Artificial Intelligence: Deep Learning

Mini project: Face Mask detection

19. November 2020

Individual report

Name	**Study ID**
Saroj Adhikari	69650

Introduction

In this mini project, I have developed a neural network which is able to detect if the person is wearing a face mask. This kind of system can be used in a place where one needs to have a face mask on.

I tried to make this solution in two different ways. First I tried by applying custom convnets with data augmentation and second I did it using pre-trained convnets. In a second solution, I just run the convolutional base over the dataset, recording the output to a Numpy array. I could not do it in another way, where we can add a dense layer on top to extend the model, because of the limited time frame I had and GPU requirements. We can also use data augmentation if we add a dense layer on the top. It helps us to reduce the overfitting and is very important while working with the small dataset.

Description

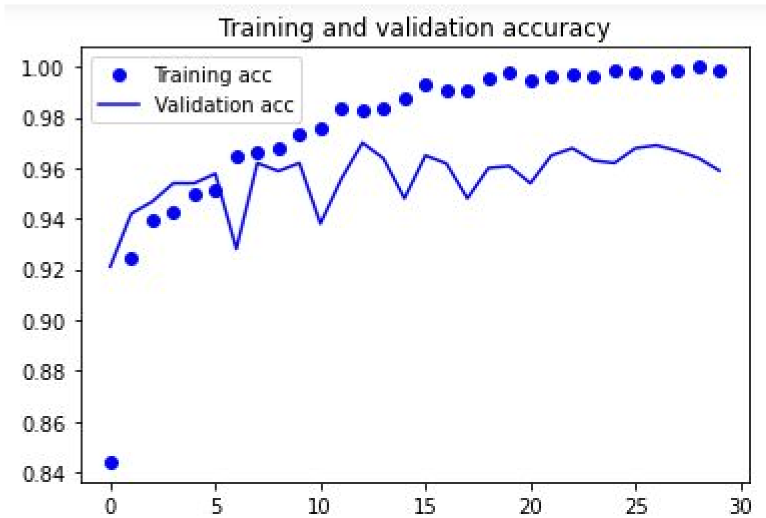
In my dataset, I have different images of people who're having and not having the masks on. Look at the screenshot below for the number of images available for different purposes.

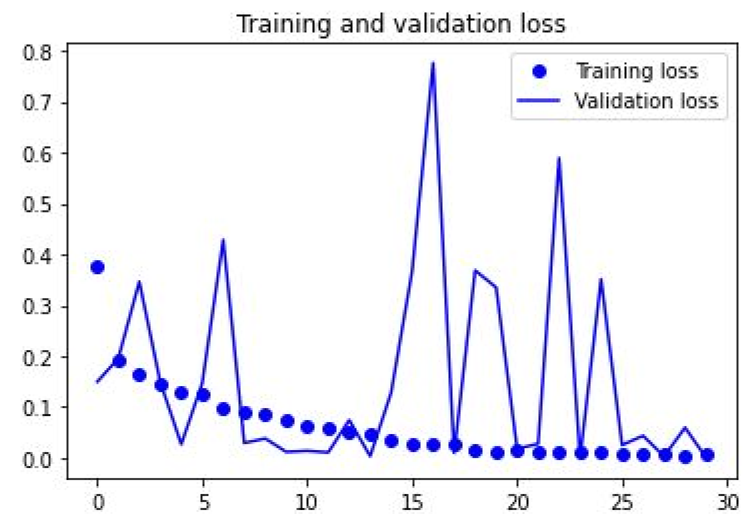
```
In [4]:  print('total with mask training images:', len(os.listdir(train_withmask_dir)))
total with mask training images: 682

In [5]:  print('total with mask validation images:', len(os.listdir(validation_withmask_dir)))
total with mask validation images: 1386

In [6]:  print('total with mask test images:', len(os.listdir(test_withmask_dir)))
total with mask test images: 700

In [7]:  print('total without mask training images:', len(os.listdir(train_withoutmask_dir)))
total without mask training images: 712

In [8]:  print('total without mask validation images:', len(os.listdir(validation_withoutmask_dir)))
total without mask validation images: 1389

In [9]:  print('total without mask test images:', len(os.listdir(test_withoutmask_dir)))
total without mask test images: 705
```

I’ve applied convnets to extract the feature map. I assigned different convolutional and pooling layers as you can see in the screenshot below.

```
In [12]:   1 model.summary()

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

From the summary, we can notice that the dimension of the model is decreasing when the depth is increasing and also have 3,453,121 trainable parameters. This is because in every layer it is collecting the features of the images and shrinking. It ended up with having a dimension of 7*7 with 128 depth in the very last pooling layer. Let’s look at the generated plot from this result.





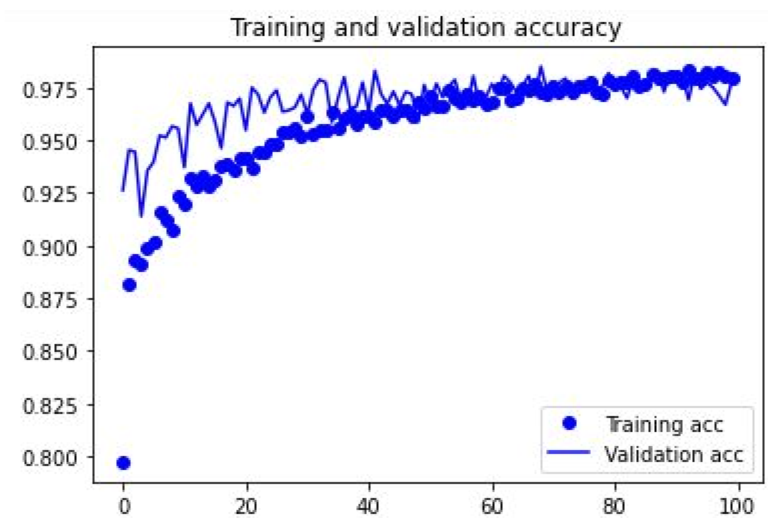
Having applied convnets with the different convolutional and pooling layers, our model is overfitted. This can be noticed by the lines of training and accuracy going two different directions. Overfitting has become the biggest problem in our model. This is mainly because our dataset is very small and we do not have any layer like “Dropout” to reduce overfitting .

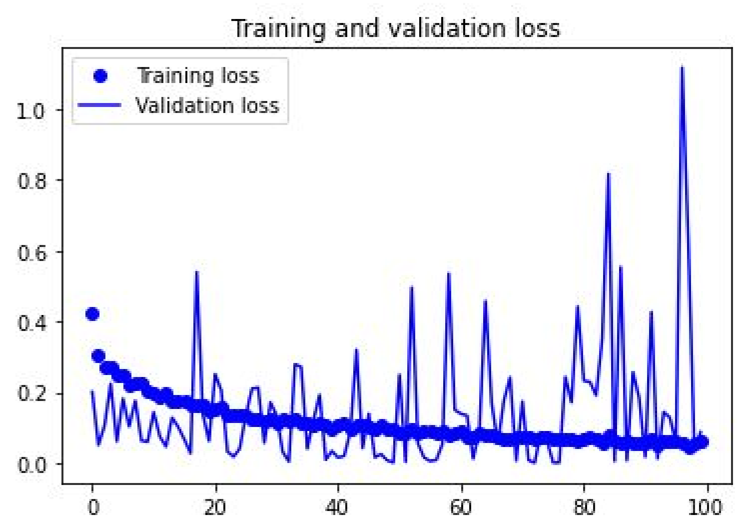
To overcome this problem we have used data augmentation which generates new samples from the existing samples and also applied “Dropout”. It alters certain transitions of the original image and makes it different. Data augmentation is one of the most important techniques to overcome overfitting, especially working with small datasets. Because of data augmentation our network never experiences the same input twice but inputs are mostly related because we are just remixing the same information. Let’s look at the options below. I have applied for the ImageDataGenerator to perform the data augmentation.

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)
```

Each option changes the specific attribute of an image.

Let’s see the plot after fitting the model with data augmentation and the dropout.





Now, we have achieved the accuracy of 98% after data augmentation.

Working with pre-trained convnet VGG16 vs Custom

VGG16 is a pre-trained convnet. We can use it in our project in two different ways. First we can just run the convolutional base over our dataset, second we can add a dense layer on top. The second one gives us the ability to use data augmentation.

Comparing customly created networks, using pre-trained convnet is faster. Also it provides better accuracy as it has been tested and developed given a long time.

Notebooks

Face mask detection - Custom convnets

In [1]:
```
import keras
keras.__version__
```

```
Using TensorFlow backend.
```

Out[1]:
```
'2.3.1'
```

In [2]:
```
import os, shutil
```

In [3]:
```
# The directory where we will
# store our smaller dataset

### MODIFY THIS LINE SO IT FITS WITH YOUR FILES!!!!!
## IF YOUR NOTEBOOK IS IN THE SAME FOLDER AS THE DATASET, YOU CAN JUST
USE RELATIVE PATHS
base_dir = 'dataset'


# Directories for our training,
# validation and test splits
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Directory with our training cat pictures
train_withmask_dir = os.path.join(train_dir, 'withmask')

# Directory with our validation cat pictures
validation_withmask_dir = os.path.join(validation_dir, 'withmask')

# Directory with our validation cat pictures
test_withmask_dir = os.path.join(test_dir, 'withmask')

# Directory with our training cat pictures
train_withoutmask_dir = os.path.join(train_dir, 'withoutmask')

# Directory with our validation cat pictures
validation_withoutmask_dir = os.path.join(validation_dir, 'withoutmask
')

# Directory with our validation cat pictures
test_withoutmask_dir = os.path.join(test_dir, 'withoutmask')
```

In [4]:
```
print('total with mask training images:', len(os.listdir(train_withmask
_dir)))
```

```
total with mask training images: 682
```

In [5]:
```
print('total with mask validation images:', len(os.listdir(validation_w
ithmask_dir)))
```

```
total with mask validation images: 1386
```

In [6]:
```
print('total with mask test images:', len(os.listdir(test_withmask_di
r)))
```

```
total with mask test images: 700
```

In [7]:
```
print('total without mask training images:', len(os.listdir(train_witho
utmask_dir)))
```

```
total without mask training images: 712
```

In [8]:
```
print('total without mask validation images:', len(os.listdir(validatio
n_withoutmask_dir)))
```

```
total without mask validation images: 1389
```

In [9]:
```
print('total without mask test images:', len(os.listdir(test_withoutmas
k_dir)))
```

```
total without mask test images: 705
```

In [10]:
```
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 [11]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
from keras import optimizers

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

In [13]:
```
from keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 1394 images belonging to 2 classes.
Found 2775 images belonging to 2 classes.
```

In [14]:
```
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,)
```

In [15]:
```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

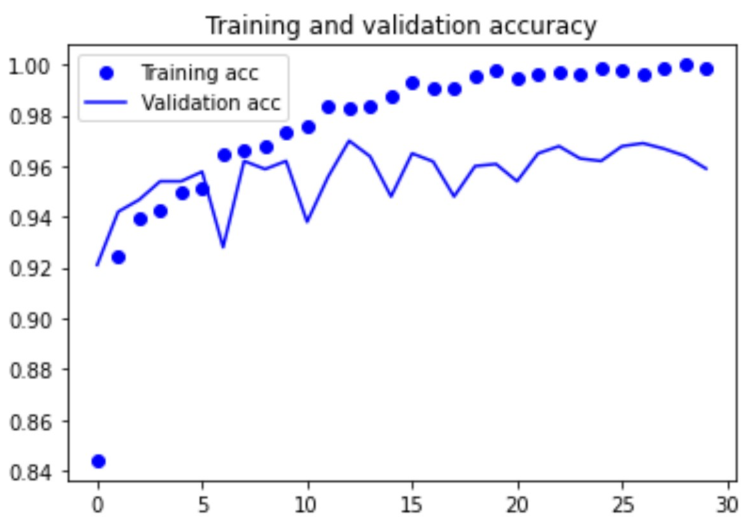
```
Epoch 1/30
 34/100 [=========>....................] - ETA: 1:09 - loss: 0.5392 -
acc: 0.7515

C:\Users\saroj\anaconda3\envs\AI_keras_env\lib\site-packages\PIL\Imag
e.py:961: UserWarning: Palette images with Transparency expressed in
bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
```

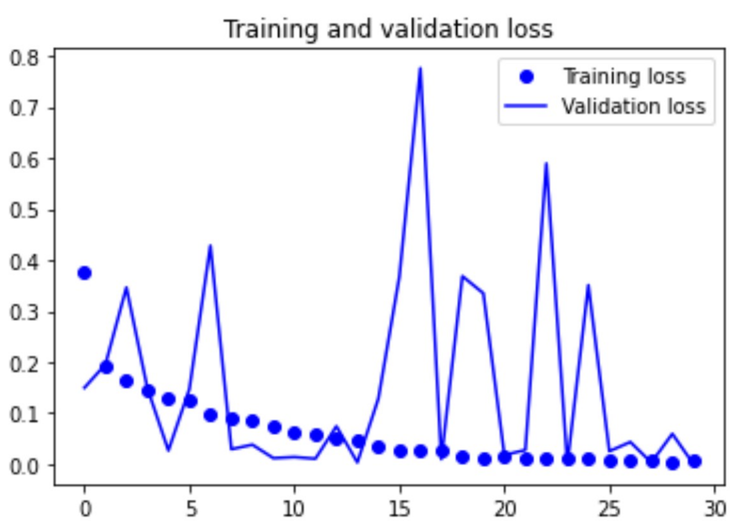
```
100/100 [==============================] - 114s 1s/step - loss: 0.377
4 - acc: 0.8440 - val_loss: 0.1498 - val_acc: 0.9210
Epoch 2/30
100/100 [==============================] - 127s 1s/step - loss: 0.193
1 - acc: 0.9245 - val_loss: 0.1955 - val_acc: 0.9420
Epoch 3/30
100/100 [==============================] - 122s 1s/step - loss: 0.163
8 - acc: 0.9393 - val_loss: 0.3463 - val_acc: 0.9467
Epoch 4/30
100/100 [==============================] - 133s 1s/step - loss: 0.146
5 - acc: 0.9427 - val_loss: 0.1489 - val_acc: 0.9540
Epoch 5/30
100/100 [==============================] - 229s 2s/step - loss: 0.129
8 - acc: 0.9493 - val_loss: 0.0268 - val_acc: 0.9540
Epoch 6/30
100/100 [==============================] - 261s 3s/step - loss: 0.124
6 - acc: 0.9512 - val_loss: 0.1472 - val_acc: 0.9578
Epoch 7/30
100/100 [==============================] - 248s 2s/step - loss: 0.098
6 - acc: 0.9644 - val_loss: 0.4282 - val_acc: 0.9280
Epoch 8/30
100/100 [==============================] - 200s 2s/step - loss: 0.088
8 - acc: 0.9659 - val_loss: 0.0294 - val_acc: 0.9620
Epoch 9/30
100/100 [==============================] - 225s 2s/step - loss: 0.086
9 - acc: 0.9678 - val_loss: 0.0382 - val_acc: 0.9588
Epoch 10/30
100/100 [==============================] - 231s 2s/step - loss: 0.073
0 - acc: 0.9734 - val_loss: 0.0119 - val_acc: 0.9620
Epoch 11/30
100/100 [==============================] - 204s 2s/step - loss: 0.063
6 - acc: 0.9759 - val_loss: 0.0140 - val_acc: 0.9380
Epoch 12/30
100/100 [==============================] - 216s 2s/step - loss: 0.058
4 - acc: 0.9835 - val_loss: 0.0107 - val_acc: 0.9558
Epoch 13/30
100/100 [==============================] - 213s 2s/step - loss: 0.049
3 - acc: 0.9824 - val_loss: 0.0744 - val_acc: 0.9700
Epoch 14/30
100/100 [==============================] - 152s 2s/step - loss: 0.046
0 - acc: 0.9834 - val_loss: 0.0035 - val_acc: 0.9638
Epoch 15/30
```

In [16]:

```
model.save('face_mask_1.h5')
```

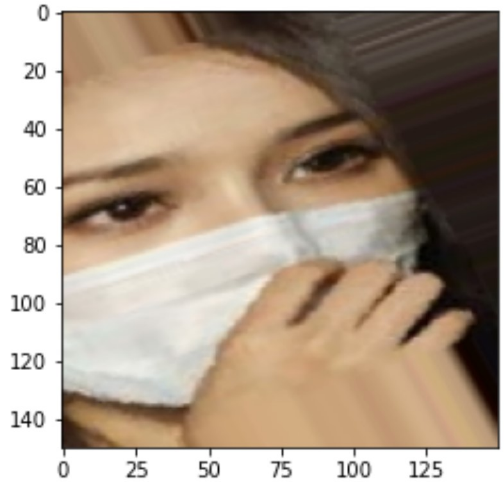
In [17]:
```
import matplotlib.pyplot as plt

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

epochs = range(len(acc))

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()
```

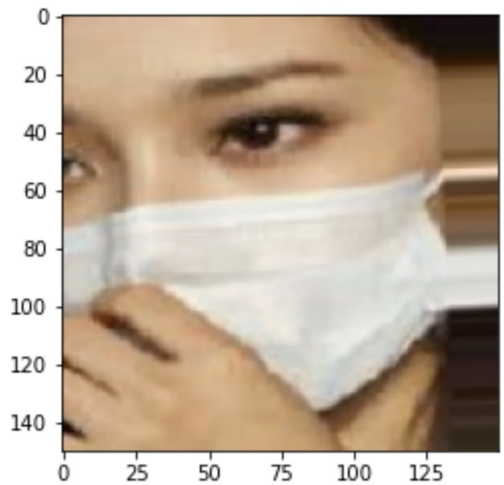


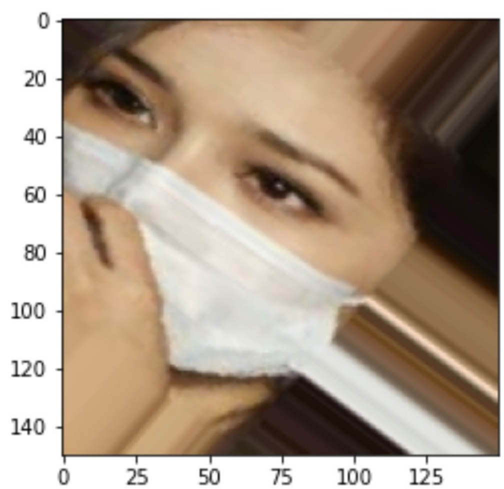


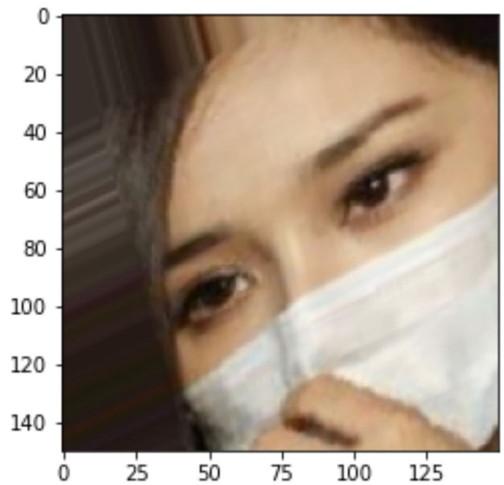
In [18]:
```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

In [19]:
```
# This is module with image preprocessing utilities
from keras.preprocessing import image

fnames = [os.path.join(train_withmask_dir, fname) for fname in os.listd
ir(train_withmask_dir)]

# We pick one image to "augment"
img_path = fnames[3]

# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)

# The .flow() command below generates batches of randomly transformed i
mages.
# It will loop indefinitely, so we need to `break` the loop at some poi
nt!
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()
```


0
20
40
60
80
100
120
140
0 25 50 75 100 125


0
20
40
60
80
100
120
140
0 25 50 75 100 125


0
20
40
60
80
100
120
140
0 25 50 75 100 125



In [20]:
```
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'])
```

In [21]:
```
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,)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 150x150
        target_size=(150, 150),
        batch_size=32,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

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

```
Found 1394 images belonging to 2 classes.
Found 2775 images belonging to 2 classes.
Epoch 1/100
 16/100 [===>.........................] - ETA: 2:27 - loss: 0.6627 -
acc: 0.5984

C:\Users\saroj\anaconda3\envs\AI_keras_env\lib\site-packages\PIL\Imag
e.py:961: UserWarning: Palette images with Transparency expressed in
bytes should be converted to RGBA images
  "Palette images with Transparency expressed in bytes should be "
```

```
100/100 [==============================] - 233s 2s/step - loss: 0.426
5 - acc: 0.7973 - val_loss: 0.2011 - val_acc: 0.9262
Epoch 2/100
100/100 [==============================] - 312s 3s/step - loss: 0.301
6 - acc: 0.8821 - val_loss: 0.0512 - val_acc: 0.9453
Epoch 3/100
100/100 [==============================] - 246s 2s/step - loss: 0.274
8 - acc: 0.8933 - val_loss: 0.1059 - val_acc: 0.9444
Epoch 4/100
100/100 [==============================] - 232s 2s/step - loss: 0.272
6 - acc: 0.8911 - val_loss: 0.2245 - val_acc: 0.9139
Epoch 5/100
100/100 [==============================] - 217s 2s/step - loss: 0.251
2 - acc: 0.8989 - val_loss: 0.0616 - val_acc: 0.9356
Epoch 6/100
100/100 [==============================] - 223s 2s/step - loss: 0.249
2 - acc: 0.9013 - val_loss: 0.1820 - val_acc: 0.9397
Epoch 7/100
100/100 [==============================] - 238s 2s/step - loss: 0.222
8 - acc: 0.9155 - val_loss: 0.1035 - val_acc: 0.9522
Epoch 8/100
100/100 [==============================] - 218s 2s/step - loss: 0.228
0 - acc: 0.9120 - val_loss: 0.1763 - val_acc: 0.9513
Epoch 9/100
100/100 [==============================] - 212s 2s/step - loss: 0.227
9 - acc: 0.9075 - val_loss: 0.0636 - val_acc: 0.9566
Epoch 10/100
100/100 [==============================] - 218s 2s/step - loss: 0.202
3 - acc: 0.9237 - val_loss: 0.0600 - val_acc: 0.9556
Epoch 11/100
100/100 [==============================] - 214s 2s/step - loss: 0.201
4 - acc: 0.9199 - val_loss: 0.1439 - val_acc: 0.9371
Epoch 12/100
100/100 [==============================] - 220s 2s/step - loss: 0.186
3 - acc: 0.9316 - val_loss: 0.0751 - val_acc: 0.9675
Epoch 13/100
100/100 [==============================] - 216s 2s/step - loss: 0.196
1 - acc: 0.9281 - val_loss: 0.0466 - val_acc: 0.9573
Epoch 14/100
100/100 [==============================] - 216s 2s/step - loss: 0.173
4 - acc: 0.9332 - val_loss: 0.1285 - val_acc: 0.9623
Epoch 15/100
100/100 [==============================] - 218s 2s/step - loss: 0.173
6 - acc: 0.9284 - val_loss: 0.1017 - val_acc: 0.9675
Epoch 16/100
100/100 [==============================] - 214s 2s/step - loss: 0.176
7 - acc: 0.9313 - val_loss: 0.0667 - val_acc: 0.9585
Epoch 17/100
100/100 [==============================] - 215s 2s/step - loss: 0.160
7 - acc: 0.9379 - val_loss: 0.0276 - val_acc: 0.9463
Epoch 18/100
100/100 [==============================] - 216s 2s/step - loss: 0.167
6 - acc: 0.9392 - val_loss: 0.5395 - val_acc: 0.9679
Epoch 19/100
100/100 [==============================] - 219s 2s/step - loss: 0.164
8 - acc: 0.9357 - val_loss: 0.1376 - val_acc: 0.9663
```

```
Epoch 20/100
100/100 [==============================] - 218s 2s/step - loss: 0.148
0 - acc: 0.9420 - val_loss: 0.0625 - val_acc: 0.9698
Epoch 21/100
100/100 [==============================] - 220s 2s/step - loss: 0.151
2 - acc: 0.9417 - val_loss: 0.2514 - val_acc: 0.9547
Epoch 22/100
100/100 [==============================] - 217s 2s/step - loss: 0.159
5 - acc: 0.9373 - val_loss: 0.2049 - val_acc: 0.9750
Epoch 23/100
100/100 [==============================] - 212s 2s/step - loss: 0.136
0 - acc: 0.9448 - val_loss: 0.0329 - val_acc: 0.9717
Epoch 24/100
100/100 [==============================] - 213s 2s/step - loss: 0.138
2 - acc: 0.9443 - val_loss: 0.0180 - val_acc: 0.9631
Epoch 25/100
100/100 [==============================] - 216s 2s/step - loss: 0.134
5 - acc: 0.9485 - val_loss: 0.0414 - val_acc: 0.9705
Epoch 26/100
100/100 [==============================] - 214s 2s/step - loss: 0.134
0 - acc: 0.9481 - val_loss: 0.1365 - val_acc: 0.9737
Epoch 27/100
100/100 [==============================] - 214s 2s/step - loss: 0.123
3 - acc: 0.9543 - val_loss: 0.2103 - val_acc: 0.9635
Epoch 28/100
100/100 [==============================] - 216s 2s/step - loss: 0.127
3 - acc: 0.9541 - val_loss: 0.2135 - val_acc: 0.9642
Epoch 29/100
100/100 [==============================] - 211s 2s/step - loss: 0.120
0 - acc: 0.9562 - val_loss: 0.0574 - val_acc: 0.9656
Epoch 30/100
100/100 [==============================] - 219s 2s/step - loss: 0.125
3 - acc: 0.9521 - val_loss: 0.1715 - val_acc: 0.9717
Epoch 31/100
100/100 [==============================] - 213s 2s/step - loss: 0.115
0 - acc: 0.9611 - val_loss: 0.1350 - val_acc: 0.9625
Epoch 32/100
100/100 [==============================] - 213s 2s/step - loss: 0.124
1 - acc: 0.9529 - val_loss: 0.0314 - val_acc: 0.9742
Epoch 33/100
100/100 [==============================] - 213s 2s/step - loss: 0.122
6 - acc: 0.9547 - val_loss: 0.0045 - val_acc: 0.9787
Epoch 34/100
100/100 [==============================] - 212s 2s/step - loss: 0.127
7 - acc: 0.9552 - val_loss: 0.2791 - val_acc: 0.9774
Epoch 35/100
100/100 [==============================] - 214s 2s/step - loss: 0.114
4 - acc: 0.9630 - val_loss: 0.2716 - val_acc: 0.9585
Epoch 36/100
100/100 [==============================] - 212s 2s/step - loss: 0.112
7 - acc: 0.9562 - val_loss: 0.0406 - val_acc: 0.9700
Epoch 37/100
100/100 [==============================] - 226s 2s/step - loss: 0.109
7 - acc: 0.9609 - val_loss: 0.1216 - val_acc: 0.9799
Epoch 38/100
100/100 [==============================] - 212s 2s/step - loss: 0.116
```

```
1 - acc: 0.9625 - val_loss: 0.1928 - val_acc: 0.9650
Epoch 39/100
100/100 [==============================] - 210s 2s/step - loss: 0.107
4 - acc: 0.9574 - val_loss: 0.0093 - val_acc: 0.9661
Epoch 40/100
100/100 [==============================] - 211s 2s/step - loss: 0.097
2 - acc: 0.9617 - val_loss: 0.0342 - val_acc: 0.9775
Epoch 41/100
100/100 [==============================] - 216s 2s/step - loss: 0.109
3 - acc: 0.9619 - val_loss: 0.0156 - val_acc: 0.9635
Epoch 42/100
100/100 [==============================] - 210s 2s/step - loss: 0.110
1 - acc: 0.9584 - val_loss: 0.0202 - val_acc: 0.9830
Epoch 43/100
100/100 [==============================] - 211s 2s/step - loss: 0.098
5 - acc: 0.9639 - val_loss: 0.0888 - val_acc: 0.9719
Epoch 44/100
100/100 [==============================] - 213s 2s/step - loss: 0.105
3 - acc: 0.9644 - val_loss: 0.3201 - val_acc: 0.9673
Epoch 45/100
100/100 [==============================] - 213s 2s/step - loss: 0.105
3 - acc: 0.9612 - val_loss: 0.0425 - val_acc: 0.9731
Epoch 46/100
100/100 [==============================] - 211s 2s/step - loss: 0.104
3 - acc: 0.9647 - val_loss: 0.1395 - val_acc: 0.9667
Epoch 47/100
100/100 [==============================] - 208s 2s/step - loss: 0.095
8 - acc: 0.9645 - val_loss: 0.0163 - val_acc: 0.9730
Epoch 48/100
100/100 [==============================] - 213s 2s/step - loss: 0.105
9 - acc: 0.9612 - val_loss: 0.0253 - val_acc: 0.9719
Epoch 49/100
100/100 [==============================] - 210s 2s/step - loss: 0.094
0 - acc: 0.9680 - val_loss: 0.0078 - val_acc: 0.9585
Epoch 50/100
100/100 [==============================] - 211s 2s/step - loss: 0.093
5 - acc: 0.9650 - val_loss: 0.0012 - val_acc: 0.9762
Epoch 51/100
100/100 [==============================] - 209s 2s/step - loss: 0.084
8 - acc: 0.9708 - val_loss: 0.2496 - val_acc: 0.9679
Epoch 52/100
100/100 [==============================] - 215s 2s/step - loss: 0.089
0 - acc: 0.9658 - val_loss: 0.0037 - val_acc: 0.9769
Epoch 53/100
100/100 [==============================] - 211s 2s/step - loss: 0.094
1 - acc: 0.9660 - val_loss: 0.4959 - val_acc: 0.9705
Epoch 54/100
100/100 [==============================] - 220s 2s/step - loss: 0.082
9 - acc: 0.9737 - val_loss: 0.0561 - val_acc: 0.9755
Epoch 55/100
100/100 [==============================] - 212s 2s/step - loss: 0.088
8 - acc: 0.9697 - val_loss: 0.0158 - val_acc: 0.9787
Epoch 56/100
100/100 [==============================] - 209s 2s/step - loss: 0.088
7 - acc: 0.9682 - val_loss: 0.0056 - val_acc: 0.9686
Epoch 57/100
```

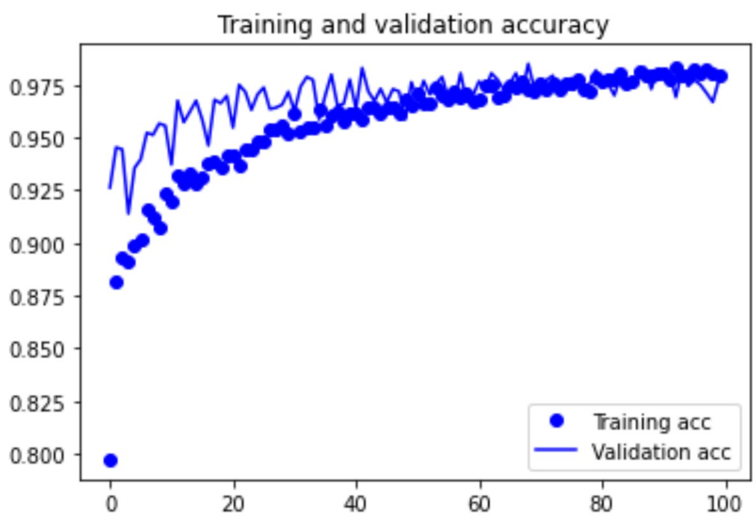
```
100/100 [==============================] - 210s 2s/step - loss: 0.087
9 - acc: 0.9715 - val_loss: 0.0087 - val_acc: 0.9669
Epoch 58/100
100/100 [==============================] - 210s 2s/step - loss: 0.090
8 - acc: 0.9691 - val_loss: 0.0549 - val_acc: 0.9805
Epoch 59/100
100/100 [==============================] - 215s 2s/step - loss: 0.078
3 - acc: 0.9712 - val_loss: 0.5352 - val_acc: 0.9669
Epoch 60/100
100/100 [==============================] - 209s 2s/step - loss: 0.085
1 - acc: 0.9674 - val_loss: 0.1503 - val_acc: 0.9698
Epoch 61/100
100/100 [==============================] - 211s 2s/step - loss: 0.090
4 - acc: 0.9683 - val_loss: 0.1392 - val_acc: 0.9767
Epoch 62/100
100/100 [==============================] - 214s 2s/step - loss: 0.073
0 - acc: 0.9751 - val_loss: 0.1342 - val_acc: 0.9725
Epoch 63/100
100/100 [==============================] - 213s 2s/step - loss: 0.073
5 - acc: 0.9751 - val_loss: 0.0123 - val_acc: 0.9805
Epoch 64/100
100/100 [==============================] - 212s 2s/step - loss: 0.082
4 - acc: 0.9687 - val_loss: 0.0922 - val_acc: 0.9775
Epoch 65/100
100/100 [==============================] - 211s 2s/step - loss: 0.081
6 - acc: 0.9699 - val_loss: 0.4577 - val_acc: 0.9730
Epoch 66/100
100/100 [==============================] - 210s 2s/step - loss: 0.077
4 - acc: 0.9750 - val_loss: 0.1701 - val_acc: 0.9756
Epoch 67/100
100/100 [==============================] - 210s 2s/step - loss: 0.071
0 - acc: 0.9741 - val_loss: 0.0562 - val_acc: 0.9805
Epoch 68/100
100/100 [==============================] - 208s 2s/step - loss: 0.069
8 - acc: 0.9763 - val_loss: 0.1772 - val_acc: 0.9717
Epoch 69/100
100/100 [==============================] - 210s 2s/step - loss: 0.066
2 - acc: 0.9732 - val_loss: 0.2420 - val_acc: 0.9850
Epoch 70/100
100/100 [==============================] - 215s 2s/step - loss: 0.074
9 - acc: 0.9719 - val_loss: 0.0070 - val_acc: 0.9730
Epoch 71/100
100/100 [==============================] - 220s 2s/step - loss: 0.076
6 - acc: 0.9753 - val_loss: 0.1743 - val_acc: 0.9744
Epoch 72/100
100/100 [==============================] - 211s 2s/step - loss: 0.075
9 - acc: 0.9729 - val_loss: 0.0082 - val_acc: 0.9774
Epoch 73/100
100/100 [==============================] - 208s 2s/step - loss: 0.069
8 - acc: 0.9754 - val_loss: 8.3167e-04 - val_acc: 0.9794
Epoch 74/100
100/100 [==============================] - 215s 2s/step - loss: 0.073
3 - acc: 0.9732 - val_loss: 0.0875 - val_acc: 0.9705
Epoch 75/100
100/100 [==============================] - 213s 2s/step - loss: 0.073
9 - acc: 0.9757 - val_loss: 0.0629 - val_acc: 0.9736
```

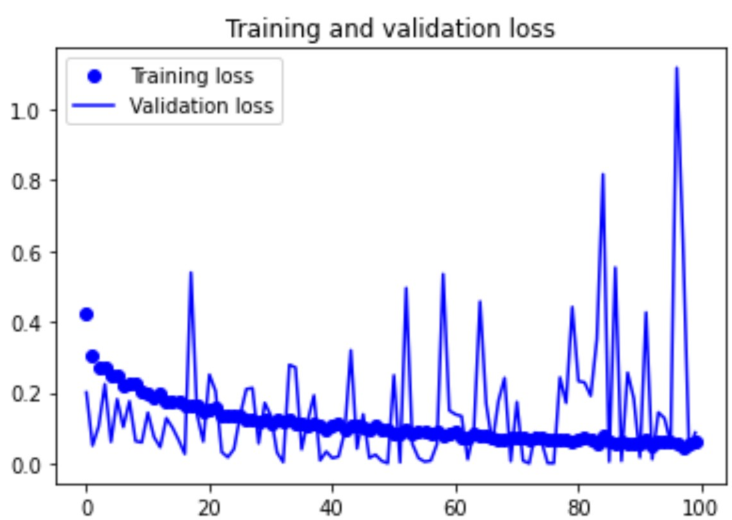
```
Epoch 76/100
100/100 [==============================] - 210s 2s/step - loss: 0.067
1 - acc: 0.9760 - val_loss: 0.0018 - val_acc: 0.9781
Epoch 77/100
100/100 [==============================] - 211s 2s/step - loss: 0.069
2 - acc: 0.9772 - val_loss: 0.0014 - val_acc: 0.9767
Epoch 78/100
100/100 [==============================] - 210s 2s/step - loss: 0.070
4 - acc: 0.9729 - val_loss: 0.2435 - val_acc: 0.9744
Epoch 79/100
100/100 [==============================] - 213s 2s/step - loss: 0.070
7 - acc: 0.9716 - val_loss: 0.1716 - val_acc: 0.9755
Epoch 80/100
100/100 [==============================] - 209s 2s/step - loss: 0.064
0 - acc: 0.9782 - val_loss: 0.4422 - val_acc: 0.9819
Epoch 81/100
100/100 [==============================] - 214s 2s/step - loss: 0.067
2 - acc: 0.9764 - val_loss: 0.2328 - val_acc: 0.9755
Epoch 82/100
100/100 [==============================] - 210s 2s/step - loss: 0.072
5 - acc: 0.9779 - val_loss: 0.2294 - val_acc: 0.9755
Epoch 83/100
100/100 [==============================] - 207s 2s/step - loss: 0.066
4 - acc: 0.9776 - val_loss: 0.1915 - val_acc: 0.9700
Epoch 84/100
100/100 [==============================] - 208s 2s/step - loss: 0.054
7 - acc: 0.9807 - val_loss: 0.3512 - val_acc: 0.9793
Epoch 85/100
```

In [22]:
```
model.save('face_mask_2.h5')
```

In [23]:

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

epochs = range(len(acc))

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()
```

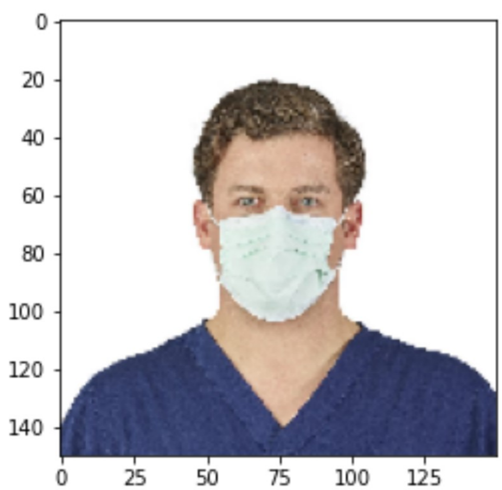




In [24]:
```
# We pick one image to "augment"
img_path = 'manwithmask.jpg'

# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)
prediction = model.predict(x)
item = prediction.item()
```

In [25]:
```
plt.imshow(img)
(item)
```

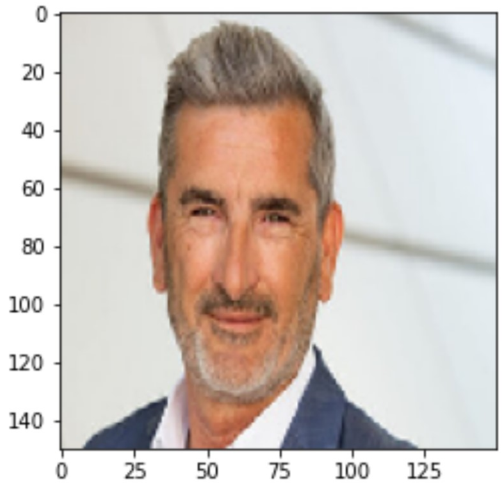
Out[25]: 0.0



In [26]:
```
# We pick one image to "augment"
img_path = 'manwithoutmask.jpg'

# Read the image and resize it
img = image.load_img(img_path, target_size=(150, 150))

# Convert it to a Numpy array with shape (150, 150, 3)
x = image.img_to_array(img)

# Reshape it to (1, 150, 150, 3)
x = x.reshape((1,) + x.shape)
prediction = model.predict(x)
item = prediction.item()
```

In [27]:
```
plt.imshow(img)
(item)
```

Out[27]: 1.0



Face mask detection - VGG16 (Without data augmentation)

In [11]:
```
import keras
keras.__version__
```

Out[11]:
```
'2.3.1'
```

Let's instantiate the VGG16 model:

In [6]:
```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

Let's look at the details of an architechture of VGG16 convolutional base.

In [12]:
```
conv_base.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (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
_________________________________________________________________
```

Let's extract images as Numphy Arrays as well as their labels

In [13]:
```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

## YOUR OWN DIRECTORY!
base_dir = 'dataset'

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_batc
h
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            # Note that since generators yield data indefinitely in a l
oop,
            # we must `break` after every image has been seen once.
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 2700)
validation_features, validation_labels = extract_features(validation_di
r, 1350)
test_features, test_labels = extract_features(test_dir, 1350)
```

```
Found 2700 images belonging to 2 classes.
Found 1350 images belonging to 2 classes.
Found 1350 images belonging to 2 classes.
```

The extracted features are currently of shape (samples, 4, 4, 512) so, we much flatten them.

In [14]:
```
train_features = np.reshape(train_features, (2700, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1350, 4 * 4 * 51
2))
test_features = np.reshape(test_features, (1350, 4 * 4 * 512))
```

Let's define our densely connected classifier and triain it on data and labels that we just recorded.

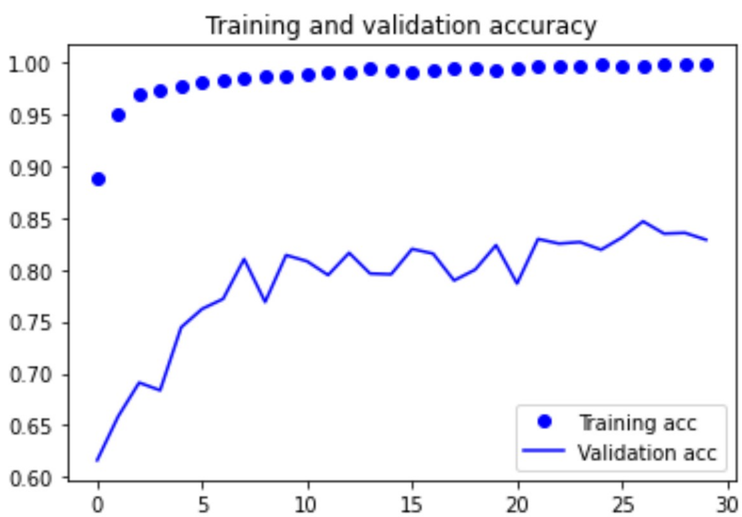
In [15]:
```
from keras import models
from keras import layers
from keras import optimizers

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_la
bels))
```

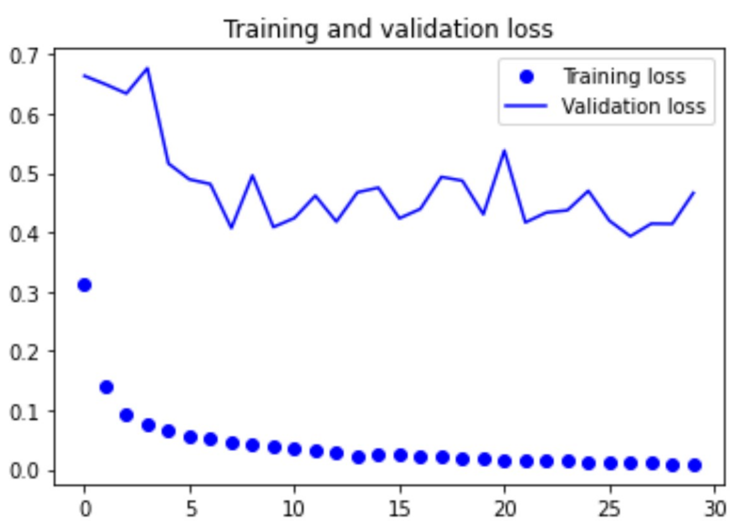
```
Train on 2700 samples, validate on 1350 samples
Epoch 1/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.31
21 - acc: 0.8885 - val_loss: 0.6637 - val_acc: 0.6163
Epoch 2/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.13
94 - acc: 0.9507 - val_loss: 0.6497 - val_acc: 0.6585
Epoch 3/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.09
45 - acc: 0.9689 - val_loss: 0.6341 - val_acc: 0.6911
Epoch 4/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.07
77 - acc: 0.9741 - val_loss: 0.6767 - val_acc: 0.6837
Epoch 5/30
2700/2700 [==============================] - 6s 2ms/step - loss: 0.06
47 - acc: 0.9778 - val_loss: 0.5161 - val_acc: 0.7444
Epoch 6/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.05
67 - acc: 0.9815 - val_loss: 0.4893 - val_acc: 0.7622
Epoch 7/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.05
16 - acc: 0.9819 - val_loss: 0.4814 - val_acc: 0.7719
Epoch 8/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.04
56 - acc: 0.9852 - val_loss: 0.4073 - val_acc: 0.8104
Epoch 9/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.04
22 - acc: 0.9874 - val_loss: 0.4959 - val_acc: 0.7689
Epoch 10/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.03
80 - acc: 0.9870 - val_loss: 0.4089 - val_acc: 0.8141
Epoch 11/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.03
55 - acc: 0.9878 - val_loss: 0.4242 - val_acc: 0.8081
Epoch 12/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.03
35 - acc: 0.9896 - val_loss: 0.4618 - val_acc: 0.7948
Epoch 13/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
95 - acc: 0.9900 - val_loss: 0.4180 - val_acc: 0.8163
Epoch 14/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
38 - acc: 0.9937 - val_loss: 0.4675 - val_acc: 0.7963
Epoch 15/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
60 - acc: 0.9922 - val_loss: 0.4753 - val_acc: 0.7956
Epoch 16/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
69 - acc: 0.9911 - val_loss: 0.4236 - val_acc: 0.8200
Epoch 17/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
16 - acc: 0.9930 - val_loss: 0.4394 - val_acc: 0.8156
Epoch 18/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
16 - acc: 0.9948 - val_loss: 0.4935 - val_acc: 0.7896
Epoch 19/30
```

```
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
86 - acc: 0.9952 - val_loss: 0.4870 - val_acc: 0.8000
Epoch 20/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.02
02 - acc: 0.9933 - val_loss: 0.4304 - val_acc: 0.8237
Epoch 21/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
62 - acc: 0.9948 - val_loss: 0.5379 - val_acc: 0.7867
Epoch 22/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
55 - acc: 0.9963 - val_loss: 0.4167 - val_acc: 0.8296
Epoch 23/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
54 - acc: 0.9970 - val_loss: 0.4335 - val_acc: 0.8252
Epoch 24/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
53 - acc: 0.9963 - val_loss: 0.4372 - val_acc: 0.8267
Epoch 25/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
27 - acc: 0.9974 - val_loss: 0.4700 - val_acc: 0.8193
Epoch 26/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
32 - acc: 0.9967 - val_loss: 0.4197 - val_acc: 0.8311
Epoch 27/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
35 - acc: 0.9959 - val_loss: 0.3934 - val_acc: 0.8467
Epoch 28/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
14 - acc: 0.9974 - val_loss: 0.4147 - val_acc: 0.8348
Epoch 29/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.01
03 - acc: 0.9978 - val_loss: 0.4143 - val_acc: 0.8356
Epoch 30/30
2700/2700 [==============================] - 5s 2ms/step - loss: 0.00
93 - acc: 0.9978 - val_loss: 0.4664 - val_acc: 0.8289
```

Let's look at the plot for loss and accuracy during the plotting.

In [16]:
```
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(len(acc))

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()
```





We reached validation accuracy around 85%. However, we can also some overfitting. This is because we cannot add data augmentation in this technique to reduce overfitting. Technique to reduce overfitting is very important while we work with small datasets.

Let's save the model

In [17]:
```
model.save('face_mask_3.h5')
```

This is the end of this notebook

Real Time detection

In [3]:

```
import cv2
import numpy as np
from PIL import Image
from keras import models

#Load the saved model
model = models.load_model('face_mask_2.h5')
video = cv2.VideoCapture(1)

labels_dict={0:'with_mask',1:'without_mask'}
color_dict={0:(0,255,0),1:(0,0,255)}

while True:
        _, frame = video.read()

        #Convert the captured frame into RGB
        im = Image.fromarray(frame, 'RGB')

        #Resizing into 128x128 because we trained the model with this i
mage size.
        im = im.resize((150,150))
        img_array = np.array(im)

        #Our keras model used a 4D tensor, (images x height x width x c
hannel)
        #So changing dimension 128x128x3 into 1x128x128x3
        img_array = np.expand_dims(img_array, axis=0)

        #Calling the predict method on model to predict 'me' on the ima
ge
        prediction = int(model.predict(img_array)[0][0])
        luna = str(model.predict_proba(img_array))

        #if prediction is 0, which means I am missing on the image, the
n show the frame in gray color.
        if prediction == 0:
                cv2.putText(frame, 'Mask! Probability:' + luna, (50, 5
0), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE_4)

        if prediction == 1:
                cv2.putText(frame, 'No mask!  Probability:' + luna, (5
0, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE_4)

        cv2.imshow("Capturing", frame)
        key=cv2.waitKey(1)
        if key == ord('q'):
                break
video.release()
cv2.destroyAllWindows()
```

Conclusion

It has been fun working with CNN to determine if the person is wearing a facemask. There are multiple ways we can choose to implement an idea. Performing without a pre-trained convnets is time consuming where we have to spend a lot of time to find out what works the best. But on the other hand if we are working with a small dataset, we still need to apply data augmentation even if we have a pre-trained network. It takes a lot of time and GPU.

Convolutional and pooling layers are very important to generate a feature map. It gathers important features from the input and shrinks the dimension. Which progressively transforms the image from human readable to non-readable.