cat_dog_bike_car_zhou_xi

using CNN to classify cat&dog&bike&car

*If the computing power of the computer is not enough, running the plot will cause the server to crash and the drawn image will not be displayed. I tried to solve the drawing problem by using the computer in the lab instead of using my own laptop.

In order to prevent the same problem from occurring, I will paste the picture drawn in the answer for reference.

In [45]:
```
import os
base_dir = './cat_dog_car_bike'
train_dir = './cat_dog_car_bike/train'
val_dir = './cat_dog_car_bike/val'
test_dir = './cat_dog_car_bike/test'

train_cats_dir=os.path.join(train_dir,'c0')
train_dogs_dir=os.path.join(train_dir,'c1')
train_cars_dir=os.path.join(train_dir,'c2')
train_motorbikes_dir=os.path.join(train_dir,'c3')
test_cats_dir=os.path.join(test_dir,'c0')
test_dogs_dir=os.path.join(test_dir,'c1')
test_cars_dir=os.path.join(test_dir,'c2')
test_motorbikes_dir=os.path.join(test_dir,'c3')
val_cats_dir=os.path.join(val_dir,'c0')
val_dogs_dir=os.path.join(val_dir,'c1')
val_cars_dir=os.path.join(val_dir,'c2')
val_motorbikes_dir=os.path.join(val_dir,'c3')
```

Question1： CNN architecture

In [46]:
```
#Use the sigmoid activation function for all layers but the last one which uses
the softmax activation function. Use default values for the parameters which ar
e not specified above.
import tensorflow
from tensorflow import keras
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),padding='same',activation='sigmoid',#add the
padding—for question1(b)
                        input_shape=(32, 32, 3))) #since we rescale the input p
ictures to 32*32,the input_shape shoude be (32,32,3)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),padding='same', activation='sigmoid')) #add
 the padding—for question1(b)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), padding='same',activation='sigmoid')) #add
the padding—for question1(b)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3),padding='same',activation='sigmoid')) #add
 the padding—for question1(b)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512,activation='sigmoid'))
model.add(layers.Dense(4, activation='softmax')) # k=4
```

In [47]:
```
## Configuration for training
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_133 (Conv2D)          (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_132 (MaxPoolin (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_134 (Conv2D)          (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_133 (MaxPoolin (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_135 (Conv2D)          (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_134 (MaxPoolin (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 4, 4, 128)         147584
_________________________________________________________________
max_pooling2d_135 (MaxPoolin (None, 2, 2, 128)         0
_________________________________________________________________
flatten_33 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_65 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_66 (Dense)             (None, 4)                 2052
=================================================================
Total params: 505,540
Trainable params: 505,540
Non-trainable params: 0
_________________________________________________________________
```

Quesion1_answers

- (a) the right value for k should be 4,since we have 4 classes:cats,dogs,cars,motorbikes,so the size of output should be 4.
- (6) The issue in the architecurte of previous CNN is 'Negative dimension appears'.The reason for the problem is that the feature map will shrink through each conv2D layer when there is no padding.We should add padding to the conv2D layers,using the code 'padding='same'.the padding parameter has two values:'valid' and 'same',‘valid’ means no padding,‘same’ means that the output size is the same as the input after using padding,the default value of the parameter is ‘valid’, we should change it to ‘same’.

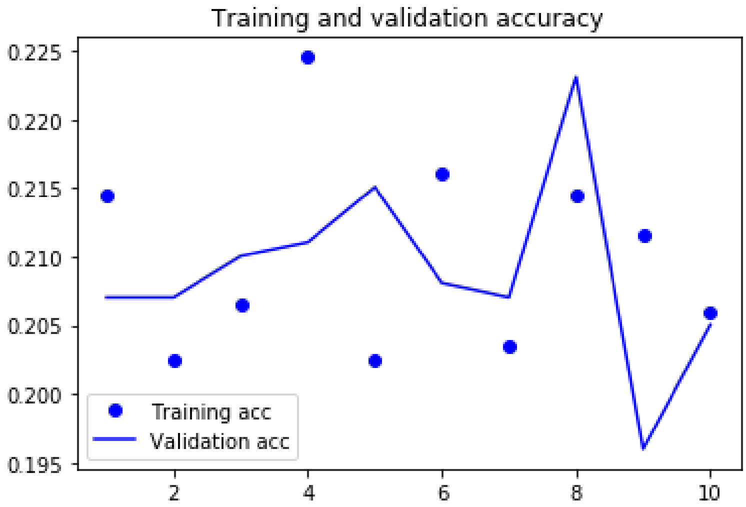
Question2：Training a small CNN from scratch

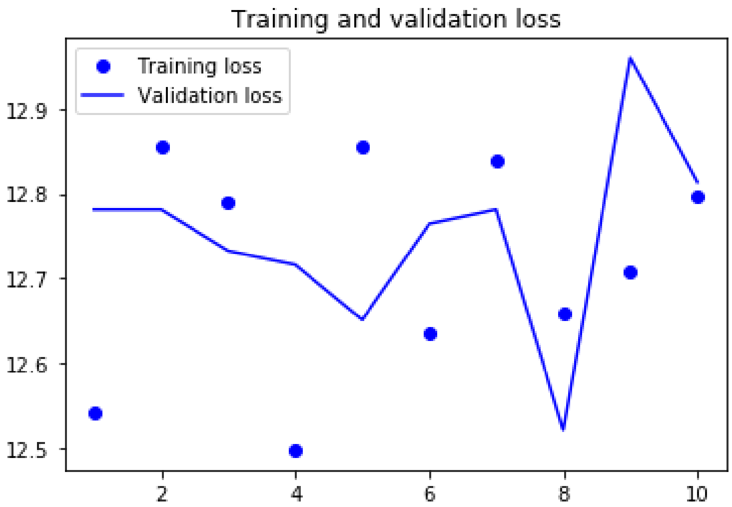
In [48]:
```python
import PIL
from keras import optimizers
model.compile(loss='categorical_crossentropy', #We should change the binary_cro
ssentropy to categorical_crossentropy,since now we have 4 classes
              optimizer=optimizers.RMSprop(lr=0.1),
              metrics=['acc'])

from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255) #rescale the tensor values t
o [0,1]
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(32, 32),
        batch_size=20,class_mode='categorical') #we have four classes

validation_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(32, 32),
        batch_size=20,
        class_mode='categorical')

print ("start..")
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=50)
```

```
Found 1675 images belonging to 4 classes.
Found 835 images belonging to 4 classes.
start..
Epoch 1/10
100/100 [==============================] - 11s 111ms/step - loss: 12.2470 - ac
c: 0.2313 - val_loss: 12.2303 - val_acc: 0.2412
Epoch 2/10
100/100 [==============================] - 9s 88ms/step - loss: 12.3330 - acc:
0.2348 - val_loss: 12.4085 - val_acc: 0.2302
Epoch 3/10
100/100 [==============================] - 9s 89ms/step - loss: 12.2767 - acc:
0.2383 - val_loss: 12.4085 - val_acc: 0.2302
Epoch 4/10
100/100 [==============================] - 9s 88ms/step - loss: 12.3061 - acc:
0.2365 - val_loss: 12.2627 - val_acc: 0.2392
Epoch 5/10
100/100 [==============================] - 9s 88ms/step - loss: 12.3491 - acc:
0.2338 - val_loss: 12.2627 - val_acc: 0.2392
Epoch 6/10
100/100 [==============================] - 9s 87ms/step - loss: 12.4216 - acc:
0.2293 - val_loss: 12.4060 - val_acc: 0.2303
Epoch 7/10
100/100 [==============================] - 9s 89ms/step - loss: 12.1180 - acc:
0.2482 - val_loss: 12.2627 - val_acc: 0.2392
Epoch 8/10
100/100 [==============================] - 9s 88ms/step - loss: 12.3867 - acc:
0.2315 - val_loss: 12.2465 - val_acc: 0.2402
Epoch 9/10
100/100 [==============================] - 9s 88ms/step - loss: 12.2767 - acc:
0.2383 - val_loss: 12.2951 - val_acc: 0.2372
Epoch 10/10
100/100 [==============================] - 9s 88ms/step - loss: 12.3599 - acc:
0.2332 - val_loss: 12.3275 - val_acc: 0.2352
```

In [14]:

```
#plot for the question2 (a):
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()
```





Question2_answers:

- loss function:change the binary_crossentropy to categorical_crossentropy,since now we have 4 classes

(a)

plots for question2(a) are showing below(with learning rate=0.1),the volatility of the data points is very large,and there is no convergence.

The main reasons are:

(1)the learning rate is too large, so the training process is overshooting;

(2)the activation functiong is sigmoid,since the output of sigmoid function is not zero-centered,we get zig-zag patterns for gradient descent.
image

image

(b)

2 parameters:

according to our analysis above,we should change the learning rate as well as activation function.After lower the learning rate and change the activation function from 'sigmoid' to 'tanh'(I have also tried 'relu',but the 'tanh' performs better in terms of accuracy),the accuracy increased.

the imapact of learning rate:

if the rate is too small,then there has to be a lot of iterations until convergence,and the training process maybe trapped in local minimum;if the rate is too large,the overshooting problem may appears and there maybe no convergence.

the imapact of activation function:

sigmoid function may cause vanishing gradient as well as zig-zag problem.

I have also plot the the training/validation accuracy and training/validation loss as a function of the epochs after tuning the 2 parameters (see below)

In [49]:
```
#change the learning rate as well as the activation function for question2(b)
from tensorflow import keras
from keras import layers
from keras import models
from keras import optimizers

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),padding='same',activation='tanh',#add the pa
dding
                        input_shape=(32, 32, 3))) #since we rescale the input p
ictures to 32*32,the input_shape shoude be (32,32,3)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),padding='same', activation='tanh')) #add the
padding
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), padding='same',activation='tanh')) #add th
e padding
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3),padding='same',activation='tanh')) #add the
padding
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512,activation='tanh'))
model.add(layers.Dense(4, activation='softmax')) # k=4

model.compile(loss='categorical_crossentropy', #change the binary_crossentropy
 to categorical_crossentropy,since now we have 4 classes
            optimizer=optimizers.RMSprop(lr=0.9e-3),
            metrics=['acc'])
```

In [50]:
```
# train_dir="../sample1000/train"
# validation_dir="../sample1000/val"
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255) #rescale the tensor values t
o [0,1]
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(32, 32),
        batch_size=20,
        class_mode='categorical') #we only have two classes

validation_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(32, 32),
        batch_size=20,
        class_mode='categorical')
```

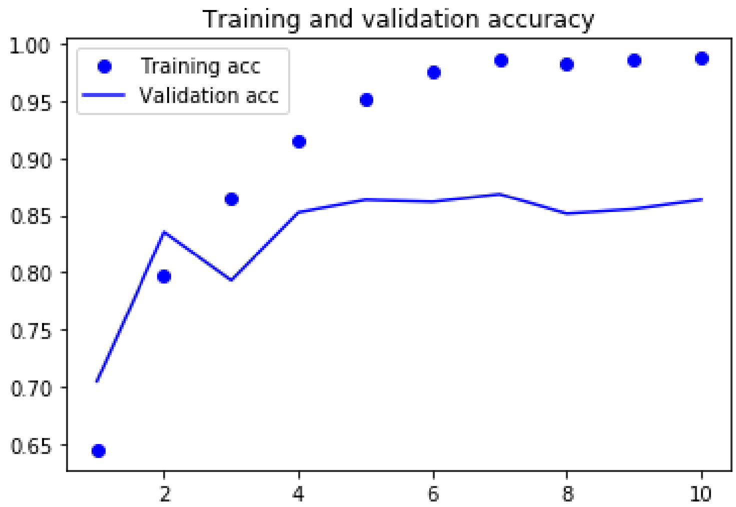
```
Found 1675 images belonging to 4 classes.
Found 835 images belonging to 4 classes.
```

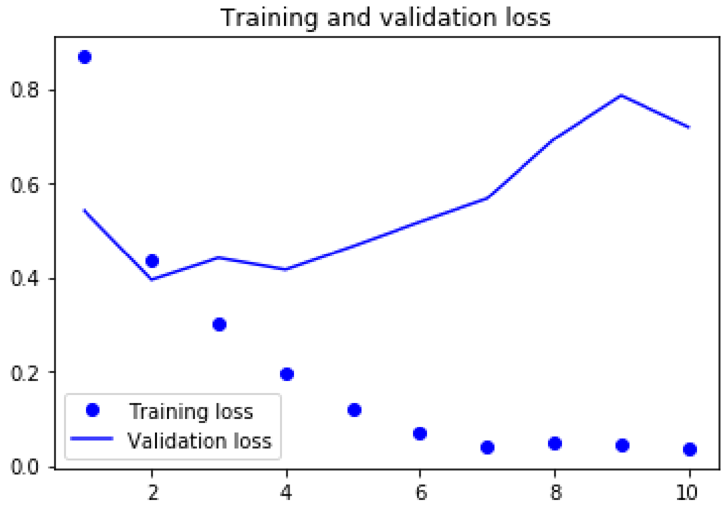
In [51]:
```
print ("start..")
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=50)
```

```
start..
Epoch 1/10
100/100 [==============================] - 12s 124ms/step - loss: 0.8671 - ac
c: 0.6455 - val_loss: 0.5409 - val_acc: 0.7045
Epoch 2/10
100/100 [==============================] - 10s 99ms/step - loss: 0.4375 - acc:
0.7963 - val_loss: 0.3945 - val_acc: 0.8352
Epoch 3/10
100/100 [==============================] - 9s 93ms/step - loss: 0.3023 - acc:
0.8643 - val_loss: 0.4411 - val_acc: 0.7930
Epoch 4/10
100/100 [==============================] - 9s 91ms/step - loss: 0.1948 - acc:
0.9160 - val_loss: 0.4161 - val_acc: 0.8523
Epoch 5/10
100/100 [==============================] - 9s 92ms/step - loss: 0.1190 - acc:
0.9513 - val_loss: 0.4654 - val_acc: 0.8633
Epoch 6/10
100/100 [==============================] - 9s 91ms/step - loss: 0.0689 - acc:
0.9760 - val_loss: 0.5172 - val_acc: 0.8616
Epoch 7/10
100/100 [==============================] - 9s 93ms/step - loss: 0.0402 - acc:
0.9868 - val_loss: 0.5667 - val_acc: 0.8683
Epoch 8/10
100/100 [==============================] - 9s 92ms/step - loss: 0.0466 - acc:
0.9830 - val_loss: 0.6931 - val_acc: 0.8513
Epoch 9/10
100/100 [==============================] - 9s 92ms/step - loss: 0.0449 - acc:
0.9855 - val_loss: 0.7855 - val_acc: 0.8553
Epoch 10/10
100/100 [==============================] - 9s 93ms/step - loss: 0.0365 - acc:
0.9875 - val_loss: 0.7190 - val_acc: 0.8633
```

In [52]:
```
model.save('cat_dog_car_bike.h5')
```

In [53]:
```python
#plot for the question2 (b):
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()
```





(I also plot both the training/validation accuracy and training/validation loss as a function of the epochs after changing these 2 parameters,and the pictures are shown below:)
image

image

End of the question 2

Question3(Optimize the learning rate)

a)

(a) try to find a wide enough learning rate range,recording the val_acc after 10 epoches:

a wide enough range can be:(1e-7,1.5e-3)

- 1e-7:0.2613 #lower bound
- 1e-6:0.5729
- 1e-5:0.6925
- 1e-4:0.8312
- 1e-3:0.8543 #The promotion is not obvious, and the accuracy has fallen during some iterations (the increase of accuracy is not stable). Overshotting may have occurred.
- 1e-2:0.2302 #Apparent overshooting has appeared, an upper bound of learning rate can be exist between 1e-3 and 1e-2
- 5e-3:0.2141
- 2.5e-3:0.2945
- 1.5e-3： 0.2824 #using 1.5e-3 as the upper bound of the learning rate

try to find the optimal learning rate:the optimal learning rate can be around 9.5e-4 (0.95e-3).

- 5e-4: 0.8291
- 7.5e-4： 0.8482
- 8.5e-4： 0.8412
- 9.5e-4:0.8523
- 9.75e-4: 0.8402
- 9.95e-4: 0.8020
- 1.05e-3： 0.8352

b)

(b) Provide an example of a learning rate which is “bad”, “good” and “very good”. Motivate your answer.

- bad: When the learning rate is 1e-7, the accuracies on the training set and the validation set are less than 0.3, because the learning rate is too low at that time, and it is not enough to learn a useful pattern within 10 epochs;

- good： When the learning rate is 4.5e-5, the acc of the validation set rises from the initial 0.6+ to 0.7799. However the highest accuracy is not achieved within 10 epochs,and the acc of first epoch is quite low,so this learning rate maybe a bit low;

- very good： When the learning rate is 0.95e-3, the accuracy of the validation set rises from the initial 0.7497 to 0.8523, and the classification accuracy of the training set increases from 0.6+ to 0.9+.

Question4 Transfer Learning

In [54]:
```
from keras.applications import VGG16
conv_base = VGG16(weights='imagenet',include_top=False,input_shape=(32, 32, 3))
#remove the top layer, VGG was trained for 1000 classes, here we only have four
conv_base.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Feature Extraction (with Data Augmentation)

We add other layers on top of conv_base

In [55]:
```
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(4, activation='sigmoid'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 1, 1, 512)         14714688  
_________________________________________________________________
flatten_35 (Flatten)         (None, 512)               0         
_________________________________________________________________
dense_69 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_70 (Dense)             (None, 4)                 1028      
=================================================================
Total params: 14,847,044
Trainable params: 14,847,044
Non-trainable params: 0
_________________________________________________________________
```

In [56]:
```
#this "freezes" the VGGNet
conv_base.trainable = False
model.summary()
```

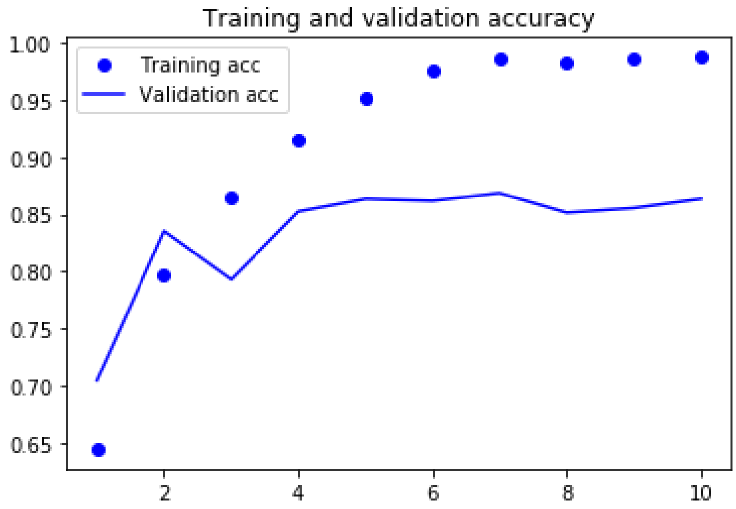
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vgg16 (Model)                (None, 1, 1, 512)         14714688  
_________________________________________________________________
flatten_35 (Flatten)         (None, 512)               0         
_________________________________________________________________
dense_69 (Dense)             (None, 256)               131328    
_________________________________________________________________
dense_70 (Dense)             (None, 4)                 1028      
=================================================================
Total params: 14,847,044
Trainable params: 132,356
Non-trainable params: 14,714,688
_________________________________________________________________
```

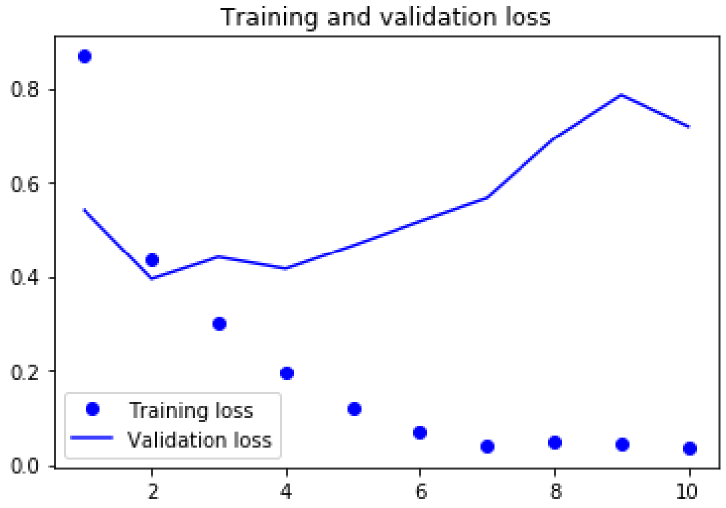
In [44]:
```
import os
base_dir = './cat_dog_car_bike'
train_dir = './cat_dog_car_bike/train'
val_dir = './cat_dog_car_bike/val'
test_dir = './cat_dog_car_bike/test'

train_cats_dir=os.path.join(train_dir,'c0')
train_dogs_dir=os.path.join(train_dir,'c1')
train_cars_dir=os.path.join(train_dir,'c2')
train_motorbikes_dir=os.path.join(train_dir,'c3')
test_cats_dir=os.path.join(test_dir,'c0')
test_dogs_dir=os.path.join(test_dir,'c1')
test_cars_dir=os.path.join(test_dir,'c2')
test_motorbikes_dir=os.path.join(test_dir,'c3')
val_cats_dir=os.path.join(val_dir,'c0')
val_dogs_dir=os.path.join(val_dir,'c1')
val_cars_dir=os.path.join(val_dir,'c2')
val_motorbikes_dir=os.path.join(val_dir,'c3')

from tensorflow import keras
from keras import layers
from keras import models
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=(32, 32),
    batch_size=20,
    class_mode='categorical')
validation_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(32, 32),
        batch_size=20,
        class_mode='categorical')
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-3),
              metrics=['acc'])
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=20,
      validation_data=validation_generator,
      validation_steps=50)
```

```
Found 1675 images belonging to 4 classes.
Found 835 images belonging to 4 classes.
Epoch 1/20
100/100 [==============================] - 12s 119ms/step - loss: 11.3538 - ac
c: 0.2728 - val_loss: 1.2838 - val_acc: 0.5960
Epoch 2/20
100/100 [==============================] - 9s 94ms/step - loss: 0.8405 - acc:
0.6735 - val_loss: 3.3989 - val_acc: 0.3015
Epoch 3/20
100/100 [==============================] - 9s 93ms/step - loss: 0.7152 - acc:
0.7110 - val_loss: 0.7520 - val_acc: 0.7296
Epoch 4/20
100/100 [==============================] - 9s 91ms/step - loss: 0.6315 - acc:
0.7440 - val_loss: 1.1743 - val_acc: 0.6442
Epoch 5/20
100/100 [==============================] - 9s 92ms/step - loss: 0.5723 - acc:
0.7788 - val_loss: 0.9154 - val_acc: 0.6985
Epoch 6/20
100/100 [==============================] - 9s 91ms/step - loss: 0.5721 - acc:
0.7627 - val_loss: 1.4446 - val_acc: 0.6020
Epoch 7/20
100/100 [==============================] - 9s 90ms/step - loss: 0.5279 - acc:
0.7970 - val_loss: 1.0888 - val_acc: 0.6472
Epoch 8/20
100/100 [==============================] - 9s 91ms/step - loss: 0.5080 - acc:
0.7910 - val_loss: 0.5246 - val_acc: 0.8090
Epoch 9/20
100/100 [==============================] - 9s 94ms/step - loss: 0.5165 - acc:
0.7933 - val_loss: 1.0001 - val_acc: 0.7015
Epoch 10/20
100/100 [==============================] - 9s 95ms/step - loss: 0.4994 - acc:
0.7927 - val_loss: 0.5438 - val_acc: 0.8020
Epoch 11/20
100/100 [==============================] - 10s 98ms/step - loss: 0.4808 - acc:
0.7987 - val_loss: 1.0419 - val_acc: 0.6990
Epoch 12/20
100/100 [==============================] - 9s 93ms/step - loss: 0.4615 - acc:
0.8120 - val_loss: 0.6963 - val_acc: 0.7698
Epoch 13/20
100/100 [==============================] - 9s 90ms/step - loss: 0.4680 - acc:
0.8118 - val_loss: 0.4302 - val_acc: 0.8271
Epoch 14/20
100/100 [==============================] - 9s 89ms/step - loss: 0.4551 - acc:
0.8233 - val_loss: 0.4198 - val_acc: 0.8372
Epoch 15/20
100/100 [==============================] - 9s 88ms/step - loss: 0.4571 - acc:
0.8235 - val_loss: 0.5001 - val_acc: 0.8281
Epoch 16/20
100/100 [==============================] - 9s 90ms/step - loss: 0.4644 - acc:
0.8185 - val_loss: 0.6735 - val_acc: 0.7606
Epoch 17/20
100/100 [==============================] - 9s 89ms/step - loss: 0.4205 - acc:
0.8192 - val_loss: 1.7540 - val_acc: 0.6533
Epoch 18/20
100/100 [==============================] - 9s 90ms/step - loss: 0.4333 - acc:
0.8300 - val_loss: 0.9663 - val_acc: 0.7497
Epoch 19/20
100/100 [==============================] - 9s 88ms/step - loss: 0.3971 - acc:
0.8377 - val_loss: 0.9512 - val_acc: 0.7487
Epoch 20/20
100/100 [==============================] - 9s 90ms/step - loss: 0.4303 - acc:
0.8352 - val_loss: 1.0471 - val_acc: 0.7327
```

In [57]:

```
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()
```





plots

accFeat.png

comments on the plot

Above we used the pretrained network VGG16, which is a network trained on ImageNet. For deep learning, if the training data set is large enough and versatile, the spatial hierarchy of the features learned by the pretrained network can be effectively used as a general model of the visual world.

The CNN model consists of two parts, one part is the convolutional base and the other is the dense layers. For CNN, feature extraction is to take out the convolutional base of the pretrained network, run new data on it, and then train a new classifier on the output.

Before using the VGG16 convolutional base, our model was trained on a very small training set, so there was a serious overfitting problem, that is, although the accuracy on the training set increased with the increase of number of the epochs, the accuracy on the evaluation set was decreasing.

This is because the model has begun to learn some of the properties specific to the training set image rather than the features common to all images.

The best way to solve overfitting is to increase the amount of data in the training set. The more training data, the better the generalization ability of the model. So we used data augment to create more images.

In addition,using the pretrained network which is trained on a large enought data set could also solve the problem of overfitting.

As can be seen from the above figure, although the accuracy of the training set is not improved, the accuracy of the validation is not high, but the gap between the two is very small, indicating that the overfitting problem is well solved. Next we try to improve the classification of accuracy by fine-tuning.

Question 5

I tried 2 methods to increase the accuracy.

The first method is fine-tuning, using VGG16 as feature extractor;

The second one is using drop out, regularization to solve the problem of overfitting, and then fine-tune the learning rate as well as change the optimizer; I have also considered that the input image maybe not centered on zero, we need to rescale the image to make it zero centered.

method1:Fine Tuning(also using the VGG16 as feature extrator)

Fine tuning complements the above feature extraction (using VGG16).

Fine tuning is to thaw the top layers of the convolutional base of the VGG16 and combine the thawed layers with the newly added fully connected dense layers.

The thawed layers participate in random initialization with the dense layers and then participate in the training process,making the resulting model more relevant to the data set being trained.

Before we do fine-tuning, check the structure of the current convolutional base：

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

The CNN structure is shown above.

We try to fine-tune the convolutional layer of block5, because the near-bottom layers encodes more general reusable features (such as arcs or lines) and the layers near the top encode features more specialized to the training dataset.

We should fine-tune these less general-purpose layers based on the specific training data set to make the model more suitable for the data set we want to classify.

I have tried to run 100 epochs to see the trend,and to save time,I also provided a 10-round version below, it seems that the 10 epochs version is as effective as 100 rounds.

In [28]:
```
#I have tried to run 100 epochs to see the trend,and to save time,I also offer
 a 10 epochs version in below:
#100 epochs version
#we freeze all layers before block5_conv1
conv_base.trainable = True
set_trainable = False## Fine Tuning
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model.compile(loss='binary_crossentropy',
                      optimizer=optimizers.RMSprop(lr=1e-5),
                      metrics=['acc'])#using a low eta to avoid breaking the st
ructure
history = model.fit_generator(train_generator,steps_per_epoch=100,epochs=100,va
lidation_data=validation_generator,validation_steps=50)
```

```
Epoch 1/100
100/100 [==============================] - 110s 1s/step - loss: 0.2956 - acc:
0.8664 - val_loss: 0.3271 - val_acc: 0.8651
Epoch 2/100
100/100 [==============================] - 109s 1s/step - loss: 0.2485 - acc:
0.8880 - val_loss: 0.3031 - val_acc: 0.8746
Epoch 3/100
100/100 [==============================] - 109s 1s/step - loss: 0.2494 - acc:
0.8872 - val_loss: 0.2717 - val_acc: 0.8852
Epoch 4/100
100/100 [==============================] - 108s 1s/step - loss: 0.2362 - acc:
0.8953 - val_loss: 0.3033 - val_acc: 0.8746
Epoch 5/100
100/100 [==============================] - 108s 1s/step - loss: 0.2288 - acc:
0.8969 - val_loss: 0.2797 - val_acc: 0.8894
Epoch 6/100
100/100 [==============================] - 108s 1s/step - loss: 0.2083 - acc:
0.9062 - val_loss: 0.2690 - val_acc: 0.8925
Epoch 7/100
100/100 [==============================] - 117s 1s/step - loss: 0.2059 - acc:
0.9086 - val_loss: 0.2706 - val_acc: 0.8899
Epoch 8/100
100/100 [==============================] - 111s 1s/step - loss: 0.2122 - acc:
0.9101 - val_loss: 0.3266 - val_acc: 0.8668
Epoch 9/100
100/100 [==============================] - 111s 1s/step - loss: 0.1884 - acc:
0.9191 - val_loss: 0.3261 - val_acc: 0.8769
Epoch 10/100
100/100 [==============================] - 114s 1s/step - loss: 0.1978 - acc:
0.9165 - val_loss: 0.2797 - val_acc: 0.8925
Epoch 11/100
100/100 [==============================] - 111s 1s/step - loss: 0.1876 - acc:
0.9185 - val_loss: 0.2699 - val_acc: 0.8942
Epoch 12/100
100/100 [==============================] - 111s 1s/step - loss: 0.1824 - acc:
0.9189 - val_loss: 0.2138 - val_acc: 0.9136
Epoch 13/100
100/100 [==============================] - 112s 1s/step - loss: 0.1829 - acc:
0.9222 - val_loss: 0.2174 - val_acc: 0.9168
Epoch 14/100
100/100 [==============================] - 112s 1s/step - loss: 0.1683 - acc:
0.9267 - val_loss: 0.2904 - val_acc: 0.8935
Epoch 15/100
100/100 [==============================] - 112s 1s/step - loss: 0.1792 - acc:
0.9224 - val_loss: 0.2158 - val_acc: 0.9103
Epoch 16/100
100/100 [==============================] - 112s 1s/step - loss: 0.1697 - acc:
0.9255 - val_loss: 0.2479 - val_acc: 0.9008
Epoch 17/100
100/100 [==============================] - 111s 1s/step - loss: 0.1652 - acc:
0.9303 - val_loss: 0.2521 - val_acc: 0.9035
Epoch 18/100
100/100 [==============================] - 110s 1s/step - loss: 0.1638 - acc:
0.9298 - val_loss: 0.2867 - val_acc: 0.8975
Epoch 19/100
100/100 [==============================] - 110s 1s/step - loss: 0.1624 - acc:
0.9300 - val_loss: 0.2787 - val_acc: 0.9010
Epoch 20/100
100/100 [==============================] - 110s 1s/step - loss: 0.1600 - acc:
0.9324 - val_loss: 0.2278 - val_acc: 0.9053
Epoch 21/100
100/100 [==============================] - 110s 1s/step - loss: 0.1623 - acc:
0.9312 - val_loss: 0.2550 - val_acc: 0.9025
Epoch 22/100
100/100 [==============================] - 110s 1s/step - loss: 0.1510 - acc:
0.9371 - val_loss: 0.2482 - val_acc: 0.9083
```

```
Epoch 23/100
100/100 [==============================] - 144s 1s/step - loss: 0.1578 - acc: 0.9337 - val_loss: 0.2820 - val_acc: 0.9038
Epoch 24/100
100/100 [==============================] - 145s 1s/step - loss: 0.1592 - acc: 0.9324 - val_loss: 0.3305 - val_acc: 0.8877
Epoch 25/100
100/100 [==============================] - 111s 1s/step - loss: 0.1504 - acc: 0.9387 - val_loss: 0.2922 - val_acc: 0.8942
Epoch 26/100
100/100 [==============================] - 107s 1s/step - loss: 0.1462 - acc: 0.9398 - val_loss: 0.2913 - val_acc: 0.8982
Epoch 27/100
100/100 [==============================] - 107s 1s/step - loss: 0.1478 - acc: 0.9370 - val_loss: 0.2132 - val_acc: 0.9201
Epoch 28/100
100/100 [==============================] - 107s 1s/step - loss: 0.1513 - acc: 0.9372 - val_loss: 0.2791 - val_acc: 0.8995
Epoch 29/100
100/100 [==============================] - 107s 1s/step - loss: 0.1368 - acc: 0.9429 - val_loss: 0.2720 - val_acc: 0.8950
Epoch 30/100
100/100 [==============================] - 107s 1s/step - loss: 0.1466 - acc: 0.9382 - val_loss: 0.2828 - val_acc: 0.9033
Epoch 31/100
100/100 [==============================] - 107s 1s/step - loss: 0.1299 - acc: 0.9476 - val_loss: 0.3042 - val_acc: 0.8970
Epoch 32/100
100/100 [==============================] - 107s 1s/step - loss: 0.1468 - acc: 0.9371 - val_loss: 0.2522 - val_acc: 0.9051
Epoch 33/100
100/100 [==============================] - 107s 1s/step - loss: 0.1279 - acc: 0.9479 - val_loss: 0.2534 - val_acc: 0.9063
Epoch 34/100
100/100 [==============================] - 110s 1s/step - loss: 0.1291 - acc: 0.9470 - val_loss: 0.2282 - val_acc: 0.9148
Epoch 35/100
100/100 [==============================] - 110s 1s/step - loss: 0.1275 - acc: 0.9458 - val_loss: 0.2421 - val_acc: 0.9156
Epoch 36/100
100/100 [==============================] - 110s 1s/step - loss: 0.1402 - acc: 0.9425 - val_loss: 0.2931 - val_acc: 0.9028
Epoch 37/100
100/100 [==============================] - 110s 1s/step - loss: 0.1257 - acc: 0.9483 - val_loss: 0.2375 - val_acc: 0.9162
Epoch 38/100
100/100 [==============================] - 110s 1s/step - loss: 0.1316 - acc: 0.9434 - val_loss: 0.2149 - val_acc: 0.9206
Epoch 39/100
100/100 [==============================] - 111s 1s/step - loss: 0.1299 - acc: 0.9463 - val_loss: 0.2748 - val_acc: 0.9038
Epoch 40/100
100/100 [==============================] - 112s 1s/step - loss: 0.1237 - acc: 0.9495 - val_loss: 0.2908 - val_acc: 0.9078
Epoch 41/100
100/100 [==============================] - 112s 1s/step - loss: 0.1295 - acc: 0.9457 - val_loss: 0.2678 - val_acc: 0.9040
Epoch 42/100
100/100 [==============================] - 112s 1s/step - loss: 0.1229 - acc: 0.9527 - val_loss: 0.3264 - val_acc: 0.8939
Epoch 43/100
100/100 [==============================] - 111s 1s/step - loss: 0.1240 - acc: 0.9475 - val_loss: 0.3255 - val_acc: 0.8955
Epoch 44/100
100/100 [==============================] - 111s 1s/step - loss: 0.1242 - acc: 0.9470 - val_loss: 0.2535 - val_acc: 0.9168
```

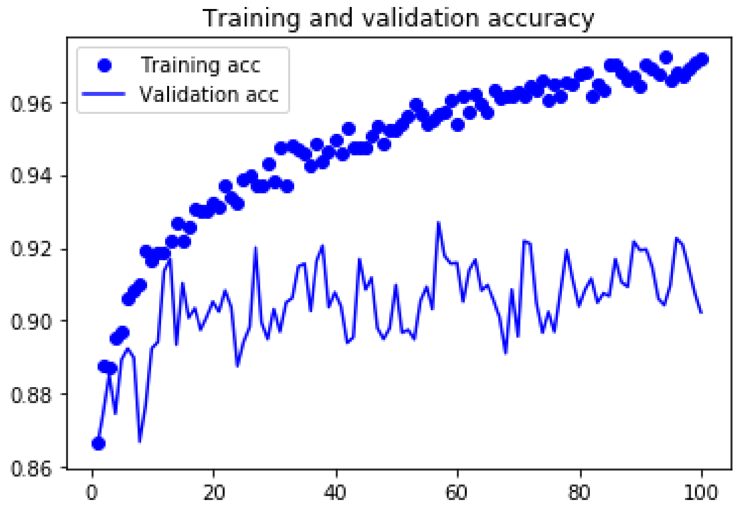
```
Epoch 45/100
100/100 [==============================] - 113s 1s/step - loss: 0.1224 - acc:
0.9475 - val_loss: 0.2544 - val_acc: 0.9085
Epoch 46/100
100/100 [==============================] - 113s 1s/step - loss: 0.1202 - acc:
0.9508 - val_loss: 0.2698 - val_acc: 0.9118
Epoch 47/100
100/100 [==============================] - 113s 1s/step - loss: 0.1145 - acc:
0.9532 - val_loss: 0.2992 - val_acc: 0.8980
Epoch 48/100
100/100 [==============================] - 109s 1s/step - loss: 0.1216 - acc:
0.9482 - val_loss: 0.3085 - val_acc: 0.8950
Epoch 49/100
100/100 [==============================] - 109s 1s/step - loss: 0.1146 - acc:
0.9520 - val_loss: 0.3318 - val_acc: 0.8980
Epoch 50/100
100/100 [==============================] - 109s 1s/step - loss: 0.1189 - acc:
0.9526 - val_loss: 0.3045 - val_acc: 0.9098
Epoch 51/100
100/100 [==============================] - 111s 1s/step - loss: 0.1150 - acc:
0.9542 - val_loss: 0.3101 - val_acc: 0.8967
Epoch 52/100
100/100 [==============================] - 111s 1s/step - loss: 0.1078 - acc:
0.9561 - val_loss: 0.3206 - val_acc: 0.8975
Epoch 53/100
100/100 [==============================] - 111s 1s/step - loss: 0.1061 - acc:
0.9590 - val_loss: 0.3166 - val_acc: 0.8949
Epoch 54/100
100/100 [==============================] - 110s 1s/step - loss: 0.1062 - acc:
0.9565 - val_loss: 0.2805 - val_acc: 0.9055
Epoch 55/100
100/100 [==============================] - 111s 1s/step - loss: 0.1092 - acc:
0.9538 - val_loss: 0.2703 - val_acc: 0.9093
Epoch 56/100
100/100 [==============================] - 112s 1s/step - loss: 0.1092 - acc:
0.9548 - val_loss: 0.2929 - val_acc: 0.9033
Epoch 57/100
100/100 [==============================] - 110s 1s/step - loss: 0.1089 - acc:
0.9567 - val_loss: 0.1980 - val_acc: 0.9269
Epoch 58/100
100/100 [==============================] - 111s 1s/step - loss: 0.1050 - acc:
0.9574 - val_loss: 0.2392 - val_acc: 0.9177
Epoch 59/100
100/100 [==============================] - 111s 1s/step - loss: 0.0965 - acc:
0.9604 - val_loss: 0.2508 - val_acc: 0.9156
Epoch 60/100
100/100 [==============================] - 110s 1s/step - loss: 0.1086 - acc:
0.9542 - val_loss: 0.2698 - val_acc: 0.9158
Epoch 61/100
100/100 [==============================] - 112s 1s/step - loss: 0.0939 - acc:
0.9617 - val_loss: 0.2991 - val_acc: 0.9053
Epoch 62/100
100/100 [==============================] - 110s 1s/step - loss: 0.1041 - acc:
0.9572 - val_loss: 0.2432 - val_acc: 0.9138
Epoch 63/100
100/100 [==============================] - 111s 1s/step - loss: 0.0977 - acc:
0.9619 - val_loss: 0.2317 - val_acc: 0.9167
Epoch 64/100
100/100 [==============================] - 112s 1s/step - loss: 0.1041 - acc:
0.9595 - val_loss: 0.2800 - val_acc: 0.9083
Epoch 65/100
100/100 [==============================] - 113s 1s/step - loss: 0.1060 - acc:
0.9569 - val_loss: 0.2729 - val_acc: 0.9098
Epoch 66/100
100/100 [==============================] - 112s 1s/step - loss: 0.0953 - acc:
0.9632 - val_loss: 0.2898 - val_acc: 0.9058
```

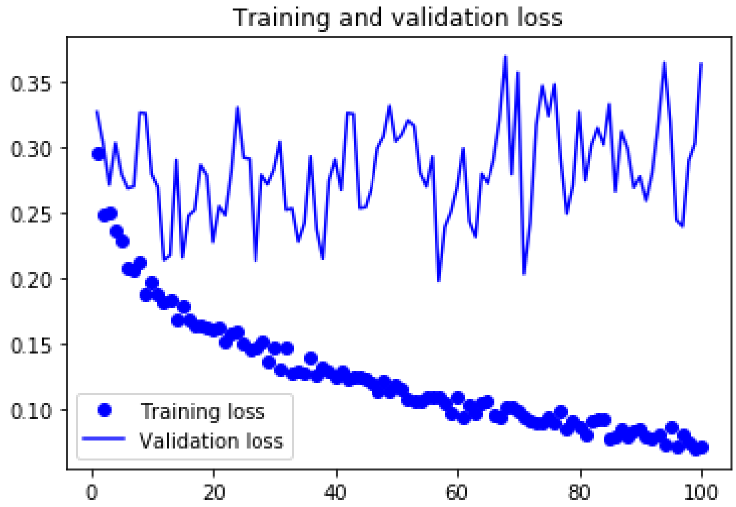
```
Epoch 67/100
100/100 [==============================] - 112s 1s/step - loss: 0.0947 - acc:
0.9610 - val_loss: 0.3210 - val_acc: 0.9010
Epoch 68/100
100/100 [==============================] - 110s 1s/step - loss: 0.1014 - acc:
0.9614 - val_loss: 0.3695 - val_acc: 0.8912
Epoch 69/100
100/100 [==============================] - 109s 1s/step - loss: 0.1011 - acc:
0.9616 - val_loss: 0.2796 - val_acc: 0.9085
Epoch 70/100
100/100 [==============================] - 110s 1s/step - loss: 0.0986 - acc:
0.9624 - val_loss: 0.3574 - val_acc: 0.8957
Epoch 71/100
100/100 [==============================] - 111s 1s/step - loss: 0.0948 - acc:
0.9615 - val_loss: 0.2033 - val_acc: 0.9219
Epoch 72/100
100/100 [==============================] - 110s 1s/step - loss: 0.0914 - acc:
0.9642 - val_loss: 0.2424 - val_acc: 0.9211
Epoch 73/100
100/100 [==============================] - 110s 1s/step - loss: 0.0897 - acc:
0.9631 - val_loss: 0.3173 - val_acc: 0.9050
Epoch 74/100
100/100 [==============================] - 111s 1s/step - loss: 0.0891 - acc:
0.9660 - val_loss: 0.3470 - val_acc: 0.8967
Epoch 75/100
100/100 [==============================] - 110s 1s/step - loss: 0.0939 - acc:
0.9602 - val_loss: 0.3241 - val_acc: 0.9025
Epoch 76/100
100/100 [==============================] - 109s 1s/step - loss: 0.0892 - acc:
0.9648 - val_loss: 0.3481 - val_acc: 0.8970
Epoch 77/100
100/100 [==============================] - 110s 1s/step - loss: 0.0984 - acc:
0.9615 - val_loss: 0.2914 - val_acc: 0.9080
Epoch 78/100
100/100 [==============================] - 110s 1s/step - loss: 0.0856 - acc:
0.9654 - val_loss: 0.2494 - val_acc: 0.9193
Epoch 79/100
100/100 [==============================] - 111s 1s/step - loss: 0.0912 - acc:
0.9648 - val_loss: 0.2712 - val_acc: 0.9111
Epoch 80/100
100/100 [==============================] - 111s 1s/step - loss: 0.0869 - acc:
0.9672 - val_loss: 0.3274 - val_acc: 0.9040
Epoch 81/100
100/100 [==============================] - 112s 1s/step - loss: 0.0801 - acc:
0.9680 - val_loss: 0.2751 - val_acc: 0.9085
Epoch 82/100
100/100 [==============================] - 110s 1s/step - loss: 0.0911 - acc:
0.9613 - val_loss: 0.3017 - val_acc: 0.9116
Epoch 83/100
100/100 [==============================] - 110s 1s/step - loss: 0.0926 - acc:
0.9645 - val_loss: 0.3150 - val_acc: 0.9050
Epoch 84/100
100/100 [==============================] - 110s 1s/step - loss: 0.0922 - acc:
0.9631 - val_loss: 0.3020 - val_acc: 0.9076
Epoch 85/100
100/100 [==============================] - 110s 1s/step - loss: 0.0779 - acc:
0.9700 - val_loss: 0.3331 - val_acc: 0.9068
Epoch 86/100
100/100 [==============================] - 110s 1s/step - loss: 0.0786 - acc:
0.9699 - val_loss: 0.2665 - val_acc: 0.9168
Epoch 87/100
100/100 [==============================] - 110s 1s/step - loss: 0.0844 - acc:
0.9680 - val_loss: 0.3123 - val_acc: 0.9106
Epoch 88/100
100/100 [==============================] - 110s 1s/step - loss: 0.0785 - acc:
0.9661 - val_loss: 0.2987 - val_acc: 0.9093
```

```
Epoch 89/100
100/100 [==============================] - 110s 1s/step - loss: 0.0843 - acc:
0.9670 - val_loss: 0.2696 - val_acc: 0.9217
Epoch 90/100
100/100 [==============================] - 110s 1s/step - loss: 0.0847 - acc:
0.9643 - val_loss: 0.2780 - val_acc: 0.9193
Epoch 91/100
100/100 [==============================] - 110s 1s/step - loss: 0.0796 - acc:
0.9702 - val_loss: 0.2593 - val_acc: 0.9196
Epoch 92/100
100/100 [==============================] - 109s 1s/step - loss: 0.0780 - acc:
0.9689 - val_loss: 0.2807 - val_acc: 0.9146
Epoch 93/100
100/100 [==============================] - 110s 1s/step - loss: 0.0805 - acc:
0.9676 - val_loss: 0.3191 - val_acc: 0.9060
Epoch 94/100
100/100 [==============================] - 110s 1s/step - loss: 0.0734 - acc:
0.9722 - val_loss: 0.3649 - val_acc: 0.9043
Epoch 95/100
100/100 [==============================] - 111s 1s/step - loss: 0.0871 - acc:
0.9661 - val_loss: 0.3199 - val_acc: 0.9096
Epoch 96/100
100/100 [==============================] - 111s 1s/step - loss: 0.0719 - acc:
0.9681 - val_loss: 0.2444 - val_acc: 0.9226
Epoch 97/100
100/100 [==============================] - 110s 1s/step - loss: 0.0804 - acc:
0.9667 - val_loss: 0.2398 - val_acc: 0.9209
Epoch 98/100
100/100 [==============================] - 111s 1s/step - loss: 0.0746 - acc:
0.9692 - val_loss: 0.2899 - val_acc: 0.9143
Epoch 99/100
100/100 [==============================] - 110s 1s/step - loss: 0.0699 - acc:
0.9705 - val_loss: 0.3030 - val_acc: 0.9075
Epoch 100/100
100/100 [==============================] - 111s 1s/step - loss: 0.0712 - acc:
0.9720 - val_loss: 0.3638 - val_acc: 0.9023
```

In [29]:

```python
#plot 100 epochs
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()
```





In [58]:
```
#change number of epoch to 10 to save time
conv_base.trainable = True
set_trainable = False## Fine Tuning
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model.compile(loss='binary_crossentropy',
                      optimizer=optimizers.RMSprop(lr=1e-5),
                      metrics=['acc'])#using a low eta to avoid breaking the st
ructure
history = model.fit_generator(train_generator,steps_per_epoch=100,epochs=10,val
idation_data=validation_generator,validation_steps=50)
```

```
Epoch 1/10
100/100 [==============================] - 112s 1s/step - loss: 0.4708 - acc:
0.7720 - val_loss: 0.3240 - val_acc: 0.8698
Epoch 2/10
100/100 [==============================] - 110s 1s/step - loss: 0.2659 - acc:
0.8967 - val_loss: 0.2393 - val_acc: 0.9008
Epoch 3/10
100/100 [==============================] - 112s 1s/step - loss: 0.2088 - acc:
0.9133 - val_loss: 0.1968 - val_acc: 0.9239
Epoch 4/10
100/100 [==============================] - 116s 1s/step - loss: 0.1648 - acc:
0.9353 - val_loss: 0.1880 - val_acc: 0.9176
Epoch 5/10
100/100 [==============================] - 114s 1s/step - loss: 0.1369 - acc:
0.9462 - val_loss: 0.1676 - val_acc: 0.9279
Epoch 6/10
100/100 [==============================] - 109s 1s/step - loss: 0.1174 - acc:
0.9534 - val_loss: 0.1640 - val_acc: 0.9308
Epoch 7/10
100/100 [==============================] - 109s 1s/step - loss: 0.1023 - acc:
0.9596 - val_loss: 0.1594 - val_acc: 0.9344
Epoch 8/10
100/100 [==============================] - 109s 1s/step - loss: 0.0908 - acc:
0.9641 - val_loss: 0.1569 - val_acc: 0.9389
Epoch 9/10
100/100 [==============================] - 109s 1s/step - loss: 0.0836 - acc:
0.9675 - val_loss: 0.1419 - val_acc: 0.9407
Epoch 10/10
100/100 [==============================] - 110s 1s/step - loss: 0.0695 - acc:
0.9751 - val_loss: 0.1581 - val_acc: 0.9324
```

We plot the results and we got
question5first.png

question5second.png

Finally I got the accuracy of 0.9720 on training set and 0.9023 on validation set.

```
acc: 0.9720 - val_loss: 0.3638 - val_acc: 0.9023
```

- ## Method2
- rescale the image to make it zero centered to avoid zig-zag problems.(compute mean of the data points and subtract the mean to each data point so that the mean of the input data points is zero.)

  using the code:

  train_datagen = ImageDataGenerator(samplewise_center=True,rescale=1./255) test_datagen = ImageDataGenerator(samplewise_center=True,rescale=1./255)

- using drop out， using drop out, regularization to solve the problem of overfitting, and then fine-tune the learning rate as well as change the optimizer；
- However,after adding 0.5 drop out， acc did not increased， the code is：

  model.add(layers.Dropout(0.5))

- Adding l2 regularizer:

  ```
  model.add(layers.Dense(4,kernel_regularizer=regularizers.l2(0.0001),activation='sof
  tmax'))
  ```

- increase number of epochs from 20 to 40,val_acc rised to 0.8482.
- Adding image augment, although the overfitting situation has eased, the acc has not improved.
- change the 'tanh' to 'relu' in order to gain faster convergence.

In [59]:

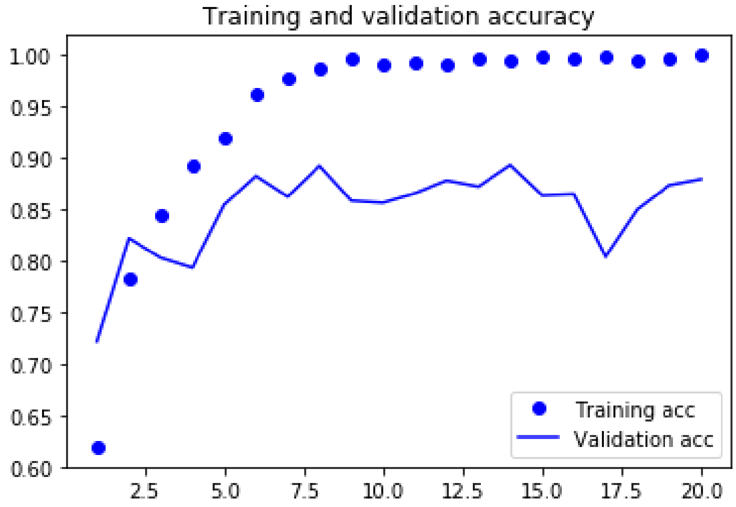
```
#using the CNN trained by myself,adding 0.5 dropout

from tensorflow import keras
from keras import regularizers #import regularizer,adding the regularizer to de
nse layers
from keras import layers
from keras import models
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import os
base_dir = './cat_dog_car_bike'
train_dir = './cat_dog_car_bike/train'
val_dir = './cat_dog_car_bike/val'
test_dir = './cat_dog_car_bike/test'

train_cats_dir=os.path.join(train_dir,'c0')
train_dogs_dir=os.path.join(train_dir,'c1')
train_cars_dir=os.path.join(train_dir,'c2')
train_motorbikes_dir=os.path.join(train_dir,'c3')
test_cats_dir=os.path.join(test_dir,'c0')
test_dogs_dir=os.path.join(test_dir,'c1')
test_cars_dir=os.path.join(test_dir,'c2')
test_motorbikes_dir=os.path.join(test_dir,'c3')
val_cats_dir=os.path.join(val_dir,'c0')
val_dogs_dir=os.path.join(val_dir,'c1')
val_cars_dir=os.path.join(val_dir,'c2')
val_motorbikes_dir=os.path.join(val_dir,'c3')

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),padding='same',activation='relu',#add the pa
dding
                        input_shape=(32, 32, 3))) #since we rescale the input p
ictures to 32*32,the input_shape shoude be (32,32,3)
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3),padding='same', activation='relu')) #add the
padding
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), padding='same',activation='relu')) #add th
e padding
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3),padding='same',activation='relu')) #add the
padding
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(4, kernel_regularizer=regularizers.l2(0.0001),activation
='softmax')) # k=4

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-3),
              metrics=['acc'])



#train_datagen = ImageDataGenerator(rescale=1./255,rotation_range=40,width_shif
t_range=0.2,height_shift_range=0.2,shear_range=0.2,zoom_range=0.2,horizontal_fl
ip=True,)
train_datagen = ImageDataGenerator(samplewise_center=True,rescale=1./255)
test_datagen = ImageDataGenerator(samplewise_center=True,rescale=1./255)


train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(32, 32),
    batch_size=32,
```

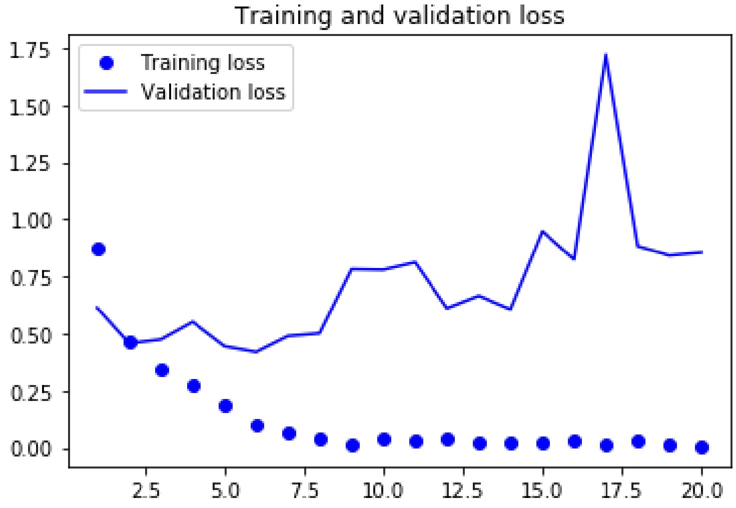
```
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(32, 32),
        batch_size=32,
        class_mode='categorical')

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=20,
      validation_data=validation_generator,
      validation_steps=50)
```

```
Found 1675 images belonging to 4 classes.
Found 835 images belonging to 4 classes.
Epoch 1/20
100/100 [==============================] - 16s 161ms/step - loss: 0.8790 - ac
c: 0.6180 - val_loss: 0.6124 - val_acc: 0.7218
Epoch 2/20
100/100 [==============================] - 13s 134ms/step - loss: 0.4631 - ac
c: 0.7838 - val_loss: 0.4573 - val_acc: 0.8217
Epoch 3/20
100/100 [==============================] - 14s 137ms/step - loss: 0.3458 - ac
c: 0.8451 - val_loss: 0.4742 - val_acc: 0.8029
Epoch 4/20
100/100 [==============================] - 13s 135ms/step - loss: 0.2715 - ac
c: 0.8922 - val_loss: 0.5509 - val_acc: 0.7931
Epoch 5/20
100/100 [==============================] - 13s 134ms/step - loss: 0.1896 - ac
c: 0.9161 - val_loss: 0.4450 - val_acc: 0.8541
Epoch 6/20
100/100 [==============================] - 13s 133ms/step - loss: 0.0986 - ac
c: 0.9625 - val_loss: 0.4197 - val_acc: 0.8820
Epoch 7/20
100/100 [==============================] - 14s 135ms/step - loss: 0.0679 - ac
c: 0.9754 - val_loss: 0.4895 - val_acc: 0.8619
Epoch 8/20
100/100 [==============================] - 13s 135ms/step - loss: 0.0384 - ac
c: 0.9869 - val_loss: 0.5012 - val_acc: 0.8917
Epoch 9/20
100/100 [==============================] - 14s 136ms/step - loss: 0.0170 - ac
c: 0.9950 - val_loss: 0.7821 - val_acc: 0.8580
Epoch 10/20
100/100 [==============================] - 14s 136ms/step - loss: 0.0422 - ac
c: 0.9909 - val_loss: 0.7795 - val_acc: 0.8560
Epoch 11/20
100/100 [==============================] - 14s 135ms/step - loss: 0.0276 - ac
c: 0.9928 - val_loss: 0.8122 - val_acc: 0.8651
Epoch 12/20
100/100 [==============================] - 13s 133ms/step - loss: 0.0359 - ac
c: 0.9909 - val_loss: 0.6104 - val_acc: 0.8774
Epoch 13/20
100/100 [==============================] - 13s 135ms/step - loss: 0.0187 - ac
c: 0.9956 - val_loss: 0.6649 - val_acc: 0.8716
Epoch 14/20
100/100 [==============================] - 14s 135ms/step - loss: 0.0232 - ac
c: 0.9941 - val_loss: 0.6058 - val_acc: 0.8924
Epoch 15/20
100/100 [==============================] - 13s 135ms/step - loss: 0.0175 - ac
c: 0.9975 - val_loss: 0.9478 - val_acc: 0.8632
Epoch 16/20
100/100 [==============================] - 13s 135ms/step - loss: 0.0279 - ac
c: 0.9950 - val_loss: 0.8243 - val_acc: 0.8645
Epoch 17/20
100/100 [==============================] - 13s 135ms/step - loss: 0.0129 - ac
c: 0.9981 - val_loss: 1.7228 - val_acc: 0.8035
Epoch 18/20
100/100 [==============================] - 13s 135ms/step - loss: 0.0338 - ac
c: 0.9947 - val_loss: 0.8823 - val_acc: 0.8495
Epoch 19/20
100/100 [==============================] - 13s 133ms/step - loss: 0.0175 - ac
c: 0.9944 - val_loss: 0.8420 - val_acc: 0.8729
Epoch 20/20
100/100 [==============================] - 13s 134ms/step - loss: 0.0052 - ac
c: 0.9991 - val_loss: 0.8555 - val_acc: 0.8787
```

In [60]:

```
#plot the result
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()
```





In [ ]: