Copyright 2019 The TensorFlow Authors.

In [ ]:

```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

데이터가 포함된 zip 파일 다운받기

In [ ]:

```
#학습세트
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/rps.zip \
    -O /tmp/rps.zip

#테스트 및 검증세트
!wget --no-check-certificate \
    https://storage.googleapis.com/laurencemoroney-blog.appspot.com/rps-test-set.zip \
    -O /tmp/rps-test-set.zip
```

```
--2020-07-02 10:24:41--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/rps.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.124.128, 172.217.212.128, 172.
217.214.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.124.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 200682221 (191M) [application/zip]
Saving to: '/tmp/rps.zip'

/tmp/rps.zip        100%[===================>] 191.38M   114MB/s    in 1.7s

2020-07-02 10:24:42 (114 MB/s) - '/tmp/rps.zip' saved [200682221/200682221]

--2020-07-02 10:24:43--  https://storage.googleapis.com/laurencemoroney-blog.appspot.com/rps-test-
set.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 74.125.124.128, 172.217.212.128, 172.
217.214.128, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|74.125.124.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 29516758 (28M) [application/zip]
Saving to: '/tmp/rps-test-set.zip'

/tmp/rps-test-set.z 100%[===================>]  28.15M  94.1MB/s    in 0.3s

2020-07-02 10:24:44 (94.1 MB/s) - '/tmp/rps-test-set.zip' saved [29516758/29516758]
```

zip파일 라이브러리를 이용해 파일의 압출 해제

In [ ]:

```
import os
import zipfile

local_zip = '/tmp/rps.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')

#현재 작업 디렉토리에 압축 해제, 상위 디렉토리의 이름을 기준으로 이미지에 자동으로 레이블이 지정됨
zip_ref.extractall('/tmp/')
zip_ref.close()
```

```
local_zip = '/tmp/rps-test-set.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp/')
zip_ref.close()
```

가위바위보 디렉토리 생성

In [ ]:

```
rock_dir = os.path.join('/tmp/rps/rock')
paper_dir = os.path.join('/tmp/rps/paper')
scissors_dir = os.path.join('/tmp/rps/scissors')

print('total training rock images:', len(os.listdir(rock_dir)))
print('total training paper images:', len(os.listdir(paper_dir)))
print('total training scissors images:', len(os.listdir(scissors_dir)))

rock_files = os.listdir(rock_dir)
print(rock_files[:10])

paper_files = os.listdir(paper_dir)
print(paper_files[:10])

scissors_files = os.listdir(scissors_dir)
print(scissors_files[:10])
```

```
total training rock images: 840
total training paper images: 840
total training scissors images: 840
['rock04-035.png', 'rock06ck02-007.png', 'rock07-k03-069.png', 'rock06ck02-044.png', 'rock02-057.p
ng', 'rock02-101.png', 'rock07-k03-102.png', 'rock05ck01-078.png', 'rock02-029.png', 'rock04-070.p
ng']
['paper07-117.png', 'paper05-060.png', 'paper04-002.png', 'paper02-063.png', 'paper05-031.png', 'p
aper05-008.png', 'paper05-064.png', 'paper05-032.png', 'paper02-020.png', 'paper01-016.png']
['scissors02-077.png', 'scissors03-094.png', 'testscissors01-093.png', 'scissors04-020.png', 'scis
sors02-105.png', 'scissors02-037.png', 'testscissors01-038.png', 'scissors01-054.png',
'testscissors03-051.png', 'scissors03-067.png']
```

In [ ]:

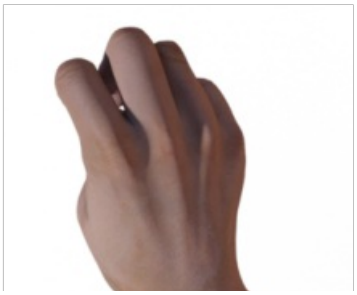
```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

pic_index = 2

next_rock = [os.path.join(rock_dir, fname)
                for fname in rock_files[pic_index-2:pic_index]]
next_paper = [os.path.join(paper_dir, fname)
                for fname in paper_files[pic_index-2:pic_index]]
next_scissors = [os.path.join(scissors_dir, fname)
                for fname in scissors_files[pic_index-2:pic_index]]

for i, img_path in enumerate(next_rock+next_paper+next_scissors):
  #print(img_path)
  img = mpimg.imread(img_path)
  plt.imshow(img)
  plt.axis('Off')
  plt.show()
```



In [ ]:

```
import tensorflow as tf
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator

#다운로드한 디렉토리에서 학습용 이미지를 생성하는 이미지 데이터 제너레이터를 생성
TRAINING_DIR = "/tmp/rps/"
training_datagen = ImageDataGenerator(
      rescale = 1./255,
     rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

VALIDATION_DIR = "/tmp/rps-test-set/"
validation_datagen = ImageDataGenerator(rescale = 1./255)

#트레이닝 제너레이터를 준비(학습 데이터 생성)
train_generator = training_datagen.flow_from_directory(
 TRAINING_DIR,
 target_size=(150,150),
 class_mode='categorical',
  batch_size=126
)

validation_generator = validation_datagen.flow_from_directory(
 VALIDATION_DIR,
 target_size=(150,150),
 class_mode='categorical',
  batch_size=126
)

#뉴럴 네트워크
model = tf.keras.models.Sequential([
    # 이미지 크기 150x150,
    # 첫 번째 convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # 두 번째 convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # 세 번째 convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # 네 번째 convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # 결과를 병합하여 DNN에 피드
    tf.keras.layers.Flatten(),
    #Dropout은 텐서 레이어로 가기 전에 뉴럴 중 일부를 버림으로써 뉴럴 네트워크의 효율성을 상승시킴
    tf.keras.layers.Dropout(0.5),
    # 512 뉴런의 숨겨진 layer
    tf.keras.layers.Dense(512, activation='relu'),
    #출력 레이어 : 3개의 뉴럴
    tf.keras.layers.Dense(3, activation='softmax')
])


model.summary()

#뉴럴 네트워크 컴파일
model.compile(loss = 'categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

#모델 핏을 호출해서 학습을 시작
history = model.fit(train_generator, epochs=25, steps_per_epoch=20, validation_data = validation_ge
nerator, verbose = 1, validation_steps=3)

model.save("rps.h5")
```

```
Found 2520 images belonging to 3 classes.
Found 372 images belonging to 3 classes.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539
=================================================================
Total params: 3,473,475
Trainable params: 3,473,475
Non-trainable params: 0
_________________________________________________________________
Epoch 1/25
20/20 [==============================] - 23s 1s/step - loss: 1.5435 - accuracy: 0.3393 - val_loss:
1.0904 - val_accuracy: 0.4812
Epoch 2/25
20/20 [==============================] - 23s 1s/step - loss: 1.1404 - accuracy: 0.3679 - val_loss:
1.0961 - val_accuracy: 0.3333
Epoch 3/25
20/20 [==============================] - 23s 1s/step - loss: 1.0718 - accuracy: 0.4206 - val_loss:
1.0490 - val_accuracy: 0.4704
Epoch 4/25
20/20 [==============================] - 23s 1s/step - loss: 0.9771 - accuracy: 0.5095 - val_loss:
0.9603 - val_accuracy: 0.3978
Epoch 5/25
20/20 [==============================] - 23s 1s/step - loss: 0.9661 - accuracy: 0.5488 - val_loss:
0.6271 - val_accuracy: 0.8763
Epoch 6/25
20/20 [==============================] - 23s 1s/step - loss: 0.7405 - accuracy: 0.6603 - val_loss:
0.5068 - val_accuracy: 0.6747
Epoch 7/25
20/20 [==============================] - 23s 1s/step - loss: 0.6593 - accuracy: 0.7111 - val_loss:
0.2501 - val_accuracy: 1.0000
Epoch 8/25
20/20 [==============================] - 23s 1s/step - loss: 0.5149 - accuracy: 0.7917 - val_loss:
0.5490 - val_accuracy: 0.6855
Epoch 9/25
20/20 [==============================] - 23s 1s/step - loss: 0.6053 - accuracy: 0.7552 - val_loss:
0.1144 - val_accuracy: 0.9758
Epoch 10/25
20/20 [==============================] - 23s 1s/step - loss: 0.4150 - accuracy: 0.8377 - val_loss:
0.1572 - val_accuracy: 0.9570
Epoch 11/25
20/20 [==============================] - 23s 1s/step - loss: 0.3159 - accuracy: 0.8694 - val_loss:
0.5614 - val_accuracy: 0.6694
Epoch 12/25
20/20 [==============================] - 23s 1s/step - loss: 0.2750 - accuracy: 0.8893 - val_loss:
0.1991 - val_accuracy: 0.9301
Epoch 13/25
20/20 [==============================] - 23s 1s/step - loss: 0.2215 - accuracy: 0.9139 - val_loss:
0.0763 - val_accuracy: 0.9812
Epoch 14/25
20/20 [==============================] - 23s 1s/step - loss: 0.2706 - accuracy: 0.8996 - val_loss:
0.0870 - val_accuracy: 0.9839
```

```
Epoch 15/25
20/20 [==============================] - 23s 1s/step - loss: 0.1515 - accuracy: 0.9464 - val_loss:
0.1441 - val_accuracy: 0.9167
Epoch 16/25
20/20 [==============================] - 23s 1s/step - loss: 0.1717 - accuracy: 0.9333 - val_loss:
0.0689 - val_accuracy: 0.9731
Epoch 17/25
20/20 [==============================] - 23s 1s/step - loss: 0.1480 - accuracy: 0.9520 - val_loss:
0.0413 - val_accuracy: 0.9731
Epoch 18/25
20/20 [==============================] - 23s 1s/step - loss: 0.1638 - accuracy: 0.9373 - val_loss:
0.0574 - val_accuracy: 0.9812
Epoch 19/25
20/20 [==============================] - 23s 1s/step - loss: 0.1256 - accuracy: 0.9611 - val_loss:
0.9084 - val_accuracy: 0.6371
Epoch 20/25
20/20 [==============================] - 23s 1s/step - loss: 0.1703 - accuracy: 0.9437 - val_loss:
0.0140 - val_accuracy: 1.0000
Epoch 21/25
20/20 [==============================] - 23s 1s/step - loss: 0.0890 - accuracy: 0.9659 - val_loss:
0.0286 - val_accuracy: 0.9892
Epoch 22/25
20/20 [==============================] - 23s 1s/step - loss: 0.1070 - accuracy: 0.9615 - val_loss:
0.1260 - val_accuracy: 0.9651
Epoch 23/25
20/20 [==============================] - 23s 1s/step - loss: 0.0881 - accuracy: 0.9738 - val_loss:
0.0318 - val_accuracy: 0.9758
Epoch 24/25
20/20 [==============================] - 23s 1s/step - loss: 0.3399 - accuracy: 0.9274 - val_loss:
0.0321 - val_accuracy: 0.9839
Epoch 25/25
20/20 [==============================] - 23s 1s/step - loss: 0.0620 - accuracy: 0.9802 - val_loss:
0.0285 - val_accuracy: 0.9892
```

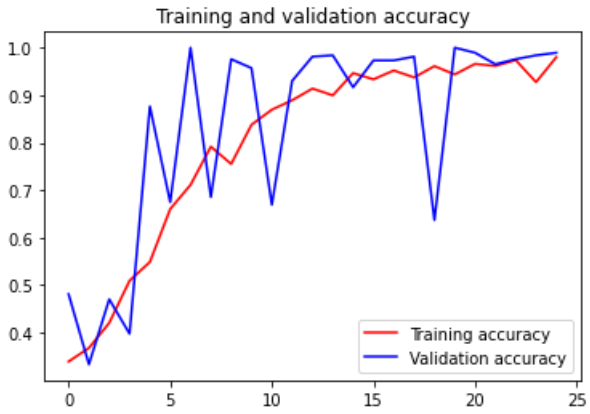
In [ ]:

```
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(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
#검증 데이터가 학습데이터보다 낮은 정확도를 보이는 것을 오버피팅(과적합)이라함.
#이전에 학습한 것을 파악하는데는 매우 능숙하지만 일반화에는 좋지 않은 경우 발생.
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()


plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```

In [ ]:

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # 이미지 예측
  path = fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])

  #이미지를 가지고 와서, 예측값을 반환
  classes = model.predict(images, batch_size=10)
  print(fn)
  print(classes)
```

```
Using TensorFlow backend.
```

Choose File No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving unnamed.png to unnamed.png
unnamed.png
[[0. 0. 1.]]
```

In [ ]:

```

```