In [1]:
```
from PIL import Image
import numpy as np
import os, glob


print("import PIL, numpy, os, glob ")
```

```
import PIL, numpy, os, glob
```

파일들 unzip

In [ ]:
```
!unzip rock_scissor_paper.zip
```

In [ ]:
```
!unzip test.zip
```

train image resize

학습할 데이터를 28x28사이즈로 resize시켜준다

In [2]:
```
import os

def resize_images(img_path):
        images=glob.glob(img_path + "/*.jpg")

        print(len(images), " images to be resized.")

    # 파일마다 모두 28x28 사이즈로 바꾸어 저장합니다.
        target_size=(28,28)
        for img in images:
                old_img=Image.open(img)
                new_img=old_img.resize(target_size,Image.ANTIALIAS)
                new_img.save(img, "JPEG")

        print(len(images), " images resized.")

# 가위, 바위, 보 이미지가 저장된 디렉토리 아래의 모든 jpg 파일을 읽어들여서
image_dir_path = "./rock_scissor_paper/scissor"
resize_images(image_dir_path)
print("가위 이미지 resize 완료!")

image_dir_path = "./rock_scissor_paper/rock"
resize_images(image_dir_path)
print("바위 이미지 resize 완료!")

image_dir_path = "./rock_scissor_paper/paper"
resize_images(image_dir_path)
print("보 이미지 resize 완료!")
```

```
100  images to be resized.
100  images resized.
가위 이미지 resize 완료!
100  images to be resized.
100  images resized.
바위 이미지 resize 완료!
100  images to be resized.
100  images resized.
보 이미지 resize 완료!
```

test image resize

test 이미지를 28x28사이즈로 resize시켜준다

In [3]:

```
image_dir_path = "./test/scissor"
resize_images(image_dir_path)
print("test 가위 이미지 resize 완료!")

image_dir_path = "./test/rock"
resize_images(image_dir_path)
print("test 바위 이미지 resize 완료!")

image_dir_path = "./test/paper"
resize_images(image_dir_path)
print("test 보 이미지 resize 완료!")
```

```
100  images to be resized.
100  images resized.
test 가위 이미지 resize 완료!
100  images to be resized.
100  images resized.
test 바위 이미지 resize 완료!
100  images to be resized.
100  images resized.
test 보 이미지 resize 완료!
```

가위: 0, 바위: 1, 보: 2 로 라벨링해준다

In [4]:

```
# 이미지 데이터와 라벨(가위 : 0, 바위 : 1, 보 : 2) 데이터를 담을 행렬(matrix) 영역을 생성
def load_data(img_path, number_of_data=300):
    img_size=28 # resize 할 크기
    color=3 # 색상은 3, 흑백은 1
    imgs=np.zeros(number_of_data*img_size*img_size*color,dtype=np.int32).reshape(number_of_data,img_size,im
    labels=np.zeros(number_of_data,dtype=np.int32)

    idx=0
    for file in glob.iglob(img_path+'/scissor/*.jpg'):
        img = np.array(Image.open(file),dtype=np.int32)
        imgs[idx,:,:,:]=img    # 데이터 영역에 이미지 행렬을 복사
        labels[idx]=0   # 가위 : 0
        idx=idx+1

    for file in glob.iglob(img_path+'/rock/*.jpg'):
        img = np.array(Image.open(file),dtype=np.int32)
        imgs[idx,:,:,:]=img    # 데이터 영역에 이미지 행렬을 복사
        labels[idx]=1   # 바위 : 1
        idx=idx+1

    for file in glob.iglob(img_path+'/paper/*.jpg'):
        img = np.array(Image.open(file),dtype=np.int32)
        imgs[idx,:,:,:]=img    # 데이터 영역에 이미지 행렬을 복사
        labels[idx]=2   # 보 : 2
        idx=idx+1

    print("학습데이터(x_train)의 이미지 개수는", idx,"입니다.")
    return imgs, labels

image_dir_path = "./rock_scissor_paper"
(x_train, y_train)=load_data(image_dir_path)
x_train_norm = x_train/255.0   # 입력은 0~1 사이의 값으로 정규화

print("x_train shape: {}".format(x_train.shape))
```

```
print("y_train shape: {}".format(y_train.shape))
print("x_train_norm: {} ".format(x_train_norm.shape))
```

학습데이터(x_train)의 이미지 개수는 300 입니다.
x_train shape: (300, 28, 28, 3)
y_train shape: (300,)
x_train_norm: (300, 28, 28, 3)

In [ ]:

```
import matplotlib.pyplot as plt
plt.imshow(x_train[40])
print('라벨: ', y_train[40])
```

test데이터 load

In [5]:

```
image_dir_path = "./test"
(x_test, y_test)=load_data(image_dir_path)
x_test_norm = x_test/255.0  # 입력은 0~1 사이의 값으로 정규화

print("x_test shape: {}".format(x_test.shape))
print("y_test shape: {}".format(y_test.shape))
print("x_test_norm : {}".format(x_test_norm.shape))
```

학습데이터(x_train)의 이미지 개수는 300 입니다.
x_test shape: (300, 28, 28, 3)
y_test shape: (300,)
x_test_norm : (300, 28, 28, 3)

딥러닝 네트워크 설계

In [240…

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

n_channel_1=45
n_channel_2=57
n_dense=50
n_train_epoch=20

model=keras.models.Sequential()
model.add(keras.layers.Conv2D(n_channel_1, (3,3), activation='relu', input_shape=(28,28,3))) # 컬러 이미지므로
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(n_channel_2, (3,3), activation='relu'))
model.add(keras.layers.MaxPooling2D((2,2)))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(n_dense, activation='relu'))
model.add(keras.layers.Dense(3, activation='softmax')) # 가위, 바위, 보이므로 클래스 3개


model.summary()
```

Model: "sequential_54"

Layer (type)	Output Shape	Param #
conv2d_108 (Conv2D)	(None, 26, 26, 45)	1260
max_pooling2d_108 (MaxPoolin	(None, 13, 13, 45)	0
conv2d_109 (Conv2D)	(None, 11, 11, 57)	23142
max_pooling2d_109 (MaxPoolin	(None, 5, 5, 57)	0

```
_________________________________________________________________
flatten_54 (Flatten)         (None, 1425)              0
_________________________________________________________________
dense_108 (Dense)            (None, 50)                71300
_________________________________________________________________
dense_109 (Dense)            (None, 3)                 153
=================================================================
Total params: 95,855
Trainable params: 95,855
Non-trainable params: 0
_________________________________________________________________
```

딥러닝 네트워크 학습시키기

In [241…

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train_norm, y_train, epochs=n_train_epoch)
```

```
Epoch 1/20
10/10 [==============================] - 1s 17ms/step - loss: 1.1121 - accuracy: 0.3300
Epoch 2/20
10/10 [==============================] - 0s 18ms/step - loss: 1.0838 - accuracy: 0.3633
Epoch 3/20
10/10 [==============================] - 0s 19ms/step - loss: 1.0491 - accuracy: 0.5067
Epoch 4/20
10/10 [==============================] - 0s 18ms/step - loss: 1.0102 - accuracy: 0.5300
Epoch 5/20
10/10 [==============================] - 0s 18ms/step - loss: 0.9492 - accuracy: 0.5467
Epoch 6/20
10/10 [==============================] - 0s 18ms/step - loss: 0.9312 - accuracy: 0.5633
Epoch 7/20
10/10 [==============================] - 0s 18ms/step - loss: 0.8713 - accuracy: 0.5933
Epoch 8/20
10/10 [==============================] - 0s 19ms/step - loss: 0.8037 - accuracy: 0.6367
Epoch 9/20
10/10 [==============================] - 0s 18ms/step - loss: 0.7545 - accuracy: 0.7000
Epoch 10/20
10/10 [==============================] - 0s 18ms/step - loss: 0.6967 - accuracy: 0.6833
Epoch 11/20
10/10 [==============================] - 0s 18ms/step - loss: 0.6501 - accuracy: 0.7467
Epoch 12/20
10/10 [==============================] - 0s 17ms/step - loss: 0.5595 - accuracy: 0.8367
Epoch 13/20
10/10 [==============================] - 0s 18ms/step - loss: 0.5165 - accuracy: 0.8000
Epoch 14/20
10/10 [==============================] - 0s 20ms/step - loss: 0.5105 - accuracy: 0.8033
Epoch 15/20
10/10 [==============================] - 0s 18ms/step - loss: 0.4210 - accuracy: 0.8633
Epoch 16/20
10/10 [==============================] - 0s 19ms/step - loss: 0.4358 - accuracy: 0.8200
Epoch 17/20
10/10 [==============================] - 0s 19ms/step - loss: 0.4045 - accuracy: 0.8433
Epoch 18/20
10/10 [==============================] - 0s 18ms/step - loss: 0.3282 - accuracy: 0.9000
Epoch 19/20
10/10 [==============================] - 0s 18ms/step - loss: 0.2933 - accuracy: 0.9233
Epoch 20/20
10/10 [==============================] - 0s 17ms/step - loss: 0.2701 - accuracy: 0.9100
```

Out[241…

```
<tensorflow.python.keras.callbacks.History at 0x7f918a42a810>
```

테스트 데이터로 성능을 확인

In [242…

```
test_loss, test_accuracy = model.evaluate(x_test_norm, y_test, verbose=2)
print("test_loss: {} ".format(test_loss))
print("test_accuracy: {}".format(test_accuracy))
```

```
10/10 - 0s - loss: 1.2031 - accuracy: 0.6400
test_loss: 1.2031381130218506
test_accuracy: 0.6399999856948853
```

In [ ]:

In [ ]:

In [ ]: