In [60]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import numpy as np
import matplotlib.pyplot as plt
```

In [61]:

```
import tensorflow as tf
```

In [62]:

```
import numpy as np
studentID = 2018711118
np.random.seed(studentID)
```

In [63]:

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

In [64]:

```
test_cats_dir = os.path.join(test_dir, 'cats')  # directory with our training cat pictures
test_dogs_dir = os.path.join(test_dir, 'dogs')  # directory with our training dog pictures
train_cats_dir = os.path.join(train_dir, 'cats')  # directory with our training cat pictures
train_dogs_dir = os.path.join(train_dir, 'dogs')  # directory with our training dog pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')  # directory with our validation cat pictures
validation_dogs_dir = os.path.join(validation_dir, 'dogs')  # directory with our validation dog pictures
```

In [65]:

```
#understand the data
```

In [66]:

```
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))

num_cats_val = len(os.listdir(validation_cats_dir))
num_dogs_val = len(os.listdir(validation_dogs_dir))

total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val
```

In [67]:

```
print('total training cat images:', num_cats_tr)
print('total training dog images:', num_dogs_tr)

print('total validation cat images:', num_cats_val)
print('total validation dog images:', num_dogs_val)
print("--")
print("Total training images:", total_train)
print("Total validation images:", total_val)
```

```
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
--
Total training images: 2000
Total validation images: 1000
```

In [109]:

```
batch_size = 20
epochs = 30
IMG_HEIGHT = 150
IMG_WIDTH = 150
```

In [69]:

```
#Data Preparation
```

In [70]:

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    brightness_range=None,
    shear_range=0.1,
    zoom_range=0.1,
    channel_shift_range=0.0,
    fill_mode="nearest",
    cval=0.0,
    horizontal_flip=True,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    dtype=None,
)
```

Out[70]:

<tensorflow.python.keras.preprocessing.image.ImageDataGenerator at 0x7fbfb0505050>

In [71]:

```
train_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our training data
validation_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our validation data
```

In [72]:

```
train_data_gen = train_image_generator.flow_from_directory(batch_size=batch_size,
                                                           directory=train_dir,
                                                           shuffle=True,
                                                           target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                           class_mode='binary')
```

Found 2000 images belonging to 2 classes.

In [73]:

```
val_data_gen = validation_image_generator.flow_from_directory(batch_size=batch_size,
                                                              directory=validation_dir,
                                                              target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                              class_mode='binary')
```

Found 1000 images belonging to 2 classes.

In [74]:

```
#Visualize training images
```

In [75]:

```
sample_training_images, _ = next(train_data_gen)
```

In [76]:

```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

In [77]:

```
plotImages(sample_training_images[:5])
```

In [78]:

```
#Create the model
#Seqeuntial + Pre-trained VGG16(except dense layer(fully connected layer), used Convolutional and Pooling laye
```

In [82]:

```
conv_base = tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(150,150,3),
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

In [83]:

```
conv_base.trainable = False
```

In [ ]:

```
# model = models.Sequential()
# model .add(conv_base)
# model .add(layers.Flatten())
# model .add(layers.Dense(10, activation='relu'))
# model .add(layers.Dense(1, activation='sigmoid'))
# model.compile('rmsprop', 'binary_crossentropy', ['accuracy'])
```

In [114]:

```
model = Sequential()
model.add(conv_base)
model .add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation="sigmoid"))
model.compile('rmsprop', 'binary_crossentropy', ['accuracy'])
# model.compile(loss = 'binary_crossentropy', optimizer = opt)
```

In [115]:

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=2e-5,
    decay_steps=10000,
    decay_rate=0.9)
#optimizer = keras.optimizers.SGD(learning_rate=lr_schedule)
```

In [116]:

```
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_25 (Dense)             (None, 256)               2097408
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [ ]:

```
#Compile the model
```

In [35]:

```
# do it for the test
# model.compile(optimizer='adam',
#               loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
#               metrics=['accuracy'])
```

In [36]:

```
# model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense (Dense)                (None, 512)               10617344
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 10,641,441
Trainable params: 10,641,441
Non-trainable params: 0
_________________________________________________________________
```

In [38]:

```
#Train the model
#Use the fit_generator() method and the ImagedataGenerator to train the network
```

In [117]:

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size
)
```

```
Epoch 1/30
100/100 [==============================] - 694s 7s/step - loss: 0.5590 - accuracy: 0.81
49 - val_loss: 0.4304 - val_accuracy: 0.8328
Epoch 2/30
100/100 [==============================] - 704s 7s/step - loss: 0.2020 - accuracy: 0.91
15 - val_loss: 0.2706 - val_accuracy: 0.8892
Epoch 3/30
100/100 [==============================] - 701s 7s/step - loss: 0.1374 - accuracy: 0.94
68 - val_loss: 0.4456 - val_accuracy: 0.8204
Epoch 4/30
100/100 [==============================] - 699s 7s/step - loss: 0.0748 - accuracy: 0.97
56 - val_loss: 0.3203 - val_accuracy: 0.8937
Epoch 5/30
100/100 [==============================] - 709s 7s/step - loss: 0.0679 - accuracy: 0.98
04 - val_loss: 0.3400 - val_accuracy: 0.8926
Epoch 6/30
100/100 [==============================] - 700s 7s/step - loss: 0.0598 - accuracy: 0.98
64 - val_loss: 0.3681 - val_accuracy: 0.8965
Epoch 7/30
100/100 [==============================] - 692s 7s/step - loss: 0.0211 - accuracy: 0.99
54 - val_loss: 0.4289 - val_accuracy: 0.8878
Epoch 8/30
100/100 [==============================] - 693s 7s/step - loss: 0.0611 - accuracy: 0.98
96 - val_loss: 0.4158 - val_accuracy: 0.8969
Epoch 9/30
100/100 [==============================] - 690s 7s/step - loss: 0.0420 - accuracy: 0.99
10 - val_loss: 0.4879 - val_accuracy: 0.8918
Epoch 10/30
100/100 [==============================] - 691s 7s/step - loss: 0.0250 - accuracy: 0.99
47 - val_loss: 0.4597 - val_accuracy: 0.8948
Epoch 11/30
100/100 [==============================] - 695s 7s/step - loss: 0.0312 - accuracy: 0.99
50 - val_loss: 0.4856 - val_accuracy: 0.8936
Epoch 12/30
100/100 [==============================] - 706s 7s/step - loss: 0.0229 - accuracy: 0.99
51 - val_loss: 0.5100 - val_accuracy: 0.8854
Epoch 13/30
100/100 [==============================] - 699s 7s/step - loss: 2.0407e-04 - accuracy:
1.0000 - val_loss: 0.5592 - val_accuracy: 0.8937
Epoch 14/30
100/100 [==============================] - 692s 7s/step - loss: 0.0231 - accuracy: 0.99
58 - val_loss: 0.5574 - val_accuracy: 0.8957
Epoch 15/30
100/100 [==============================] - 701s 7s/step - loss: 0.0401 - accuracy: 0.99
50 - val_loss: 0.5685 - val_accuracy: 0.8910
Epoch 16/30
100/100 [==============================] - 701s 7s/step - loss: 0.0481 - accuracy: 0.99
56 - val_loss: 0.6532 - val_accuracy: 0.8868
Epoch 17/30
100/100 [==============================] - 703s 7s/step - loss: 2.7312e-05 - accuracy:
1.0000 - val_loss: 0.6321 - val_accuracy: 0.8964
```

```
Epoch 18/30
100/100 [==============================] - 702s 7s/step - loss: 0.0503 - accuracy: 0.99
59 - val_loss: 0.6039 - val_accuracy: 0.8895
Epoch 19/30
100/100 [==============================] - 697s 7s/step - loss: 0.0295 - accuracy: 0.99
52 - val_loss: 0.7139 - val_accuracy: 0.8819
Epoch 20/30
100/100 [==============================] - 705s 7s/step - loss: 2.3295e-05 - accuracy:
1.0000 - val_loss: 0.6747 - val_accuracy: 0.8911
Epoch 21/30
100/100 [==============================] - 702s 7s/step - loss: 0.0491 - accuracy: 0.99
58 - val_loss: 0.7273 - val_accuracy: 0.8895
Epoch 22/30
100/100 [==============================] - 699s 7s/step - loss: 0.0240 - accuracy: 0.99
50 - val_loss: 0.9107 - val_accuracy: 0.8557
Epoch 23/30
100/100 [==============================] - 703s 7s/step - loss: 2.3521e-04 - accuracy:
1.0000 - val_loss: 0.6713 - val_accuracy: 0.8971
Epoch 24/30
100/100 [==============================] - 702s 7s/step - loss: 0.0518 - accuracy: 0.99
61 - val_loss: 0.9030 - val_accuracy: 0.8710
Epoch 25/30
100/100 [==============================] - 700s 7s/step - loss: 5.5842e-05 - accuracy:
1.0000 - val_loss: 0.7303 - val_accuracy: 0.8921
Epoch 26/30
100/100 [==============================] - 703s 7s/step - loss: 0.0286 - accuracy: 0.99
52 - val_loss: 0.8869 - val_accuracy: 0.8760
Epoch 27/30
100/100 [==============================] - 706s 7s/step - loss: 2.4152e-05 - accuracy:
1.0000 - val_loss: 0.7330 - val_accuracy: 0.8914
Epoch 28/30
100/100 [==============================] - 699s 7s/step - loss: 0.0369 - accuracy: 0.99
69 - val_loss: 0.9808 - val_accuracy: 0.8724
Epoch 29/30
100/100 [==============================] - 700s 7s/step - loss: 5.4066e-05 - accuracy:
1.0000 - val_loss: 0.7565 - val_accuracy: 0.8913
Epoch 30/30
100/100 [==============================] - 699s 7s/step - loss: 0.0422 - accuracy: 0.99
55 - val_loss: 1.8261 - val_accuracy: 0.8333
```

In [119]:

```
#Visualize training results
#Now visualize after trainning the network
#Q1. plot the trainning accuracy and validation accuray
#Q2. plot the trainning loss and validation loss
```

In [118]:

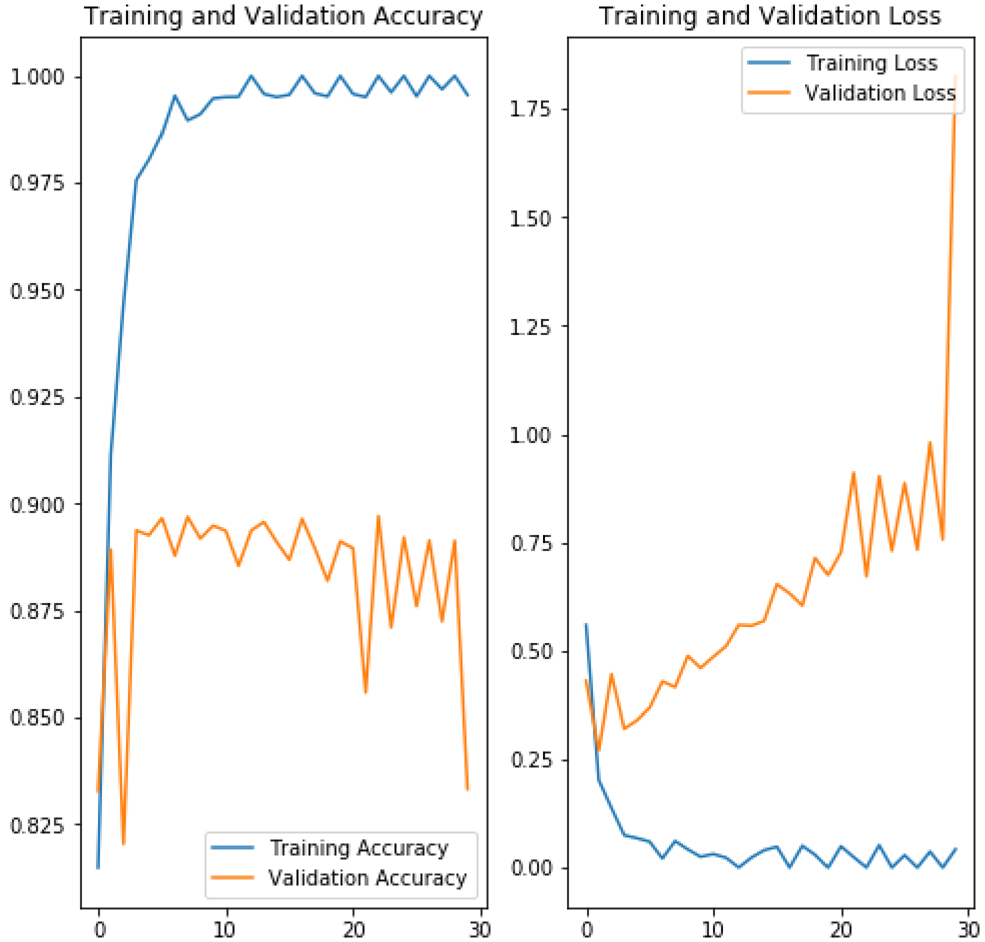
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



In [120]:

```
#Q. two different optimizers of your choice for your model show corresponding accuracy values
```

In [125]:

```
model = Sequential()
model.add(conv_base)
model .add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation="sigmoid"))
model .add(Flatten())
model.compile('SGD', 'binary_crossentropy', ['accuracy'])
# model.compile(loss = 'binary_crossentropy', optimizer = opt)
```

In [126]:

```
model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_27 (Dense)             (None, 256)               2097408
_________________________________________________________________
dense_28 (Dense)             (None, 1)                 257
_________________________________________________________________
flatten_3 (Flatten)          (None, 1)                 0
=================================================================
Total params: 16,812,353
Trainable params: 2,097,665
Non-trainable params: 14,714,688
_________________________________________________________________
```

In [127]:

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size
)
```

```
Epoch 1/30
100/100 [==============================] - 700s 7s/step - loss: 0.5348 - accuracy: 0.72
24 - val_loss: 0.4353 - val_accuracy: 0.7916
Epoch 2/30
100/100 [==============================] - 692s 7s/step - loss: 0.3609 - accuracy: 0.84
23 - val_loss: 0.3154 - val_accuracy: 0.8686
Epoch 3/30
100/100 [==============================] - 689s 7s/step - loss: 0.2893 - accuracy: 0.87
87 - val_loss: 0.2978 - val_accuracy: 0.8720
Epoch 4/30
100/100 [==============================] - 693s 7s/step - loss: 0.2624 - accuracy: 0.89
46 - val_loss: 0.2863 - val_accuracy: 0.8755
Epoch 5/30
100/100 [==============================] - 689s 7s/step - loss: 0.2244 - accuracy: 0.91
16 - val_loss: 0.3027 - val_accuracy: 0.8634
Epoch 6/30
100/100 [==============================] - 692s 7s/step - loss: 0.2306 - accuracy: 0.90
58 - val_loss: 0.2906 - val_accuracy: 0.8779
Epoch 7/30
100/100 [==============================] - 687s 7s/step - loss: 0.1871 - accuracy: 0.92
98 - val_loss: 0.2680 - val_accuracy: 0.8806
Epoch 8/30
100/100 [==============================] - 686s 7s/step - loss: 0.1922 - accuracy: 0.92
38 - val_loss: 0.3045 - val_accuracy: 0.8728
Epoch 9/30
100/100 [==============================] - 687s 7s/step - loss: 0.1704 - accuracy: 0.93
49 - val_loss: 0.3196 - val_accuracy: 0.8726
Epoch 10/30
100/100 [==============================] - 689s 7s/step - loss: 0.1708 - accuracy: 0.93
34 - val_loss: 0.2976 - val_accuracy: 0.8680
Epoch 11/30
100/100 [==============================] - 688s 7s/step - loss: 0.1518 - accuracy: 0.94
42 - val_loss: 0.2696 - val_accuracy: 0.8811
Epoch 12/30
100/100 [==============================] - 690s 7s/step - loss: 0.1296 - accuracy: 0.95
52 - val_loss: 0.3885 - val_accuracy: 0.8435
Epoch 13/30
100/100 [==============================] - 689s 7s/step - loss: 0.1322 - accuracy: 0.95
14 - val_loss: 0.2807 - val_accuracy: 0.8849
Epoch 14/30
100/100 [==============================] - 700s 7s/step - loss: 0.1098 - accuracy: 0.96
32 - val_loss: 0.5488 - val_accuracy: 0.7832
Epoch 15/30
100/100 [==============================] - 703s 7s/step - loss: 0.1187 - accuracy: 0.95
64 - val_loss: 0.2713 - val_accuracy: 0.8817
Epoch 16/30
100/100 [==============================] - 701s 7s/step - loss: 0.1023 - accuracy: 0.96
63 - val_loss: 0.2671 - val_accuracy: 0.8832
Epoch 17/30
100/100 [==============================] - 704s 7s/step - loss: 0.1232 - accuracy: 0.95
88 - val_loss: 0.2800 - val_accuracy: 0.8822
```

```
Epoch 18/30
100/100 [==============================] - 704s 7s/step - loss: 0.0751 - accuracy: 0.98
07 - val_loss: 0.2726 - val_accuracy: 0.8814
Epoch 19/30
100/100 [==============================] - 706s 7s/step - loss: 0.1131 - accuracy: 0.96
21 - val_loss: 0.2754 - val_accuracy: 0.8811
Epoch 20/30
100/100 [==============================] - 702s 7s/step - loss: 0.0661 - accuracy: 0.98
38 - val_loss: 0.3140 - val_accuracy: 0.8832
Epoch 21/30
100/100 [==============================] - 700s 7s/step - loss: 0.0740 - accuracy: 0.98
00 - val_loss: 0.2997 - val_accuracy: 0.8869
Epoch 22/30
100/100 [==============================] - 706s 7s/step - loss: 0.0619 - accuracy: 0.98
50 - val_loss: 0.3108 - val_accuracy: 0.8867
Epoch 23/30
100/100 [==============================] - 707s 7s/step - loss: 0.0538 - accuracy: 0.98
88 - val_loss: 0.2810 - val_accuracy: 0.8840
Epoch 24/30
100/100 [==============================] - 703s 7s/step - loss: 0.0475 - accuracy: 0.99
29 - val_loss: 0.2869 - val_accuracy: 0.8822
Epoch 25/30
100/100 [==============================] - 702s 7s/step - loss: 0.0449 - accuracy: 0.99
36 - val_loss: 0.2979 - val_accuracy: 0.8832
Epoch 26/30
100/100 [==============================] - 703s 7s/step - loss: 0.0398 - accuracy: 0.99
50 - val_loss: 0.2940 - val_accuracy: 0.8862
Epoch 27/30
100/100 [==============================] - 707s 7s/step - loss: 0.0377 - accuracy: 0.99
62 - val_loss: 0.2904 - val_accuracy: 0.8782
Epoch 28/30
100/100 [==============================] - 704s 7s/step - loss: 0.0348 - accuracy: 0.99
74 - val_loss: 0.2965 - val_accuracy: 0.8862
Epoch 29/30
100/100 [==============================] - 699s 7s/step - loss: 0.0317 - accuracy: 0.99
85 - val_loss: 0.3002 - val_accuracy: 0.8777
Epoch 30/30
100/100 [==============================] - 701s 7s/step - loss: 0.0297 - accuracy: 0.99
93 - val_loss: 0.3008 - val_accuracy: 0.8827
```

In [128]:

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

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

In [ ]:

```
model = Sequential()
model.add(conv_base)
model .add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation="sigmoid"))
model .add(Flatten())
model.compile('binary_crossentropy', ['accuracy'])
# model.compile(loss = 'binary_crossentropy', optimizer = opt)
```

In [ ]:

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size
)
```

##교수님 모델하나 돌리는데 시간이 너무 오래걸려서 (위에 1모델당 거의 8-9시간씩걸리네요.) 나머지 문제는 따로 돌려서 결과값을 붙여넣습니다.
실제 돌려본결과값입니다.

Q4. five different learning rate values of your choice and show the corresponding accuracy value

In [ ]:

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=2e-5, #change here
    decay_steps=10000,
    decay_rate=0.9)
```

In [ ]:

```
lrf = learn.lr_find() #러닝 파라미터값 확인으로 fit해보기
```

In [ ]:

```
learn.shed,plot() # -> 1e-5구간이 loss가 가장 낮은것을 확인할 수 있다.
```

Q5.build another performing well method

In [ ]:

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH ,3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])
```

In [ ]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [ ]:

```
model.summary()
```

In [ ]:

```
history = model.fit_generator(
    train_data_gen,
    steps_per_epoch=total_train // batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=total_val // batch_size
)
```

In [ ]:

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

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Q6. use 2 face of selfie of mine and show what they are classified as.note that probabilities of each being cat or dog
photo1.jpeg -> 0.6282 dog
photo2.jpeg -> 0.4439 dog

In [ ]:

```
import numpy as np
from keras.preprocessing import image
test_image = image.load_img('test/your_picture/photo1.jpeg', target_size = (64, 64))
```

In [ ]:

```
test_image = image.img_to_array(test_image)
```

In [ ]:

```
test_image = np.expand_dims(test_image, axis = 0)
```

In [ ]:

```
result = classifier.predict(test_image)
```

In [ ]:

```
training_set.class_indices
```

#Theoretically explain your model how to recognized when input the your selfie data as a cat and dog classifier?
#what is the correct way to approach the task of classifiying human faces?

VGG16 베이스라인에 dense layer 256을 relu로 결과값을 평가하고
다시 dense layer1 로 최종적으로 시그모이드로 classifier 값을 결정한다. 라벨에따라 고양이 0 개 1로 binary classification한다.
옵티마이저는 'rmsprop'과 'SGD'를 사용했으며 나머지 파라메터는 가이드에 주어진대로 셋팅했다.

사람의 얼굴을 분류하는 방법은 결국 멀티클래스 classification으로 별도 라벨링하는 방법밖엔 없다고 생각한다.
사람의 얼굴이미지도 컨볼루션으로 학습해서 최종적으로 별도 라벨링을 하는것이 분류할 수 있는 최적방법이라고 생각한다.

Q8. Fix your model to handle the human face image classifier further more in recent your cat and dog classifier model.
I'll code it down the below

Q8 Answer. I guess it need to be multi classfier to use smaller VGGNET that's going to be recognized multiple labels from uploaded images
I'll code it down the below

In [ ]:

```
import matplotlib
matplotlib.use("Agg")
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from pyimagesearch.smallervggnet import SmallerVGGNet
import matplotlib.pyplot as plt
from imutils import paths
import numpy as np
import argparse
import random
import pickle
import cv2
import os
```

In [ ]:

```
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
            help="path to input dataset (i.e., directory of images)")
ap.add_argument("-m", "--model", required=True,
            help="path to output model")
ap.add_argument("-l", "--labelbin", required=True,
            help="path to output label binarizer")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
            help="path to output accuracy/loss plot")
args = vars(ap.parse_args())
```

In [ ]:

```
EPOCHS = 30
INIT_LR = 2e-5
BS = 20
IMAGE_DIMS = (150, 150, 3)
```

In [ ]:

```
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(studentID)
random.shuffle(imagePaths)
data = []
labels = []
```

In [ ]:

```
for imagePath in imagePaths:
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
    image = img_to_array(image)
    data.append(image)
    l = label = imagePath.split(os.path.sep)[-2].split("_")
    labels.append(l)
```

In [ ]:

```
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
print("[INFO] data matrix: {} images ({:.2f}MB)".format(len(imagePaths), data.nbytes / (1024 * 1000.0)))
```

In [ ]:

```
print("[INFO] class labels:")
mlb = MultiLabelBinarizer()
labels = mlb.fit_transform(labels)
# loop over each of the possible class labels and show them
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i + 1, label))
```

In [ ]:

```
(trainX, testX, trainY, testY) = train_test_split(data,labels, test_size=0.2, random_state=42)
aug = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,height_shift_range=0.1, shear_range=0.2,
```

In [ ]:

```
print("[INFO] compiling model...")
model = SmallerVGGNet.build(width=IMAGE_DIMS[1], height=IMAGE_DIMS[0],depth=IMAGE_DIMS[2], classes
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
```

In [ ]:

```
model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])
print("[INFO] training network...")
H = model.fit(x=aug.flow(trainX, trainY, batch_size=BS),validation_data=(testX, testY),steps_per_epoch=len(train
```

In [ ]:

```
print("[INFO] serializing network...")
model.save(args["model"], save_format="h5")
print("[INFO] serializing label binarizer...")
f = open(args["labelbin"], "wb")
f.write(pickle.dumps(mlb))
f.close()
```

In [ ]:

```
plt.style.use("ggplot")
plt.figure()
N = EPOCHS
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="upper left")
plt.savefig(args["plot"])
```

Q9. your fixed model generalized approach?
what happen when input the image like fish, panda in your fixed model?

small VGGnet은 VGG의 perfomance적인 아쉬운 부분을 심플화 하여 이미지 분류기로 일반적으로 사용한다고 생각한다.
돌린결과로는 VGG16과 큰 차이는 없다.

Q10. explain the way that handle the different case of image fed in your new model? (like fish, bird or panda)

Q10 Answer: 소프트맥스의 원리처럼 더 섞인 이미지(not in case 개 or 고양이 another one)의 비중을 맞추어서 더 많이 섞인 이미지를 해당 라벨에 더가깝게 라벨링한다.
네트워크의 마지막의 soft max activation을 sigmoid activation으로 바꾼다.
기존의 binary cross entropy 옵션을 categorial cross entropy 옵션으로 바꾼다.
결국 개, 고양이 이외의 사진이 추가삽입될 경우에는 multi classifier로 가는것이 최적이다.
그러나 muli classifier의 경우 train 된적없는 데이터에 대해서는 절대 예측할 수 없다.
multi classifier의경우 라벨의 복잡도가 높아 정확한 예측이 떨어질 수도 있다.

In [ ]:

```
import the necessary packages
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import imutils
import pickle
import cv2
import os

ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,help="path to trained model model")
ap.add_argument("-l", "--labelbin", required=True,help="path to label binarizer")
ap.add_argument("-i", "--image", required=True,help="path to input image")
args = vars(ap.parse_args())
```

In [ ]:

```
image = cv2.imread(args["image"])
output = imutils.resize(image, width=400)

image = cv2.resize(image, (96, 96))
image = image.astype("float") / 255.0
image = img_to_array(image)
image = np.expand_dims(image, axis=0)
```

In [ ]:

```
print("[INFO] loading network...")
model = load_model(args["model"])
mlb = pickle.loads(open(args["labelbin"], "rb").read())
print("[INFO] classifying image...")
proba = model.predict(image)[0]
idxs = np.argsort(proba)[::-1][:2]
```

In [ ]:

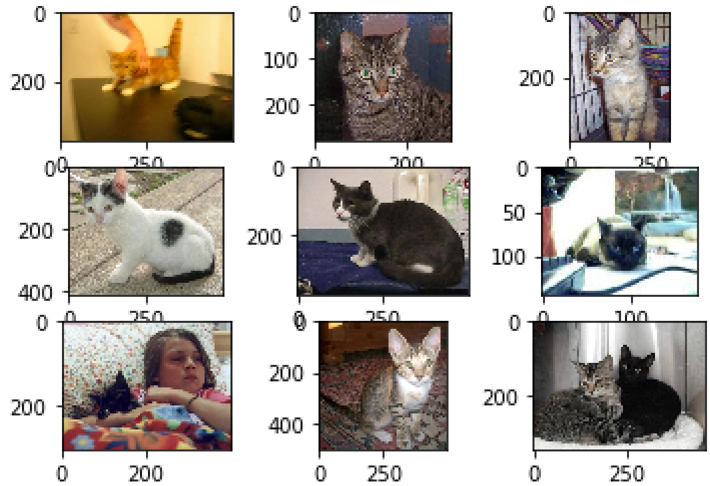
```
for (i, j) in enumerate(idxs):
    label = "{}: {:.2f}%".format(mlb.classes_[j], proba[j] * 100)
    cv2.putText(output, label, (10, (i * 30) + 25), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
for (label, p) in zip(mlb.classes_, proba):
    print("{}: {:.2f}%".format(label, p * 100))
    cv2.imshow("Output", output)
    cv2.waitKey(0)
```

In [ ]:

```
#### test the data #####
```

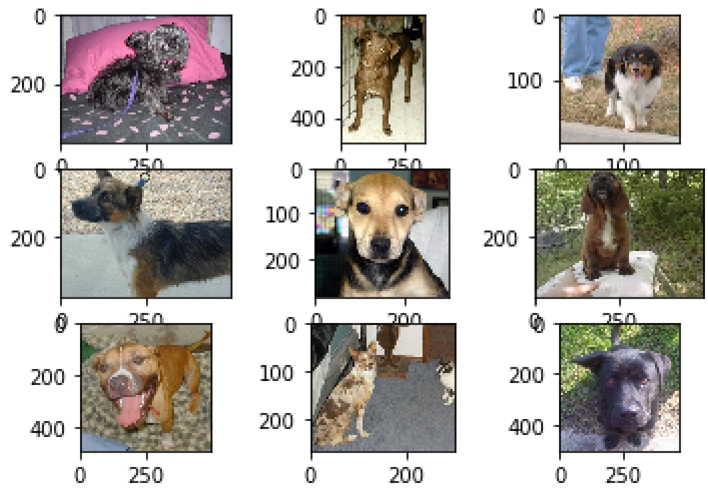
In [7]:

```
from matplotlib import pyplot
from matplotlib.image import imread
folder = 'train/cats/'
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    filename = folder + 'cat.' + str(i) + '.jpg'
    image = imread(filename)
    pyplot.imshow(image)
pyplot.show()
```



In [8]:

```
# plot cat photos from the dogs vs cats dataset
from matplotlib import pyplot
from matplotlib.image import imread
folder = 'train/dogs/'
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    filename = folder + 'dog.' + str(i) + '.jpg'
    image = imread(filename)
    pyplot.imshow(image)
pyplot.show()
```



In [ ]:

```
from os import listdir
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
folder = 'train/cats'
photos, labels = list(), list()
for file in listdir(folder):
    output = 0.0
    if file.startswith('cat'):
        output = 1.0
    photo = load_img(folder + file, target_size=(200, 200))
    photo = img_to_array(photo)
    photos.append(photo)
    labels.append(output)
photos = asarray(photos)
labels = asarray(labels)
print(photos.shape, labels.shape)
save('dogs_vs_cats_photos.npy', photos)
save('dogs_vs_cats_labels.npy', labels)
```