

# CNN - 가위바위보

<https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors> 에서 배포하는 데이터셋

아래 URL을 통해 기본 파일 정리가 수행된 파일을 내려 받는다.

<https://drive.google.com/file/d/1x6YsEBCSuxAKbmUbF-U0ntXoNwELoTVr/view?usp=sharing>

## #01. 준비작업

### [1] 패키지 참조

```
# 연결된 모듈이 업데이트 되면 즉시 자동 로드함
%load_ext autoreload
%autoreload 2

import warnings
warnings.filterwarnings(action="ignore")

from hossam.util import *
from hossam.plot import *
from hossam.tensor import *

import zipfile

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

The autoreload extension is already loaded. To reload it, use:  
%reload\_ext autoreload

### [2] 데이터셋 준비하기

#### (1) 파일 압축 해제

데이터셋을 캐글로부터 다운로드 받은 후 적절한 위치에 압축을 해제한다.

압축을 해제하면 cats 폴더와 dogs 폴더에 각각 5000장의 이미지가 포함되어 있다.

```
# 압축파일의 경로
workspace_dir = "D:\\\"
file_path = os.path.join(workspace_dir, "rock-paper-scissors.zip")

# 압축을 해제할 경로
extract_dir = os.path.join(workspace_dir, "rock-paper-scissors")

# 해당 폴더가 없다면 폴더를 생성하고 파일의 압축을 해제
if not os.path.exists(extract_dir):
    os.mkdir(extract_dir)

zip_ref = zipfile.ZipFile(file_path, "r")
```

```
zip_ref.extractall(extract_dir)
zip_ref.close()
```

## (2) 임의의 이미지 확인

이미지

실행시마다 표시 이미지가 랜덤하게 바뀐다.

```
subdir = os.listdir(extract_dir)
subdir
```

```
['paper', 'rock', 'scissors']
```

```
for s in subdir:
    path = os.path.join(extract_dir, s)
    print(path)

    image_list = os.listdir(path)
    image_count = len(image_list)

    rand = np.random.random_integers(0, image_count - 1, 5)

    fig, ax = plt.subplots(1, 5, figsize=(20, 3), dpi=100)

    for i in range(0, len(ax)):
        file_path = os.path.join(path, image_list[rand[i]])
        img = load_image(file_path)
        ax[i].imshow(img)
        ax[i].axis("off")
        ax[i].set_title(image_list[rand[i]])

    plt.show()
    plt.close()
```

D:\rock-paper-scissors\paper

OhkFvTOiBqphyG0l.png



qhhaVBNDIWDdNR2Y.png



cOHWFfHkyo8sqWnpW.png



TmPdVzKg9yHUefja.png



FT9FiK5l1lohxfxl.png



D:\rock-paper-scissors\rock

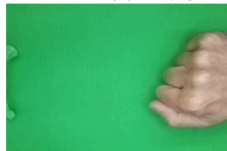
UDjD30Rke7seRcaD.png



yeinQlqGeaqSgvoN.png



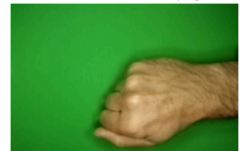
YmNXS0VhjQjldK3P.png



fQPCA8oYZ8ywDgyv.png



8WivsxbidGW1wnnx.png



D:\rock-paper-scissors\scissors

OLq5gEcrMDGftxeG.png



R9GIL5dx9nmjXWIS.png



HJ3qSJKzOvM3lwZR.png



AORIZvib6x3RaA82.png



cPuxKnJCdcTotM5k.png



## #02. 이미지 데이터 전처리

### [1] 이미지 전처리 생성

```
image_gen = ImageDataGenerator(  
    rescale=1.0 / 255, # 정규화(색상값을 0~1사이로 변경함)  
    rotation_range=30, # 이미지 무작위 회전 (30도 이내)  
    width_shift_range=0.2, # 가로 방향 이동 범위 (무작위 20% 이내)  
    height_shift_range=0.2, # 세로 방향 이동 범위 (무작위 20% 이내)  
    shear_range=0.2, # 층 밀리기 강도 (무작위 20% 이내)  
    zoom_range=0.2, # 줌 범위 (무작위 20% 이내)  
    brightness_range=[0.5, 1.0], # 이미지 밝기  
    horizontal_flip=True, # 수평 뒤집기  
    vertical_flip=True, # 수직 뒤집기  
    fill_mode="nearest", # 이미지 변형 시 채울 픽셀  
    validation_split=0.2, # 검증 데이터 비율  
)
```

### [2] 이미지 데이터 전처리 수행

#### (1) 훈련용 이미지 데이터 생성

```
classes = os.listdir(extract_dir)  
classes
```

```
['paper', 'rock', 'scissors']
```

```
train_set = image_gen.flow_from_directory(  
    extract_dir, # 이미지 파일이 위치한 폴더  
    classes=classes, # 분류할 클래스명  
    batch_size=16, # 배치 사이즈  
    class_mode="categorical", # 다항분류용임을 명시(binary or categorical)  
    target_size=(64, 64), # 변환될 이미지 해상도  
    shuffle=True, # 이미지 섞기  
    color_mode="rgb", # 컬러 이미지  
    seed=get_random_state(), # 랜덤 시드값  
    subset="training", # 훈련용 데이터 생성임을 명시  
)
```

```
train_set.class_indices
```

Found 1751 images belonging to 3 classes.

```
{'paper': 0, 'rock': 1, 'scissors': 2}
```

## (2) 검증용 데이터 생성

```
test_set = image_gen.flow_from_directory(  
    extract_dir, # 이미지 파일이 위치한 폴더  
    classes=classes, # 분류할 클래스명  
    batch_size=16, # 배치 사이즈  
    class_mode="categorical", # 다항분류용임을 명시(binary or categorical)  
    target_size=(64, 64), # 변환될 이미지 해상도  
    shuffle=True, # 이미지 섞기  
    color_mode="rgb", # 컬러 이미지  
    seed=get_random_state(), # 랜덤 시드값  
    subset="validation", # 검증용 데이터 생성임을 명시  
)  
  
test_set.class_indices
```

Found 437 images belonging to 3 classes.

```
{'paper': 0, 'rock': 1, 'scissors': 2}
```

## #03. 훈련 모델 적합

### [1] 모델 정의하기

```
model = tf_create(  
    layer=[  
        # cnn (1)  
        {  
            "type": "conv2d",  
            "filters": 16,  
            "kernel_size": 6,  
            "padding": "same",  
            "input_shape": (64, 64, 3),  
        },  
        {"type": "batchnorm"},  
        {"type": "activation", "function": "relu"},  
        {"type": "maxpooling", "pool_size": (2, 2)},  
        {"type": "dropout", "rate": 0.1},  
        # cnn (2)  
        {  
            "type": "conv2d",  
            "filters": 32,  
            "kernel_size": 5,  
            "padding": "same",  
        },  
    ],  
)
```

```

{"type": "batchnorm"},
{"type": "activation", "function": "relu"},
{"type": "maxpooling", "pool_size": (2, 2)},
{"type": "dropout", "rate": 0.1},
# cnn (3)
{
    "type": "conv2d",
    "filters": 64,
    "kernel_size": 4,
    "padding": "same",
},
{"type": "batchnorm"},
{"type": "activation", "function": "relu"},
{"type": "maxpooling", "pool_size": (2, 2)},
{"type": "dropout", "rate": 0.1},
# cnn (4)
{
    "type": "conv2d",
    "filters": 128,
    "kernel_size": 3,
    "padding": "same",
},
{"type": "batchnorm"},
{"type": "activation", "function": "relu"},
{"type": "maxpooling", "pool_size": (2, 2)},
{"type": "dropout", "rate": 0.1},
# 단일층
{"type": "flatten"},
{"type": "dense", "units": 64},
{"type": "batchnorm"},
{"type": "activation", "function": "relu"},
{"type": "dense", "units": 3}, # 출력층의 수는 클래스의 수와 동일해야 함
{"type": "batchnorm"},
{"type": "activation", "function": "softmax"}, # 다중클래스 분류용 활성화 함수
],
optimizer="adam",
loss="categorical_crossentropy", # 다중클래스 분류용 손실함수
metrics=["acc"],
)

```

```
model.summary()
```

```

{'type': 'conv2d', 'filters': 16, 'kernel_size': 6, 'padding': 'same', 'input_shape':
{'type': 'batchnorm'}
{'type': 'activation', 'function': 'relu'}
{'type': 'maxpooling', 'pool_size': (2, 2)}
{'type': 'dropout', 'rate': 0.1}
{'type': 'conv2d', 'filters': 32, 'kernel_size': 5, 'padding': 'same'}
{'type': 'batchnorm'}
{'type': 'activation', 'function': 'relu'}
{'type': 'maxpooling', 'pool_size': (2, 2)}
{'type': 'dropout', 'rate': 0.1}
{'type': 'conv2d', 'filters': 64, 'kernel_size': 4, 'padding': 'same'}
{'type': 'batchnorm'}
{'type': 'activation', 'function': 'relu'}
{'type': 'maxpooling', 'pool_size': (2, 2)}

```

```

{'type': 'dropout', 'rate': 0.1}
{'type': 'conv2d', 'filters': 128, 'kernel_size': 3, 'padding': 'same'}
{'type': 'batchnorm'}
{'type': 'activation', 'function': 'relu'}
{'type': 'maxpooling', 'pool_size': (2, 2)}
{'type': 'dropout', 'rate': 0.1}
{'type': 'flatten'}
{'type': 'dense', 'units': 64}
{'type': 'batchnorm'}
{'type': 'activation', 'function': 'relu'}
{'type': 'dense', 'units': 3}
{'type': 'batchnorm'}
{'type': 'activation', 'function': 'softmax'}

```

**Model: "sequential"**

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 16)	1,744
batch_normalization (BatchNormalization)	(None, 64, 64, 16)	64
activation (Activation)	(None, 64, 64, 16)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 16)	0
dropout (Dropout)	(None, 32, 32, 16)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	12,832
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 32)	128
activation_1 (Activation)	(None, 32, 32, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
dropout_1 (Dropout)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	32,832
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
activation_2 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_2 (Dropout)	(None, 8, 8, 64)	0
conv2d_3 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 128)	512
activation_3 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_3 (Dropout)	(None, 4, 4, 128)	0

flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 64)	131,136
batch_normalization_4 (BatchNormalization)	(None, 64)	256
activation_4 (Activation)	(None, 64)	0
dense_1 (Dense)	(None, 3)	195
batch_normalization_5 (BatchNormalization)	(None, 3)	12
activation_5 (Activation)	(None, 3)	0

Total params: 253,823 (991.50 KB)

Trainable params: 253,209 (989.10 KB)

Non-trainable params: 614 (2.40 KB)

## [2] 학습하기





























```
%%time

batch_size = 16

result = tf_train(
    model=model,
    x_train=train_set,
    x_test=test_set,
    epochs=1000,
    steps_per_epoch=train_set.samples // batch_size,
    validation_steps=test_set.samples // batch_size,
    verbose=1
)

tf_result(result)
```

```
Epoch 1/1000
109/109 ————— 18s 127ms/step - acc: 0.5927 - loss: 0.9202 - val_acc: 0.3
Epoch 2/1000
109/109 ————— 0s 511us/step - acc: 0.9375 - loss: 0.3634 - val_acc: 0.40
Epoch 3/1000
109/109 ————— 6s 55ms/step - acc: 0.8823 - loss: 0.4352 - val_acc: 0.333
Epoch 4/1000
109/109 ————— 0s 287us/step - acc: 0.8750 - loss: 0.4013 - val_acc: 0.40
Epoch 5/1000
109/109 ————— 6s 55ms/step - acc: 0.9446 - loss: 0.3185 - val_acc: 0.432
Epoch 6/1000
109/109 ————— 0s 287us/step - acc: 0.9375 - loss: 0.2597 - val_acc: 0.80
Epoch 7/1000
109/109 ————— 7s 62ms/step - acc: 0.9454 - loss: 0.2904 - val_acc: 0.523
```

```
Epoch 8/1000
109/109  0s 306us/step - acc: 0.9375 - loss: 0.2407 - val_acc: 0.60
Epoch 9/1000
109/109  6s 56ms/step - acc: 0.9502 - loss: 0.2612 - val_acc: 0.620
Epoch 10/1000
109/109  0s 222us/step - acc: 0.9375 - loss: 0.2482 - val_acc: 0.40
Epoch 11/1000
109/109  7s 61ms/step - acc: 0.9534 - loss: 0.2260 - val_acc: 0.983
Epoch 12/1000
109/109  0s 213us/step - acc: 1.0000 - loss: 0.2124 - val_acc: 1.00
Epoch 13/1000
109/109  7s 58ms/step - acc: 0.9691 - loss: 0.1988 - val_acc: 0.958
Epoch 14/1000
109/109  0s 185us/step - acc: 0.9375 - loss: 0.1948 - val_acc: 0.80
Epoch 15/1000
109/109  7s 62ms/step - acc: 0.9668 - loss: 0.1800 - val_acc: 0.893
Epoch 16/1000
  1/109  1s 13ms/step - acc: 1.0000 - loss: 0.1819
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.000100000000474974513.
109/109  0s 185us/step - acc: 1.0000 - loss: 0.1819 - val_acc: 0.80
Epoch 17/1000
109/109  6s 57ms/step - acc: 0.9676 - loss: 0.1673 - val_acc: 0.983
Epoch 18/1000
109/109  0s 167us/step - acc: 1.0000 - loss: 0.0890 - val_acc: 1.00
Epoch 19/1000
109/109  6s 57ms/step - acc: 0.9775 - loss: 0.1570 - val_acc: 0.979
Epoch 20/1000
109/109  0s 259us/step - acc: 0.9375 - loss: 0.2398 - val_acc: 1.00
Epoch 21/1000
109/109  7s 59ms/step - acc: 0.9900 - loss: 0.1349 - val_acc: 0.979
Epoch 22/1000
109/109  0s 250us/step - acc: 0.9375 - loss: 0.1313 - val_acc: 1.00
Epoch 23/1000
109/109  7s 58ms/step - acc: 0.9911 - loss: 0.1339 - val_acc: 0.960
Epoch 24/1000
109/109  0s 222us/step - acc: 0.9375 - loss: 0.2137 - val_acc: 1.00
Epoch 25/1000
109/109  7s 57ms/step - acc: 0.9859 - loss: 0.1279 - val_acc: 0.953
Epoch 26/1000
109/109  0s 176us/step - acc: 1.0000 - loss: 0.1224 - val_acc: 1.00
Epoch 27/1000
109/109  6s 57ms/step - acc: 0.9814 - loss: 0.1482 - val_acc: 0.993
Epoch 28/1000
109/109  0s 176us/step - acc: 1.0000 - loss: 0.0956 - val_acc: 1.00
Epoch 29/1000
109/109  0s 47ms/step - acc: 0.9719 - loss: 0.1548
Epoch 29: ReduceLROnPlateau reducing learning rate to 1.00000000474974514e-05.
109/109  7s 57ms/step - acc: 0.9720 - loss: 0.1546 - val_acc: 0.981
Epoch 30/1000
109/109  0s 185us/step - acc: 1.0000 - loss: 0.1006 - val_acc: 1.00
Epoch 31/1000
109/109  7s 59ms/step - acc: 0.9834 - loss: 0.1287 - val_acc: 0.986
Epoch 32/1000
109/109  0s 278us/step - acc: 1.0000 - loss: 0.1391 - val_acc: 1.00
Epoch 33/1000
109/109  6s 56ms/step - acc: 0.9830 - loss: 0.1305 - val_acc: 0.990
```

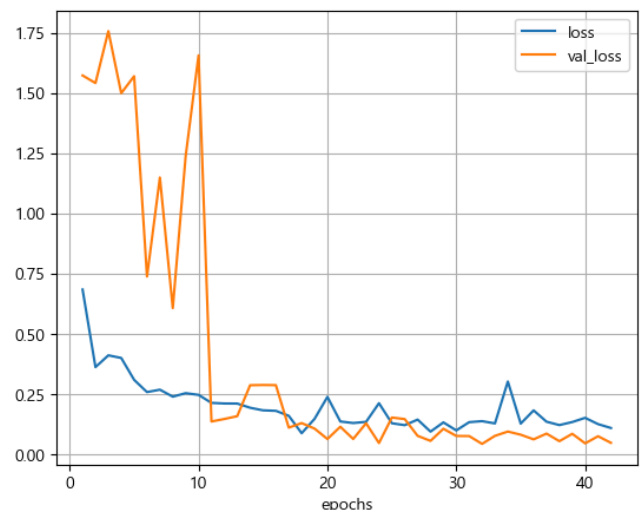
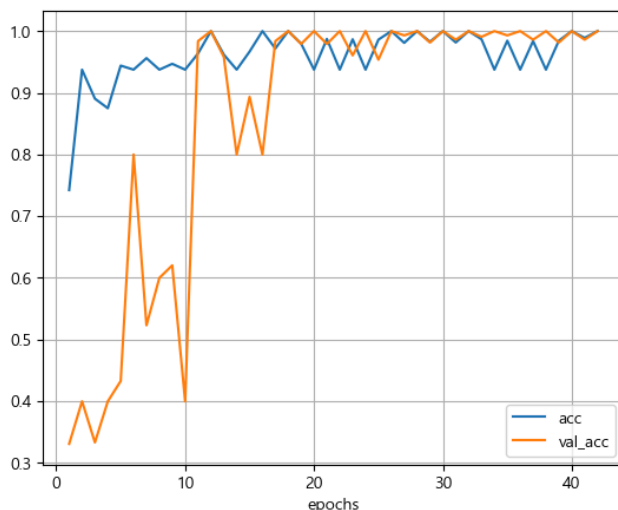


```

Epoch 34/1000
109/109 ————— 0s 185us/step - acc: 0.9375 - loss: 0.3036 - val_acc: 1.00
Epoch 35/1000
109/109 ————— 6s 56ms/step - acc: 0.9857 - loss: 0.1248 - val_acc: 0.993
Epoch 36/1000
109/109 ————— 0s 185us/step - acc: 0.9375 - loss: 0.1838 - val_acc: 1.00
Epoch 37/1000
109/109 ————— 0s 45ms/step - acc: 0.9822 - loss: 0.1399
Epoch 37: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.
109/109 ————— 6s 56ms/step - acc: 0.9823 - loss: 0.1399 - val_acc: 0.986
Epoch 38/1000
109/109 ————— 0s 185us/step - acc: 0.9375 - loss: 0.1227 - val_acc: 1.00
Epoch 39/1000
109/109 ————— 6s 56ms/step - acc: 0.9851 - loss: 0.1337 - val_acc: 0.981
Epoch 40/1000
109/109 ————— 0s 185us/step - acc: 1.0000 - loss: 0.1529 - val_acc: 1.00
Epoch 41/1000
109/109 ————— 6s 56ms/step - acc: 0.9867 - loss: 0.1325 - val_acc: 0.986
Epoch 42/1000
  1/109 ————— 2s 22ms/step - acc: 1.0000 - loss: 0.1106
Epoch 42: ReduceLROnPlateau reducing learning rate to 1.0000001111620805e-07.
109/109 ————— 0s 185us/step - acc: 1.0000 - loss: 0.1106 - val_acc: 1.00
Epoch 42: early stopping
Restoring model weights from the end of the best epoch: 32.

```

	acc	loss
train	0.99486	0.0666689
test	0.98627	0.0877104



CPU times: total: 8min 23s

Wall time: 2min 36s

## #04. 학습 결과 확인

### [1] 검증 데이터의 라벨 확인

```
y = test_set.classes
y
```

[illegible]

## [2] 검증 데이터에 대한 예측값

```
y_pred_proba = model.predict(test_set)
y_pred_proba
```

28/28  1s 43ms/step

```
array([[0.01205405, 0.97127783, 0.01666812],
       [0.01011027, 0.02459688, 0.96529293],
       [0.9426583 , 0.03240826, 0.0249335 ],
       ...,
       [0.01544976, 0.0221088 , 0.9624413 ],
       [0.82828116, 0.02427813, 0.14744073],
       [0.009068 , 0.02585837, 0.9650736 ]], dtype=float32)
```

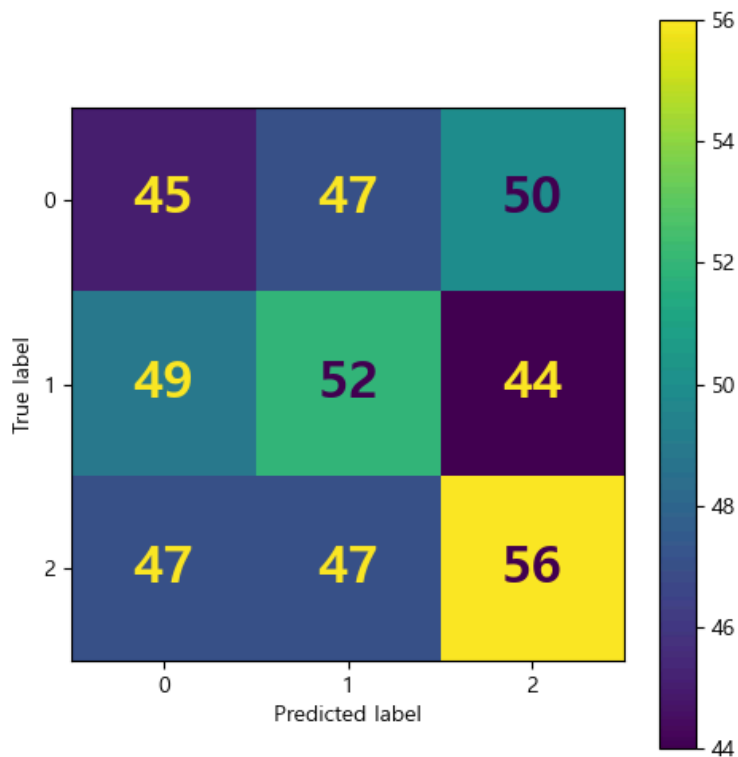
```
y_pred = np.argmax(y_pred_proba, axis=1)
y_pred
```

```
array([1, 2, 0, 0, 1, 1, 0, 1, 0, 2, 1, 0, 0, 2, 1, 1, 0, 1, 0, 2, 0, 1,
       0, 2, 2, 2, 0, 1, 1, 2, 0, 1, 1, 0, 1, 0, 0, 0, 2, 1, 1, 1, 1, 0,
       1, 0, 0, 2, 1, 2, 0, 1, 1, 2, 1, 1, 1, 0, 0, 1, 1, 2, 2, 1, 0, 0,
       2, 2, 2, 0, 1, 2, 2, 1, 1, 0, 0, 2, 1, 2, 2, 2, 2, 2, 1, 1, 0, 2,
       2, 0, 0, 0, 2, 0, 2, 1, 2, 0, 1, 1, 2, 0, 0, 2, 2, 2, 1, 2, 0, 0,
       0, 2, 1, 1, 1, 1, 1, 2, 0, 2, 2, 2, 2, 0, 1, 2, 1, 0, 2, 0, 1, 2,
       2, 0, 2, 1, 2, 2, 0, 0, 2, 2, 0, 0, 1, 1, 0, 2, 2, 2, 0, 2, 1, 0])
```

```
1, 1, 2, 2, 2, 0, 1, 0, 2, 0, 1, 0, 2, 1, 2, 0, 2, 2, 0, 2, 2, 2,
2, 1, 0, 2, 1, 1, 1, 1, 2, 1, 1, 1, 0, 0, 2, 0, 1, 1, 1, 1, 1,
1, 0, 0, 1, 0, 1, 2, 2, 1, 0, 1, 2, 0, 2, 1, 0, 2, 1, 1, 1, 0, 1,
0, 1, 2, 0, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 0, 0, 1, 0, 2, 1, 0,
2, 0, 1, 1, 0, 2, 2, 2, 0, 0, 0, 2, 0, 2, 0, 2, 1, 1, 0, 2, 1, 0,
2, 1, 0, 2, 1, 1, 0, 2, 0, 2, 0, 0, 0, 2, 0, 0, 2, 0, 1, 0, 1, 0,
0, 2, 1, 0, 0, 1, 0, 2, 1, 2, 0, 1, 0, 1, 0, 0, 2, 1, 0, 1, 2, 2,
2, 2, 1, 0, 1, 0, 0, 0, 0, 1, 2, 2, 1, 2, 2, 1, 0, 0, 1, 0, 2, 0,
1, 2, 0, 0, 2, 2, 1, 0, 1, 2, 0, 2, 1, 2, 2, 0, 1, 2, 1, 2, 1, 0,
0, 0, 1, 0, 2, 1, 0, 2, 1, 2, 0, 1, 2, 2, 2, 0, 1, 0, 0, 2, 1, 2,
2, 2, 0, 0, 1, 2, 2, 1, 1, 2, 2, 1, 2, 0, 2, 1, 1, 2, 1, 1, 0, 1,
2, 1, 0, 2, 1, 2, 2, 2, 2, 0, 1, 0, 0, 2, 0, 1, 0, 0, 1, 2, 2, 1,
2, 1, 0, 1, 0, 0, 2, 1, 2, 1, 2, 1, 2, 0, 1, 2, 2, 0, 2],
dtype=int64)
```

### [3] 혼동 행렬

```
my_confusion_matrix(y, y_pred, figsize=(5, 5), dpi=100)
```



## #05. 학습 결과 적용

### [1] 임의의 이미지 가져오기

```
가위_img = load_image("res/가위.jpg")
가위_img
```



```
바위_img = load_image("res/바위.jpg")  
바위_img
```



```
보_img = load_image("res/보.jpg")  
보_img
```



## [2] 이미지 전처리

```

# 훈련 데이터와 동일한 크기로 리사이즈
가위_tune = tune_image(가위_img, size=(64, 64), contrast=1.5)
바위_tune = tune_image(바위_img, size=(64, 64), contrast=1.5)
보_tune = tune_image(보_img, size=(64, 64), contrast=1.5)

# 이미지 데이터 변환
가위_flow = image_gen.flow(np.array([가위_tune]))
바위_flow = image_gen.flow(np.array([바위_tune]))
보_flow = image_gen.flow(np.array([보_tune]))




# 예측값 생성
가위_pred_proba = model.predict(가위_flow)
가위_pred = np.argmax(가위_pred_proba)
print(가위_pred_proba, 가위_pred)

바위_pred_proba = model.predict(바위_flow)
바위_pred = np.argmax(바위_pred_proba)
print(바위_pred_proba, 바위_pred)

보_pred_proba = model.predict(보_flow)
보_pred = np.argmax(보_pred_proba)
print(보_pred_proba, 보_pred)

```

```

1/1  0s 159ms/step
[[0.55396247 0.3398054 0.10623207]] 0
1/1  0s 25ms/step
[[0.14617248 0.75611955 0.09770793]] 1
1/1  0s 24ms/step
[[0.4540972 0.46823904 0.07766376]] 1

```

### [3] 예측 결과 확인

```

fig, ax = plt.subplots(1, 3, figsize=(15, 5), dpi=100)

ax[0].imshow(가위_img)
ax[0].axis("off")
ax[0].set_title(classes[가위_pred])

ax[1].imshow(바위_img)
ax[1].axis("off")
ax[1].set_title(classes[바위_pred])

ax[2].imshow(보_img)
ax[2].axis("off")
ax[2].set_title(classes[보_pred])

plt.show()
plt.close()

```

paper



rock



rock

