배열과 차원

1차원 배열 (벡터)

```
import numpy as np
x = np.array([7, 9])
```

```
x.shape
```

1차원 → 2차원

```
np.expand_dims(x, 0)
```

1행 2열의 행렬로 변환

```
np.expand_dims(x, 1)
```

2행 1열의 행렬로 변환

2차원 배열 (행렬)

```
x = np.array([
    [1, 2, 3],
    [4, 5, 6]])
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

3차원 배열

```
x = np.array(
    [[[ 1,  2,  3,  4],
      [ 5,  6,  7,  8],
      [ 9, 10, 11, 12]],

     [[13, 14, 15, 16],
      [17, 18, 19, 20],
      [21, 22, 23, 24]]])
```

파일로 학습하기

파일 다운로드

```
!wget -c https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
!unzip -qq cats_and_dogs_filtered.zip
```

경로 설정

```
from pathlib import Path
```

데이터 폴더

```
data_root = Path('cats_and_dogs_filtered')
```

훈련용

```
train_dir = data_root / 'train'
```

테스트용

```
test_dir = data_root / 'validation'
```

열어 보기

```
import PIL.Image
```

```
PIL.Image.open(train_dir / 'cats' / 'cat.0.jpg')
```

데이터 크기

```
len(list(train_dir.glob('*/*.jpg')))
```

하이퍼파라미터

```
batch_size = 32
epochs = 15
img_height = 150
img_width = 150
```

training set 로딩

```
import tensorflow as tf
```

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  train_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

`validation_split=0.2` : 데이터 중 무작위로 20%는 validation 용으로 유보

`seed=123` : 데이터를 무작위로 고를 때의 난수 생성의 초기값을 고정.

validation set 로딩

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  train_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

`seed=123` : training set에서와 난수 생성을 동일하게 하도록 설정

범주 보기

```
train_ds.class_names
```

1개의 미니배치 보기

```
image_batch, labels_batch = next(iter(train_ds))
print(image_batch.shape)
print(labels_batch.shape)
```

캐시

```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

`.cache()` 데이터를 불러와서 메모리에 보존.

`.prefetch()` 데이터를 불러오는 과정을 병렬로 진행

전처리

전처리 함수를 정의.

```
def preproc(images, labels):
    images = images / 255.0
    images = tf.image.flip_left_right(images)

    # 흑백으로 변환(적용 여부 확인용, 실제로 학습시에는 삭제)
    images = tf.image.rgb_to_grayscale(images)

    return images, labels
```

전처리된 데이터셋 만들기

```
aug_ds = train_ds.map(preproc)
```

`.map` 으로 전처리 함수를 지정하면 데이터를 로딩할 때 자동으로 전처리를 적용한다

전처리 결과 확인

```
image_batch, labels_batch = next(iter(aug_ds))
print(image_batch.shape)
print(labels_batch.shape)
```

모형 정의

```
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D

model = tf.keras.Sequential([
    Conv2D(16, 3, padding='same', activation='relu',
           input_shape=(img_height, img_width ,3)),
    MaxPooling2D(),
    Conv2D(16, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

학습 설정

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

모형 요약

```
model.summary()
```

학습

```
history = model.fit(
  aug_ds,
  validation_data=val_ds,
)
```

테스트 데이터셋

불러오기

```
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
  test_dir,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

캐시 설정

```
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

성능 테스트

```
model.evaluate(test_ds)
```

혼동 행렬

예측된 범주와 실제 범주 비교

```
y_pred = []
y_test = []
threshold = 0.5
for images, labels in test_ds:
    probs = model.predict(images)
    prediction = tf.where(probs > 0.5, 1, 0)
    prediction = tf.squeeze(prediction).numpy().tolist()
    y_pred.extend(prediction)
    y_test.extend(labels.numpy().tolist())
```

혼동 행렬

```
import sklearn.metrics as m
```

혼동 행렬

```
m.confusion_matrix(y_test, y_pred)
```

혼동 행렬 기반의 지표

정확도

```
m.accuracy_score(y_test, y_pred)
```

정밀도

```
m.precision_score(y_test, y_pred)
```

재현도

```
m.recall_score(y_test, y_pred)
```

Batch Normalization

Batch Normalization

```
layer = tf.keras.layers.BatchNormalization()
```

학습시

```
x1 = tf.convert_to_tensor(
    [
        [1, 10, 100],
        [2, 20, 200],
        [3, 30, 300]
    ],
    =tf.float32)
```

```
layer(x1, training=True)
```

`model.fit`을 할 때는 자동으로 `training=True`로 사용

예측시

```
x2 = tf.convert_to_tensor(
    [
        [5, 10, -100],
        [6, 20, -200],
        [7, 30, -300]
    ],
    dtype=tf.float32)
```

```
layer(x1, training=False)
```

학습시 추정된 평균과 표준편차로 표준화

`model.predict`을 할 때는 자동으로 `training=False`로 사용

ResNet

텐서플로 허브

```
import tensorflow_hub as hub
resnet = hub.KerasLayer(
    "https://tfhub.dev/google/imagenet/resnet_v2_50/classification/4")
```

모형에 추가

```
model = tf.keras.Sequential([resnet])
model.build((None, 224, 224, 3))
```

이미지넷 레이블

```
import numpy as np

labels_path = tf.keras.utils.get_file(
    'ImageNetLabels.txt',
    'https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt')

imagenet_labels = np.array(open(labels_path).read().splitlines())
```

이미지 변환

```
img = x_train[1:2]
img = img / 255.0
img = tf.image.resize(img, (224, 224))
```

예측

```
probs = model.predict(img)
```

예측 결과의 레이블 확인

```
i = tf.argmax(probs, axis=-1).numpy().tolist()[0]
imagenet_labels[i]
```

전이 학습

분류기가 없는 모형

모형 주소가 앞의 ResNet 모형과 다름

```
resnet = hub.KerasLayer(
    "https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/4",
    trainable=False),
```

기존 모형 + 분류기

```
model = tf.keras.Sequential([
    resnet,
    tf.keras.layers.Dense(10, activation='softmax')
])

model.build([None, 224, 224, 3])  # Batch input shape.
```

학습 가능한 모형

```
resnet = hub.KerasLayer(
    "https://tfhub.dev/google/imagenet/resnet_v2_50/feature_vector/4",
    trainable=True)
```

ResNet 부분도 함께 학습시킬 때 사용