In this project, it is necessary that all the images are the same size. Taking both information reservation as well as the speed and memory consumption when reading those data into account, all the images are transformed into 128*128 pixels. In this case, data reading takes about 3 minutes.

```
width=128
```

Firstly, since the process of reading information of training, validation and testing sets should be almost the same, a function is defined to read training, validation and testing data. The function takes a string which is the file name as an argument and returns the training data (which is an n*128*128*3 array) and their corresponding labels (which is an n*1 array). The details of the function are explained below:

```
def readdata(filename):
    train=[]

    with open(filename) as f:
        line = f.readline()
        while line:
            line=line.split()
            train.append(line)
            # print line
            line = f.readline()
```

Read the information line by line and split the address and labels, and then store it into a big 2D list.

```
    train=pd.DataFrame(train)
    #print train


    add=train[0]
    if (train.shape[1]<2):
        y_train=[]
    else:
        y_train=np.array(train[1], dtype='uint8').reshape(len(add),1)
```

For convenience, transform the 2D list into a 2-column DataFrame. The first column is the addresses of the images while the second column is the corresponding labels. For test data, there are no labels so only an empty list will be stored.

```
    X_train=[]

    for i in add:

        img = Image.open(i)
        img=img.resize((width,width))

        data = img.getdata()
        data = np.array(data)
        data=data.tolist()
        #data = data.reshape(width,width,3)

        #new_im = Image.fromarray(data.astype('uint8'))
        #plt.imshow(new_im)


        X_train.append(data)
        #X_train=np.array(X_train)

    X_train=np.array(X_train, dtype='uint8')
    X_train=X_train.reshape(len(add), width, width,3)

    #plt.imshow(img)
    #X_train=pd.DataFrame(X_train)

    return (X_train, y_train)
```

Then, open the images according to the addresses and transform the images into 128*128. After that, transform the images into numerical arrays of size length*width*3 (RGB). The “Image” class is imported from package PIL.

And the function will be called like that:

```
(X_train, y_train)=readdata('train.txt')
print("train data read complete")

(X_test, y_test)=readdata('val.txt')
print("test data read complete")

(X_task, _)=readdata('test.txt')
print("task data read complete")
```

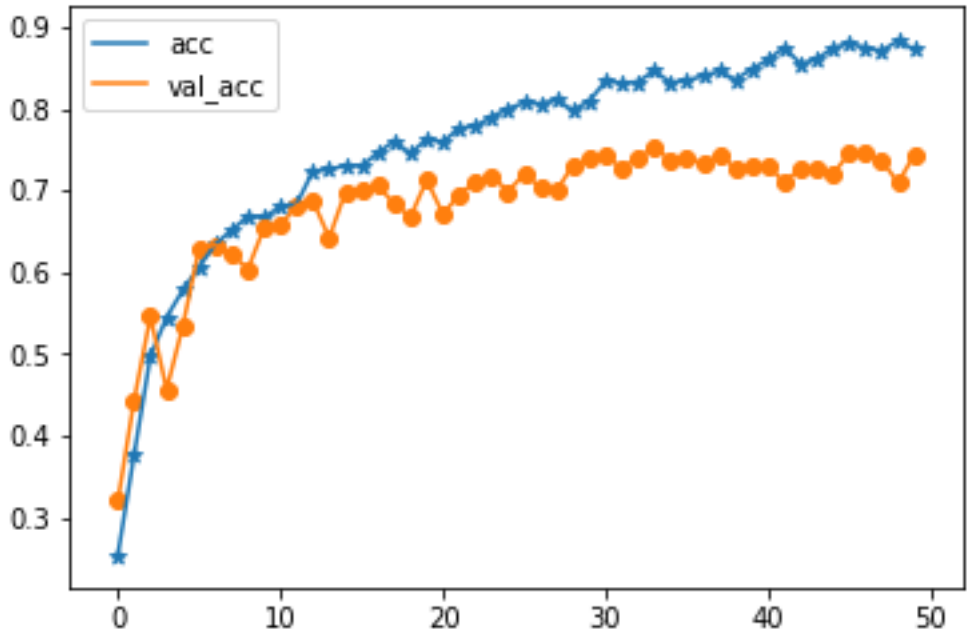
After reading data, some preprocessing will be conducted:

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

y_train = utils.to_categorical(y_train, 5)
y_test = utils.to_categorical(y_test, 5)
```

Normalize the RGB values so that they fall in [0,1], and categorize the labels from numerical values.

And then convolutional neural network is built. (The screenshot of the codes are emitted.) It contains firstly a convolutional layer, followed by 5 convolutional layers and 5 max-pooling layers placed one by one. Afterwards there is a flatten layer and 3 fully-connected layers. Also a dropout layer is added at appropriate positions to avoid overfit. The structure of the model and those hyperparameters are tuned after dozens of times of trials and experiments, so the validation accuracy could be improved from 50% to more than 70%.



After 50 epochs of fitting, the validation accuracy converges to about 75%. This may be due to the lack of hardware support since without GPU, the training speed was too slow to afford more complex models, which may further improve the performance.

```
2569/2569 [==============================] - 486s 189ms/step - loss: 0.3420 - acc:
0.8739 - val_loss: 0.8412 - val_acc: 0.7200
Epoch 46/50
2569/2569 [==============================] - 487s 190ms/step - loss: 0.3210 - acc:
0.8801 - val_loss: 0.8373 - val_acc: 0.7455
Epoch 47/50
2569/2569 [==============================] - 486s 189ms/step - loss: 0.3281 - acc:
0.8743 - val_loss: 0.7951 - val_acc: 0.7473
Epoch 48/50
2569/2569 [==============================] - 485s 189ms/step - loss: 0.3371 - acc:
0.8708 - val_loss: 0.7771 - val_acc: 0.7364
Epoch 49/50
2569/2569 [==============================] - 485s 189ms/step - loss: 0.3160 - acc:
0.8828 - val_loss: 0.9146 - val_acc: 0.7091
Epoch 50/50
2569/2569 [==============================] - 486s 189ms/step - loss: 0.3404 - acc:
0.8723 - val_loss: 0.8859 - val_acc: 0.7436
```

Finally, the task is to make a prediction and save the results to a txt file.

```
y_task=model.predict(X_task)

prediction=[]
for i in y_task:
    for j in range(0,5):
        if i[j]>0.5:
            prediction.append(j)

with open('project2_20461901.txt',"w") as f:
    for i in prediction:
        f.write(str(i))
        f.write("\n")
f.close()
```