Exercise Activity Classification using Convolutional Neural Networks

1st Jinsol Kim
Yonsei University
Seoul, South Korea
jinsolkim@yonsei.ac.kr

***Abstract*— Many people use the fitness app on their smartphones to exercise and enjoy their own exercise life or take care of their health. These fitness apps measure the amount of exercise a user performs according to the set value when the user sets up the desired exercise. In this study, we would like to conduct an initial study to develop a system in which the fitness app can automatically record the amount of exercise that users are currently doing without having to set what kind of exercise they are going to do. For this purpose, images of four sports motions are collected to make a system that learns about human motion movements and posture using CNN (Convolutional Neural Network). Finally, experiments with test motion images confirm the utility and validity of the system proposed in this paper.**

Keywords—Human Activity Classification, CNN, fitness app

I. INTRODUCTION

Along with the development of advanced information and communication technology, the development and spread of smart devices such as smartphones and tablets are emerging at a rapid pace. Along with this, various mobile apps have emerged, and much of everyday life, such as using public transportation, banking, shopping, and exercising, is being carried out through mobile apps. According to [6], mobile has become an essential part of everyday life, and smartphone users spend 3 hours and 40 minutes on mobile apps throughout the day. Among the numerous smart industries created by mobile apps, the smart health and fitness market has recently been in the spotlight as individuals can easily manage their health and exercise anytime, anywhere.

In the recent era of artificial intelligence, mobile healthcare and fitness and related content are drawing attention due to the growing interest in health and exercise of the general public. The spread of trends that utilize smartphones' devices to manage individual health in real time and health care and fitness-related devices are receiving much attention. Exercisers actually use fitness apps to record, track and set up new exercise plans. It is their own exercise management platform which allows them to monitor their own exercise comfortably. The fitness app has become providing a life care platform that can continue to improve the overall quality of human life.

However, there are some inconveniences when users use these fitness apps. Whenever they exercise, they have to set up the app to record their workout. For example, when going for a run, you have to access the app and start the function that allows you to record your running. Also, when you desire to change your workout while exercising, you have to access the app again and turn on the function to start another exercise. Not many fitness apps even yet have the ability to record the amount of exercise in various sports. In this case, only sports supported by the app are allowed to record their own sports and other sports. This study begins with the expectation that the app will automatically recognize human activity motion and record the amount of exercise. In other words, the goal is to develop a system that makes possible the app to automatically recognize and record his or her movements instead of us manually record our sports activities.

This paper is a study for the development of mobile fitness app, which effectively reflects user convenience, and aims to help develop mobile fitness system in the future. To this end, a deep learning algorithm, CNN (Convolutional Neural Network), known for its excellent performance in static image learning is used, and the process and results of learning about actual motion images are shown.

The order of this paper is as follows. Chapter 2 describes the relevant research. Chapter 3 The experimental method describes the data set used in the test and the structure of the deep learning model for learning. Chapter 4 shows the results of the experiment after learning by the method described in Chapter 3. The final chapter describes the conclusions and future studies.

II. RELATED WORK

Automatic learning of interactions between human behavior and the environment has been actively studied in various fields of study due to their applicability in various fields. In these fields of study, the focus is on modeling human behavior in many aspects, including emotion, attitude, and behavior. [1] noted that the CNN model, which consists of 2D convnets to recognize these human behaviors, has ignored changes in time flow, but helps distinguish human behavior classification.

In recent deep learning studies, instead of designing background separation or feature extraction algorithms using human heuristics, they are performing well by learning models that play their role with a big amount of data, and among them, CNN-affiliated models stand out. Recent studies on human behavior analysis, especially in the fields of fitness, have been conducted mainly based on sensor data. In 2009, [2] analyzed five exercise motions (walking, running, cycling, driving, and sports), which were analyzed using the decision tree, naïve Bayes and naïve Bayes with PCA to achieve 72.3% accuracy. Also by 2011, [3] achieved 91.1% accuracy by using multi-layer perceptron classifier for classifying five human actions: running, walking, lying, standing and sitting. In 2017, [5] classified human exercise using wearable sensor data through CNN model, recording a high accuracy of 92.1%.

Fig. 1. Exercise activities used for the research: pull-up, push-up, running, squat.

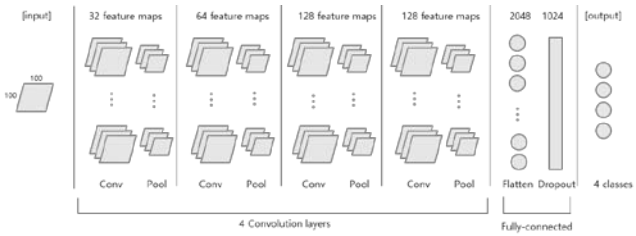


Fig. 2. The CNN architecture with four convolutional layers followed by a fully connected layer.

However, there are still few studies that classify motion through physical exercise rather than motion recognition and analysis using wearable-based sensor data. In this study, as an initial study for the development of a system that recognizes and analyzes users' movements in real time through the fitness app in the near future, this study intends to develop a system that recognizes exercise through the user's exercise image.

III. CNN ARCHITECTURE FOR ACTIVITY IMAGE DATA

A. Human activity image data

The data used in this study are images of people exercising, covering four sports: pull-up, push-up, running, and squats. The reason for selecting pull-up, push-up, running, and squats for sports was that they were the most easy-accessible and popular sports for home training and regular exercise routine. In fact, if a recognition and analysis system service is provided for many types of sports in the fitness app, these four types will be the most popular sports for people.

Image data were collected through Google search engine and performed web crawling using Python 3.0. About 2000~3000 number of data were collected for each type of sports. After data pre-processing, 1000 data for each class, a total of 4000 image data, were generated for this research. In pre-processing data step, pictures with incorrect movement posture, poor image quality, and only some parts of the body seen were excluded.

Especially, while collecting image data, different types of forms were used for search term for all four classes of sports activity classification for this research. For instance, for pull-up images, ‘pull-up’, ‘short grip pull-up’, ‘reverse pull-up’, vertical grip pull-up’ were used as search terms. This way, collected data showed many different poses even in one class of sports. For instance, for pull-up, images show a person with his arms stretched out to pull up and also, a person with a chin up on an iron bar. For push-up, the images show a person with their arms stretched up from the ground, and also a person whose body is almost touching the ground. For running, the images show runners running with their backs on them, their faces, their sides, and all the different angles of the runner. For squat, the image shows the person who is on their get-ready form, or full-squat form. By using datasets including various exercise postures for different sports activity classes, this research desired to reflect both different characteristics of people and different exercise poses made in real time. The sample of the images are shown in Fig. 1.

The collected data were pre-processed with a floating-point type tensor for application to the CNN algorithm. The pixel value scale of [0, 255] was adjusted to [0, 1] and the ImageDataGenerator utility of keras was used to handle these steps.

B. CNN architecture

CNN architecture is the most commonly used algorithm for deep learning, in which models classify images, videos, text, or sounds directly. CNN was originally developed for 2D image recognition. Therefore, CNN is particularly useful in finding patterns to recognize objects, faces and scenes in images. Without the burden for humans to use patterns of data to classify images and manually extract features, CNN learns directly from the data. In addition to this convenience, CNN is showing a high level of recognition accuracy, and these aspects of CNN makes the number of uses of CNN in deep learning is increasing rapidly.

The CNN architecture design for exercise activity recognition is presented in Fig. 2. In the CNN architecture, four convolution layers, which have 32, 64, 128, and 128 feature maps, are followed by a fully connected layer which has 2048 nodes. Rectified units are employed as activation functions and softmax functions are used for evaluating the final 4 output node values. For optimization, the RMSProp is employed with a learning rate of 0.0001. Also, dropout with a probability of 0.5 is applied to each layer to avoid the overfitting problem on training model.

The data used as input images were scaled to 100x100 using ImageDataGenerator, as described in Section 3.1, and are color images. The input data passes through a 2D Convolution Layer (Conv2D layer) with an activation function of ReLU. Output data from the Conv2D layer is then sampled with max pooling procedure. A total of four such convolutional layers are used for the same conditions as the Conv2D layer and max pooling as described above. Output data from the last hidden layer is used as input data to the output layer, through the flatten process of converting to 1D vector form, with an active function of Softmax and four output nodes.

IV. EXPERIMENTS

A total of 4,000 image data were used for the experiment. The data were classified by the ratio of training data, verification data and test data to 7:1:2. As a result, a total of 2,800 training images, 800 verification images and 1,600 test images were collected. Each segmented data contains the same number of samples for each class. Since this

Fig 3. Images after data augmentation is performed.

classification is balanced multiclass problem, accuracy was used to measure performance.

A. Baseline CNN architecture

MaxPooling2D steps were added to the Conv2D layer for the baseline CNN architecture. This process was done for the capacity of the network for the use of image data and reduce the size of the feature map so that the size of the Flatten layer does not become too large. Starting with an input size of 100x100 and the feature map size decreases as it passes through the MaxPooling layer. Since this experiment is multi-classification problem, the network used four units and Softmax activation functions. This results in the output of probability distributions for the final four classes. In addition, since the best loss function for the multi-classification problem is known as categorical_cossentropy, it is used for the network's loss function. This function measures the distance between two probability distributions. That is, measuring the distance between the probability distribution output by the network and the distribution of the true label. Minimizing the distance between the two distributions will train the model to produce as close an output as possible to the true label.

Specifically, for hyperparameters, steps per epoch is set to 120, the number of epochs to 30, and the validation steps to 40. As a result, 80.50% accuracy was recorded. Checking the 'Training and validation accuracy' graph, a problem with overfitting was found. Therefore, the need for data aggregation and dropout was decided, judging that there was a problem with the small amount of data.

B. Data augmentation & Dropout

Overfitting occurs when there are too few samples to learn to train models that can be generalized to new data. With an infinite amount of data given, the model will be able to learn all possible aspects of the data distribution. Data augmentation is a method of generating more training data from existing training samples. This method increases the sample by applying several random transformations to create a plausible image. Therefore, it helps generalize by allowing models to learn different aspects of the data. Using ImageDataGenerator in Keras, multiple kinds of random transformations were set up to be applied to images.

Specific parameters and their settings are as follows: rotation range=5, width shift range=0.1, height shift range=0.1, shear range=0.1, zoom range=0.2, horizontal flip=True, fill mode='nearest.' These settings were determined according to several experiments, which were decided to be most suitable according to the characteristics of motion. The considered human motion characteristics are as follows. When doing pull-up, push-up, squats and running, the angle does not tilt significantly in the position of standing or lying on the ground. Also, the shearing transformation is not significant because the images represent human, and that when pictures are enlarged, the person is cut out of the image. The image of the sample as a result of data aggregation is shown in Fig. 3.

When training a new network using data augmentation, the same input data is not used into the network twice. However, because it was created from a small number of original images, there is still a great correlation between the input data. In other words, there is no creation of new information and just recombination of existing information. Therefore, it may not be enough to eliminate overfitting completely. Thus, a dropout layer was added before the full-connected layer to further restrain overfitting. Dropout is set to 0.5.

As a result of learning with a new model with data aggregation and dropout, 81.50% accuracy was recorded. Although the performance was better than before data augmentation and dropout, the degree of improvement was small. In addition, the training and validation loss graph confirmed that the training accuracy only reached 75%. Therefore, it was necessary to increase the number of epochs and train the model more raps.

C. Effect on the number of epochs

To examine how the changes in the number of hidden layers are affecting the performance of the model, a change in the number of hidden layers was given. The experiment was conducted by increasing the number of hidden layers to 5. All other hyperparameter conditions are the same. As a result, accuracy was 80.50%, the same performance as when data aggregation and dropout were not carried out when the hidden layer was composed of only 4 layers. Judging that the hidden layer no longer needs to be added, the number of hidden layers decided to be fixed at 4.

D. Effect on the number of epochs

Baseline CNN architecture set the number of epochs to 30, but the train accuracy only reached 70%, so the new experiment was determined to conduct at the number of epochs to 100. All other hyperparameter conditions were the same as before. As a result, the train accuracy increased significantly as well as the validation accuracy. Also, it turned out that overfitting problems have been solved significantly. This allowed to determine the suitability of the hyperparameters used in this experiment. But the training was over when the train accuracy was still 88.5%, so it was decided to raise the epochs much higher. In this next experiment, the accuracy of the test data was recorded at 88.38%.

The new experiment was conducted by setting the number of epochs to 500 and the train accuracy reached up to 99.9%. In addition, the validation accuracy also increased, but around at levels of 180 epochs, the validation accuracy was found to be static without any further increase.

Therefore, it was confirmed that more than 180 epochs of training was meaningless. In this experiment, the test accuracy was able to achieve the highest figure of 91.75%.

Furthermore, another experiment was conducted by setting the initial number of epochs to 180, which is considered to be the start of the best performance. All other hyperparameters stayed the same. As a result, the test accuracy hit 89.38 percent, lower than when the number of epochs was 500. This shows that we can find models with the best performance by setting up a slightly larger number of epochs than the expected number of epochs.

V. CONCLUSION

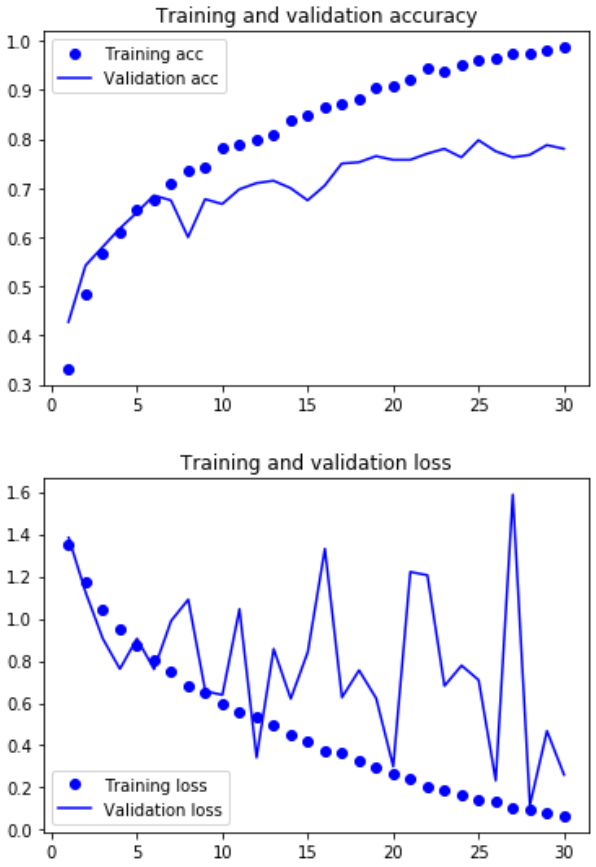
In this paper, the process and results of learning with a two-dimensional CNN model on the actual exercise image was shown. The best performance model was when there were four input layers, Softmax as an active function, causal_crossentropy as a loss function, and RMSProp as an optimizer. The problem of overfitting due to the small amount of data could be solved using data aggregation and dropout. Through this research, it was found that the number of suitable hidden layers varies depending on the nature of the data. In other words, it is not only good to train more by setting up many hidden layers and deepening the layers. In addition, this research concludes that the sufficient number of epochs would increase the train accuracy, making it an environment in which the validation accuracy could be reached.

This study is meaningful in that it has developed a system that allows different forms of posture to be recognized as one type of sport. It is significantly important because even if a fitness app user tries to take a specific posture while exercising, different positions appear every time the person tries and according to each person. Also, the posture of exercise changes in real time while exercising even if it is the same person. As a result, in order to develop a real-time exercise activity recognition system, a high-performance exercise activity classification algorithm that can be classified into the same exercise category even with various postures must be developed.

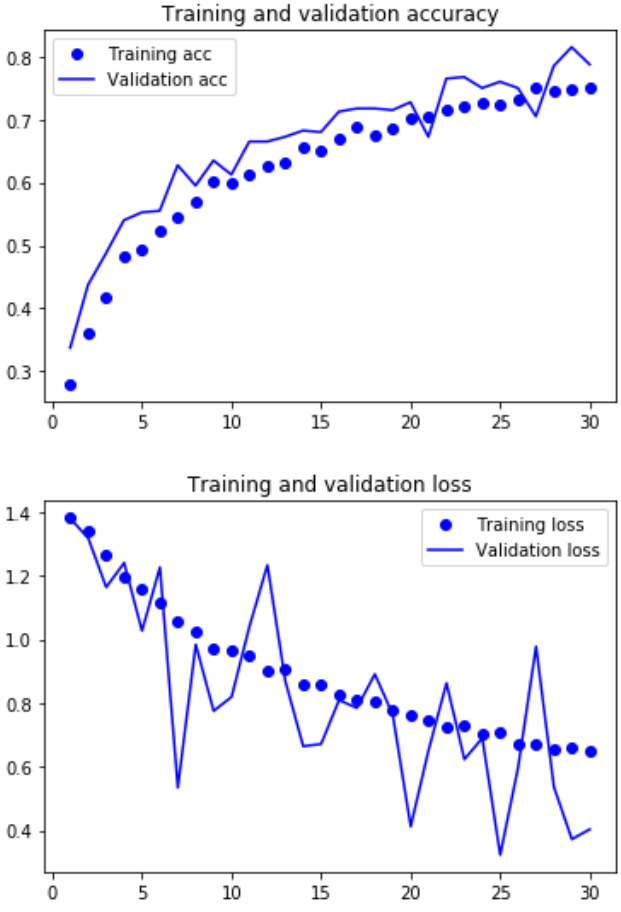
The fitness app will improve the user's exercise efficiency greatly if the app can capture the user's exercise in real-time and recognize it. As an early step to enter this stage, this study is meaningful in that it developed an exercise activity image recognition algorithm with high performance prior to motion recognition. The model proposed in this study showed a high performance at 91.75%, which will greatly contribute to the development of the field of fitness motion recognition research in the future.

APPENDIX

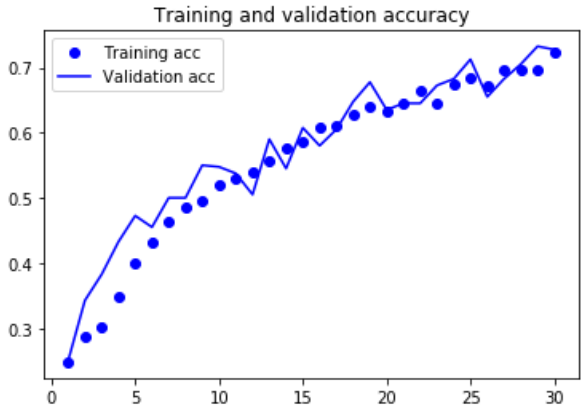
A. Baseline CNN architecture (acc: 80.50%)

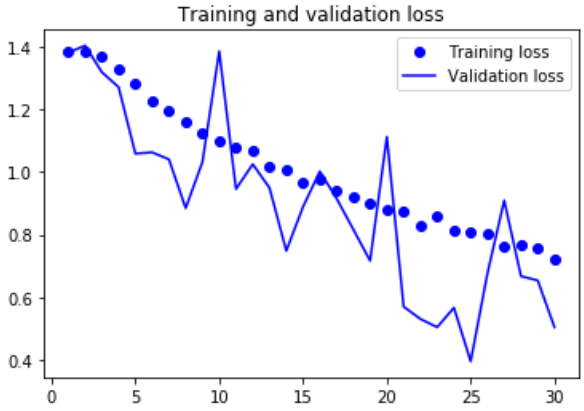


B. Data augmentation & Dropout (acc: 81.50%)

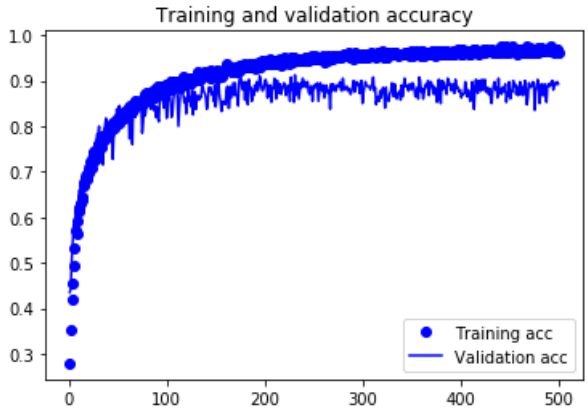


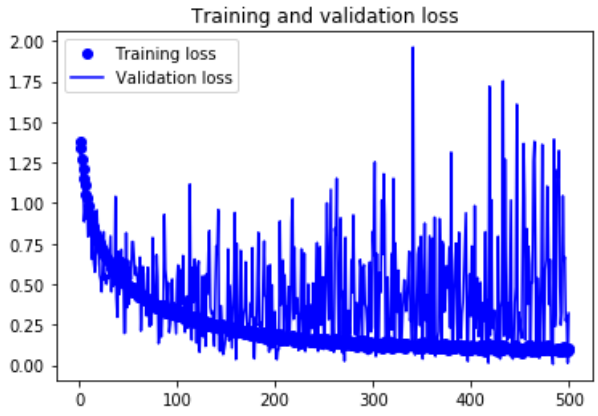
C. Effect on the number of epochs (acc: 80.50%)



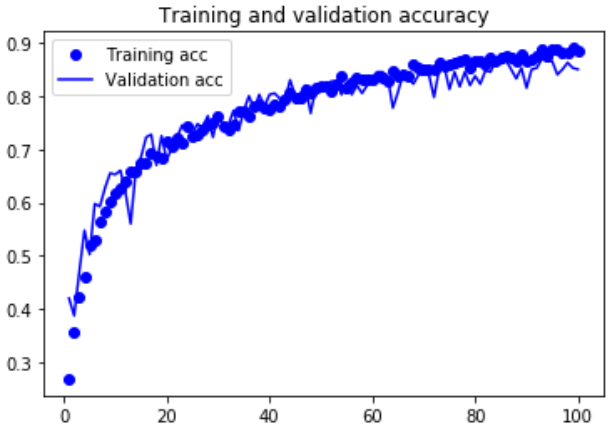


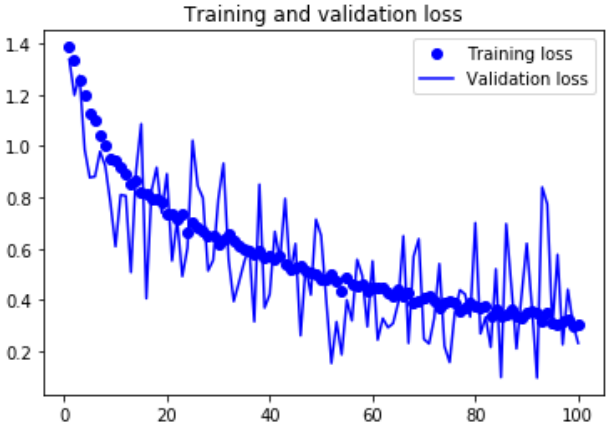
– 500 epochs (acc: 91.75%)



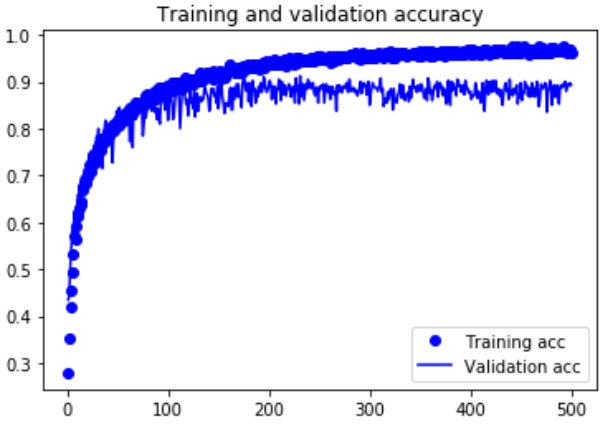


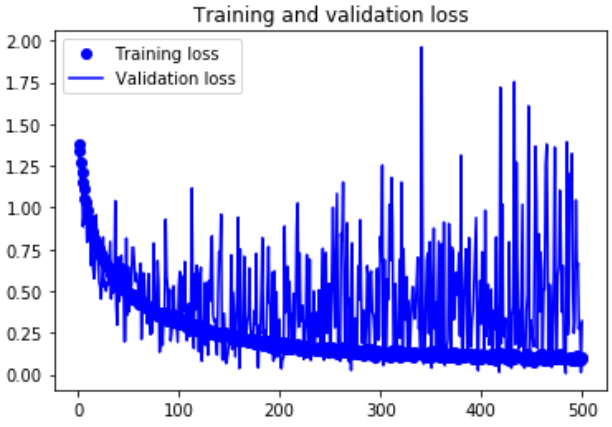
D. Effect on the number of epoch
– 100 epochs (acc: 88.38%)





– 180 epochs (acc: 88.38%)





REFERENCES

[1] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, Atilla Baskurt, “Sequential Deep Learning for Human Action Recognition”, Lecture Notes in Computer Science, 2011.

[2] X. Long, B. Yin, and R. M. Aarts, “Single-accelerometer-based daily physical activity classification,” in 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Sept 2009, pp. 6107–6110.

[3] S. Chernbumroong, A. S. Atkins, and H. Yu, “Activity classification using a single wrist-worn accelerometer,” in Software, Knowledge Information, Industrial Management and Applications (SKIMA), 2011 5th International Conference on, Sept 2011, pp. 1–6.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[5] Um, Terry Taewoong, Vahid Babakeshizadeh, and Dana Kulić. "Exercise motion classification from large-scale wearable sensor data using convolutional neural networks." 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.

[6] App Annie, “2020 Mobile Usage Report.” 2019.

[Python codes]

In [1]:
```
import os, shutil
```

making directory

In [2]:
```
original_dataset_dir = './datasets/train'
```

In [3]:
```
# making directory for datasets

base_dir = './datasets_class'
os.mkdir(base_dir)
```

In [4]:
```
# making directory for traing/validation/test

train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)

validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)

test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
```

In [5]:
```
# making training data directory for 'pullup / pushup / running / squat'

train_pullup_dir = os.path.join(train_dir, 'pullup')
os.mkdir(train_pullup_dir)

train_pushup_dir = os.path.join(train_dir, 'pushup')
os.mkdir(train_pushup_dir)

train_running_dir = os.path.join(train_dir, 'running')
os.mkdir(train_running_dir)

train_squat_dir = os.path.join(train_dir, 'squat')
os.mkdir(train_squat_dir)
```

In [6]:
```
# making validation data directory for 'pullup / pushup / running / squat'

validation_pullup_dir = os.path.join(validation_dir, 'pullup')
os.mkdir(validation_pullup_dir)

validation_pushup_dir = os.path.join(validation_dir, 'pushup')
os.mkdir(validation_pushup_dir)

validation_running_dir = os.path.join(validation_dir, 'running')
os.mkdir(validation_running_dir)

validation_squat_dir = os.path.join(validation_dir, 'squat')
os.mkdir(validation_squat_dir)
```

In [7]:
```
# making test data directory for 'pullup / pushup / running / squat'

test_pullup_dir = os.path.join(test_dir, 'pullup')
os.mkdir(test_pullup_dir)

test_pushup_dir = os.path.join(test_dir, 'pushup')
os.mkdir(test_pushup_dir)

test_running_dir = os.path.join(test_dir, 'running')
os.mkdir(test_running_dir)

test_squat_dir = os.path.join(test_dir, 'squat')
os.mkdir(test_squat_dir)
```

train / validation / test datasets

train

In [8]:
```
# copying 700 pullup images to 'train_pullup_dir' (70%)

fnames = ['pullup ({}).jpg'.format(i) for i in range(1, 701)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_pullup_dir, fname)
    shutil.copyfile(src, dst)
```

In [9]:
```
# copying 700 pushup images to 'train_pushup_dir' (70%)

fnames = ['pushup ({}).jpg'.format(i) for i in range(1, 701)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_pushup_dir, fname)
    shutil.copyfile(src, dst)
```

In [10]:
```
# copying 700 running images to 'train_running_dir' (70%)

fnames = ['running ({}).jpg'.format(i) for i in range(1, 701)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_running_dir, fname)
    shutil.copyfile(src, dst)
```

In [11]:
```
# copying 700 squat images to 'train_squat_dir' (70%)

fnames = ['squat ({}).jpg'.format(i) for i in range(1, 701)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_squat_dir, fname)
    shutil.copyfile(src, dst)
```

validation

In [12]:
```
# copying 100 pullup images to 'validation_pullup_dir' (10%)

fnames = ['pullup ({}).jpg'.format(i) for i in range(701, 801)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_pullup_dir, fname)
    shutil.copyfile(src, dst)
```

In [13]:
```
# copying 100 pushup images to 'validation_pushup_dir' (10%)

fnames = ['pushup ({}).jpg'.format(i) for i in range(701, 801)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_pushup_dir, fname)
    shutil.copyfile(src, dst)
```

In [14]:
```
# copying 100 running images to 'validation_running_dir' (10%)

fnames = ['running ({}).jpg'.format(i) for i in range(701, 801)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_running_dir, fname)
    shutil.copyfile(src, dst)
```

In [15]:
```
# copying 100 squat images to 'validation_squat_dir' (10%)

fnames = ['squat ({}).jpg'.format(i) for i in range(701, 801)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_squat_dir, fname)
    shutil.copyfile(src, dst)
```

test

In [16]:
```
# copying 200 pullup images to 'test_pullup_dir'

fnames = ['pullup ({}).jpg'.format(i) for i in range(801, 1001)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_pullup_dir, fname)
    shutil.copyfile(src, dst)
```

In [17]:
```
# copying 200 pushup images to 'test_pushup_dir'

fnames = ['pushup ({}).jpg'.format(i) for i in range(801, 1001)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_pushup_dir, fname)
    shutil.copyfile(src, dst)
```

In [18]:
```
# copying 200 running images to 'test_running_dir'

fnames = ['running ({}).jpg'.format(i) for i in range(801, 1001)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_running_dir, fname)
    shutil.copyfile(src, dst)
```

In [19]:
```
# copying 200 squat images to 'test_squat_dir'

fnames = ['squat ({}).jpg'.format(i) for i in range(801, 1001)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_squat_dir, fname)
    shutil.copyfile(src, dst)
```

In [20]:
```
print('total number of training pullup images:', len(os.listdir(train_pullup_dir)))
print('total number of training pushup images:', len(os.listdir(train_pushup_dir)))
print('total number of training running images:', len(os.listdir(train_running_dir)))
print('total number of training squat images:', len(os.listdir(train_squat_dir)))
```

```
total number of training pullup images: 700
total number of training pushup images: 700
total number of training running images: 700
total number of training squat images: 700
```

model

In [21]:
```
from keras import layers
from keras import models
```

```
Using TensorFlow backend.
C:\Users\Owner\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Owner\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Owner\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Owner\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Owner\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Owner\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [22]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 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(4, activation='softmax'))
```

```
WARNING:tensorflow:From C:\Users\Owner\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:4070: The name tf.nn.max_pool is de
precated. Please use tf.nn.max_pool2d instead.
```

In [23]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 98, 98, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 47, 47, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1049088
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 2052
=================================================================
Total params: 1,291,972
Trainable params: 1,291,972
Non-trainable params: 0
```

In [24]:
```
from keras import optimizers

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

data preprocessing

In [25]:
```
from keras.preprocessing.image import ImageDataGenerator
```

In [26]:
```
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

```
Found 2800 images belonging to 4 classes.
Found 400 images belonging to 4 classes.
```

model training

In [27]:
```
history = model.fit_generator(train_generator,
                              steps_per_epoch=120,
                              epochs=30,
                              validation_data=validation_generator,
                              validation_steps=40)
```

```
120/120 [==============================] - 36s 303ms/step - loss: 0.2458 - acc: 0.9204 - val_loss: 1.2212 - val_acc: 0.7575
Epoch 22/30
120/120 [==============================] - 36s 302ms/step - loss: 0.2011 - acc: 0.9425 - val_loss: 1.2058 - val_acc: 0.7700
Epoch 23/30
```

In [28]:
```
model.save('model_1.h5')
```

In [29]:
```
import keras

model_1 = keras.models.load_model('model_1.h5')
```

In [30]:
```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

```
Found 800 images belonging to 4 classes.
```

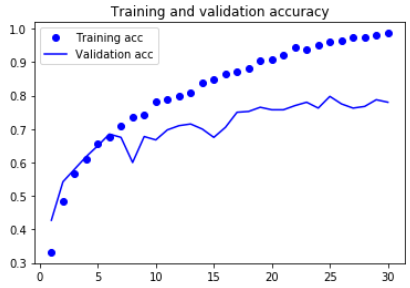
In [32]:
```
test_loss, test_acc = model_1.evaluate_generator(test_generator, steps=(800/20))
print('test acc:', test_acc)
```

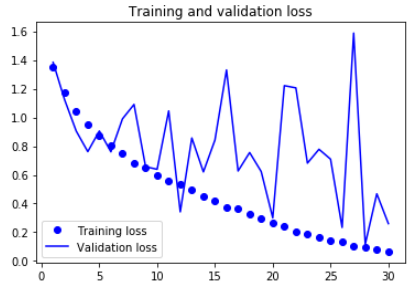
```
test acc: 0.8050000071525574
```

acc, loss graph

In [33]:
```
import matplotlib.pyplot as plt
```

In [34]:
```
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()
```





Data augmentation

In [41]:
```
datagen = ImageDataGenerator(rotation_range=2,
                             width_shift_range=0.1,
                             height_shift_range=0.1,
                             shear_range=0.1,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')
```

In [43]:
```
from keras.preprocessing import image

fnames = sorted([os.path.join(train_pullup_dir, fname) for
                 fname in os.listdir(train_pullup_dir)])

img_path = fnames[3]

img = image.load_img(img_path, target_size=(100, 100))

x = image.img_to_array(img)
x = x.reshape((1,) + x.shape)

i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break
plt.show()
```

Model 2

New model (with Data augmentation & Dropout)

In [44]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 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(4, activation='softmax'))

from keras import optimizers

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

Training new model using Data augmentions & Dropout

In [45]:
```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   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=(100, 100),
                                                    batch_size=20,
                                                    class_mode='categorical')

history = model.fit_generator(train_generator,
                              steps_per_epoch=120,
                              epochs=30,
                              validation_data=validation_generator,
                              validation_steps=40)
```

```
120/120 [==============================] - 39s 326ms/step - loss: 0.7459 - acc: 0.7038 - val_loss: 0.6519 - val_acc: 0.6725
Epoch 22/30
120/120 [==============================] - 39s 326ms/step - loss: 0.7224 - acc: 0.7167 - val_loss: 0.8624 - val_acc: 0.7650
Epoch 23/30
120/120 [==============================] - 39s 326ms/step - loss: 0.7298 - acc: 0.7212 - val_loss: 0.6231 - val_acc: 0.7675
Epoch 24/30
120/120 [==============================] - 39s 327ms/step - loss: 0.7051 - acc: 0.7254 - val_loss: 0.6895 - val_acc: 0.7500
Epoch 25/30
120/120 [==============================] - 39s 326ms/step - loss: 0.7107 - acc: 0.7246 - val_loss: 0.3229 - val_acc: 0.7600
Epoch 26/30
120/120 [==============================] - 42s 352ms/step - loss: 0.6684 - acc: 0.7321 - val_loss: 0.5979 - val_acc: 0.7500
Epoch 27/30
120/120 [==============================] - 40s 335ms/step - loss: 0.6683 - acc: 0.7508 - val_loss: 0.9771 - val_acc: 0.7050
Epoch 28/30
120/120 [==============================] - 39s 327ms/step - loss: 0.6565 - acc: 0.7442 - val_loss: 0.5356 - val_acc: 0.7850
Epoch 29/30
120/120 [==============================] - 39s 327ms/step - loss: 0.6600 - acc: 0.7471 - val_loss: 0.3719 - val_acc: 0.8150
Epoch 30/30
120/120 [==============================] - 39s 326ms/step - loss: 0.6497 - acc: 0.7517 - val_loss: 0.4032 - val_acc: 0.7875
```

In [46]:
```
model.save('model_2.h5')
```

In [47]:
```
import keras

model_2 = keras.models.load_model('model_2.h5')
```

In [48]:
```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

```
Found 800 images belonging to 4 classes.
```

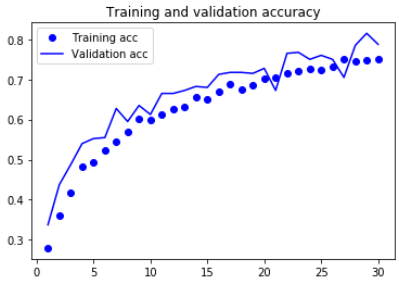
In [49]:
```
test_loss, test_acc = model_2.evaluate_generator(test_generator, steps=(800/20))
print('test acc:', test_acc)
```

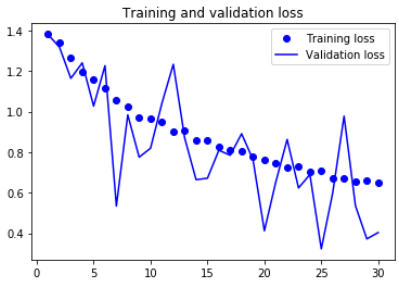
```
test acc: 0.8149999976158142
```

acc, loss graph

In [50]:
```
import matplotlib.pyplot as plt
```

In [51]:
```
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()
```





- training accuracy가 75%까지 밖에 못 갔다. epochs 수를 늘려줄 필요가 있다.
- hidden layer의 갯수의 증감에 따라 모델의 정확도가 어떻게 변하는지도 확인해보자.

Model 3

hidden layer 5개로 증가.

In [ ]:
```
# creating model

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 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.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(4, activation='softmax'))

from keras import optimizers

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

In [53]:
```
# model training

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   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=(100, 100),
                                                    batch_size=20,
                                                    class_mode='categorical')

history = model.fit_generator(train_generator,
                              steps_per_epoch=120,
                              epochs=30,
                              validation_data=validation_generator,
                              validation_steps=40)
```

```
120/120 [==============================] - 36s 301ms/step - loss: 0.8771 - acc: 0.6438 - val_loss: 0.5696 - val_acc: 0.6450
Epoch 22/30
120/120 [==============================] - 36s 302ms/step - loss: 0.8304 - acc: 0.6650 - val_loss: 0.5303 - val_acc: 0.6450
Epoch 23/30
120/120 [==============================] - 36s 301ms/step - loss: 0.8599 - acc: 0.6454 - val_loss: 0.5039 - val_acc: 0.6725
Epoch 24/30
120/120 [==============================] - 36s 300ms/step - loss: 0.8123 - acc: 0.6746 - val_loss: 0.5664 - val_acc: 0.6825
Epoch 25/30
120/120 [==============================] - 36s 302ms/step - loss: 0.8096 - acc: 0.6842 - val_loss: 0.3948 - val_acc: 0.7125
Epoch 26/30
120/120 [==============================] - 36s 301ms/step - loss: 0.8046 - acc: 0.6729 - val_loss: 0.6762 - val_acc: 0.6550
Epoch 27/30
120/120 [==============================] - 36s 300ms/step - loss: 0.7642 - acc: 0.6954 - val_loss: 0.9094 - val_acc: 0.6825
Epoch 28/30
120/120 [==============================] - 36s 301ms/step - loss: 0.7668 - acc: 0.6971 - val_loss: 0.6671 - val_acc: 0.7050
Epoch 29/30
120/120 [==============================] - 36s 303ms/step - loss: 0.7582 - acc: 0.6967 - val_loss: 0.6534 - val_acc: 0.7325
Epoch 30/30
120/120 [==============================] - 36s 300ms/step - loss: 0.7200 - acc: 0.7221 - val_loss: 0.5038 - val_acc: 0.7275
```

In [54]:
```
model.save('model_3.h5')
```

In [55]:
```
import keras

model_3 = keras.models.load_model('model_3.h5')
```

In [56]:
```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

```
Found 800 images belonging to 4 classes.
```

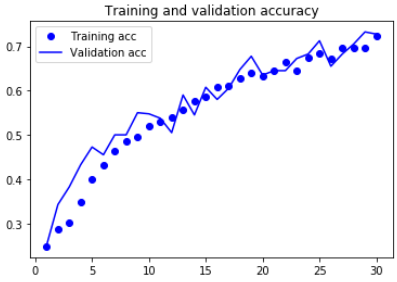
In [57]:
```
test_loss, test_acc = model_3.evaluate_generator(test_generator, steps=(800/20))
print('test acc:', test_acc)
```

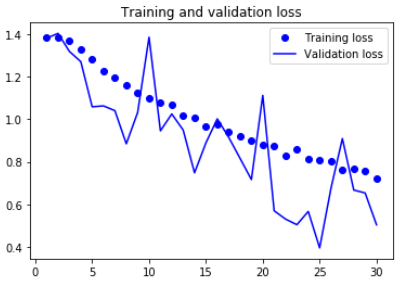
```
test acc: 0.8050000071525574
```

acc, loss graph

In [58]:
```
import matplotlib.pyplot as plt
```

In [59]:
```
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()
```





- Model 3에서와 마찬가지로 training accuracy가 72%까지 밖에 못 갔다. epochs 수를 늘려줄 필요가 있다.
- hidden layer 수를 5개로 늘렸는데 오히려 정확도가 감소하였다. hidden layer 수는 4개 이상으로 늘리지 말고, 현재로서는 4개로 진행해보자.

Model 4

hidden layer 4개 / epochs 100

In [62]:
```
# creating model

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 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(4, activation='softmax'))

from keras import optimizers

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

In [63]:
```
# model training

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   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=(100, 100),
                                                    batch_size=20,
                                                    class_mode='categorical')

history = model.fit_generator(train_generator,
                              steps_per_epoch=120,
                              epochs=100,
                              validation_data=validation_generator,
                              validation_steps=40)
```

```
120/120 [==============================] - 38s 320ms/step - loss: 0.3547 - acc: 0.8700 - val_loss: 0.3769 - val_acc: 0.8500
Epoch 92/100
120/120 [==============================] - 38s 318ms/step - loss: 0.3503 - acc: 0.8758 - val_loss: 0.0966 - val_acc: 0.8525
Epoch 93/100
120/120 [==============================] - 38s 318ms/step - loss: 0.3156 - acc: 0.8888 - val_loss: 0.8397 - val_acc: 0.8700
Epoch 94/100
120/120 [==============================] - 38s 318ms/step - loss: 0.3474 - acc: 0.8758 - val_loss: 0.7734 - val_acc: 0.8650
Epoch 95/100
120/120 [==============================] - 38s 318ms/step - loss: 0.3099 - acc: 0.8867 - val_loss: 0.2934 - val_acc: 0.8650
Epoch 96/100
120/120 [==============================] - 39s 325ms/step - loss: 0.3028 - acc: 0.8875 - val_loss: 0.5771 - val_acc: 0.8400
Epoch 97/100
120/120 [==============================] - 39s 322ms/step - loss: 0.3208 - acc: 0.8800 - val_loss: 0.2268 - val_acc: 0.8500
Epoch 98/100
120/120 [==============================] - 38s 320ms/step - loss: 0.3259 - acc: 0.8817 - val_loss: 0.4411 - val_acc: 0.8625
Epoch 99/100
120/120 [==============================] - 38s 320ms/step - loss: 0.2998 - acc: 0.8900 - val_loss: 0.3060 - val_acc: 0.8525
Epoch 100/100
120/120 [==============================] - 38s 318ms/step - loss: 0.3036 - acc: 0.8850 - val_loss: 0.2325 - val_acc: 0.8500
```

In [64]:
```
model.save('model_4.h5')
```

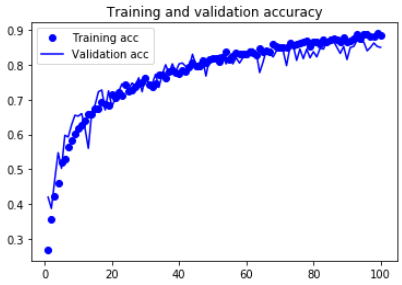
In [67]:
```
import keras

model_4 = keras.models.load_model('model_4.h5')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

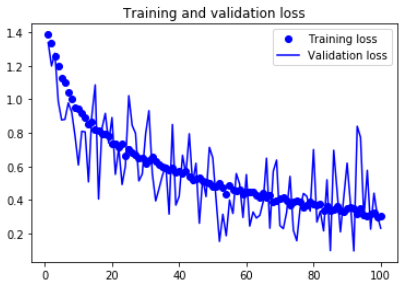
```
Found 800 images belonging to 4 classes.
```

In [68]:
```
test_loss, test_acc = model_4.evaluate_generator(test_generator, steps=(800/20))
print('test acc:', test_acc)
```

```
test acc: 0.8837500214576721
```

In [69]:
```
# acc, loss graph

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()
```





- overfitting 문제가 많이 해결된 것으로 나타났다. 적정한 파라미터를 찾은 듯 하다.
- train accuracy가 88% 정도까지 올라가고, test accuracy도 마찬가지로 88% 정도로 나타났다. epochs를 아직 더 올려야 할 것으로 판단된다.

Model 5

hidden layer 4개 / epochs 500

In [70]:
```
# creating model

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 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(4, activation='softmax'))

from keras import optimizers

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

In [71]:
```
# model training

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   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=(100, 100),
                                                    batch_size=20,
                                                    class_mode='categorical')

history = model.fit_generator(train_generator,
                              steps_per_epoch=120,
                              epochs=500,
                              validation_data=validation_generator,
                              validation_steps=40)
```

```
120/120 [==============================] - 39s 328ms/step - loss: 0.1016 - acc: 0.9658 - val_loss: 0.4902 - val_acc: 0.8875
Epoch 492/500
120/120 [==============================] - 39s 322ms/step - loss: 0.0918 - acc: 0.9683 - val_loss: 0.2479 - val_acc: 0.8950
Epoch 493/500
120/120 [==============================] - 38s 320ms/step - loss: 0.0840 - acc: 0.9742 - val_loss: 0.3262 - val_acc: 0.8900
Epoch 494/500
120/120 [==============================] - 38s 321ms/step - loss: 0.0965 - acc: 0.9688 - val_loss: 1.0445 - val_acc: 0.9000
Epoch 495/500
120/120 [==============================] - 39s 321ms/step - loss: 0.1166 - acc: 0.9621 - val_loss: 0.6439 - val_acc: 0.8800
Epoch 496/500
120/120 [==============================] - 39s 324ms/step - loss: 0.0996 - acc: 0.9671 - val_loss: 0.6663 - val_acc: 0.8950
Epoch 497/500
120/120 [==============================] - 41s 338ms/step - loss: 0.1044 - acc: 0.9692 - val_loss: 0.2734 - val_acc: 0.8900
Epoch 498/500
120/120 [==============================] - 40s 333ms/step - loss: 0.0896 - acc: 0.9675 - val_loss: 0.3057 - val_acc: 0.8950
Epoch 499/500
120/120 [==============================] - 39s 324ms/step - loss: 0.0999 - acc: 0.9696 - val_loss: 0.0137 - val_acc: 0.8975
Epoch 500/500
120/120 [==============================] - 39s 326ms/step - loss: 0.0912 - acc: 0.9629 - val_loss: 0.3204 - val_acc: 0.8925
```

In [72]:
```
model.save('model_5.h5')
```

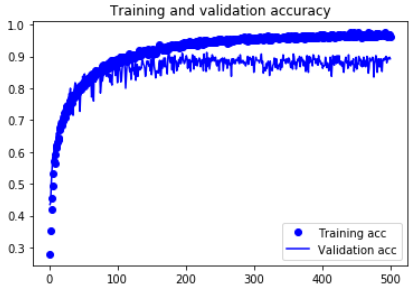
In [73]:
```
import keras

model_5 = keras.models.load_model('model_5.h5')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

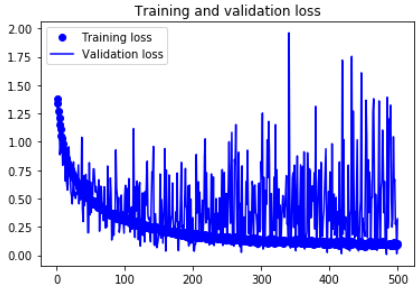
```
Found 800 images belonging to 4 classes.
```

In [74]:
```
test_loss, test_acc = model_5.evaluate_generator(test_generator, steps=(800/20))
print('test acc:', test_acc)
```

```
test acc: 0.9175000190734863
```

In [75]:
```
# acc, loss graph

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()
```





Model 6

hidden layer 47개 / epochs 180

In [76]:
```
# creating model

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 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(4, activation='softmax'))

from keras import optimizers

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

In [77]:
```
# model training

train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=2,
                                   width_shift_range=0.1,
                                   height_shift_range=0.1,
                                   shear_range=0.1,
                                   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=(100, 100),
                                                    batch_size=20,
                                                    class_mode='categorical')

history = model.fit_generator(train_generator,
                              steps_per_epoch=120,
                              epochs=180,
                              validation_data=validation_generator,
                              validation_steps=40)
```

```
120/120 [==============================] - 39s 322ms/step - loss: 0.2420 - acc: 0.9104 - val_loss: 0.5058 - val_acc: 0.8650
Epoch 137/180
120/120 [==============================] - 39s 325ms/step - loss: 0.2154 - acc: 0.9212 - val_loss: 0.6085 - val_acc: 0.8650
Epoch 138/180
120/120 [==============================] - 39s 323ms/step - loss: 0.2598 - acc: 0.9079 - val_loss: 0.1432 - val_acc: 0.8675
Epoch 139/180
120/120 [==============================] - 39s 324ms/step - loss: 0.2590 - acc: 0.9075 - val_loss: 0.1093 - val_acc: 0.8625
Epoch 140/180
120/120 [==============================] - 39s 324ms/step - loss: 0.2289 - acc: 0.9179 - val_loss: 0.6352 - val_acc: 0.8775
Epoch 141/180
120/120 [==============================] - 39s 324ms/step - loss: 0.2374 - acc: 0.9162 - val_loss: 0.4757 - val_acc: 0.8825
Epoch 142/180
120/120 [==============================] - 39s 322ms/step - loss: 0.2436 - acc: 0.9158 - val_loss: 0.5859 - val_acc: 0.8675
Epoch 143/180
120/120 [==============================] - 39s 324ms/step - loss: 0.2288 - acc: 0.9196 - val_loss: 0.5106 - val_acc: 0.8575
Epoch 144/180
120/120 [==============================] - 39s 323ms/step - loss: 0.2159 - acc: 0.9225 - val_loss: 0.0456 - val_acc: 0.8800
Epoch 145/180
120/120 [==============================] - 39s 322ms/step - loss: 0.2296 - acc: 0.9183 - val_loss: 0.1732 - val_acc: 0.8525
```

In [78]:
```
model.save('model_6.h5')
```

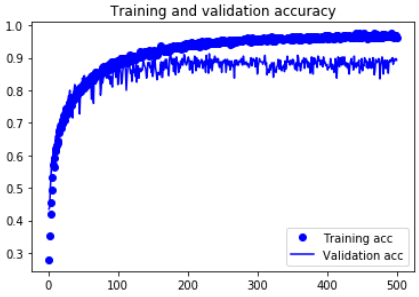
In [79]:
```
import keras

model_6 = keras.models.load_model('model_6.h5')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(100, 100),
    batch_size=20,
    class_mode='categorical')
```

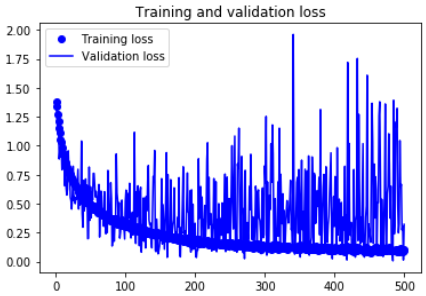
```
Found 800 images belonging to 4 classes.
```

In [80]:
```
test_loss, test_acc = model_6.evaluate_generator(test_generator, steps=(800/20))
print('test acc:', test_acc)
```

```
test acc: 0.893750011920929
```

In [75]:
```
# acc, loss graph

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()
```


Training and validation accuracy
Training acc
Validation acc
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
100
200
300
400
500


Training and validation loss
Training loss
Validation loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
100
200
300
400
500