In [1]:
```
import tensorflow as tf
import numpy as np

print(tf.__version__)
print(np.__version__)
```

```
2.6.0
1.22.2
```

Data Load & Resize

In [2]:
```
from PIL import Image
import glob
import os

print("PIL 라이브러리 import 완료!")
```

```
PIL 라이브러리 import 완료!
```

In [3]:
```
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 이미지가 저장된 디렉토리 아래의 모든 jpg 파일을 읽어들여서
image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/scissor"
resize_images(image_dir_path)

print("가위 이미지 resize 완료!")
```

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

In [4]:
```
image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/rock"

resize_images(image_dir_path)

print("바위 이미지 resize 완료!")
```

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

In [5]:
```
image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/paper"
resize_images(image_dir_path)

print("보 이미지 resize 완료!")
```

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

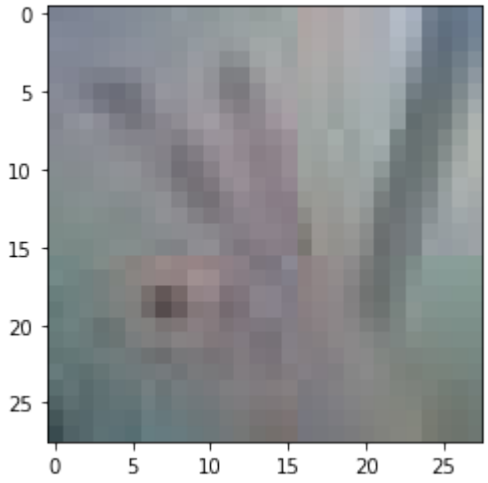
load_data

In [6]:
```
import numpy as np

def load_data(img_path, number_of_data=300):  # 가위바위보 이미지 개수 총합에 주의하
    # 가위 : 0, 바위 : 1, 보 : 2
    img_size=28
    color=3
    #이미지 데이터와 라벨(가위 : 0, 바위 : 1, 보 : 2) 데이터를 담을 행렬(matrix) 영역을 생
    imgs=np.zeros(number_of_data*img_size*img_size*color,dtype=np.int32).res
    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 = os.getenv("HOME") + "/aiffel/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))
```

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

In [7]:
```
import matplotlib.pyplot as plt

plt.imshow(x_train[0])
print('라벨: ', y_train[0])
```

```
라벨:  0
```



DL Network 설계

In [12]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np

# 하이퍼파라미터 option 1.
n_channel_1=16
n_channel_2=32
n_dense=32
n_train_epoch=10

model=keras.models.Sequential()
model.add(keras.layers.Conv2D(n_channel_1, (3,3), activation='relu', input_s
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'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 16)        448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                25632
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 99
=================================================================
Total params: 30,819
Trainable params: 30,819
Non-trainable params: 0
_________________________________________________________________
```

DL Network 학습 시키기

In [13]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 훈련
model.fit(x_train, y_train, epochs=n_train_epoch)
```

```
Epoch 1/10
10/10 [==============================] - 1s 16ms/step - loss: 27.4018 - acc
uracy: 0.3433
Epoch 2/10
10/10 [==============================] - 0s 20ms/step - loss: 6.5183 - accu
racy: 0.3933
Epoch 3/10
10/10 [==============================] - 0s 17ms/step - loss: 5.1546 - accu
racy: 0.3700
Epoch 4/10
10/10 [==============================] - 0s 17ms/step - loss: 2.1869 - accu
racy: 0.3733
Epoch 5/10
10/10 [==============================] - 0s 17ms/step - loss: 1.8398 - accu
racy: 0.4133
Epoch 6/10
10/10 [==============================] - 0s 19ms/step - loss: 1.3513 - accu
racy: 0.4233
Epoch 7/10
10/10 [==============================] - 0s 20ms/step - loss: 1.5213 - accu
racy: 0.4533
Epoch 8/10
10/10 [==============================] - 0s 19ms/step - loss: 1.6081 - accu
racy: 0.4267
Epoch 9/10
10/10 [==============================] - 0s 18ms/step - loss: 1.1531 - accu
racy: 0.5133
Epoch 10/10
10/10 [==============================] - 0s 20ms/step - loss: 1.5141 - accu
racy: 0.4667
```

Out[13]:
```
<keras.callbacks.History at 0x781b89705a00>
```

얼마나 잘만들었는지 확인하기(test)

In [14]:
```
image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/test/sciss
resize_images(image_dir_path)

image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/test/rock"
resize_images(image_dir_path)

image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/test/paper"
resize_images(image_dir_path)

image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/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))
```

```
100  images to be resized.
100  images resized.
100  images to be resized.
100  images resized.
100  images to be resized.
100  images resized.
학습데이터(x_train)의 이미지 개수는 300 입니다.
x_test shape: (300, 28, 28, 3)
y_test shape: (300,)
```

In [15]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print("test_loss: {} ".format(test_loss))
print("test_accuracy: {}".format(test_accuracy))
```

```
10/10 - 0s - loss: 4.8252 - accuracy: 0.3400
test_loss: 4.82517671585083
test_accuracy: 0.3400000035762787
```

2nd try

In [27]:
```
# 하이퍼파라미터 option 2.
n_channel_1=16
n_channel_2=32
n_dense=36
n_train_epoch=10

model=keras.models.Sequential()
model.add(keras.layers.Conv2D(n_channel_1, (3,3), activation='relu', input_s
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'))

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 26, 26, 16)        448
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 11, 11, 32)        4640
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_8 (Dense)              (None, 36)                28836
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 111
=================================================================
Total params: 34,035
Trainable params: 34,035
Non-trainable params: 0
_________________________________________________________________
```

In [28]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
# 모델 훈련
model.fit(x_train, y_train, epochs=n_train_epoch)
```

```
Epoch 1/10
10/10 [==============================] - 1s 17ms/step - loss: 28.3330 - acc
uracy: 0.3200
Epoch 2/10
10/10 [==============================] - 0s 20ms/step - loss: 8.7310 - accu
racy: 0.3533
Epoch 3/10
10/10 [==============================] - 0s 19ms/step - loss: 4.4289 - accu
racy: 0.3700
Epoch 4/10
10/10 [==============================] - 0s 18ms/step - loss: 1.7981 - accu
racy: 0.4267
Epoch 5/10
10/10 [==============================] - 0s 16ms/step - loss: 1.5435 - accu
racy: 0.3900
Epoch 6/10
10/10 [==============================] - 0s 17ms/step - loss: 1.5695 - accu
racy: 0.4000
Epoch 7/10
10/10 [==============================] - 0s 20ms/step - loss: 1.2096 - accu
racy: 0.5167
Epoch 8/10
10/10 [==============================] - 0s 19ms/step - loss: 1.0239 - accu
racy: 0.5433
Epoch 9/10
10/10 [==============================] - 0s 19ms/step - loss: 0.9321 - accu
racy: 0.6167
Epoch 10/10
10/10 [==============================] - 0s 20ms/step - loss: 0.8000 - accu
racy: 0.6533
```

Out[28]:
```
<keras.callbacks.History at 0x781b8b21e6a0>
```

In [29]:
```
image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/test/scisso
resize_images(image_dir_path)

image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/test/rock"
resize_images(image_dir_path)

image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/test/paper"
resize_images(image_dir_path)

image_dir_path = os.getenv("HOME") + "/aiffel/rock_scissor_paper/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))
```

```
100  images to be resized.
100  images resized.
100  images to be resized.
100  images resized.
100  images to be resized.
100  images resized.
학습데이터(x_train)의 이미지 개수는 300 입니다.
x_test shape: (300, 28, 28, 3)
y_test shape: (300,)
```

In [30]:
```
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print("test_loss: {} ".format(test_loss))
print("test_accuracy: {}".format(test_accuracy))
```

```
10/10 - 0s - loss: 4.1697 - accuracy: 0.4733
test_loss: 4.169671058654785
test_accuracy: 0.47333332896232605
```

3rd try: 데이터셋 추가 및 재분할

In [31]:
```python
# 데이터 로드 및 numpy 배열로 변환
def load_images_from_folder(folder_list):
    images = []
    for folder in folder_list:
        for filename in glob.glob(folder + '/*.jpg'):
            img = Image.open(filename).convert('RGB')
            img = img.resize((28, 28))
            img_array = np.array(img)
            images.append(img_array)
    return images

base_dir = '/aiffel/aiffel/rock_scissor_paper'
categories = {
    'paper': ['paper', 'paper_SE', 'paper_test'],
    'rock': ['rock', 'rock_SE', 'rock_test'],
    'scissor': ['scissor', 'scissor_SE', 'scissor_test']
}

all_images = {}
all_labels = []

# 각 카테고리별로 이미지 로드
for label, folders in categories.items():
    folder_paths = [os.path.join(base_dir, folder) for folder in folders]
    folder_paths.extend([os.path.join(base_dir, 'test', folder) for folder :
    images = load_images_from_folder(folder_paths)
    all_images[label] = images
    all_labels.extend([label] * len(images))

# 이미지와 레이블을 numpy 배열로 변환
all_images_np = np.concatenate([np.array(all_images[label]) for label in ca
all_labels_np = np.array(all_labels)
```

In [32]:
```python
from sklearn.model_selection import train_test_split

# 데이터 분할
x_train, x_test, y_train, y_test = train_test_split(all_images_np, all_labe

print("Training set size:", x_train.shape)
print("Testing set size:", x_test.shape)
```

```
Training set size: (888, 28, 28, 3)
Testing set size: (223, 28, 28, 3)
```

In [33]:
```python
# 모델 생성/정의
model = keras.models.Sequential([
    keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28,
    keras.layers.MaxPool2D(2, 2),
    keras.layers.Conv2D(32, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(2, 2),
    keras.layers.Flatten(),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 26, 26, 16)        448
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 11, 11, 32)        4640
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                25632
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 99
=================================================================
Total params: 30,819
Trainable params: 30,819
Non-trainable params: 0
_________________________________________________________________
```

In [34]:
```
# 모델 컴파일
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [37]:
```
from sklearn.preprocessing import LabelEncoder

# 레이블 인코더 생성
label_encoder = LabelEncoder()

# 레이블 인코딩 수행
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

print("Before encoding:", y_train[:5])
print("After encoding:", y_train_encoded[:5])
```

```
Before encoding: ['scissor' 'scissor' 'scissor' 'scissor' 'scissor']
After encoding: [2 2 2 2 2]
```

In [38]:
```
# 데이터 정규화
x_train_norm = x_train / 255.0
x_test_norm = x_test / 255.0
```

In [39]:
```
# 모델 훈련
model.fit(x_train_norm, y_train_encoded, epochs=10)
```

```
Epoch 1/10
28/28 [==============================] - 1s 20ms/step - loss: 1.0682 - accu
racy: 0.4268
Epoch 2/10
28/28 [==============================] - 1s 20ms/step - loss: 0.9895 - accu
racy: 0.5270
Epoch 3/10
28/28 [==============================] - 1s 21ms/step - loss: 0.8607 - accu
racy: 0.6295
Epoch 4/10
28/28 [==============================] - 1s 21ms/step - loss: 0.7414 - accu
racy: 0.7275
Epoch 5/10
28/28 [==============================] - 1s 21ms/step - loss: 0.6448 - accu
racy: 0.7466
Epoch 6/10
28/28 [==============================] - 1s 20ms/step - loss: 0.5506 - accu
racy: 0.7872
Epoch 7/10
28/28 [==============================] - 1s 20ms/step - loss: 0.4753 - accu
racy: 0.8367
Epoch 8/10
28/28 [==============================] - 1s 20ms/step - loss: 0.4279 - accu
racy: 0.8525
Epoch 9/10
28/28 [==============================] - 1s 22ms/step - loss: 0.3838 - accu
racy: 0.8637
Epoch 10/10
28/28 [==============================] - 1s 21ms/step - loss: 0.3369 - accu
racy: 0.8953
```

Out[39]:
```
<keras.callbacks.History at 0x781bb55ad4f0>
```

In [40]:
```
# 모델 평가
test_loss, test_accuracy = model.evaluate(x_test_norm, y_test_encoded, verbo
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_accuracy}")
```

```
7/7 - 0s - loss: 0.3566 - accuracy: 0.8789
Test loss: 0.3566466271877289
Test accuracy: 0.878923773765564
```

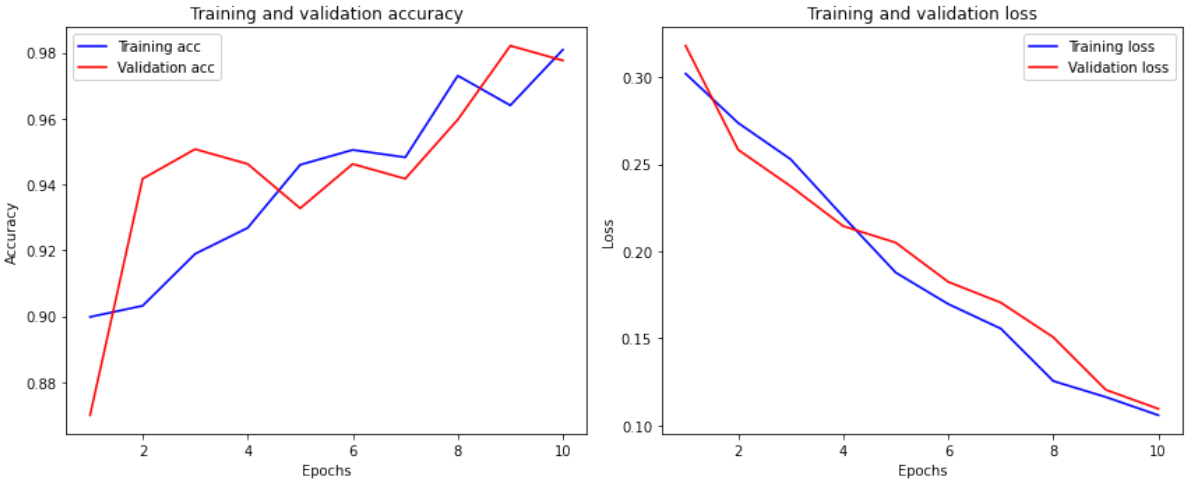
Visualization

In [41]:
```
# 모델 훈련
history = model.fit(x_train_norm, y_train_encoded, epochs=10, validation_dat
```

```
Epoch 1/10
28/28 [==============================] - 1s 24ms/step - loss: 0.3021 - accu
racy: 0.8998 - val_loss: 0.3181 - val_accuracy: 0.8700
Epoch 2/10
28/28 [==============================] - 1s 23ms/step - loss: 0.2738 - accu
racy: 0.9032 - val_loss: 0.2583 - val_accuracy: 0.9417
Epoch 3/10
28/28 [==============================] - 1s 24ms/step - loss: 0.2529 - accu
racy: 0.9189 - val_loss: 0.2374 - val_accuracy: 0.9507
Epoch 4/10
28/28 [==============================] - 1s 23ms/step - loss: 0.2200 - accu
racy: 0.9268 - val_loss: 0.2145 - val_accuracy: 0.9462
Epoch 5/10
28/28 [==============================] - 1s 23ms/step - loss: 0.1878 - accu
racy: 0.9459 - val_loss: 0.2050 - val_accuracy: 0.9327
Epoch 6/10
28/28 [==============================] - 1s 23ms/step - loss: 0.1697 - accu
racy: 0.9505 - val_loss: 0.1824 - val_accuracy: 0.9462
Epoch 7/10
28/28 [==============================] - 1s 23ms/step - loss: 0.1556 - accu
racy: 0.9482 - val_loss: 0.1705 - val_accuracy: 0.9417
Epoch 8/10
28/28 [==============================] - 1s 22ms/step - loss: 0.1255 - accu
racy: 0.9730 - val_loss: 0.1506 - val_accuracy: 0.9596
Epoch 9/10
28/28 [==============================] - 1s 23ms/step - loss: 0.1163 - accu
racy: 0.9640 - val_loss: 0.1204 - val_accuracy: 0.9821
Epoch 10/10
28/28 [==============================] - 1s 22ms/step - loss: 0.1058 - accu
racy: 0.9809 - val_loss: 0.1095 - val_accuracy: 0.9776
```

In [42]:

```python
# 훈련 과정에서의 손실과 정확도 그래프 그리기
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# 정확도 그래프
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# 손실 그래프
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



4th. 데이터 재추가 및 재분할

In [43]:
```
# 데이터 로드 및 numpy 배열로 변환
def load_images_from_folder(folder_list):
    images = []
    for folder in folder_list:
        for filename in glob.glob(folder + '/*.jpg'):
            img = Image.open(filename).convert('RGB')
            img = img.resize((28, 28))
            img_array = np.array(img, dtype=np.float32)
            images.append(img_array)
    return images

base_dir = '/aiffel/aiffel/rock_scissor_paper'
categories = {
    'paper': ['paper', 'paper_SE', 'paper_test', 'paper_SW'],
    'rock': ['rock', 'rock_SE', 'rock_test', 'rock_SW'],
    'scissor': ['scissor', 'scissor_SE', 'scissor_test', 'scissor_SW']
}

all_images = []
all_labels = []

# 각 카테고리별로 이미지 로드
for label, folders in categories.items():
    folder_paths = [os.path.join(base_dir, folder) for folder in folders]
    folder_paths.extend([os.path.join(base_dir, 'test', folder) for folder :
    images = load_images_from_folder(folder_paths)
    all_images.extend(images)
    all_labels.extend([label] * len(images))

# 이미지와 레이블을 numpy 배열로 변환
all_images_np = np.array(all_images)
all_labels_np = np.array(all_labels)

print("Loaded images shape:", all_images_np.shape)
print("Loaded labels shape:", all_labels_np.shape)
```

```
Loaded images shape: (1411, 28, 28, 3)
Loaded labels shape: (1411,)
```

In [44]:
```
# 데이터 정규화
all_images_np = all_images_np / 255.0

# 훈련 데이터와 테스트 데이터로 분할
x_train, x_test, y_train, y_test = train_test_split(all_images_np, all_labe
```

```
print("Training set size:", x_train.shape)
print("Testing set size:", x_test.shape)
```

```
Training set size: (1128, 28, 28, 3)
Testing set size: (283, 28, 28, 3)
```

In [45]:
```
# 레이블 인코더 생성
label_encoder = LabelEncoder()

# 레이블 인코딩 수행
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

print("Before encoding:", y_train[:5])
print("After encoding:", y_train_encoded[:5])
```

```
Before encoding: ['rock' 'scissor' 'paper' 'paper' 'paper']
After encoding: [1 2 0 0 0]
```

In [46]:
```
# 모델 생성/정의
model = keras.models.Sequential([
    keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(28, 28,
    keras.layers.MaxPool2D(2, 2),
    keras.layers.Conv2D(32, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D(2, 2),
    keras.layers.Flatten(),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 16)        448
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 32)        4640
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 5, 5, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_12 (Dense)             (None, 32)                25632
_________________________________________________________________
dense_13 (Dense)             (None, 3)                 99
=================================================================
Total params: 30,819
Trainable params: 30,819
Non-trainable params: 0
_________________________________________________________________
```

In [47]:
```
# 모델 컴파일
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [48]:
```
# 모델 훈련
history = model.fit(x_train, y_train_encoded, epochs=10, validation_data=(x_
```

```
Epoch 1/10
36/36 [==============================] - 1s 24ms/step - loss: 1.1002 - accu
racy: 0.3564 - val_loss: 1.0678 - val_accuracy: 0.4452
Epoch 2/10
36/36 [==============================] - 1s 22ms/step - loss: 1.0505 - accu
racy: 0.4592 - val_loss: 1.0186 - val_accuracy: 0.6007
Epoch 3/10
36/36 [==============================] - 1s 21ms/step - loss: 0.9757 - accu
racy: 0.5390 - val_loss: 0.9114 - val_accuracy: 0.5583
Epoch 4/10
36/36 [==============================] - 1s 21ms/step - loss: 0.8636 - accu
racy: 0.6321 - val_loss: 0.7784 - val_accuracy: 0.7244
Epoch 5/10
36/36 [==============================] - 1s 22ms/step - loss: 0.7362 - accu
racy: 0.6986 - val_loss: 0.7284 - val_accuracy: 0.6820
Epoch 6/10
36/36 [==============================] - 1s 22ms/step - loss: 0.6142 - accu
racy: 0.7713 - val_loss: 0.5996 - val_accuracy: 0.7668
Epoch 7/10
36/36 [==============================] - 1s 22ms/step - loss: 0.5213 - accu
racy: 0.8094 - val_loss: 0.5134 - val_accuracy: 0.8057
Epoch 8/10
36/36 [==============================] - 1s 22ms/step - loss: 0.4329 - accu
racy: 0.8617 - val_loss: 0.3937 - val_accuracy: 0.8834
Epoch 9/10
36/36 [==============================] - 1s 21ms/step - loss: 0.3527 - accu
racy: 0.8963 - val_loss: 0.3740 - val_accuracy: 0.8622
Epoch 10/10
36/36 [==============================] - 1s 22ms/step - loss: 0.3270 - accu
racy: 0.8972 - val_loss: 0.3200 - val_accuracy: 0.9117
```

In [49]:

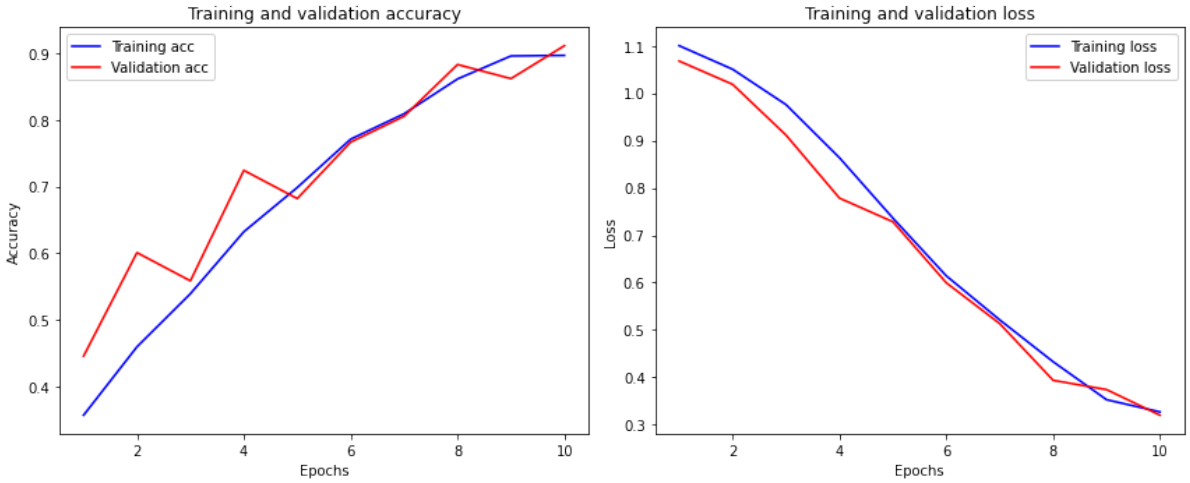
```
# 모델 평가
test_loss, test_accuracy = model.evaluate(x_test, y_test_encoded, verbose=2)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_accuracy}")
```

```
9/9 - 0s - loss: 0.3200 - accuracy: 0.9117
Test loss: 0.31995633244514465
Test accuracy: 0.9116607904434204
```

In [50]:

```
# 훈련 과정에서의 손실과 정확도 그래프
import matplotlib.pyplot as plt

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

# 정확도 그래프
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'b', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# 손실 그래프
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
```

```
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



+ Dropout, Batch Normalization

In [52]:

```
# 드롭아웃 및 배치 정규화 적용
model = keras.models.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(2, 2),
    keras.layers.Dropout(0.25),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(2, 2),
    keras.layers.Dropout(0.25),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(3, activation='softmax')
])

model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 26, 26, 32)        896
_________________________________________________________________
batch_normalization_2 (Batch (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                102464
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_17 (Dense)             (None, 3)                 195
=================================================================
Total params: 122,435
Trainable params: 122,243
Non-trainable params: 192
_________________________________________________________________
```

In [53]:
```
# 모델 컴파일
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [54]:
```
# 모델 훈련
history = model.fit(x_train, y_train_encoded, epochs=10, validation_data=(x_
```

```
Epoch 1/10
36/36 [==============================] - 3s 59ms/step - loss: 1.4637 - accu
racy: 0.4619 - val_loss: 1.0738 - val_accuracy: 0.4311
Epoch 2/10
36/36 [==============================] - 2s 52ms/step - loss: 0.8406 - accu
racy: 0.5842 - val_loss: 1.0355 - val_accuracy: 0.3640
Epoch 3/10
36/36 [==============================] - 2s 52ms/step - loss: 0.7201 - accu
racy: 0.6622 - val_loss: 1.1737 - val_accuracy: 0.3569
Epoch 4/10
36/36 [==============================] - 2s 52ms/step - loss: 0.6326 - accu
racy: 0.6968 - val_loss: 1.1811 - val_accuracy: 0.3569
Epoch 5/10
36/36 [==============================] - 2s 53ms/step - loss: 0.5435 - accu
racy: 0.7500 - val_loss: 1.0465 - val_accuracy: 0.4664
Epoch 6/10
36/36 [==============================] - 2s 52ms/step - loss: 0.5057 - accu
racy: 0.7686 - val_loss: 0.9580 - val_accuracy: 0.4947
Epoch 7/10
36/36 [==============================] - 2s 53ms/step - loss: 0.4547 - accu
racy: 0.7970 - val_loss: 0.8735 - val_accuracy: 0.6113
Epoch 8/10
36/36 [==============================] - 2s 52ms/step - loss: 0.4072 - accu
racy: 0.8378 - val_loss: 0.8225 - val_accuracy: 0.5830
Epoch 9/10
36/36 [==============================] - 2s 53ms/step - loss: 0.3476 - accu
racy: 0.8449 - val_loss: 0.6383 - val_accuracy: 0.7032
Epoch 10/10
36/36 [==============================] - 2s 53ms/step - loss: 0.3344 - accu
racy: 0.8590 - val_loss: 0.4744 - val_accuracy: 0.8021
```

In [55]:

```
# 모델 평가
test_loss, test_accuracy = model.evaluate(x_test, y_test_encoded, verbose=2)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_accuracy}")
```

```
9/9 - 0s - loss: 0.4744 - accuracy: 0.8021
Test loss: 0.47438690066337585
Test accuracy: 0.8021201491355896
```

In [56]:

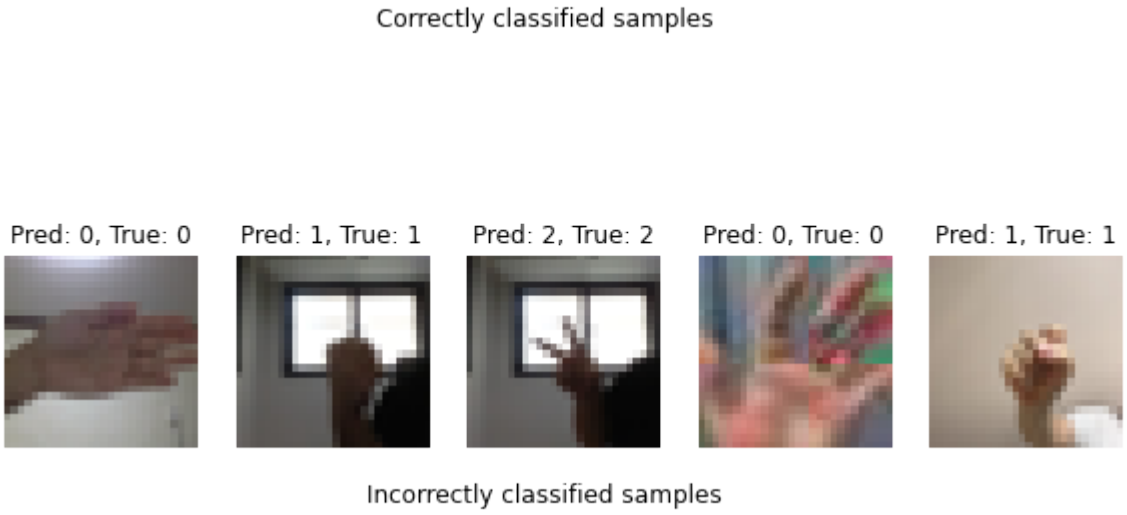
```
# 테스트 데이터에 대한 예측 생성
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)

# 올바르게 분류된 샘플과 잘못 분류된 샘플의 인덱스 찾기
correct_indices = np.where(y_pred_classes == y_test_encoded)[0]
incorrect_indices = np.where(y_pred_classes != y_test_encoded)[0]

print(f"올바르게 분류된 샘플 수: {len(correct_indices)}")
print(f"잘못 분류된 샘플 수: {len(incorrect_indices)}")
```

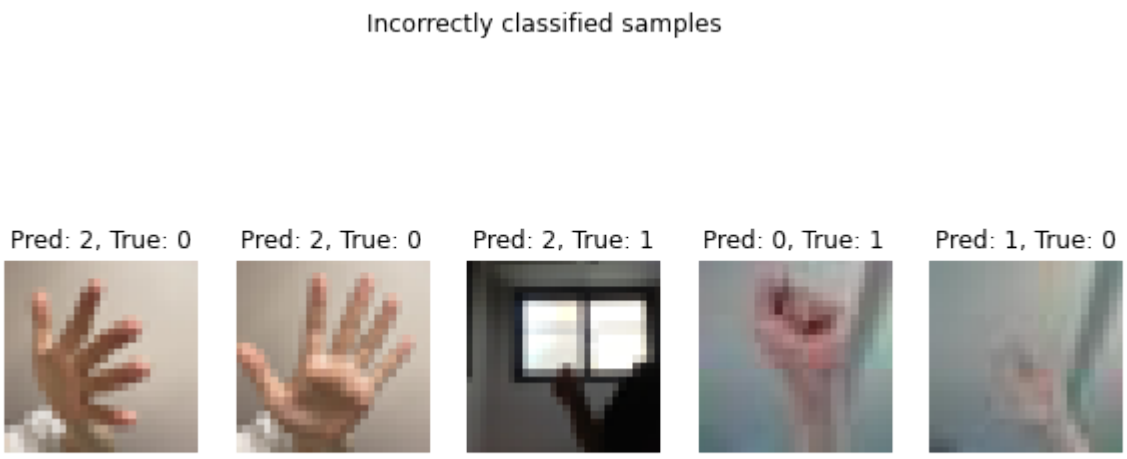
```
올바르게 분류된 샘플 수: 227
잘못 분류된 샘플 수: 56
```

In [57]:

```
import matplotlib.pyplot as plt

# 올바르게 분류된 샘플 시각화
plt.figure(figsize=(10, 5))
for i, idx in enumerate(correct_indices[:5]):
    plt.subplot(1, 5, i + 1)
    plt.imshow(x_test[idx])
    plt.title(f"Pred: {y_pred_classes[idx]}, True: {y_test_encoded[idx]}")
    plt.axis('off')
plt.suptitle("Correctly classified samples")
plt.show()
```

```
# 잘못 분류된 샘플 시각화
plt.figure(figsize=(10, 5))
for i, idx in enumerate(incorrect_indices[:5]):
    plt.subplot(1, 5, i + 1)
    plt.imshow(x_test[idx])
    plt.title(f"Pred: {y_pred_classes[idx]}, True: {y_test_encoded[idx]}")
    plt.axis('off')
plt.suptitle("Incorrectly classified samples")
plt.show()
```





In [ ]: