Project Report

Paper Title: Cat or Dog Image Detection Using 'Convolution Neural Network VGG16' Model.

Course: Machine Learning

Course Code: CSE475

Section: 1

Submitted To

Dr. Md Rifat Ahmmad Rashid

Assistant Professor, Department of CSE

East West University

Submitted By

Name: Ali Akbar Aurnab **ID:** 2017-2-60-095

Name: Md. Sabbir Hossain Khan **ID:** 2017-2-60-127

Name: Debobroto Ghosh **ID:** 2017-2-60-125

Date of Submission: 08-09-2022

Introduction:

In this project we have proposed a mechine learning algorithm using CNN-VGG16 model. We used Cats-vs-Dogs image dataset belonging to two classes from kaggle for this project. To train the model we took 501 cat and 501 dog pictures, a total of 1002 pictures where 902 pictures are used for training and 100 pictures are used for validation. After training the model can predict a single image and able to tell whether the image is a cat or a dog.

Model Description:

Convolution neural network (CNN) architecture VGG16 was employed to win the 2014 ILSVR (Imagenet) competition. It is regarded as one of the best vision model architectures created to date. The most distinctive feature of VGG16 is that it prioritized having convolution layers of 3x3 filters with a stride 1 and always utilized the same padding and maxpool layer of 2x2 filters with a stride 2. Throughout the entire architecture, convolution and max pool layers are arranged in the same manner. It concludes with two fully connected layers (FC) and a softmax for output. The 16 in VGG16 denotes the fact that there are 16 layers with weights. This network has roughly 138 million parameters, making it a fairly .

CNN Layers:

In [42]:
```
x=Flatten()(vgg.output)
prediction=Dense(len(folders),activation='softmax')(x)
model=Model(inputs=vgg.input,outputs=prediction)
model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_2 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

Model Training:

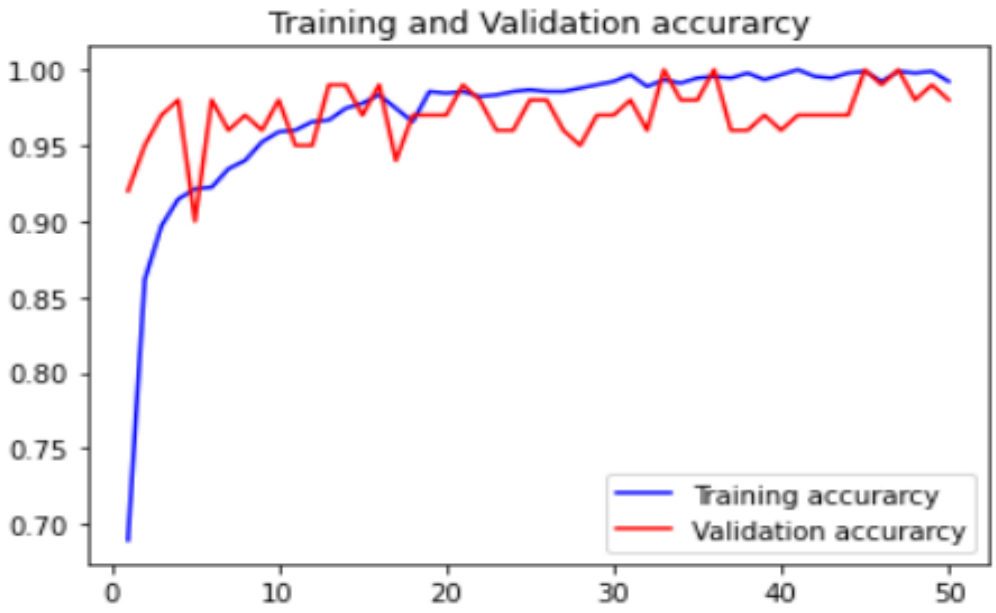
```
Epoch 1/50
15/15 [==============================] - 317s 22s/step - loss: 0.6415 - accuracy: 0.6896 - val_loss: 0.3175 - val_accuracy:
0.9200
Epoch 2/50
15/15 [==============================] - 333s 22s/step - loss: 0.3504 - accuracy: 0.8614 - val_loss: 0.1881 - val_accuracy:
0.9500
Epoch 3/50
15/15 [==============================] - 305s 21s/step - loss: 0.2710 - accuracy: 0.8969 - val_loss: 0.1664 - val_accuracy:
0.9700
Epoch 4/50
15/15 [==============================] - 334s 22s/step - loss: 0.2270 - accuracy: 0.9146 - val_loss: 0.1383 - val_accuracy:
0.9800
Epoch 5/50
15/15 [==============================] - 301s 20s/step - loss: 0.2207 - accuracy: 0.9213 - val_loss: 0.2088 - val_accuracy:
0.9000
Epoch 6/50
15/15 [==============================] - 295s 20s/step - loss: 0.1910 - accuracy: 0.9224 - val_loss: 0.1215 - val_accuracy:
0.9800
Epoch 7/50
```

```
Epoch 7/50
15/15 [==============================] - 292s 21s/step - loss: 0.1911 - accuracy: 0.9346 - val_loss: 0.1155 - val_accuracy:
0.9600
Epoch 8/50
15/15 [==============================] - 291s 19s/step - loss: 0.1687 - accuracy: 0.9401 - val_loss: 0.1176 - val_accuracy:
0.9700
Epoch 9/50
15/15 [==============================] - 289s 19s/step - loss: 0.1506 - accuracy: 0.9523 - val_loss: 0.1147 - val_accuracy:
0.9600
Epoch 10/50
15/15 [==============================] - 295s 19s/step - loss: 0.1302 - accuracy: 0.9590 - val_loss: 0.1051 - val_accuracy:
0.9800
Epoch 11/50
15/15 [==============================] - 287s 20s/step - loss: 0.1209 - accuracy: 0.9601 - val_loss: 0.1213 - val_accuracy:
0.9500
Epoch 12/50
15/15 [==============================] - 285s 19s/step - loss: 0.1238 - accuracy: 0.9656 - val_loss: 0.1175 - val_accuracy:
0.9500
Epoch 13/50
```

```
15/15 [==============================] - 314s 21s/step - loss: 0.1197 - accuracy: 0.9667 - val_loss: 0.0902 - val_accuracy:
0.9900
Epoch 14/50
15/15 [==============================] - 328s 22s/step - loss: 0.1018 - accuracy: 0.9745 - val_loss: 0.0878 - val_accuracy:
0.9900
Epoch 15/50
15/15 [==============================] - 353s 23s/step - loss: 0.1005 - accuracy: 0.9778 - val_loss: 0.0993 - val_accuracy:
0.9700
Epoch 16/50
15/15 [==============================] - 320s 21s/step - loss: 0.0922 - accuracy: 0.9834 - val_loss: 0.0755 - val_accuracy:
0.9900
Epoch 17/50
15/15 [==============================] - 304s 21s/step - loss: 0.0928 - accuracy: 0.9745 - val_loss: 0.1221 - val_accuracy:
0.9400
Epoch 18/50
15/15 [==============================] - 316s 21s/step - loss: 0.1011 - accuracy: 0.9656 - val_loss: 0.0781 - val_accuracy:
0.9700
Epoch 19/50
15/15 [==============================] - 303s 20s/step - loss: 0.0821 - accuracy: 0.9856 - val_loss: 0.0923 - val_accuracy:
```

```
0.9700
Epoch 45/50
15/15 [==============================] - 269s 18s/step - loss: 0.0285 - accuracy: 0.9989 - val_loss: 0.0426 - val_accuracy:
1.0000
Epoch 46/50
15/15 [==============================] - 268s 18s/step - loss: 0.0379 - accuracy: 0.9922 - val_loss: 0.0450 - val_accuracy:
0.9900
Epoch 47/50
15/15 [==============================] - 269s 18s/step - loss: 0.0314 - accuracy: 0.9989 - val_loss: 0.0432 - val_accuracy:
1.0000
Epoch 48/50
15/15 [==============================] - 271s 18s/step - loss: 0.0302 - accuracy: 0.9978 - val_loss: 0.0510 - val_accuracy:
0.9800
Epoch 49/50
15/15 [==============================] - 270s 18s/step - loss: 0.0295 - accuracy: 0.9989 - val_loss: 0.0510 - val_accuracy:
0.9900
Epoch 50/50
15/15 [==============================] - 267s 19s/step - loss: 0.0413 - accuracy: 0.9922 - val_loss: 0.0503 - val_accuracy:
0.9800
```

Training and Validation Accuracy Graph:



Training and Validation Loss Graph:

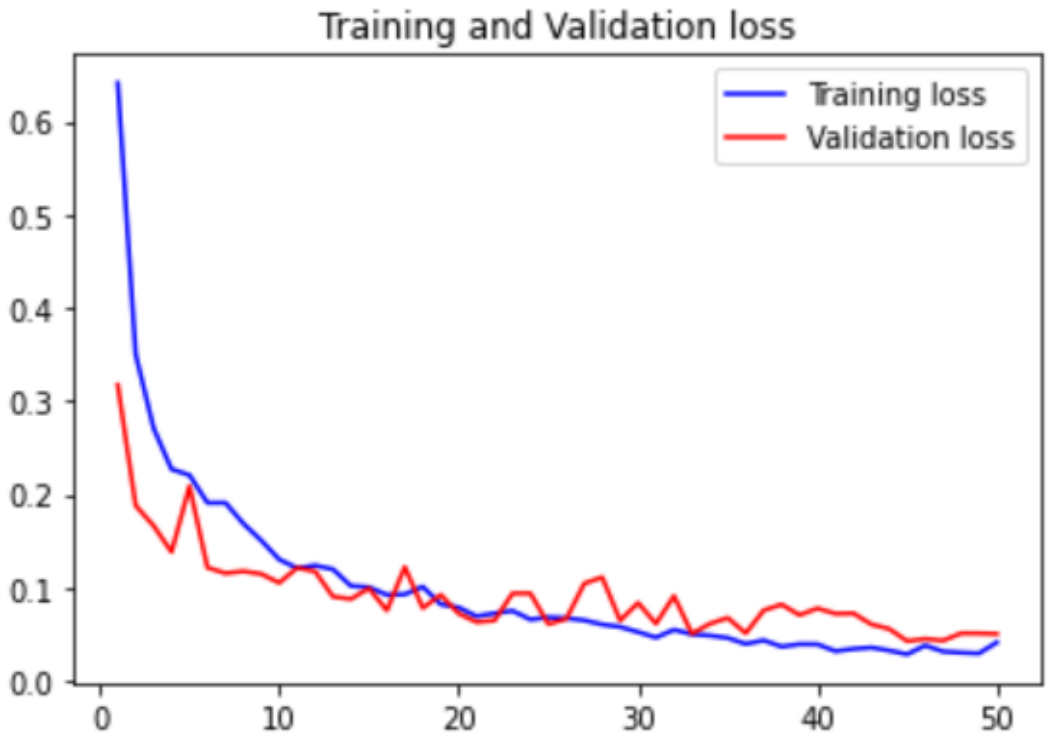


Image Prediction:

In [41]:
```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
import numpy as np

img_pred=image.load_img(r"C:\Users\Subrata Ghose\Desktop\Cat vs Dog\Dog_test\586.JPG",target_size=(224,224))

img_pred=image.img_to_array(img_pred)
img_pred=np.expand_dims(img_pred, axis=0)


rslt= model.predict(img_pred)

print(rslt)
if rslt[0][0]>rslt[0][1]:
    prediction="Cat"
else:
    prediction="Dog"
print(prediction)
```

```
1/1 [==============================] - 0s 134ms/step
[[0. 1.]]
Dog
```