Import Libraries

In [1]:

```
import numpy as np
import pandas as pd

from keras.preprocessing.image import ImageDataGenerator, load_img
from keras.models import Sequential, Model,load_model
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D, GlobalMaxPooling
2D
from keras.layers import BatchNormalization, merge, Input, Activation, Dropout
, Flatten, Dense
from keras import backend as K
from keras.layers import LeakyReLU
from keras.applications import VGG16
from sklearn.metrics import accuracy_score
from keras import layers
from keras import optimizers

import matplotlib.pyplot as plt
import tensorflow as tf

from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras.optimizers import SGD, Adam
import matplotlib.pyplot as plt
import random
import os
```

```
Using TensorFlow backend.
```

In [2]:

```
input_shape = (128, 128, 3)
```

Import VGG16

To Solve this classification problem, we use VGG16. A state of the art classification model which is pre trained for classifying such image instances. We use Weights given by imagenet.
The model is predefined and has been optimized for these weights.

In [4]:

```
pre_trained_model = VGG16(input_shape=input_shape, include_top=False, weights=
"imagenet")
```

In [4]:

```
last_layer = pre_trained_model.get_layer('block5_pool')
last_output = last_layer.output
```

In [5]:

```
x = GlobalMaxPooling2D()(last_output)
x = Dense(512, activation='relu')(x)
x = Dropout(0.3)(x)
x = layers.Dense(2, activation='sigmoid')(x)
```

```
WARNING:tensorflow:From C:\Users\pmven\Anaconda3\lib\site-packages\keras\back
end\tensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.n
n_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
```

In [6]:

```
model = Model(pre_trained_model.input, x)
model.compile(loss='binary_crossentropy', optimizer=optimizers.SGD(lr=1e-4, mo
mentum=0.9), metrics=['accuracy'])
model.summary()
```

```
WARNING:tensorflow:From C:\Users\pmven\Anaconda3\lib\site-packages\keras\opti
mizers.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compa
t.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\pmven\Anaconda3\lib\site-packages\tensorflow
\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tenso
rflow.python.ops.array_ops) is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 128, 128, 3)       0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0         
_________________________________________________________________
global_max_pooling2d_1 (Glob (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 2)                 1026      
=================================================================
Total params: 14,978,370
Trainable params: 14,978,370
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
# model.load_weights('VGG_v1.h5')
```

Loading Data and Test-Train split

In [8]:

```
filenames = os.listdir(r'C:\Users\pmven\Downloads\dogs-vs-cats\train\\')
test_filenames = os.listdir(r'C:\Users\pmven\Downloads\dogs-vs-cats\test\\')

categories = []

for i in filenames:
    category = i.split('.')[0]
    if category == 'dog':
        categories.append('dog')
    else:
        categories.append('cat')

df = pd.DataFrame({'filenames':filenames,'categories':categories})
```

In [9]:

```
train_df, validate_df = train_test_split(df, test_size=0.20, random_state=42)
test_df = pd.DataFrame({'filename': test_filenames})

train_df = train_df.reset_index(drop=True)
validate_df = validate_df.reset_index(drop=True)

total_train = train_df.shape[0]
total_validate = validate_df.shape[0]
total_test = test_df.shape[0]
```

We use Image generators to convert these images into Matrix format which can be easily read by the Neural Network. WE use a target size of 128 x 128 for the same.

In [10]:
```
train_datagen = ImageDataGenerator(rescale=1./255)
test_gen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(dataframe = train_df,  dir
ectory = r'C:\Users\pmven\Downloads\dogs-vs-cats\train\\',  x_col='filenames',
\
                                                    y_col='categories',  class
_mode="categorical", target_size=(128,128), batch_size = 15)

validation_generator = train_datagen.flow_from_dataframe( dataframe = validate
_df,  directory = r'C:\Users\pmven\Downloads\dogs-vs-cats\train\\',  x_col='fi
lenames', \
                                                          y_col='categories',
class_mode="categorical", target_size=(128,128), batch_size = 15)

test_generator = test_gen.flow_from_dataframe(test_df,  directory = r'C:\Users
\pmven\Downloads\dogs-vs-cats\test\\', x_col='filename', y_col=None,
                                              class_mode=None,  target_size=(1
28,128), batch_size=15, shuffle=False)
```

```
Found 20000 validated image filenames belonging to 2 classes.
Found 5000 validated image filenames belonging to 2 classes.
Found 12500 validated image filenames.
```

Model Training

In [11]:
```
history = model.fit_generator(train_generator, epochs=5, validation_data=valid
ation_generator, validation_steps=total_validate//16, steps_per_epoch=total_tr
ain//16)
```

```
Epoch 1/5
1250/1250 [==============================] - 168s 134ms/step - loss: 0.2719 -
acc: 0.8737 - val_loss: 0.1393 - val_acc: 0.9463
Epoch 2/5
1250/1250 [==============================] - 153s 122ms/step - loss: 0.1137 -
acc: 0.9561 - val_loss: 0.0974 - val_acc: 0.9615
Epoch 3/5
1250/1250 [==============================] - 154s 123ms/step - loss: 0.0839 -
acc: 0.9667 - val_loss: 0.0809 - val_acc: 0.9679
Epoch 4/5
1250/1250 [==============================] - 163s 130ms/step - loss: 0.0648 -
acc: 0.9757 - val_loss: 0.0849 - val_acc: 0.9649
Epoch 5/5
1250/1250 [==============================] - 163s 130ms/step - loss: 0.0456 -
acc: 0.9834 - val_loss: 0.0859 - val_acc: 0.9654
```

After Multiple Trial and Errors, We finalize on 5 epochs for the model to reduce overfitting.

In [18]:
```
model.save('VGG_model2.h5')
```

Predictions

In [12]:

```
predict = model.predict_generator(test_generator, steps=np.ceil(total_test/15
), verbose=1)
```

```
834/834 [==============================] - 86s 103ms/step
```

In [14]:

```
test_df['label'] = list(predict[:,1])
```

In [15]:

```
train_df.shape
```

Out[15]:

```
(20000, 2)
```

In [16]:

```
test_df['id'] = test_df['filename'].str.replace('.jpg', '')
test_df['id'] = test_df['id'].astype(int)
test_df.sort_values(by = 'id')
test_df.head()
```

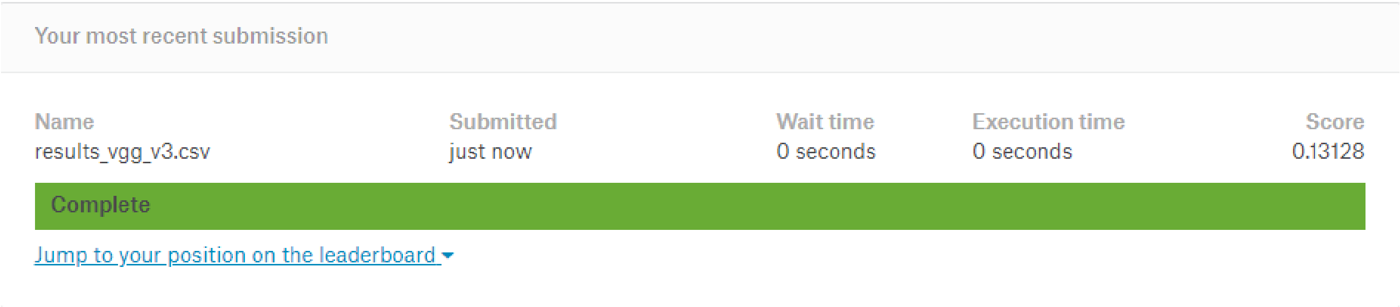
Out[16]:

	filename	label	id
0	1.jpg	0.999992	1
1	10.jpg	0.000052	10
2	100.jpg	0.000625	100
3	1000.jpg	0.999948	1000
4	10000.jpg	0.913598	10000

In [17]:

```
test_df[['id', 'label']].to_csv('results_vgg_v3.csv', index = False)
```

Results



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
results_vgg_v3.csv	just now	0 seconds	0 seconds	0.13128

Complete

Jump to your position on the leaderboard ▾

These redults can be further improved by Stacking various such algorithms, to get the logloss score even higher.

In [ ]: