50 [==============================] - 101s 2s/step - loss: 0.6014 - acc: 0.6731 - val_loss: 0.5561 - val_acc: 0.71
75
```

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 5:

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, valida
```

```
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


  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,valic
```

```
Epoch 1/15
50/50 [==============================] - 568s 11s/step - loss: 0.6141 - acc: 0.6794 - val_loss: 0.5039 - val_acc: 0.7
875
Epoch 2/15
50/50 [==============================] - 500s 10s/step - loss: 0.5406 - acc: 0.7450 - val_loss: 0.4339 - val_acc: 0.8
169
Epoch 3/15
50/50 [==============================] - 378s 8s/step - loss: 0.4749 - acc: 0.7831 - val_loss: 0.4062 - val_acc: 0.81
75
Epoch 4/15
50/50 [==============================] - 349s 7s/step - loss: 0.4534 - acc: 0.8000 - val_loss: 0.3633 - val_acc: 0.84
69
Epoch 5/15
50/50 [==============================] - 355s 7s/step - loss: 0.4394 - acc: 0.7956 - val_loss: 0.3467 - val_acc: 0.85
06
Epoch 6/15
50/50 [==============================] - 354s 7s/step - loss: 0.4167 - acc: 0.8106 - val_loss: 0.3305 - val_acc: 0.85
31
Epoch 7/15
50/50 [==============================] - 344s 7s/step - loss: 0.4080 - acc: 0.8250 - val_loss: 0.3197 - val_acc: 0.86
12
Epoch 8/15
50/50 [==============================] - 342s 7s/step - loss: 0.3831 - acc: 0.8381 - val_loss: 0.3093 - val_acc: 0.86
06
Epoch 9/15
50/50 [==============================] - 342s 7s/step - loss: 0.3700 - acc: 0.8406 - val_loss: 0.3080 - val_acc: 0.85
75
Epoch 10/15
50/50 [==============================] - 362s 7s/step - loss: 0.3704 - acc: 0.8375 - val_loss: 0.2988 - val_acc: 0.86
50
```

```
Epoch 11/15
50/50 [==============================] - 354s 7s/step - loss: 0.3553 - acc: 0.8388 - val_loss: 0.3036 - val_acc: 0.86
56
Epoch 12/15
50/50 [==============================] - 343s 7s/step - loss: 0.3524 - acc: 0.8500 - val_loss: 0.2844 - val_acc: 0.88
19
Epoch 13/15
50/50 [==============================] - 344s 7s/step - loss: 0.3878 - acc: 0.8175 - val_loss: 0.2772 - val_acc: 0.88
31
Epoch 14/15
50/50 [==============================] - 342s 7s/step - loss: 0.3587 - acc: 0.8350 - val_loss: 0.2749 - val_acc: 0.88
19
Epoch 15/15
50/50 [==============================] - 348s 7s/step - loss: 0.3319 - acc: 0.8606 - val_loss: 0.2805 - val_acc: 0.87
50
```

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,valic
```

```
Epoch 1/10
25/25 [==============================] - 276s 11s/step - loss: 0.3723 - acc: 0.8375 - val_loss: 0.2758 - val_acc: 0.8
825
Epoch 2/10
25/25 [==============================] - 275s 11s/step - loss: 0.3322 - acc: 0.8487 - val_loss: 0.2587 - val_acc: 0.8
969
Epoch 3/10
25/25 [==============================] - 278s 11s/step - loss: 0.3034 - acc: 0.8775 - val_loss: 0.2554 - val_acc: 0.8
888
Epoch 4/10
25/25 [==============================] - 351s 14s/step - loss: 0.3259 - acc: 0.8612 - val_loss: 0.2488 - val_acc: 0.8
931
Epoch 5/10
25/25 [==============================] - 457s 18s/step - loss: 0.2844 - acc: 0.8875 - val_loss: 0.2392 - val_acc: 0.9
000
Epoch 6/10
25/25 [==============================] - 465s 19s/step - loss: 0.3071 - acc: 0.8687 - val_loss: 0.2269 - val_acc: 0.8
975
Epoch 7/10
25/25 [==============================] - 474s 19s/step - loss: 0.2778 - acc: 0.8675 - val_loss: 0.2262 - val_acc: 0.9
081
Epoch 8/10
25/25 [==============================] - 485s 19s/step - loss: 0.2672 - acc: 0.9000 - val_loss: 0.2198 - val_acc: 0.9
106
Epoch 9/10
25/25 [==============================] - 486s 19s/step - loss: 0.2939 - acc: 0.8687 - val_loss: 0.2288 - val_acc: 0.9
094
Epoch 10/10
25/25 [==============================] - 487s 19s/step - loss: 0.2711 - acc: 0.8825 - val_loss: 0.2472 - val_acc: 0.9
```

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 6

#Printing Test Accuracy & Loss score

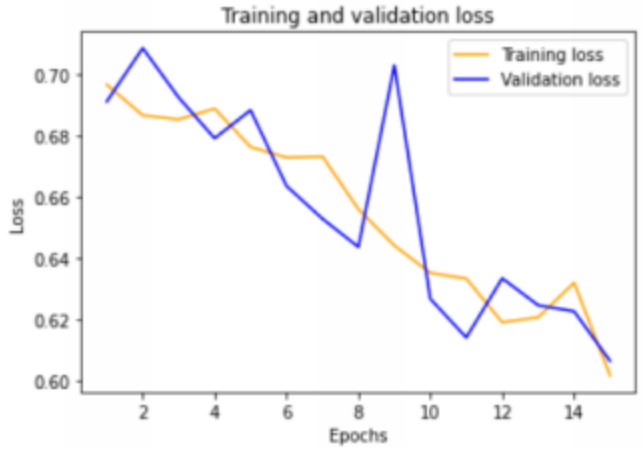
```
score = model2.evaluate(test_features, test_labels, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

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

#Accuracy and Loss curve for CONVNET 1 Model

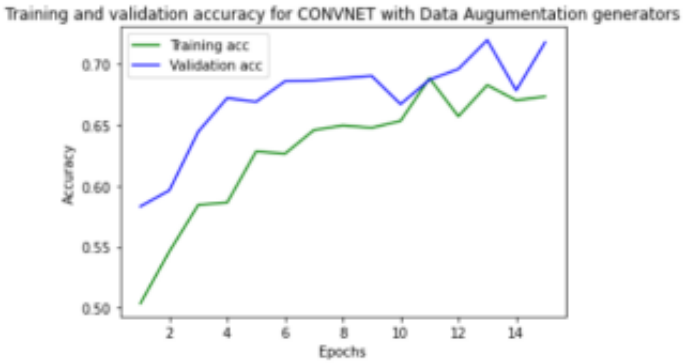
```
acc = history.history['acc']
val_acc = history.history['val_acc']

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()
```

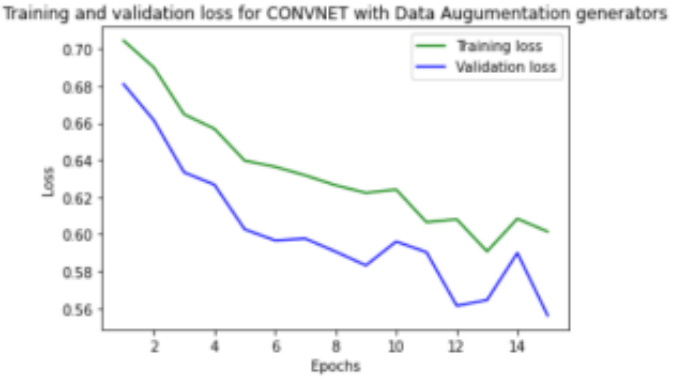


#Accuracy and Loss curve for CONVNET 2 Model

```
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()
```

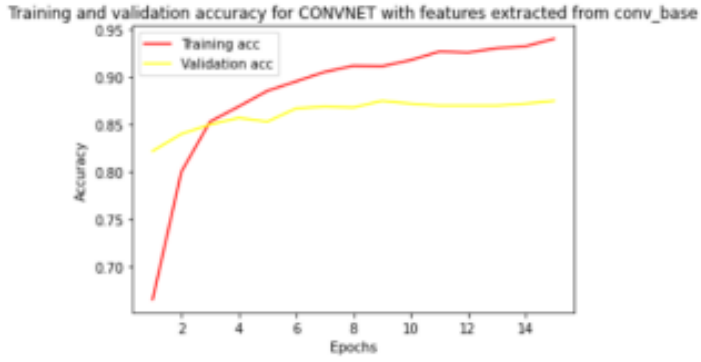



```
loss = history1.history['loss']
val_loss = history1.history['val_loss']

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()
```

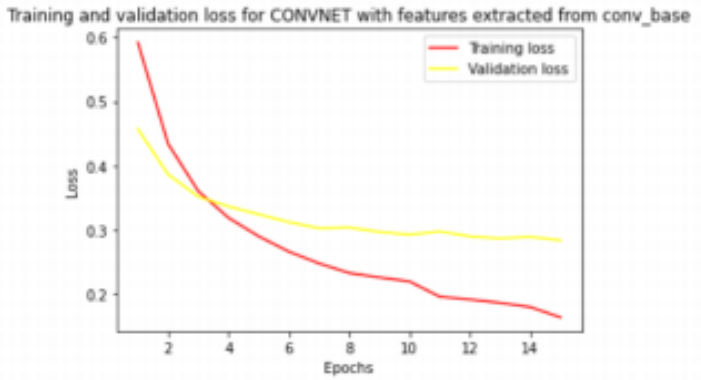


#Accuracy and Loss curve for CONVNET 3 Model

```
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()
```

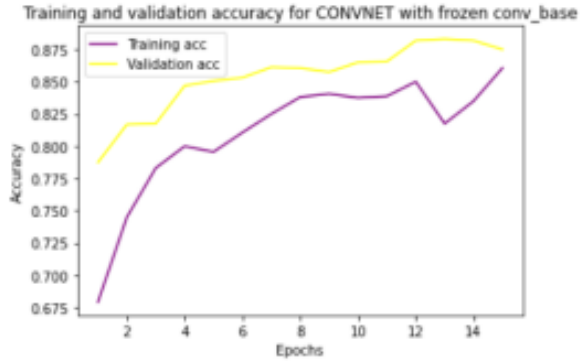



```
loss = history2.history['loss']
val_loss = history2.history['val_loss']

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()
```

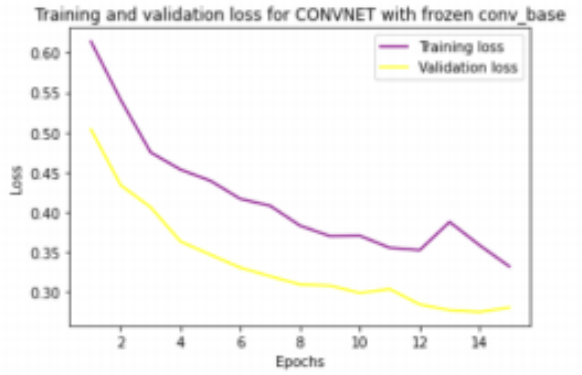


#Accuracy and Loss curve for CONVNET 4a Model

```
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()
```

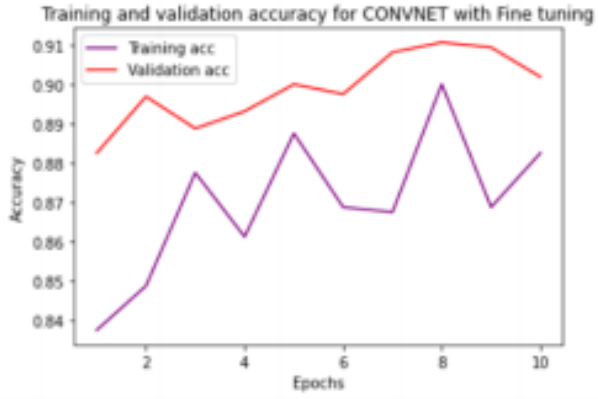



```
loss = history3.history['loss']
val_loss = history3.history['val_loss']

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()
```

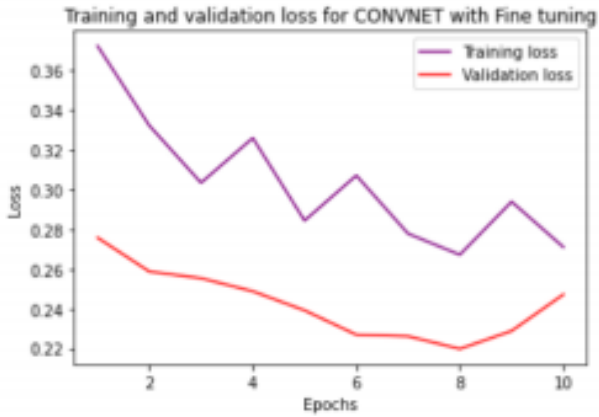


#Accuracy and Loss curve for CONVNET 4b Model

```
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 7

Image 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))
```

```
Husky (33.76%)
```

Image 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))
```

```
Persian_cat (28.96%)
```

Conclusion

CNN Models with transfer learning applications will enhance the performance of the model. Computational power for classifying image dataset is lesser when compared to other learning models because it uses advanced feature extraction techniques. 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

References

1. https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
2. Image Classification - Dogs and Cats, Deep Learning A-Z™: Download Practice Datasets
3. Deep Learning A-Z: Hands On Artificial Neural Networks by Udemy

Result

Hence a better accuracy is achieved using transfer learning of pretrained models.Done using VGG16 model.