(DNN) 옷 이미지 분류

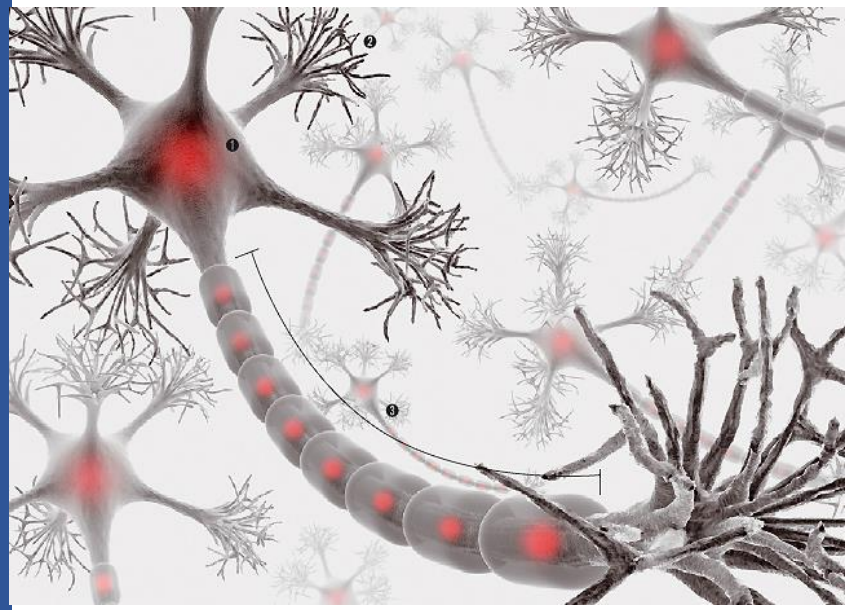
학습 목표

- 운동화나 셔츠 같은 옷 이미지를 분류하는 신경망 모델을 만들어 본다.

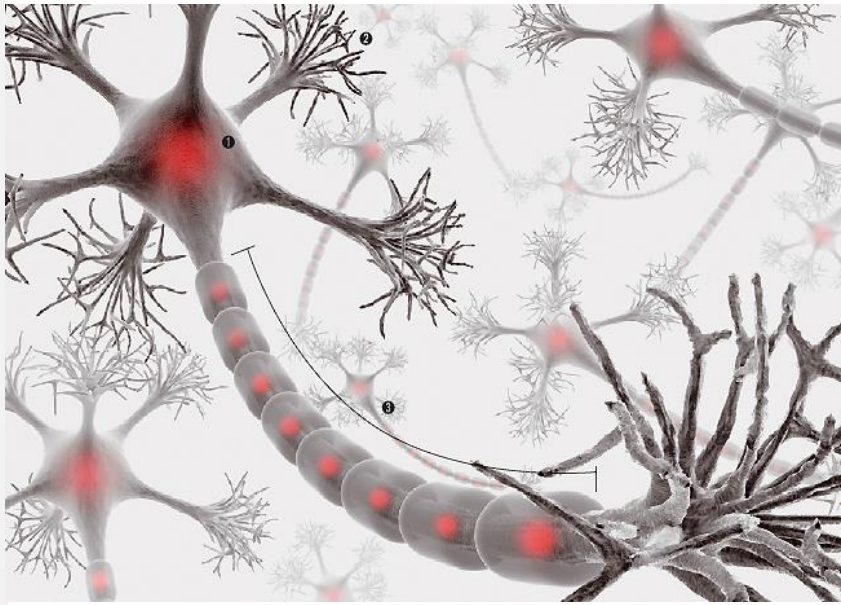
주요 내용

- 1. 문제 정의
- 2. 데이터 준비
- 3. 모델 정의 및 훈련, 검증

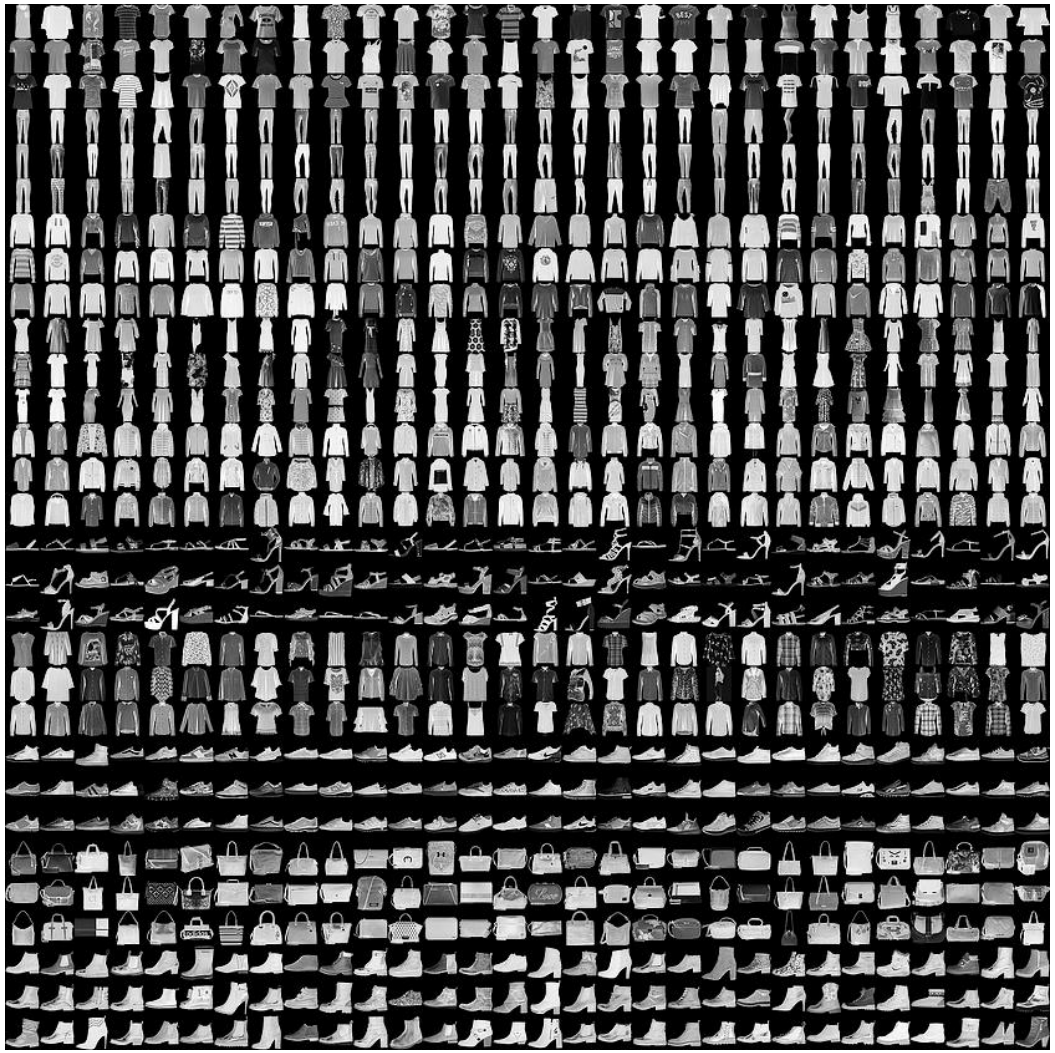
https://www.tensorflow.org/tutorials/keras/classification



1 문제 정의



Fashion MNIST Dataset



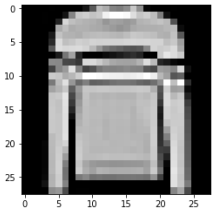
- 다양한 옷, 구두, 핸드백 등의 이미지 데이터 셋
- 10개의 category와 70,000개의 흑백 이미지로 구성
- 이미지 해상도는 28x28, 픽셀 값은 0과 255 사이
- 레이블(label)은 0에서 9까지의 정수

레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

https://github.com/zalandoresearch/fashion-mnist

Network 구성

28x28 이미지

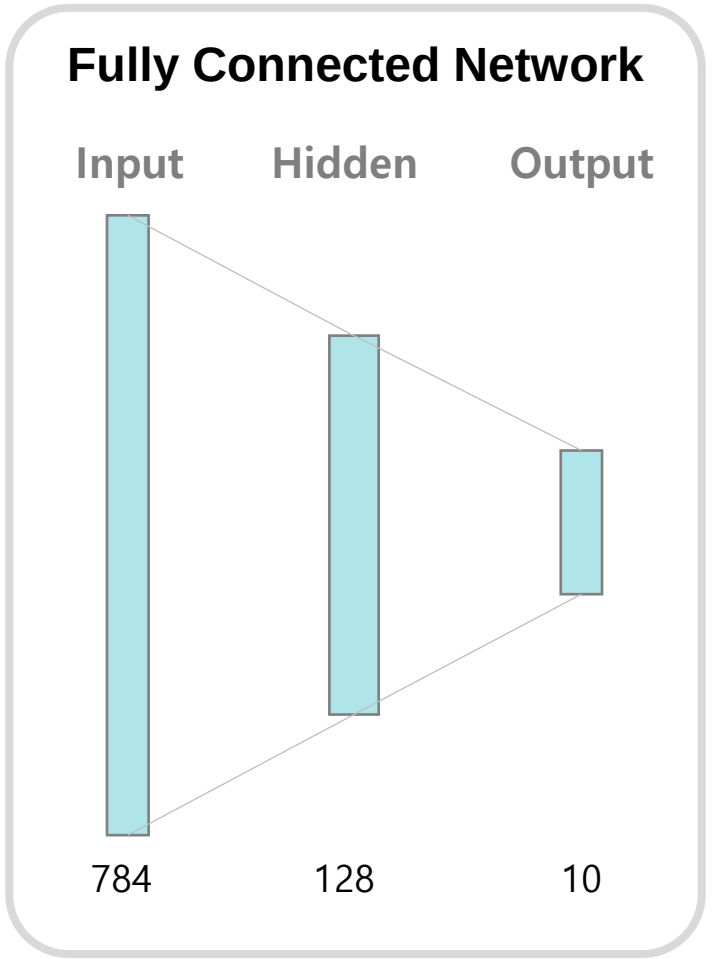


flatten

$x = (x_1, x_2, x_3, \dots, x_{784})$

784차원의 입력 벡터

28x28x1 이미지를 1차원 벡터로 변환

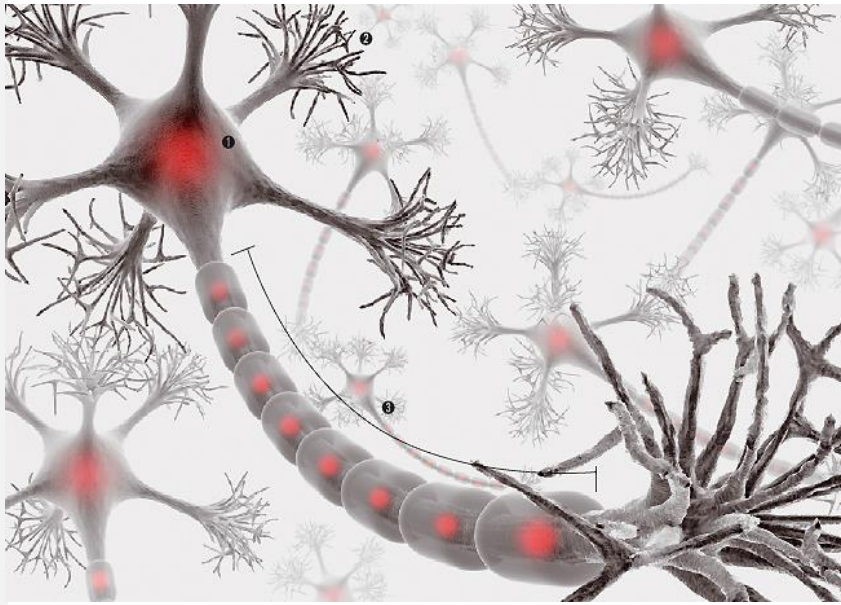


$y = (y_1, y_2, y_3, \dots, y_{10})$

0에서 9까지의 클래스에 속할 확률

- 1-hidden layers
- 128 neurons

2 데이터 준비



텐서플로와 다른 라이브러리 임포트

```
from __future__ import absolute_import, division, print_function, unicode_literals, unicode_literals

# tensorflow와 tf.keras를 임포트합니다
import tensorflow as tf
from tensorflow import keras

# 헬퍼(helper) 라이브러리를 임포트합니다
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
2.0.0-dev20190524
```

데이터셋 로드

```
fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

데이터 탐색

훈련 세트에 60,000개의 이미지가 있고 각 이미지의 크기는 28x28

```
train_images.shape
```

```
(60000, 28, 28)
```

훈련 세트에는 60,000개의 레이블이 있음

```
len(train_labels)
```

```
60000
```

각 레이블은 0과 9사이의 정수

```
train_labels
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

데이터 탐색

테스트 세트에 10,000개의 이미지가 있고 각 이미지의 크기는 28x28

```
test_images.shape
```

```
(10000, 28, 28)
```

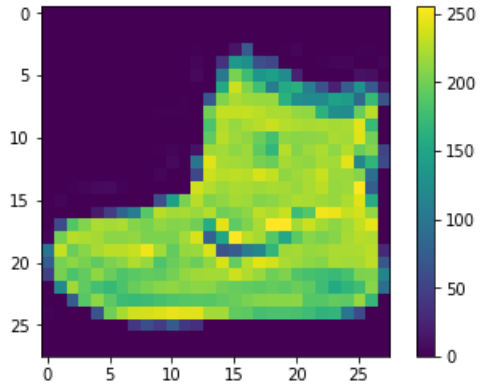
테스트 세트에는 10,000개의 레이블이 있음

```
len(test _labels)
```

```
10000
```

데이터 전처리

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



픽셀 값의 범위가 0~255 사이

이 값의 범위를 0~1 사이로 조정

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

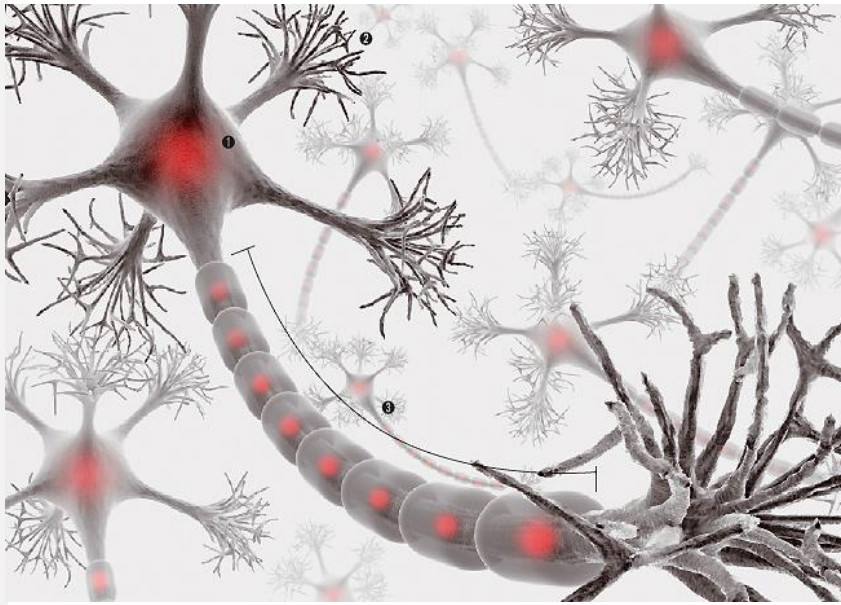
*훈련 세트*와 *테스트 세트*를 동일한 방식으로 전처리하는 것이 중요

데이터 전처리 확인 (문제)

훈련 세트에서 처음 25개 이미지와 클래스 이름을 출력

```
plt.figure(figsize=(10,10))
for i in range(25):
   # your code
   # 5x5 형태로 이미지를 클래스 이름과 함께 출력
plt.show()
```

3 모델 정의 및 훈련, 검증

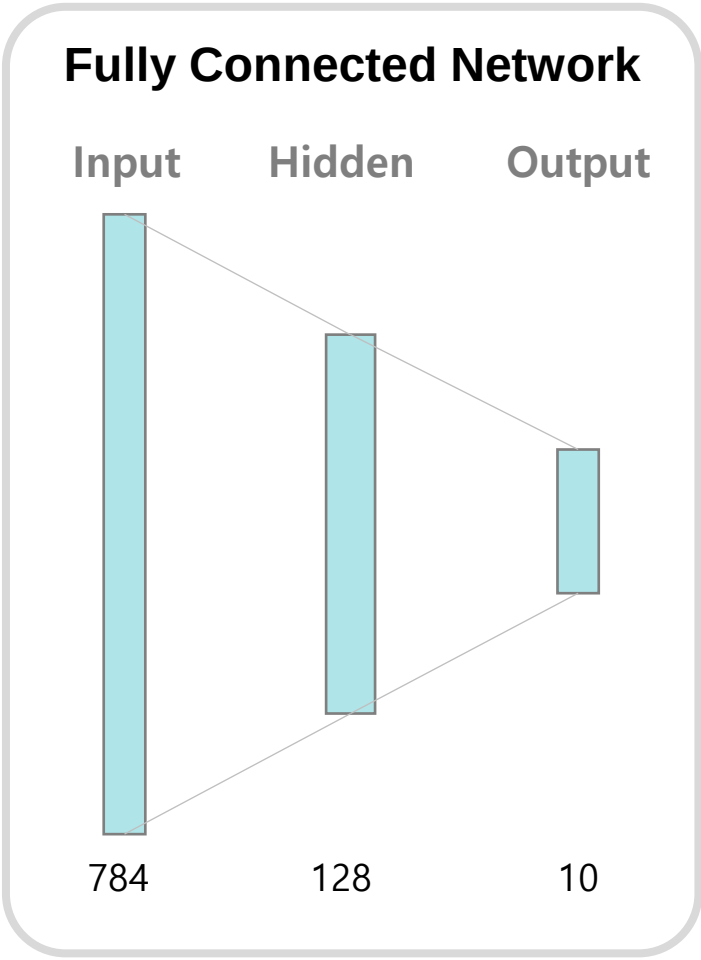


모델 정의 (문제)

Input Layer에 2차원 배열(28 x 28 픽셀)의 이미지 포맷을 28 * 28 = 784 픽셀의 1차원 배열로 변환
(Hint : keras.layers.Flatten, keras.layers.Dense 사용)

```
model = keras.Sequential([
    # your code # input layer,
    # your code # hidden layer,
    # your code # output layer
])
```

(마지막) 층은 10개의 Class를 출력하는 *소프트맥스*(softmax) 층입니다



참고 tf.keras.layers.Flatten

```
tf.keras.layers.Flatten(
    data_format=None, **kwargs
)
```

- Batch Size는 유지하고 나머지 차원만 flatten시킴
- **data_format**: Input Image의 channel 표기가 처음에 있는지 마지막에 있는지 알려주는 옵션
  If data_format='channels_last': 4D tensor with shape (batch_size, rows, cols, channels)
  If data_format='channels_first': 4D tensor with shape (batch_size, channels, rows, cols)

Flatten은 언제 사용하나요?

- **이미지와 같은 다차원 데이터를 Fully Connected Network에 입력할 때**
  - Sequential Model에서 첫번째로 사용할 때는 input_shape = (4, 28, 28, 3) 과 같이 크기 지정
- **Convolution layer의 출력을 Fully Connected Layer의 입력으로 변환할 때**

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten

참고 tf.keras.layers.Dense

```
tf.keras.layers.Dense(
    units, activation=None, use_bias=True, kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None, bias_constraint=None,
    **kwargs
)
```

- **units**: 뉴런 개수, Positive integer, dimensionality of the output space.
- **activation**: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix.
- **bias_initializer**: Initializer for the bias vector.
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix.
- **bias_regularizer**: Regularizer function applied to the bias vector.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation")..
- **kernel_constraint**: Constraint function applied to the kernel weights matrix.
- **bias_constraint**: Constraint function applied to the bias vector.

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

모델 컴파일 (문제)

훈련에 필요한 Optimizer, Loss, Metrics 설정
(단, Optimizer는 Adam, Loss는 Label을 One-Hot Encoding 해주는 Cross Entropy로 Metric은 accuracy로 설정)

```
model.compile(optimizer= #your code,
              loss= #your code,
              metrics= #your code)
```

https://www.tensorflow.org/api_docs/python/tf/keras/Model

One-hot encoding 이란?

클래스를 나타내는 숫자

cat → 0
mat → 1
on → 2
set → 3
the → 4

One-hot encoding

cat	1	0	0	0	0
mat	0	1	0	0	0
on	0	0	1	0	0
set	0	0	0	1	0
the	0	0	0	0	1

차원 : 클래스 수

- 클래스를 나타나내는 원소는 1로 표기하고 나머지는 0으로 표기하는 벡터 표기법
- 분류 문제에서 모델이 Categorical Distribution를 예측할 때, 타깃은 One-Hot Vector로 만들어서 Cross Entropy Loss를 계산한다.

참고 tf.keras.losses.SparseCategoricalCrossentropy

```
tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=False, reduction=losses_utils.ReductionV2.AUTO,
    name='sparse_categorical_crossentropy'
)
```

from_logits: Model의 출력 값이 Softmax를 거치기 전이면 True, Softmax를 실행했다면 False

SparseCategoricalCrossentropy vs. CategoricalCrossentropy

- SparseCategoricalCrossentropy는 label을 One-Hot vector로 자동 변환해 줌
- CategoricalCrossentropy는 label이 이미 One-Hot vector로 준비된 경우에 사용

https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

모델 훈련 (문제)

모델을 훈련하기 위해 training set, label, epoch 등을 지정

```
model.fit(#your code)
```

모델이 훈련되면서 손실과 정확도 지표가 출력

```
Epoch 1/5
60000/60000 [==============================] - 4s 70us/sample - loss: 0.4988 - accuracy: 0.8264
Epoch 2/5
60000/60000 [==============================] - 4s 68us/sample - loss: 0.3792 - accuracy: 0.8629
Epoch 3/5
60000/60000 [==============================] - 4s 66us/sample - loss: 0.3394 - accuracy: 0.8766
Epoch 4/5
60000/60000 [==============================] - 4s 66us/sample - loss: 0.3142 - accuracy: 0.8859
Epoch 5/5
60000/60000 [==============================] - 4s 65us/sample - loss: 0.2977 - accuracy: 0.8907
```

0.88(88%) 정도의 정확도를 달성

https://www.tensorflow.org/api_docs/python/tf/keras/Model

정확도 평가 (문제)

테스트 셋으로 모델의 Loss와 Metric을 평가

```
test_loss, test_acc = model.evaluate(#your code)

print('테스트 정확도:', test_acc)
```

```
10000/10000 [==============================] - 0s 38us/sample - loss: 0.3369 - accuracy: 0.8758
테스트 정확도: 0.8758
```

테스트 세트의 정확도가 훈련 세트의 정확도보다 조금 낮음

https://www.tensorflow.org/api_docs/python/tf/keras/Model

예측 만들기 (문제)

테스트 세트에 있는 각 이미지의 레이블을 예측

```
predictions = model.predict(#your code)
```

첫 번째 예측 확인

```
predictions[0]
```

```
array([4.5235288e-06, 2.9474028e-05, 3.7621589e-06, 2.4097352e-07,
       1.9792578e-06, 9.8094158e-03, 1.3787034e-05, 4.6312016e-02,
       1.5913237e-04, 9.4366562e-01], dtype=float32)
```

https://www.tensorflow.org/api_docs/python/tf/keras/Model

예측 만들기

모델은 이 이미지가 앵클 부츠(class_name[9])라고 가장 확신

```
np.argmax(predictions[0])
```

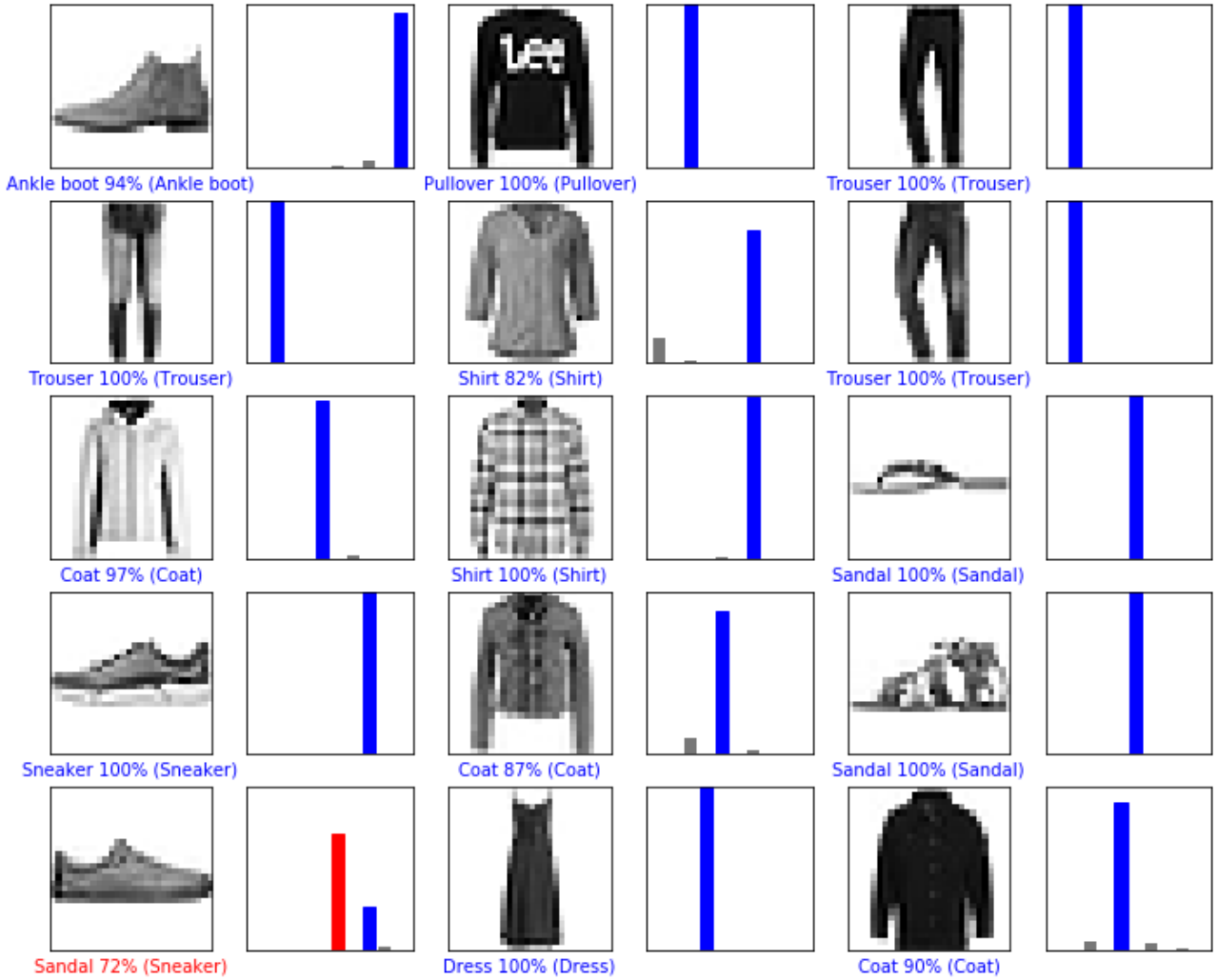
```
9
```

테스트 레이블을 확인

```
test_labels[0]
```

```
9
```

예측 만들기



올바르게 예측된 레이블은 파란색이고
잘못 예측된 레이블은 빨강색

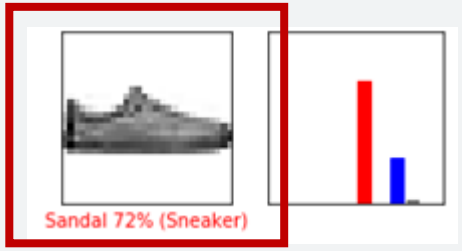
예측 만들기

입력 이미지와 예측 레이블 (예측 클래스, 예측 확률, 실제 클래스) 생성

```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary) # 입력 이미지

  predicted_label = np.argmax(predictions_array) # 예측 클래스 인덱스
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label], # 예측 클래스
                                       100*np.max(predictions_array), # 예측 확률
                                       class_names[true_label]), # 실제 클래스
                                       color=color)
```

예측이 맞는 경우

예측이 틀린 경우

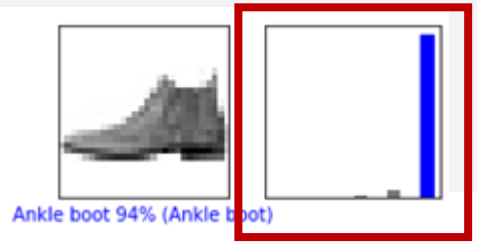


예측 만들기

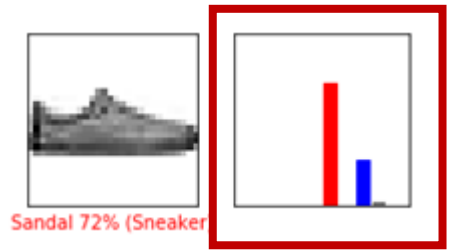
10개 클래스의 예측 확률을 바 그래프로 표현

```
def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777") # 10개 클래스 별 예측 확률
  plt.ylim([0, 1])

  predicted_label = np.argmax(predictions_array)
  thisplot[predicted_label].set_color('red') # 예측 클래스
  thisplot[true_label].set_color('blue') # 실제 클래스
```

예측이 맞는 경우



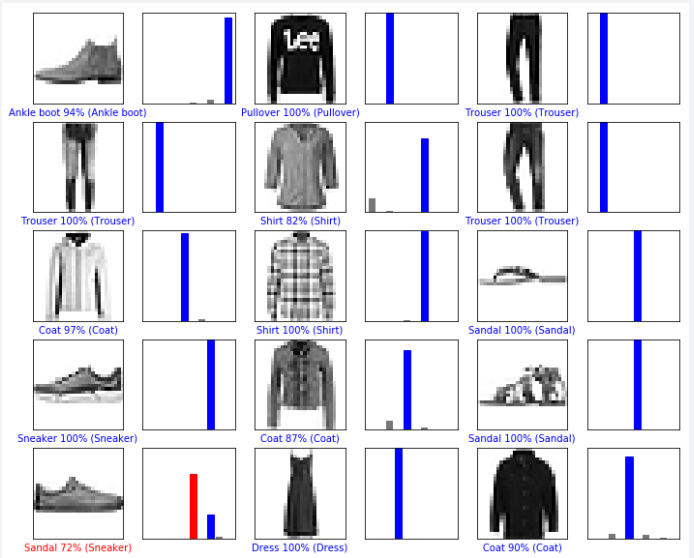
예측이 틀린 경우



예측 만들기

배열 형태로 예측 결과 확인

```
# 처음 X 개의 테스트 이미지와 예측 레이블, 진짜 레이블을 출력합니다
# 올바른 예측은 파랑색으로 잘못된 예측은 빨강색으로 나타냅니다
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))

for i in range(num_images):
  # 이미지 출력
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions, test_labels, test_images)
  # 바 그래프 출력
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)

plt.show()
```



num_rows = 5

num_cols = 3

Thank you!