Project on Computer Vision using transfer learning

By,
Sarikaa S,
2018103058,
12-05-2021.

Problem Statement:

To classify images using convnet and pre-trained VGG16 models.

Modules for Convnets:

- Loading the dataset.
- Creating Directories for training,testing and validation data.
- Defining the model
- Feed the data to the network ,compile and fit the model
- Test the performance
- Plot Training and Validation graphs

Convnets:

Convnet is a class of deep neural network.It is composed of multiple layers of artificial neurons.

Instantiating a small convnet:

In [1]:
```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
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(64, (3, 3), activation='relu'))
```

In [2]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

The summary() function displays the layers ,shape and number of parameters of the convnet model.

Adding a classifier on top of the convnet:

In [3]:
```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

In [4]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

Since the output has 10 labels in the MNIST dataset ,there are 10 neurons in the last Dense layer.

Training the convnet on MNIST images:

In [5]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
```

In [6]:
```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 39s 39ms/step - loss: 0.3952 - accuracy: 0.8736
Epoch 2/5
938/938 [==============================] - 37s 39ms/step - loss: 0.0524 - accuracy: 0.9832
Epoch 3/5
938/938 [==============================] - 37s 39ms/step - loss: 0.0328 - accuracy: 0.9900
Epoch 4/5
938/938 [==============================] - 37s 40ms/step - loss: 0.0231 - accuracy: 0.9929
Epoch 5/5
938/938 [==============================] - 37s 40ms/step - loss: 0.0173 - accuracy: 0.9945
```

Out[6]:
```
<tensorflow.python.keras.callbacks.History at 0x24db9924a90>
```

Evaluating the model on the test data:

In [7]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
test_acc
```

```
313/313 [==============================] - 3s 8ms/step - loss: 0.0286 - accuracy: 0.9922
```

Out[7]:
```
0.9922000169754028
```

The accuracy of the test data is 99.22%.

Model without max-pooling layer:

In [8]:
```
model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
model_no_max_pool.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

In [9]:
```
model_no_max_pool.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

The disadvantages of not using max pooling that :

1. It doesn't learn the spatial hierarchy of features.
2. It has a large number of parameters which will lead to overfitting.

Training a convnet on cat and dog dataset

The cat and dog dataset is downloaded from Kaggle.

Copying images to training, validation, and test directories:

In [2]:

```
import os, shutil
original_dataset_dir = 'C:/Users/ACER/Downloads/train'
base_dir = 'C:/Users/ACER/Downloads/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)
```

In [4]:

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)
```

In [5]:

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_cats_dir, fname)
    shutil.copyfile(src, dst)
```

In [6]:

```
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)
```

In [8]:

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

In [9]:

```
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

In [10]:
```
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

In [11]:
```
print('total training cat images:', len(os.listdir(train_cats_dir)))
print('total training dog images:', len(os.listdir(train_dogs_dir)))
print('total validation cat images:', len(os.listdir(validation_cats_dir)))
print('total validation dog images:', len(os.listdir(validation_dogs_dir)))
print('total test cat images:', len(os.listdir(test_cats_dir)))
print('total test dog images:', len(os.listdir(test_dogs_dir)))
```

```
total training cat images: 1000
total training dog images: 1000
total validation cat images: 500
total validation dog images: 500
total test cat images: 500
total test dog images: 500
```

There are 1000 cat and 1000 dog images in the training directory. And there are 500 cat and 500 dog in validation and testing directory

Instantiating a small convnet for dogs vs. cats classification:

In [12]:
```
from keras import layers
from keras import models
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'))
```

In [13]:

```
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
_________________________________________________________________
```

Configuring the model for training:

In [14]:

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

Using ImageDataGenerator to read images from directories:

In [16]:

```
from keras.preprocessing.image import ImageDataGenerator
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 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

In [18]:
```
for data_batch, labels_batch in train_generator:
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
    break
```

```
data batch shape: (20, 150, 150, 3)
labels batch shape: (20,)
```

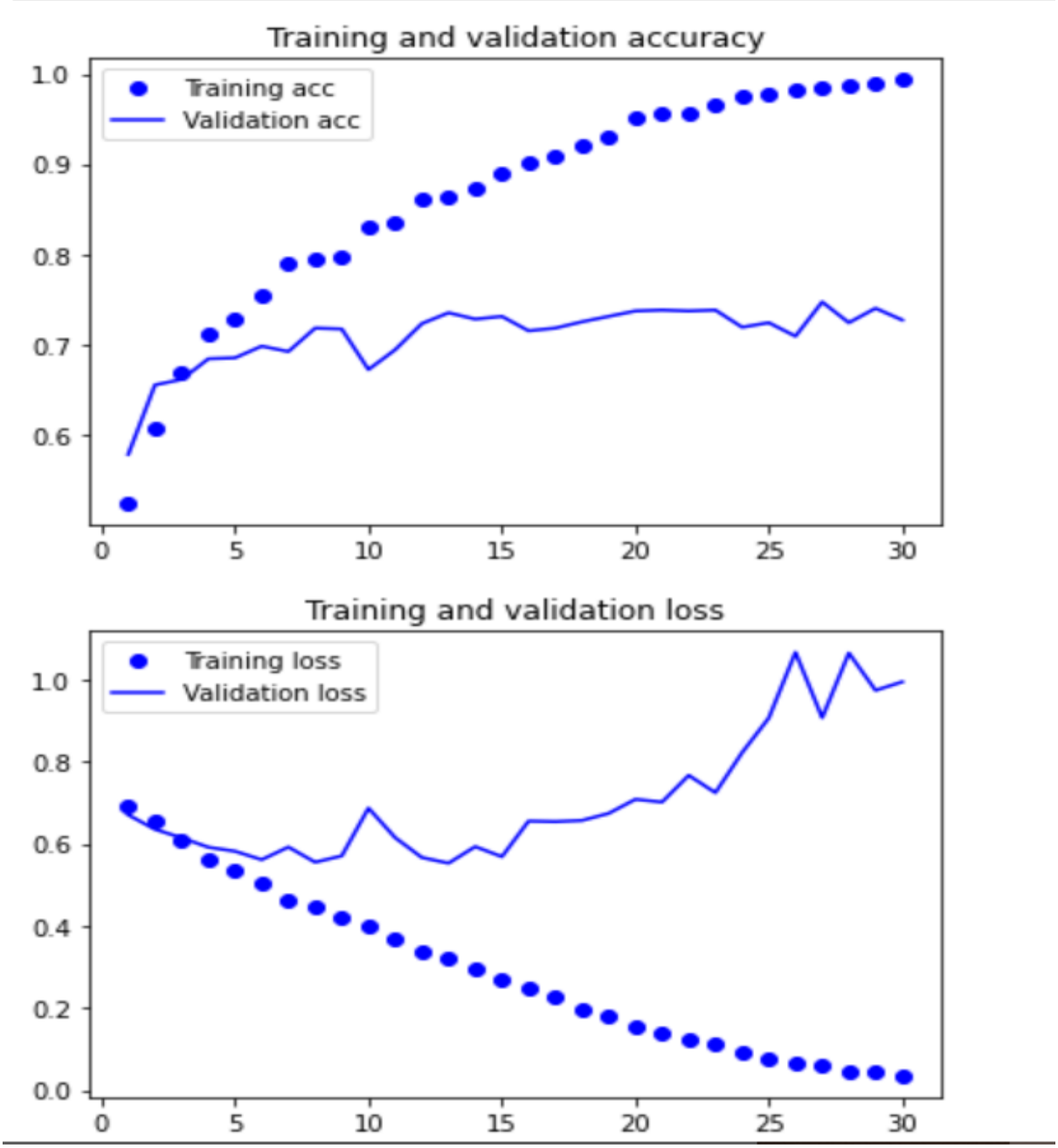
Fitting the model using a batch generator:

In [19]:
```
history = model.fit_generator(
train_generator,
steps_per_epoch=100,
epochs=30,
validation_data=validation_generator,
validation_steps=50)
```

```
C:\Users\ACER\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator` is
d in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/30
100/100 [==============================] - 59s 576ms/step - loss: 0.7046 - acc: 0.4783 - val_loss: 0.6705 - val_acc: 0.5780
Epoch 2/30
100/100 [==============================] - 53s 528ms/step - loss: 0.6647 - acc: 0.5900 - val_loss: 0.6354 - val_acc: 0.6550
Epoch 3/30
100/100 [==============================] - 50s 499ms/step - loss: 0.6236 - acc: 0.6656 - val_loss: 0.6147 - val_acc: 0.6610
Epoch 4/30
100/100 [==============================] - 50s 501ms/step - loss: 0.5724 - acc: 0.7030 - val_loss: 0.5917 - val_acc: 0.6840
Epoch 5/30
100/100 [==============================] - 49s 494ms/step - loss: 0.5348 - acc: 0.7290 - val_loss: 0.5819 - val_acc: 0.6850
Epoch 6/30
100/100 [==============================] - 49s 495ms/step - loss: 0.5039 - acc: 0.7664 - val_loss: 0.5613 - val_acc: 0.6980
Epoch 7/30
100/100 [==============================] - 49s 492ms/step - loss: 0.4672 - acc: 0.7858 - val_loss: 0.5922 - val_acc: 0.6920
Epoch 8/30
100/100 [==============================] - 50s 496ms/step - loss: 0.4462 - acc: 0.8050 - val_loss: 0.5552 - val_acc: 0.7180
Epoch 9/30
100/100 [==============================] - 48s 479ms/step - loss: 0.4298 - acc: 0.7845 - val_loss: 0.5706 - val_acc: 0.7170
Epoch 10/30
100/100 [==============================] - 52s 518ms/step - loss: 0.3985 - acc: 0.8355 - val_loss: 0.6872 - val_acc: 0.6720
Epoch 11/30
100/100 [==============================] - 51s 512ms/step - loss: 0.3734 - acc: 0.8289 - val_loss: 0.6151 - val_acc: 0.6940
Epoch 12/30
100/100 [==============================] - 53s 526ms/step - loss: 0.3341 - acc: 0.8502 - val_loss: 0.5667 - val_acc: 0.7230
Epoch 13/30
100/100 [==============================] - 50s 494ms/step - loss: 0.3089 - acc: 0.8743 - val_loss: 0.5528 - val_acc: 0.7350
Epoch 14/30
100/100 [==============================] - 50s 499ms/step - loss: 0.2680 - acc: 0.8901 - val_loss: 0.5933 - val_acc: 0.7280
```

Displaying curves of loss and accuracy during training:

In [21]:
```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



Since the gap between training loss curve and validation loss curve is large so overfitting occurs.

Setting up a data augmentation configuration via ImageDataGenerator:

Data augmentation takes the approach of generating more training data from existing training samples, by augmenting the samples via a number of random transformations

In [22]:

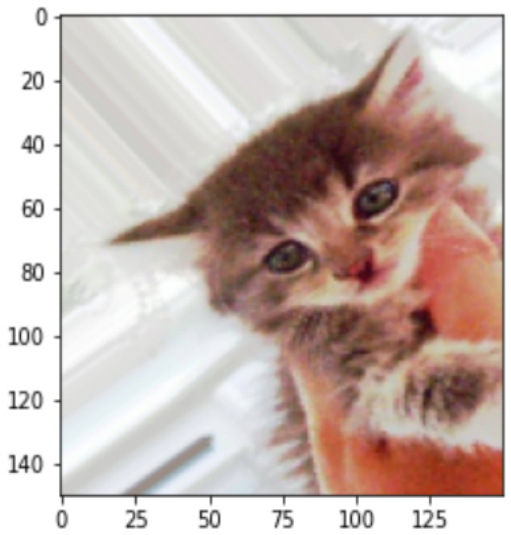
```
datagen = ImageDataGenerator(
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,
fill_mode='nearest')
```

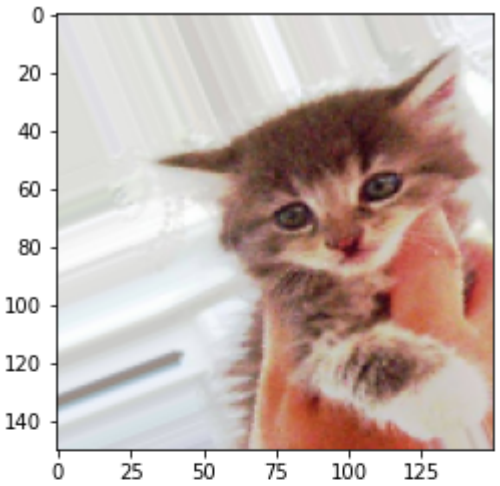
Displaying some randomly augmented training images:

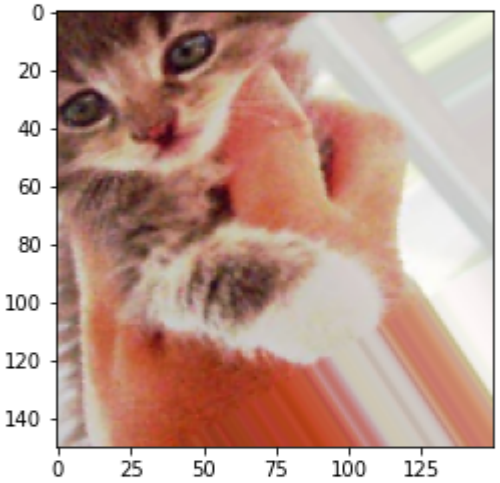

```
fill_mode='nearest')
```

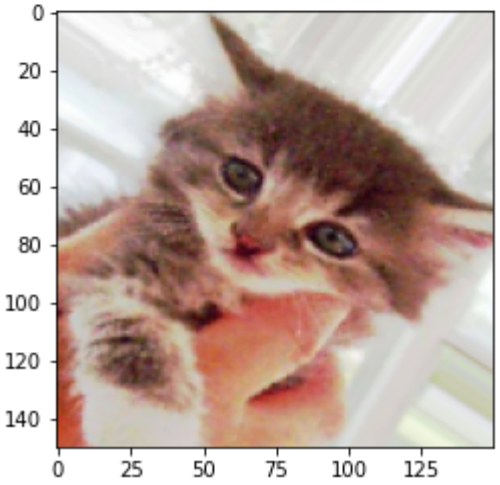
In [23]:

```
from keras.preprocessing import image
fnames = [os.path.join(train_cats_dir, fname) for
fname in os.listdir(train_cats_dir)]
img_path = fnames[3]
img = image.load_img(img_path, target_size=(150, 150))
x = image.img_to_array(img)
x = x.reshape((1,) + x.shape)
i=0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break
plt.show()
```









Defining a new convnet that includes dropout:

In [24]:

```
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.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-4),
metrics=['acc'])
```

Training the convnet using data-augmentation generators:

In [28]:

```
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,)
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 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

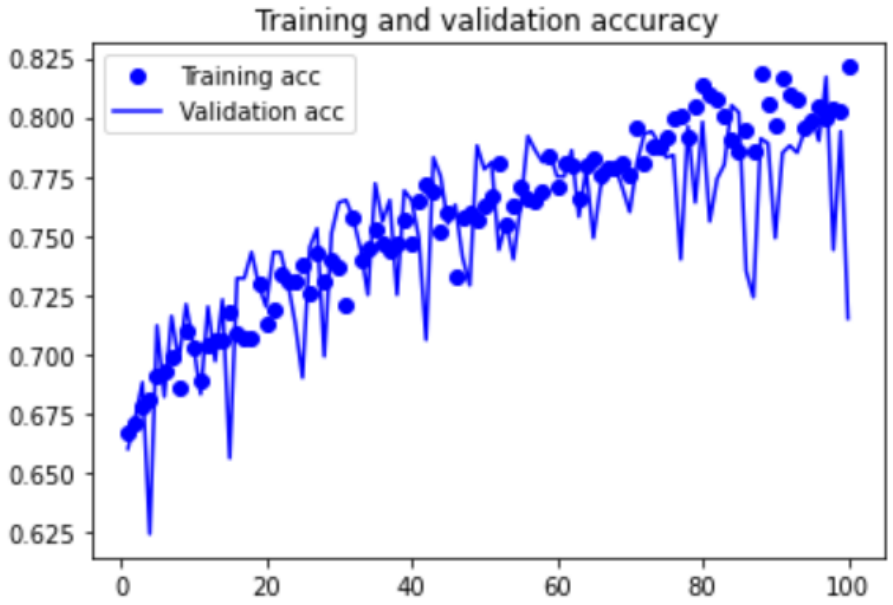
In [32]:

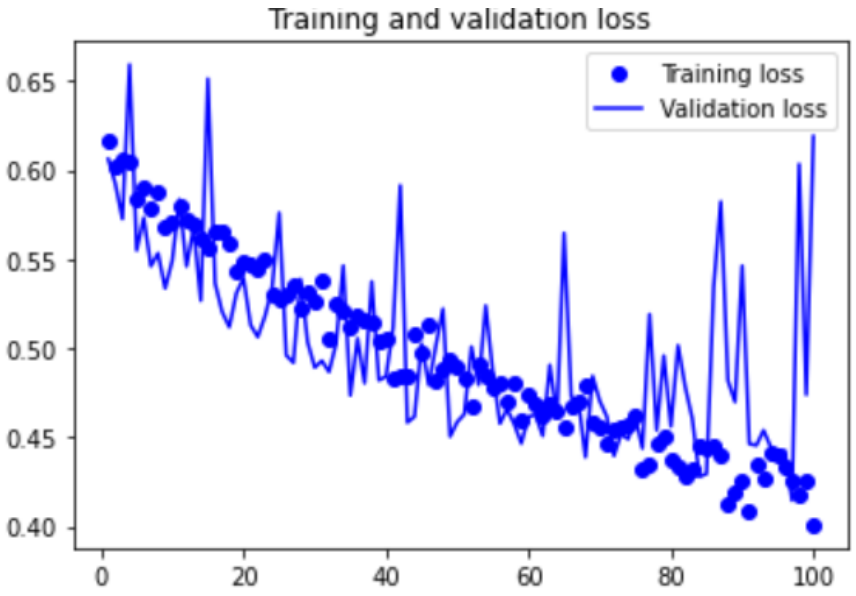
```
history = model.fit_generator(
train_generator,
steps_per_epoch=50,
epochs=100,
validation_data=validation_generator,
validation_steps=50)
```

```
Epoch 1/100
50/50 [==============================] - ETA: 0s - loss: 0.6160 - acc: 0.6673
Epoch 100/100
50/50 [==============================] - ETA: 0s - loss: 0.4008 - acc: 0.8213
```

In [39]:

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Using a pre trained convnet(Transfer Learning)

Modules:

- Loading the dataset.
- Loading the VGG16 model
- Feed the data to the network ,compile and fit the model
- Test the performance
- Plot Training and Validation graphs

Instantiating the VGG16 convolutional base:

In [34]:

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 9s 0us/step
```


```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Extracting features using the pretrained convolutional base:

In [40]:

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
base_dir = 'C:/Users/ACER/Downloads/cats_and_dogs_small'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
    directory,
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')
    i=0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels
```

In [41]:

```
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 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

In [42]:

```
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))
```

Defining and training the densely connected classifier:

In [43]:

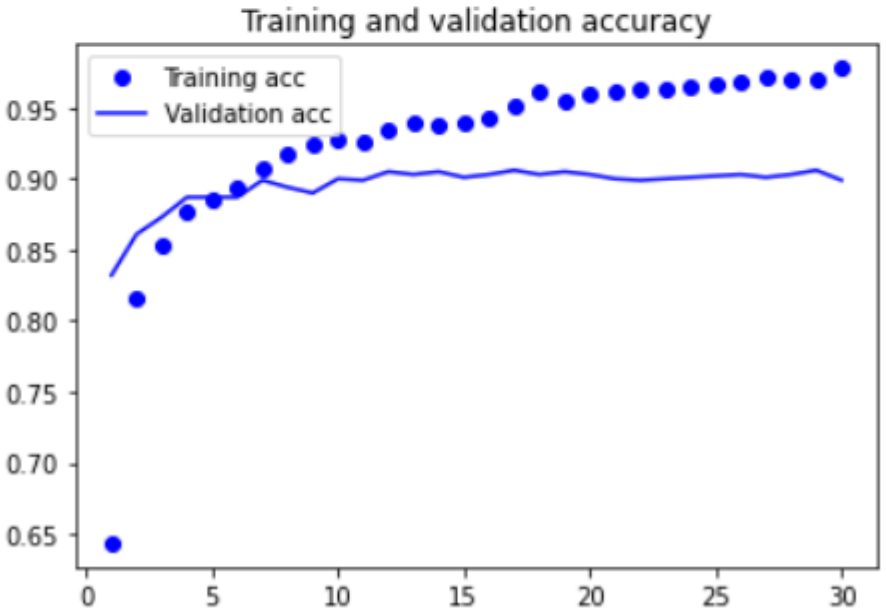
```
from keras import models
from keras import layers
from keras import optimizers
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(train_features, train_labels,
epochs=30,
batch_size=20,
validation_data=(validation_features, validation_labels))
```

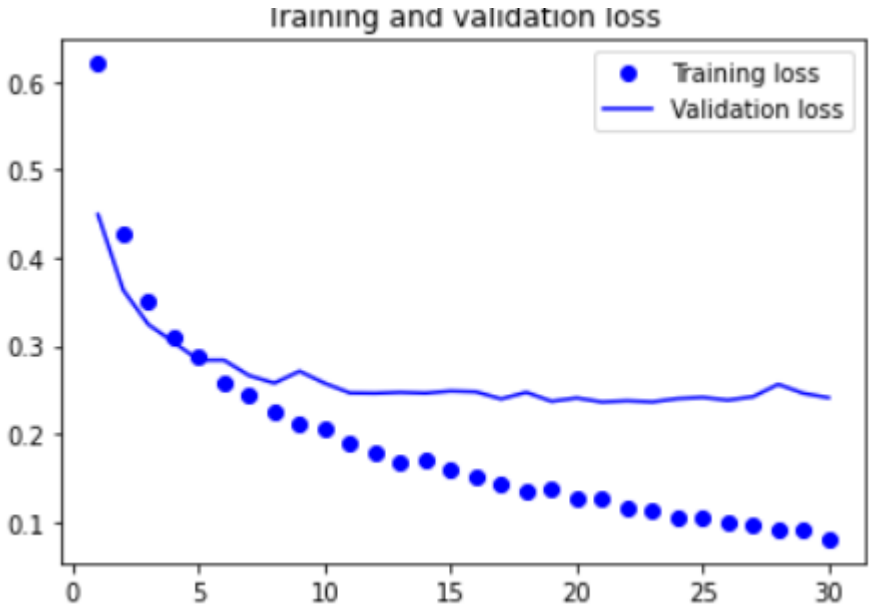
```
Epoch 1/30
100/100 [==============================] - 4s 34ms/step - loss: 0.7039 - acc: 0.5675 - val_loss: 0.4490 - val_acc: 0.8320
Epoch 2/30
100/100 [==============================] - 3s 32ms/step - loss: 0.4479 - acc: 0.8088 - val_loss: 0.3634 - val_acc: 0.8610
Epoch 3/30
100/100 [==============================] - 3s 32ms/step - loss: 0.3646 - acc: 0.8445 - val_loss: 0.3242 - val_acc: 0.8730
Epoch 4/30
100/100 [==============================] - 3s 32ms/step - loss: 0.3226 - acc: 0.8714 - val_loss: 0.3042 - val_acc: 0.8870
Epoch 5/30
100/100 [==============================] - 3s 33ms/step - loss: 0.2883 - acc: 0.8913 - val_loss: 0.2837 - val_acc: 0.8870
Epoch 6/30
100/100 [==============================] - 3s 32ms/step - loss: 0.2617 - acc: 0.8945 - val_loss: 0.2838 - val_acc: 0.8870
Epoch 7/30
100/100 [==============================] - 3s 32ms/step - loss: 0.2373 - acc: 0.9174 - val_loss: 0.2662 - val_acc: 0.8990
Epoch 8/30
100/100 [==============================] - 3s 33ms/step - loss: 0.2284 - acc: 0.9120 - val_loss: 0.2576 - val_acc: 0.8940
Epoch 9/30
100/100 [==============================] - 3s 33ms/step - loss: 0.2102 - acc: 0.9211 - val_loss: 0.2712 - val_acc: 0.8900
Epoch 10/30
100/100 [==============================] - 3s 33ms/step - loss: 0.2053 - acc: 0.9292 - val_loss: 0.2574 - val_acc: 0.9000
Epoch 11/30
100/100 [==============================] - 3s 34ms/step - loss: 0.1834 - acc: 0.9359 - val_loss: 0.2465 - val_acc: 0.8990
Epoch 12/30
100/100 [==============================] - 3s 30ms/step - loss: 0.1810 - acc: 0.9360 - val_loss: 0.2461 - val_acc: 0.9050
Epoch 13/30
100/100 [==============================] - 3s 30ms/step - loss: 0.1689 - acc: 0.9437 - val_loss: 0.2471 - val_acc: 0.9030
Epoch 14/30
100/100 [==============================] - 3s 30ms/step - loss: 0.1681 - acc: 0.9436 - val_loss: 0.2463 - val_acc: 0.9050
Epoch 15/30
100/100 [==============================] - 3s 29ms/step - loss: 0.1572 - acc: 0.9384 - val_loss: 0.2486 - val_acc: 0.9010
```

Plotting the results:

In [44]:

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Adding a densely connected classifier on top of the convolutional base:

In [45]:
```
from keras import models
from keras import layers
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [46]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (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: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

In [47]:
```
print('This is the number of trainable weights '
'before freezing the conv base:', len(model.trainable_weights))
conv_base.trainable = False
print('This is the number of trainable weights '
'after freezing the conv base:', len(model.trainable_weights))
```

```
This is the number of trainable weights before freezing the conv base: 30
This is the number of trainable weights after freezing the conv base: 4
```

Training the model end to end with a frozen convolutional base:

In [48]:

```
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
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=20,
class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
validation_dir,
target_size=(150, 150),
batch_size=20,
class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

In [49]:

```
model.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(lr=2e-5),
metrics=['acc'])
history = model.fit_generator(
train_generator,
steps_per_epoch=100,
epochs=30,
validation_data=validation_generator,
validation_steps=50)
```

```
Epoch 1/30
100/100 [==============================] - 179s 2s/step - loss: 0.6106 - acc: 0.6838 - val_loss: 0.4186 - val_acc: 0.8330
Epoch 2/30
100/100 [==============================] - 180s 2s/step - loss: 0.4734 - acc: 0.7778 - val_loss: 0.3546 - val_acc: 0.8600
Epoch 3/30
100/100 [==============================] - 173s 2s/step - loss: 0.4129 - acc: 0.8182 - val_loss: 0.3174 - val_acc: 0.8760
Epoch 4/30
100/100 [==============================] - 170s 2s/step - loss: 0.3877 - acc: 0.8387 - val_loss: 0.3060 - val_acc: 0.8700
Epoch 5/30
100/100 [==============================] - 170s 2s/step - loss: 0.3893 - acc: 0.8366 - val_loss: 0.2889 - val_acc: 0.8800
Epoch 6/30
100/100 [==============================] - 168s 2s/step - loss: 0.3760 - acc: 0.8335 - val_loss: 0.2842 - val_acc: 0.8800
Epoch 7/30
100/100 [==============================] - 167s 2s/step - loss: 0.3495 - acc: 0.8540 - val_loss: 0.2758 - val_acc: 0.8810
Epoch 8/30
100/100 [==============================] - 174s 2s/step - loss: 0.3282 - acc: 0.8600 - val_loss: 0.2706 - val_acc: 0.8890
Epoch 9/30
100/100 [==============================] - 176s 2s/step - loss: 0.3339 - acc: 0.8643 - val_loss: 0.2584 - val_acc: 0.8960
Epoch 10/30
100/100 [==============================] - 179s 2s/step - loss: 0.3385 - acc: 0.8568 - val_loss: 0.2573 - val_acc: 0.8990
Epoch 11/30
100/100 [==============================] - 181s 2s/step - loss: 0.3218 - acc: 0.8642 - val_loss: 0.2540 - val_acc: 0.8980
Epoch 12/30
100/100 [==============================] - 187s 2s/step - loss: 0.3277 - acc: 0.8583 - val_loss: 0.2655 - val_acc: 0.8860
Epoch 13/30
100/100 [==============================] - 186s 2s/step - loss: 0.3349 - acc: 0.8570 - val_loss: 0.2507 - val_acc: 0.8960
Epoch 14/30
100/100 [==============================] - 181s 2s/step - loss: 0.3153 - acc: 0.8660 - val_loss: 0.2522 - val_acc: 0.8970
Epoch 15/30
100/100 [==============================] - 174s 2s/step - loss: 0.3028 - acc: 0.8720 - val_loss: 0.2474 - val_acc: 0.9060
```

In [50]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688
_________________________________________________________________
```

Freezing all layers up to a specific one:

In [51]:
```
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
```

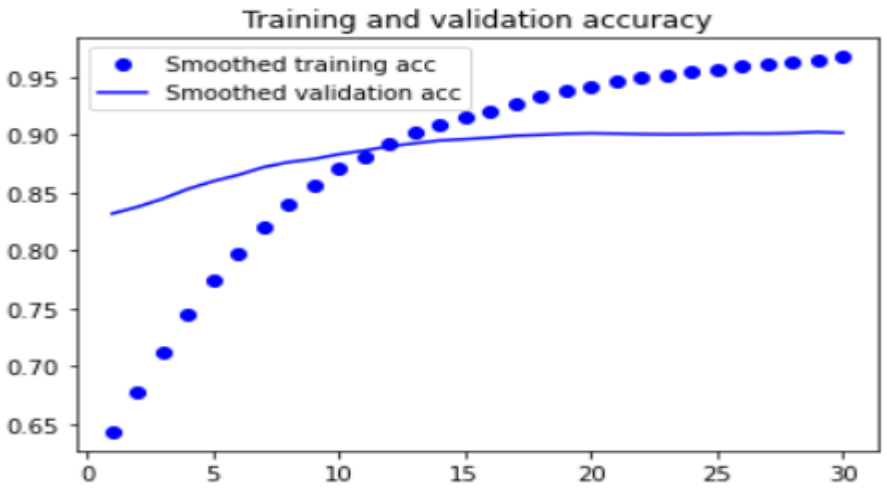
In [52]:
```
model.compile(loss='binary_crossentropy',
optimizer=optimizers.RMSprop(lr=1e-5),
metrics=['acc'])
history = model.fit_generator(
train_generator,
steps_per_epoch=100,
epochs=100,
validation_data=validation_generator,
validation_steps=50)
```

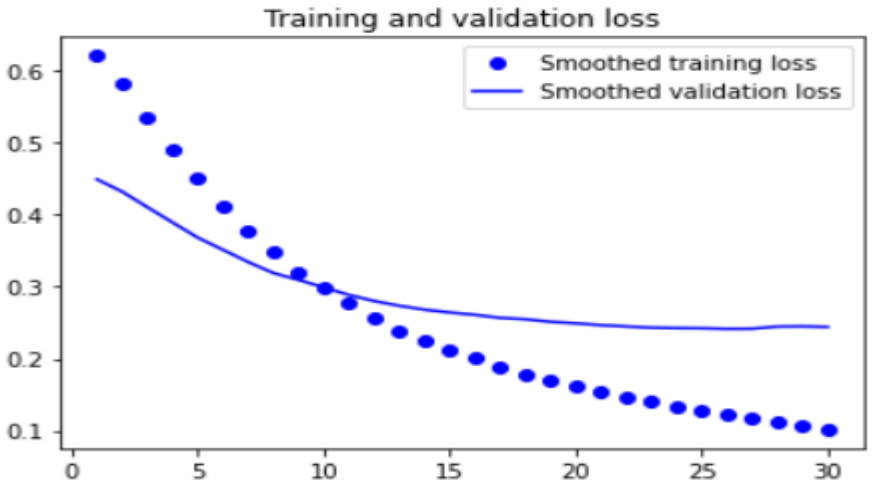
```
Epoch 1/100
100/100 [==============================] - 227s 2s/step - loss: 0.3005 - acc: 0.8740 - val_loss: 0.2392 - val_acc: 0.9130
Epoch 2/100
100/100 [==============================] - 208s 2s/step - loss: 0.2596 - acc: 0.8911 - val_loss: 0.2114 - val_acc: 0.9170
Epoch 3/100
100/100 [==============================] - 206s 2s/step - loss: 0.2603 - acc: 0.8926 - val_loss: 0.2096 - val_acc: 0.9120
Epoch 4/100
100/100 [==============================] - 205s 2s/step - loss: 0.2133 - acc: 0.9082 - val_loss: 0.2364 - val_acc: 0.9050
Epoch 5/100
100/100 [==============================] - 205s 2s/step - loss: 0.2028 - acc: 0.9126 - val_loss: 0.2022 - val_acc: 0.9240
Epoch 6/100
100/100 [==============================] - 204s 2s/step - loss: 0.1966 - acc: 0.9227 - val_loss: 0.1837 - val_acc: 0.9280
Epoch 7/100
100/100 [==============================] - 207s 2s/step - loss: 0.1729 - acc: 0.9307 - val_loss: 0.1961 - val_acc: 0.9300
Epoch 8/100
100/100 [==============================] - 216s 2s/step - loss: 0.1623 - acc: 0.9353 - val_loss: 0.1883 - val_acc: 0.9300
Epoch 9/100
100/100 [==============================] - 229s 2s/step - loss: 0.1672 - acc: 0.9360 - val_loss: 0.2084 - val_acc: 0.9250
Epoch 10/100
100/100 [==============================] - 228s 2s/step - loss: 0.1507 - acc: 0.9469 - val_loss: 0.2007 - val_acc: 0.9330
```

Smoothing the plots:

In [54]:

```
def smooth_curve(points, factor=0.8):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points
plt.plot(epochs,
smooth_curve(acc), 'bo', label='Smoothed training acc')
plt.plot(epochs,
smooth_curve(val_acc), 'b', label='Smoothed validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs,
smooth_curve(loss), 'bo', label='Smoothed training loss')
plt.plot(epochs,
smooth_curve(val_loss), 'b', label='Smoothed validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Evaluate the model:

In [55]:

```
test_generator = test_datagen.flow_from_directory(
test_dir,
target_size=(150, 150),
batch_size=20,
class_mode='binary')
test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

```
Found 1000 images belonging to 2 classes.
test acc: 0.9190000295639038
```

Visualizing what convnets learn:

In [57]:

```
from keras.models import load_model
model = load_model('cats_and_dogs_small_2.h5')
model.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
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

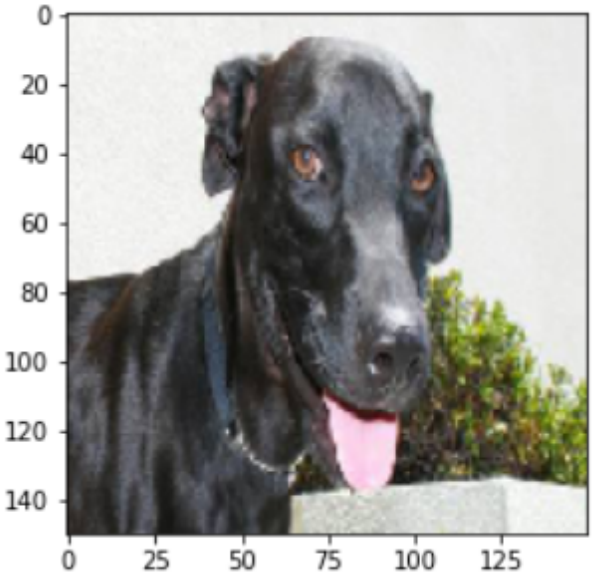
Preprocessing a single image:

In [61]:
```
img_path = 'C:/Users/ACER/Downloads/test1/23.jpg'
from keras.preprocessing import image
import numpy as np
img = image.load_img(img_path, target_size=(150, 150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis=0)
img_tensor /= 255.

print(img_tensor.shape)
```

```
(1, 150, 150, 3)
```

In [62]:
```
import matplotlib.pyplot as plt
plt.imshow(img_tensor[0])
plt.show()
```



Instantiating a model from an input tensor and a list of output tensors and Visualizing:

In [63]:
```
from keras import models
layer_outputs = [layer.output for layer in model.layers[:8]]
activation_model = models.Model(inputs=model.input, outputs=layer_outputs)
```

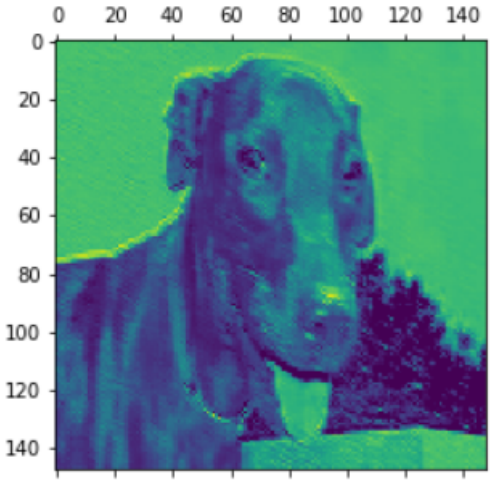
In [64]:
```
activations = activation_model.predict(img_tensor)
```

In [65]:
```
first_layer_activation = activations[0]
print(first_layer_activation.shape)
```

```
(1, 148, 148, 32)
```

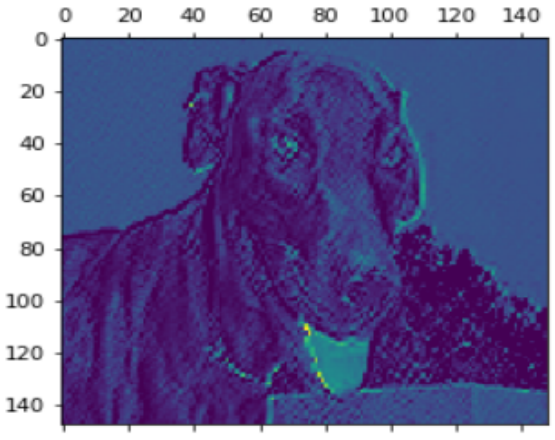
In [66]:
```
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :, :, 4], cmap='viridis')
```

Out[66]:
```
<matplotlib.image.AxesImage at 0x18a02d28490>
```



In [67]:
```
plt.matshow(first_layer_activation[0, :, :, 7], cmap='viridis')
```

Out[67]:
```
<matplotlib.image.AxesImage at 0x18a02d09d30>
```



Visualizing convnet filters

Defining the loss tensor for filter visualization:

In [78]:
```
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights='imagenet',
include_top=False)
layer_name = 'block3_conv1'
filter_index = 0
layer_output = model.get_layer(layer_name).output
loss = K.mean(layer_output[:, :, :, filter_index])
```

In [79]:
```
grads = K.gradients(loss, model.input)[0]
```

In [80]:
```
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
```

In [81]:
```
iterate = K.function([model.input], [loss, grads])
import numpy as np
loss_value, grads_value = iterate([np.zeros((1, 150, 150, 3))])
```

In [83]:
```
input_img_data = np.random.random((1, 150, 150, 3)) * 20 + 128.
step = 1.
for i in range(40):
    loss_value, grads_value = iterate([input_img_data])
    input_img_data += grads_value * step
```

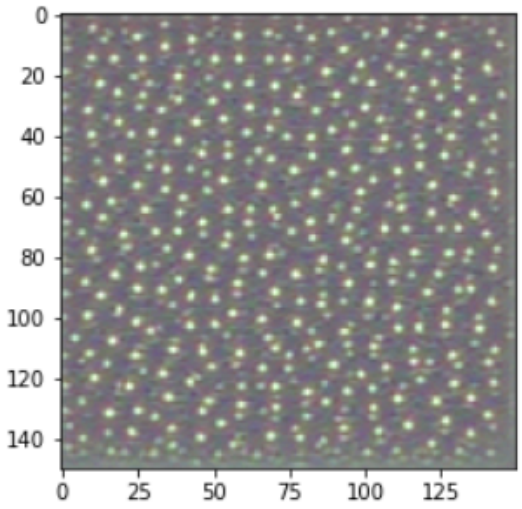
In [84]:
```
def deprocess_image(x):
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1
    x += 0.5
    x = np.clip(x, 0, 1)
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

Function to generate filter visualizations

In [86]:
```
def generate_pattern(layer_name, filter_index, size=150):
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])
    grads = K.gradients(loss, model.input)[0]
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
    iterate = K.function([model.input], [loss, grads])
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step
        img = input_img_data[0]
    return deprocess_image(img)
```

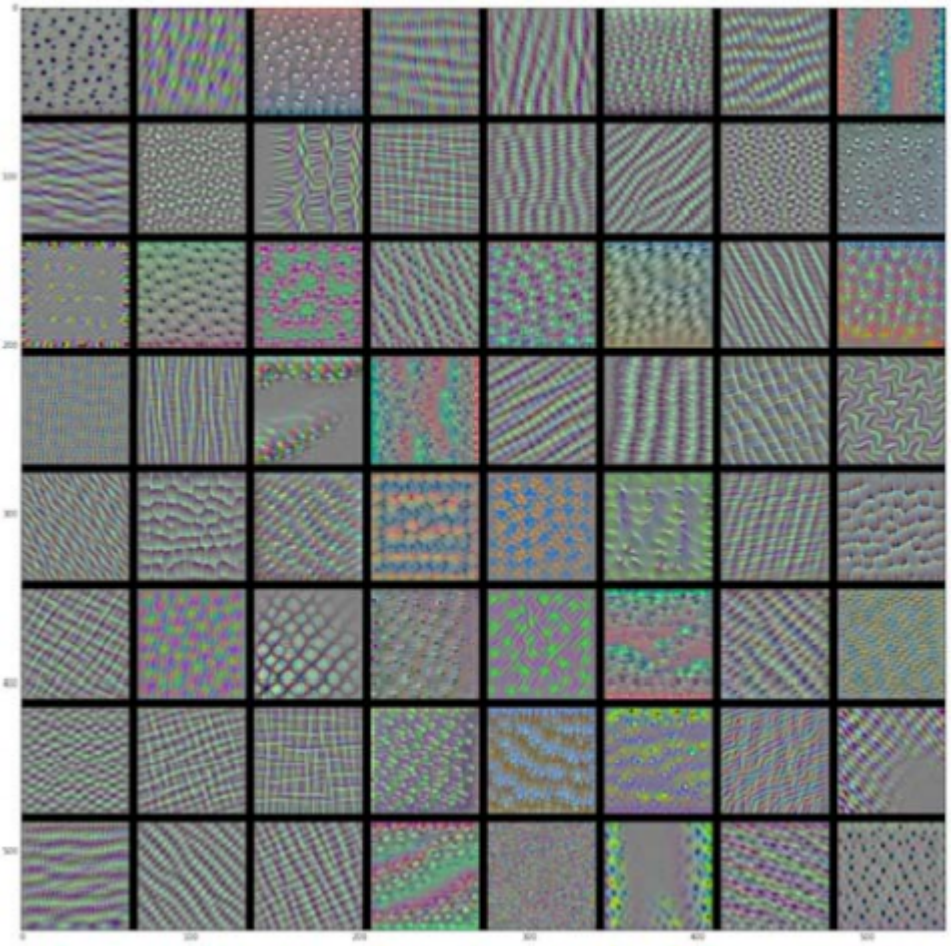
In [87]:
```
plt.imshow(generate_pattern('block3_conv1', 0))
```

Out[87]:
```
<matplotlib.image.AxesImage at 0x18a075113a0>
```



Generating a grid of all filter response patterns in a layer:

In [89]:
```
layer_name = 'block1_conv1'
size = 64
margin = 5
results = np.zeros((8 * size+7* margin, 8 * size+7* margin, 3))
for i in range(8):
    for j in range(8):
        filter_img = generate_pattern(layer_name, i + (j * 8), size=size)
        horizontal_start = i * size + i * margin
        horizontal_end = horizontal_start + size
        vertical_start = j * size + j * margin
        vertical_end = vertical_start + size
        results[horizontal_start: horizontal_end,
        vertical_start: vertical_end, :] = filter_img
plt.figure(figsize=(20, 20))
plt.imshow(results)
```



Loading the VGG16 network with pretrained weights:

In [90]:
```
from keras.applications.vgg16 import VGG16
model = VGG16(weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 2104s 4us/step
```

In [91]:
```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np
img_path = 'C:/Users/ACER/Downloads/test1/50.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
```

In [92]:
```
preds = model.predict(x)
print('Predicted:', decode_predictions(preds, top=3)[0])
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==============================] - 0s 3us/step
Predicted: [('n02105855', 'Shetland_sheepdog', 0.167318), ('n02110185', 'Siberian_husky', 0.11742018), ('n02106166', 'Border_collie', 0.039714873)]
```

In [93]:
```
np.argmax(preds[0])
```

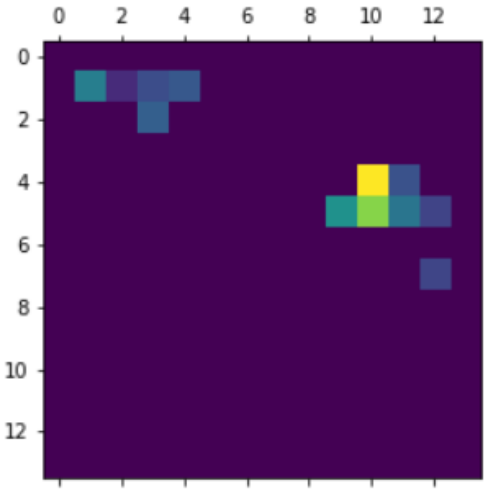
Out[93]: 230

Setting up the Grad-CAM algorithm:

In [95]:
```
cat = model.output[:, 386]
last_conv_layer = model.get_layer('block5_conv3')
grads = K.gradients(cat, last_conv_layer.output)[0]
pooled_grads = K.mean(grads, axis=(0, 1, 2))
iterate = K.function([model.input],
[pooled_grads, last_conv_layer.output[0]])
pooled_grads_value, conv_layer_output_value = iterate([x])
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]
    heatmap = np.mean(conv_layer_output_value, axis=-1)
```

In [96]:
```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```

Out[96]: <matplotlib.image.AxesImage at 0x18a0a808040>



In [100…
```
import cv2
img = cv2.imread(img_path)
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
superimposed_img = heatmap * 0.4 + img
cv2.imwrite('C:/Users/ACER/Downloads/test1/50.jpg', superimposed_img)
```

Out[100… True

Results:

- From the accuracy graphs VGG16(96%) pre-trained model performed well compared to the convnet(82%) that is trained.

Conclusion:

Thus cat and dog images are classified using convnets and VGG16 models and accuracy is evaluated.Data Augmentation was used to avoid overfitting.Filters learned by the convnets are visualized.

References:

- How to Classify Photos of Dogs and Cats (with 97% accuracy) (machinelearningmastery.com)
- Cats And Dogs Image Classification Using Keras – Pythonista Planet
- Keras Implementation of VGG16 Architecture from Scratch with Dogs Vs Cat Data Set | MLK - Machine Learning Knowledge