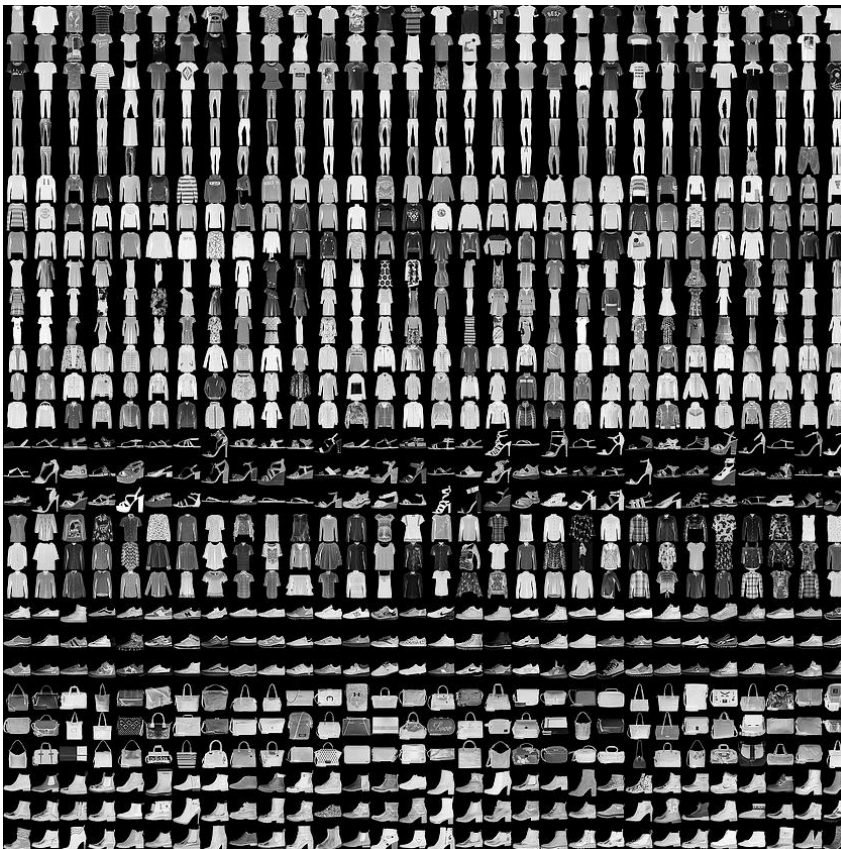


MNIST 패션 실행

# 패션 MNIST 데이터셋

- 티셔츠, 부츠 등 패션의 10 개 분류
  - 손글씨와 구조는 비슷
    - 60000, 10000개, 28 X 28 이미지 구조, 10개의 분류



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

# Fashion-MNIST 데이터 저장

- 미리 섞여진 fashion-mnist의 학습 데이터와 테스트 데이터 로드

```
# 필요 모듈 임포트
# tensorflow와 tf.keras를 임포트합니다
import tensorflow as tf
from tensorflow import keras
```

```
# ① 문제와 정답 데이터 지정
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
# 10 개의 분류 이름 지정
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

```
# 데이터 전처리
# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
train_images, test_images = train_images / 255.0, test_images / 255.0
```



# 모델 구성

## • 3개의 은닉층 구성

# ② 모델 구성(생성)

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
```

# ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택

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

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401920
dense_1 (Dense)	(None, 256)	131328
dense_2 (Dense)	(None, 128)	32896
dense_3 (Dense)	(None, 10)	1290

Total params: 567,434

Trainable params: 567,434

Non-trainable params: 0

Train on 60000 samples

# 학습과 평가

## • 8회 학습

```
# ④ 생성된 모델로 훈련 데이터 학습
# 모델을 훈련 데이터로 총 5번 훈련
```

```
model.fit(train_images, train_labels, epochs=8)
```

```
# ⑤ 테스트 데이터로 성능 평가
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('\n테스트 정확도:', test_acc)
```

```
Train on 60000 samples
```

```
Epoch 1/8
```

```
60000/60000 [=====] - 5s 85us/sample - loss: 0.4725 - accuracy: 0.8293
```

```
Epoch 2/8
```

```
60000/60000 [=====] - 5s 75us/sample - loss: 0.3603 - accuracy: 0.8669
```

```
Epoch 3/8
```

```
60000/60000 [=====] - 5s 76us/sample - loss: 0.3243 - accuracy: 0.8798
```

```
Epoch 4/8
```

```
60000/60000 [=====] - 5s 75us/sample - loss: 0.2994 - accuracy: 0.8888
```

```
Epoch 5/8
```

```
60000/60000 [=====] - 5s 76us/sample - loss: 0.2831 - accuracy: 0.8931
```

```
Epoch 6/8
```

```
60000/60000 [=====] - 5s 75us/sample - loss: 0.2676 - accuracy: 0.8986
```

```
Epoch 7/8
```

```
60000/60000 [=====] - 5s 75us/sample - loss: 0.2554 - accuracy: 0.9028
```

```
Epoch 8/8
```

```
60000/60000 [=====] - 5s 76us/sample - loss: 0.2450 - accuracy: 0.9069
```

```
테스트 정확도: 0.8846
```

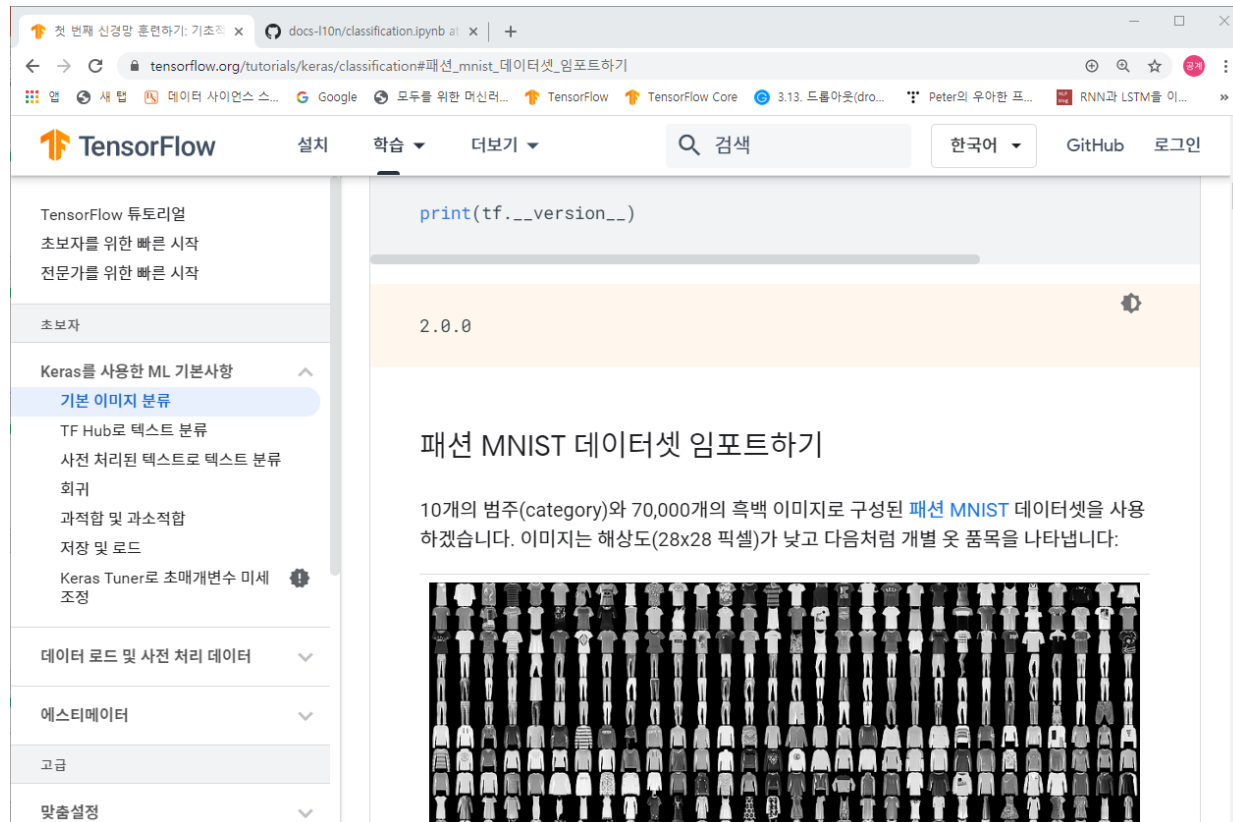
MNIST 패션 이해를 위한  
다양한 구현

# 텐서플로 튜토리얼

## • 메뉴

### – 초보자 | 기본 이미지 분류

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



# 파일

- `fashion_MNIST_vis.ipynb`



# 훈련과 테스트 데이터 이해

- shape

```
In [1]: 1 # tensorflow와 tf.keras를 임포트합니다
        2 import tensorflow as tf
        3 from tensorflow import keras
        4
        5 # 헬퍼(helper) 라이브러리를 임포트합니다
        6 import numpy as np
        7 import matplotlib.pyplot as plt
        8
        9 print(tf.__version__)
```

2.0.0

```
In [2]: 1 fashion_mnist = keras.datasets.fashion_mnist
        2 (train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
In [3]: 1 train_images.shape, test_images.shape
```

Out[3]: ((60000, 28, 28), (10000, 28, 28))

```
In [4]: 1 train_labels.shape, test_labels.shape
```

Out[4]: ((60000,), (10000,))

# 패션의 종류인 10 개 분류와 그리기

```
In [5]: 1 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
2                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
3 class_names
```

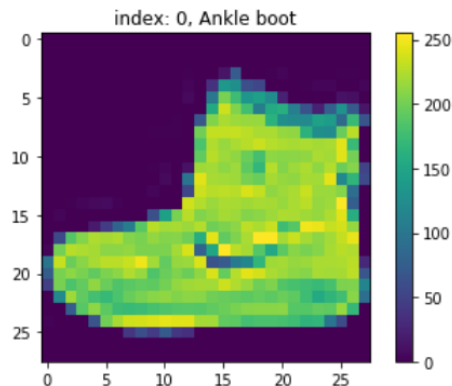
```
Out[5]: ['T-shirt/top',
'Trouser',
'Pullover',
'Dress',
'Coat',
'Sandal',
'Shirt',
'Sneaker',
'Bag',
'Ankle boot']
```

정답을 실제 이름으로 출력

```
In [6]: 1 print(train_labels[0])
2 print(class_names[train_labels[0]])
```

```
9
Ankle boot
```

```
In [7]: 1 # 첫번째 훈련 데이터 그림
2 plt.figure()
3 plt.imshow(train_images[0])
4 plt.title('index: 0, ' + class_names[train_labels[0]])
5 plt.colorbar()
6 #plt.grid(False)
7 plt.show()
```

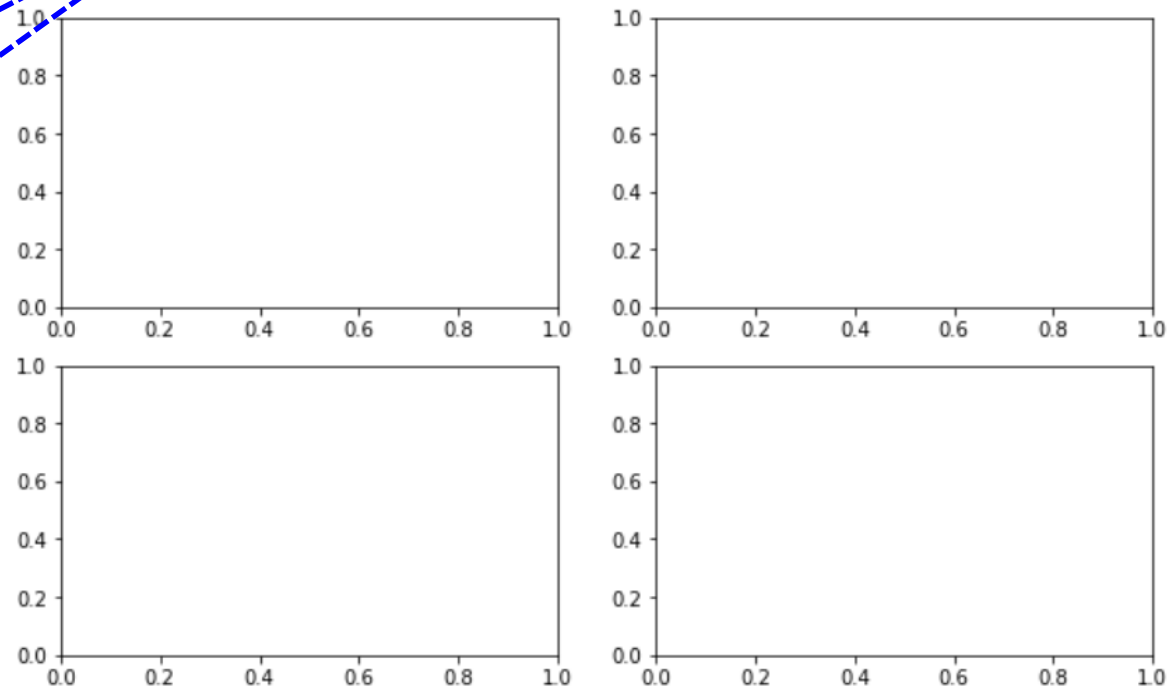


# Matplotlib

- subplot

```
In [36]: 1 plt.figure(figsize = (10, 6))  
2  
3 plt.subplot(2, 2, 1)  
4 plt.subplot(2, 2, 2)  
5 plt.subplot(2, 2, 3)  
6 plt.subplot(2, 2, 4)  
7  
8 plt.show()
```

행, 열, 순번(1번부터 ...)



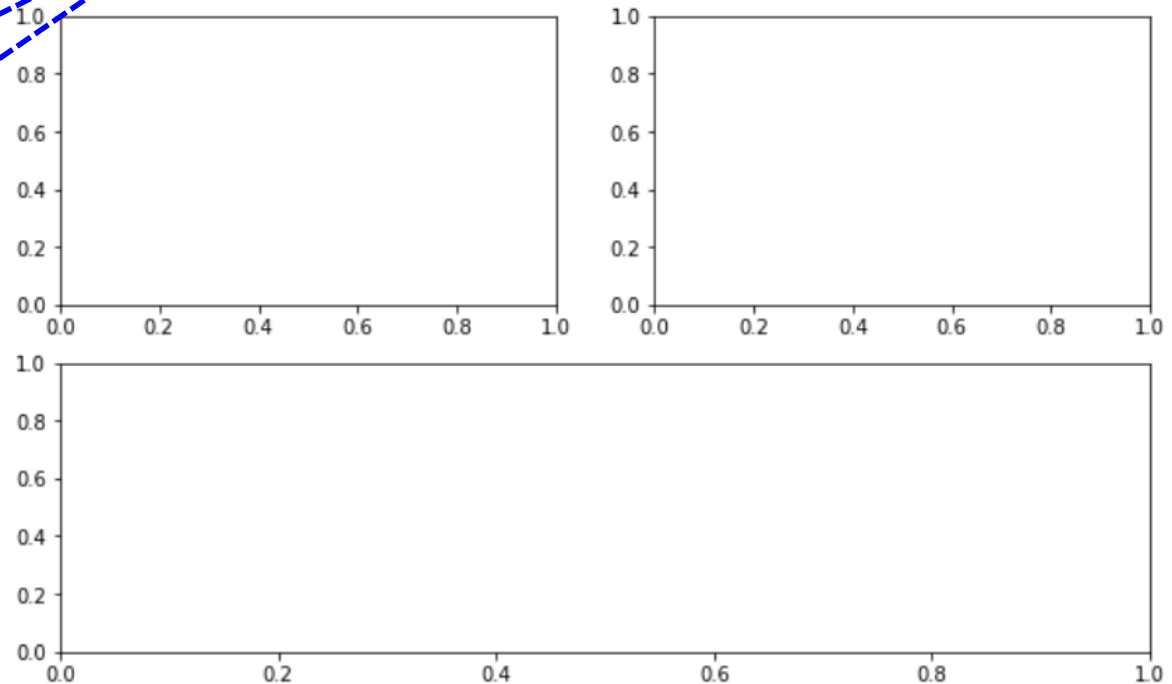
# Matplotlib

- subplot

In [42]:

```
1 plt.figure(figsize = (10, 6))
2
3 plt.subplot(2, 2, 1)
4 plt.subplot(2, 2, 2)
5 #2행 1열로 잡은 공간에 2번째로 지정하게 되면서 위 그림 상 밑에 긴 plot 공간이 잡함
6 plt.subplot(2, 1, 2)
7
8 plt.show()
```

2행 1열 상태에서 2번  
(그러므로 2행의 전체)

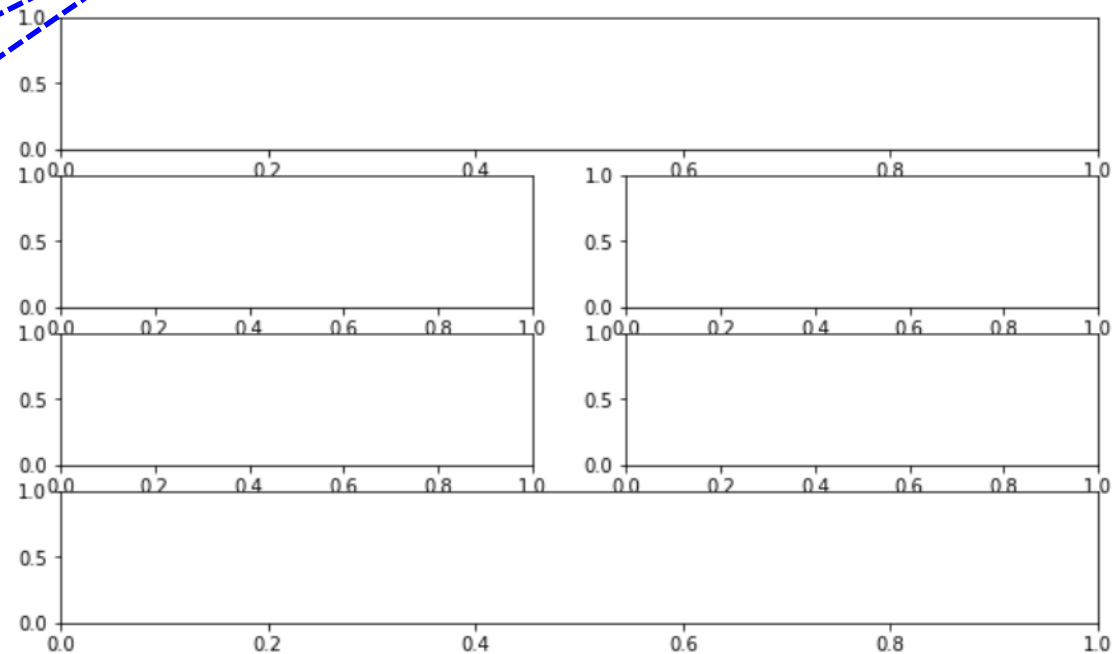


# Matplotlib

- subplot

```
In [44]: 1 plt.figure(figsize = (10, 6))
          2
          3 plt.subplot(411)
          4 plt.subplot(423)
          5 plt.subplot(424)
          6 plt.subplot(425)
          7 plt.subplot(426)
          8 #plt.subplot(413)
          9 plt.subplot(414)
          10
          11 plt.show()
```

4행 1열 상태에서 4번  
(그러므로 4행의 전체)



# subplot() 그리기

```
import numpy as np

x = np.arange(0, 5, 0.01)

plt.figure(figsize=(10, 12))

plt.subplot(411)
plt.plot(x, np.sqrt(x))
plt.grid()

plt.subplot(423)
plt.plot(x, x**2)
plt.grid()

plt.subplot(424)
plt.plot(x, x**3)
plt.grid()

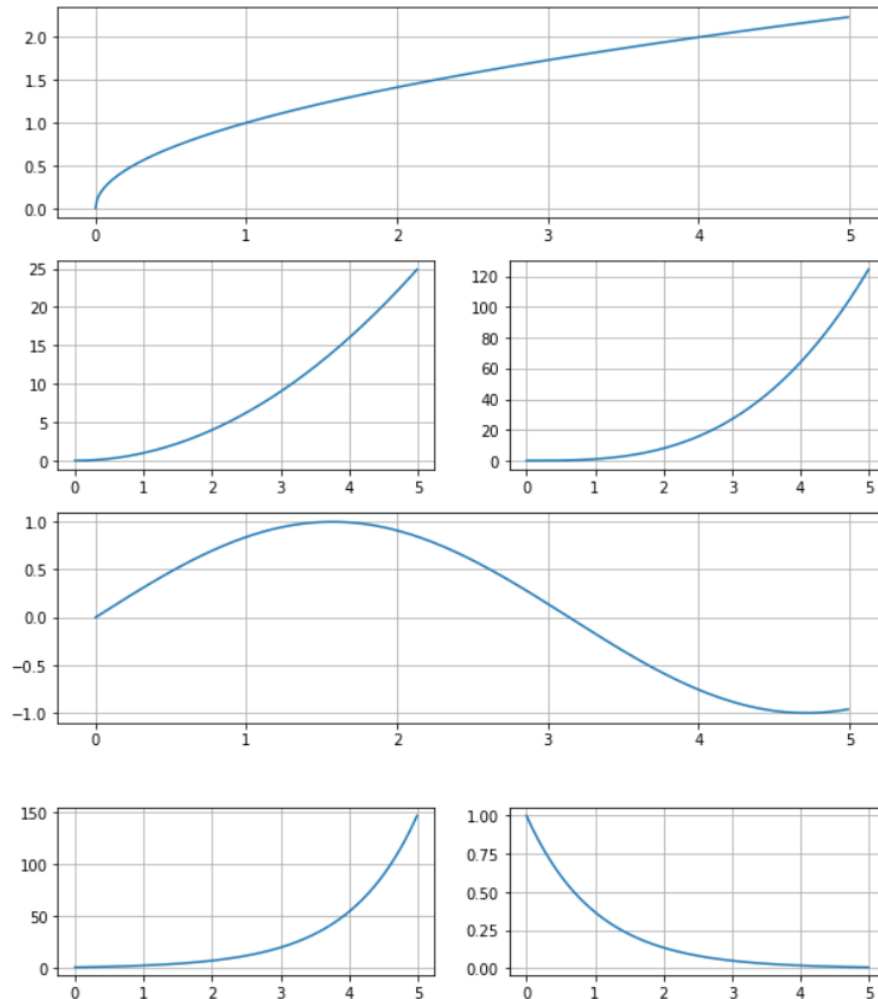
plt.subplot(413)
plt.plot(x, np.sin(x))
plt.grid()

plt.subplot(414)
plt.plot(x, np.cos(x))
plt.grid()

plt.subplot(529)
plt.plot(x, np.exp(x))
plt.grid()

plt.subplot(5, 2, 10)
plt.plot(x, np.exp(-x))
plt.grid()

plt.show()
```



# 훈련 자료 첫 30개 그려 보기

```
In [31]: 1 # 첫 30개의 훈련용 자료를 그려 보자.
2 plt.figure(figsize = (12, 10))
3 for i in range(30):
4     plt.subplot(5, 6, i+1)
5     plt.xticks([])
6     plt.yticks([])
7     plt.imshow(train_images[i], cmap = plt.cm.binary)
8     plt.xlabel(class_names[train_labels[i]])
9 plt.show()
```



# 랜덤하게 30개의 훈련용 자료 그리기

In [48]:

```

1 # 랜덤하게 30개의 훈련용 자료를 그려 보자.
2 from random import sample
3
4 rows, cols = 5, 6 #출력 가로 세로 수
5 idx = sorted(sample(range(len(train_images)), rows * cols)) # 출력할 첨자 선정
6 print(idx)
7
8 cnt = 0
9 plt.figure(figsize=(12, 10))
10
11 for i in idx:
12     cnt += 1
13     plt.subplot(rows, cols, cnt)
14     tmp = str(i) + ' ' + str(class_names[train_labels[i]])
15     plt.title(tmp)
16     plt.xticks([])
17     plt.yticks([])
18     plt.imshow(train_images[i], cmap='Greys')
19
20 plt.tight_layout()
21 plt.show()

```



[5975, 7555, 8044, 11865, 14152, 16391, 17566, 19044, 20387, 20632, 22648, 23502, 24881, 30100, 34190, 36927, 38024, 39446, 42394, 43260, 45729, 46799, 51879, 52137, 52883, 54149, 57035, 57203, 57207, 59311]



# 훈련과 테스트

```

train_images, test_images = train_images / 255.0, test_images / 255.0

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

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

history = model.fit(train_images, train_labels, epochs = 10, validation_split=0.2)

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose = 3)
print('\n테스트 정확도:', test_acc)

```

```

Epoch 7/10
1875/1875 [=====] - 11s 6ms/step - loss: 0.2559 - accuracy: 0.9032
Epoch 8/10
1875/1875 [=====] - 11s 6ms/step - loss: 0.2449 - accuracy: 0.9067
Epoch 9/10
1875/1875 [=====] - 11s 6ms/step - loss: 0.2354 - accuracy: 0.9111
Epoch 10/10
1875/1875 [=====] - 11s 6ms/step - loss: 0.2262 - accuracy: 0.9140

```

테스트 정확도: 0.8895999789237976

# 학습 과정 시각화

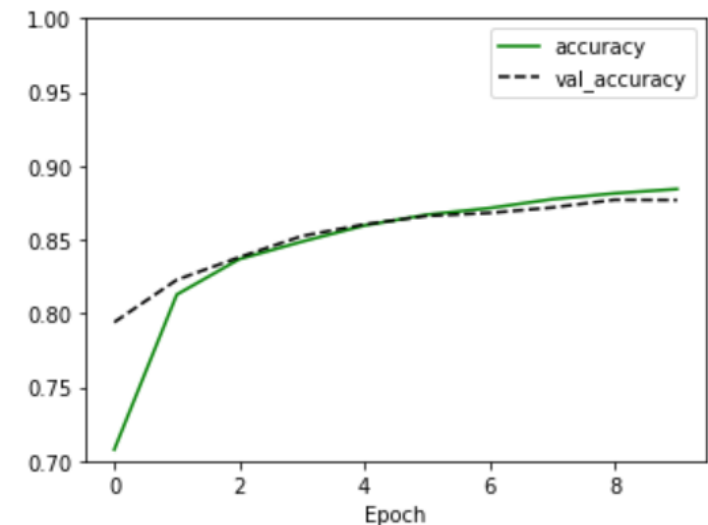
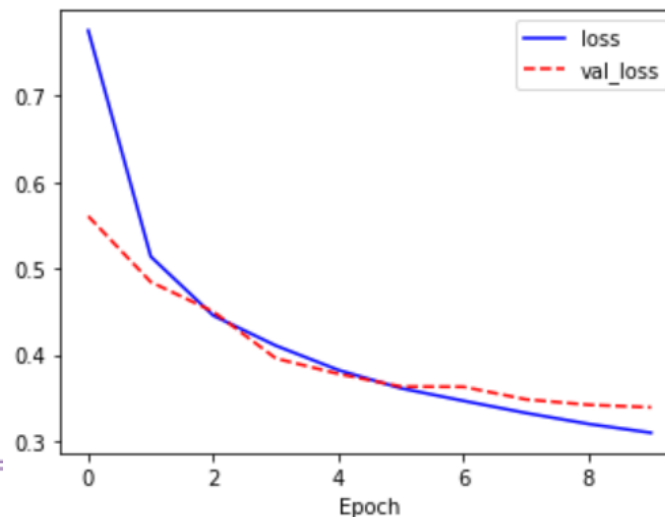
# Fashion MNIST 분류 모델 학습 결과 시각화

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
```

```
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
```

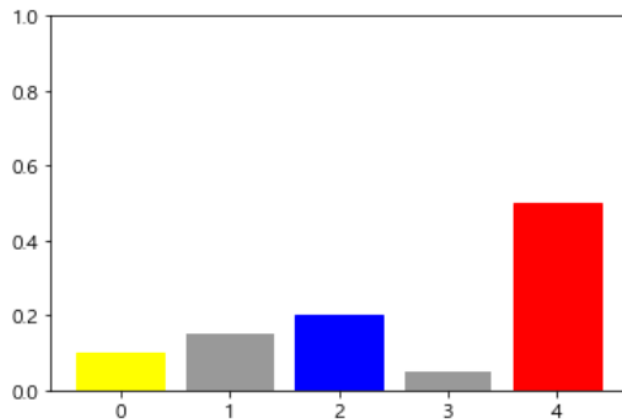
```
plt.show()
```



# 막대 그래프

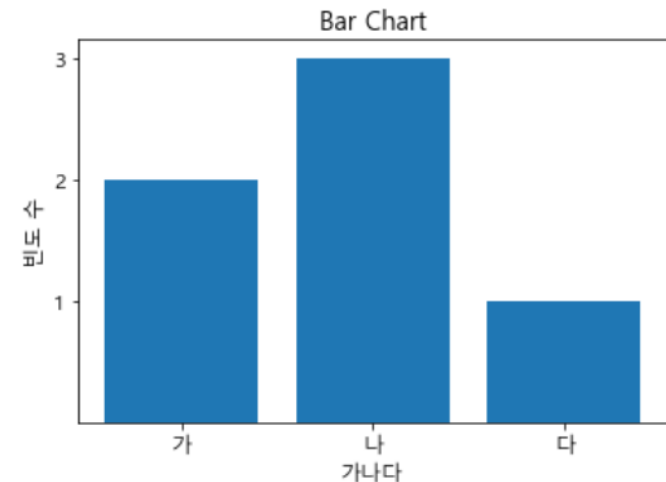
## • plt.bar

```
In [65]: 1 predictions_array = [.1, .15, .2, .05, .5]
2 predicted_label = np.argmax(predictions_array) # 4
3 true_label = 2;
4
5 # 막대 그래프 그리기 (x, y, 막대 색상)
6 thisplot = plt.bar(range(5), predictions_array, color="#999999")
7 # 부분 막대의 색상 수정
8 thisplot[0].set_color('yellow')
9 thisplot[predicted_label].set_color('red')
10 thisplot[true_label].set_color('blue')
11
12 plt.ylim([0, 1])
13 plt.show()
```



In [58]:

```
1 import matplotlib as mpl
2 import matplotlib.pyplot as plt
3
4 #한글 처리
5 plt.rcParams['font.family'] = "Malgun Gothic"
6 plt.rcParams['font.size'] = 12
7 plt.rcParams['axes.unicode_minus'] = False
8
9 y = [2, 3, 1]
10 x = np.arange(len(y))
11 xlabel = ['가', '나', '다']
12 plt.title("Bar Chart")
13 plt.bar(x, y)
14 plt.xticks(x, xlabel)
15 plt.yticks(sorted(y))
16 plt.xlabel("가나다")
17 plt.ylabel("빈도 수")
18 plt.show()
```



# 이미지 그리기

## • X 레이블 색상



Ankle boot 99% (Ankle boot)

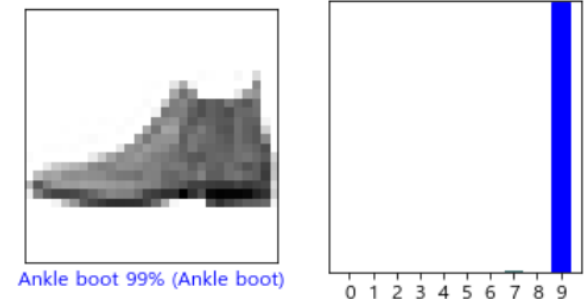
'예측 값 확률% (정답)' 형식으로 출력  
맞으면 색상이 파란, 틀리면 빨간

```
[ ] 1 # 이미지 그리기(첨자, 예측(확률 값), 결과(일반숫자), 이미지)
2 # 하단에 예측한 이름과 확률을 출력(정답이면 파란색, 오답이면 빨간색)
3 def plot_image(i, predictions_array, true_label, img):
4     predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
5     #plt.grid(False)
6     plt.xticks([])
7     plt.yticks([])
8
9     predicted_label = np.argmax(predictions_array) #예측한 확률 값 추출
10    # 하단 X 축 글: 정답이면 파란색, 오답이면 빨간색으로 그리기
11    if predicted_label == true_label:
12        color = 'blue'
13    else:
14        color = 'red'
15
16    maxprob = np.max(predictions_array) # 예측의 최대 확률 값
17    plt.xlabel("{} {:.2f}% ({})" .format(class_names[predicted_label],
18        100 * maxprob, class_names[true_label]), color = color)
19
20    plt.imshow(img, cmap = plt.cm.binary)
```

# 그림 예측 결과와 확률 값 시각화

- 결과인 원 핫 인코딩
  - 10개 값의 시각화

```
[28] 1 # 예측 값을 막대그래프로 그리기(정답이면 파란색, 오답이면 빨간색)
      2 def plot_value_array(i, predictions_array, true_label):
      3     predictions_array, true_label = predictions_array[i], true_label[i]
      4     #plt.grid(False)
      5     plt.xticks([])
      6     plt.yticks([])
      7     plt.ylim([0, 1])
      8     thisplot = plt.bar(range(10), predictions_array, color="#447777")
      9
     10     predicted_label = np.argmax(predictions_array)
     11     thisplot[predicted_label].set_color('red')
     12     thisplot[true_label].set_color('blue')
     13     # X 레이블을 그리기 위해
     14     x = np.arange(10)
     15     xlabel = np.arange(10)
     16     plt.xticks(x, xlabel)
```



```
[29] 1 # 테스트 데이터의 첫번째 원소의 이미지, 예측, 신뢰도 점수 배열을 확인
      2 predictions = model.predict(test_images)
      3
      4 i = 0
      5 plt.figure(figsize=(6,3))
      6 plt.subplot(1,2,1)
      7 plot_image(i, predictions, test_labels, test_images)
      8 plt.subplot(1,2,2)
      9 plot_value_array(i, predictions, test_labels)
     10 plt.show()
```

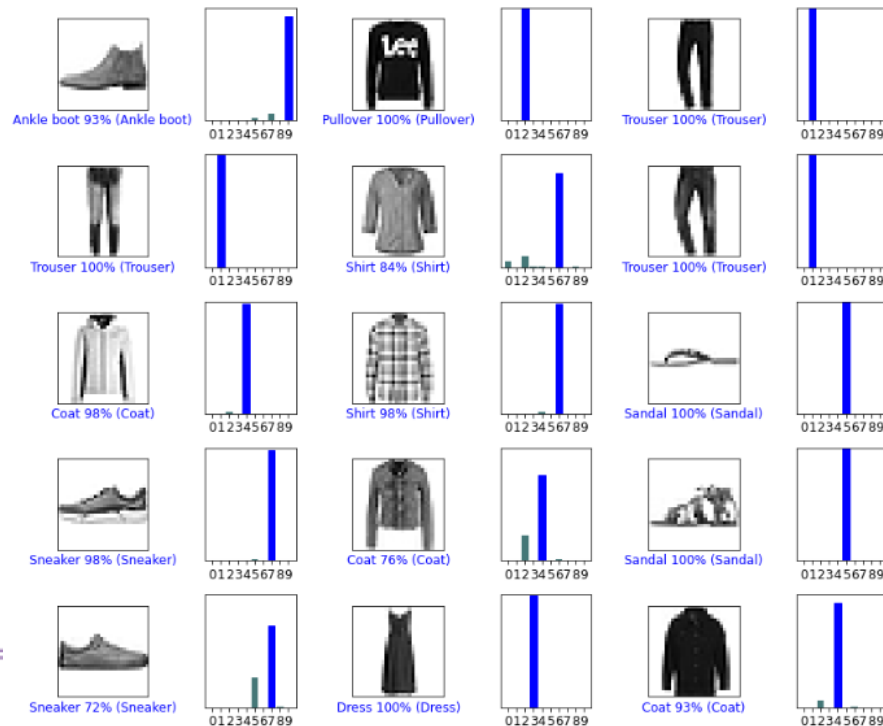
# 테스트 데이터 첫 15 개 그리기

```
# 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):
    plt.subplot(num_rows, 2*num_cols, 2*i+1)
    plot_image(i, predictions, test_labels, test_images)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions, test_labels)
```

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



# 예측이 잘못된 30 개 그리기

```

In [79]: 1 # 예측 틀린 것 첨자를 저장할 리스트
2 pred_result = model.predict(test_images)
3 # 원핫 인코딩을 일반 데이터로 변환
4 pred_labels = tf.argmax(pred_result, axis=1)
5
6 mispred = []
7 for i in range(0, len(test_labels)):
8     if pred_labels[i] != test_labels[i]:
9         mispred.append(i)
10 print('정답이 틀린 수', len(mispred))
11
12 # 랜덤하게 틀린 것 30개의 첨자 리스트 생성
13 samples = sorted(sample(mispred, 30))
14 print(samples)
15
16 # 틀린 것 30개 그리기
17 count = 0
18 rows, cols = 6, 5
19 plt.figure(figsize=(12,10))
20 for i in samples:
21     count += 1
22     plt.subplot(rows, cols, count)
23     plt.xticks([])
24     plt.yticks([])
25     plt.imshow(test_images[i].reshape(28, 28), cmap='Greys', interpolation='nearest')
26     tmp = str(class_names[test_labels[i]]) + ", pred:" + str(class_names[pred_labels[i]])
27     plt.title(tmp)
28
29 plt.tight_layout()
30 plt.show()

```



정답이 틀린 수 1174

[12, 49, 1277, 1376, 1945, 1955, 2278, 2500, 2508, 2935, 3205, 3540, 3861, 3940, 4109, 4995, 5575, 5745, 6329, 6475, 6649, 6727, 6816, 7288, 7719, 8420, 8832, 8950, 9032, 9256]

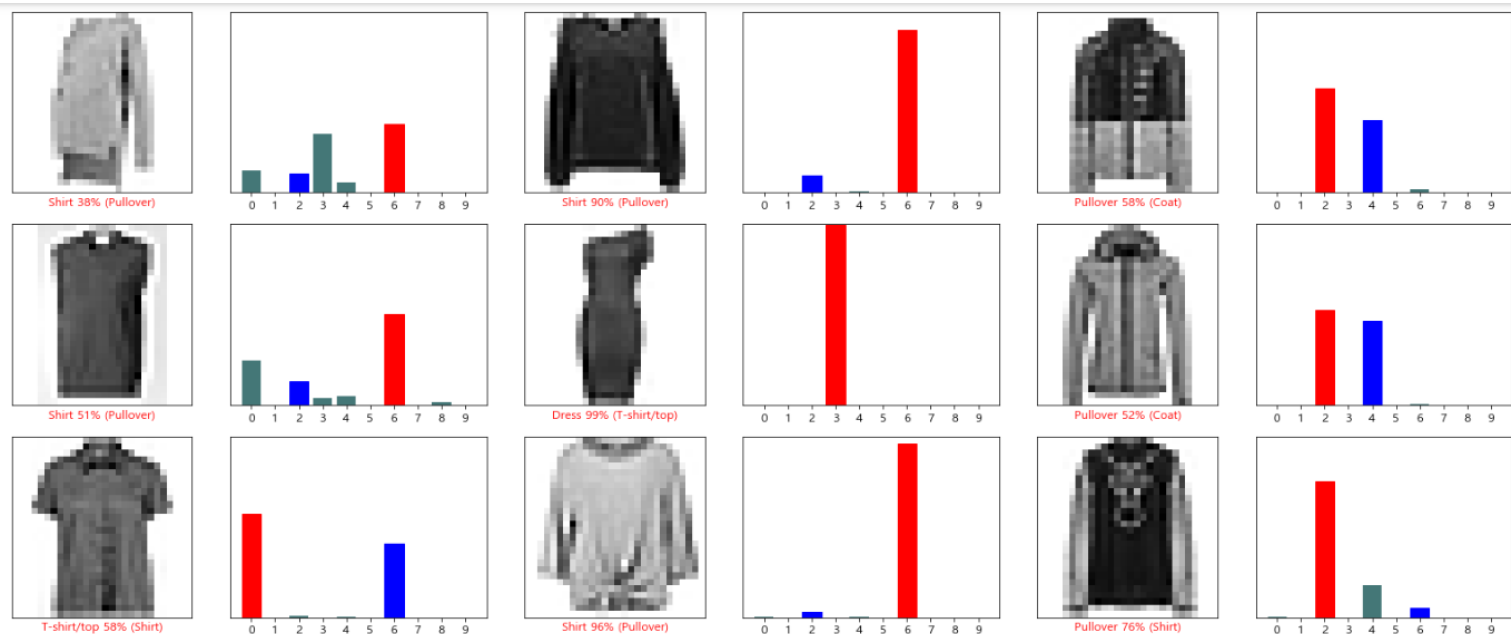
# 예측이 잘못된 30 개 예측 확률과 함께 그리기

In [83]:

```

1 # 틀린 것 30개 그리기
2 count = 0
3 rows, cols = 10, 6
4 plt.figure(figsize=(22, 30))
5
6 for i in samples:
7     count += 1
8     plt.subplot(rows, cols, count)
9     plot_image(i, pred_result, test_labels, test_images)
10    count += 1
11    plt.subplot(rows, cols, count)
12    plot_value_array(i, pred_result, test_labels)
13
14 plt.tight_layout()
15 plt.show()

```





# 손실 함수 직접 구하기

- **tf.keras.losses.SparseCategoricalCrossentropy()**
  - 정답의 원핫인코딩 불필요

In [86]:

```
1 import tensorflow as tf
2
3 #@tf.function
4 def get_loss(label, prob):
5     cce = tf.keras.losses.SparseCategoricalCrossentropy()
6     loss = cce(tf.convert_to_tensor(label), tf.convert_to_tensor(prob))
7     return loss
8
9 print('CCE: ', get_loss([2], [[0.2, .1, .3, .4]]).numpy())
10 print('CCE: ', get_loss(2, [0.2, .1, .3, .4]).numpy())
11 print('CCE: ', get_loss(3, [0.2, .1, .3, .4]).numpy())
12 print('CCE: ', get_loss(5, [3.5794352e-07, 1.6668336e-09, 1.2257135e-07, 3.7462252e-07, 4.4686758e-06,
13     6.8782866e-02, 1.0355227e-06, 5.1593245e-04, 2.1532403e-09, 9.3069482e-01]).numpy())
14 print('CCE: ', get_loss(9, [3.5794352e-07, 1.6668336e-09, 1.2257135e-07, 3.7462252e-07, 4.4686758e-06,
15     6.8782866e-02, 1.0355227e-06, 5.1593245e-04, 2.1532403e-09, 9.3069482e-01]).numpy())
```

정답과 예측 확률

```
CCE: 1.2039728
CCE: 1.2039728
CCE: 0.91629076
CCE: 2.6768007
CCE: 0.07182401
```

# 훈련 데이터에 대해 크로스 엔트로피 값 모두 알기

```
In [90]: 1 def get_loss(label, prob):
2         cce = tf.keras.losses.SparseCategoricalCrossentropy()
3         loss = cce(tf.convert_to_tensor(label), tf.convert_to_tensor(prob))
4         return loss
5
6 pred_labels = model.predict(test_images)
7
8 # 모든 테스트 데이터의 크로스 엔트로피 값을 저장
9 cces = []
10 for i in range(len(test_images)):
11     cces.append(get_loss(test_labels[i], pred_labels[i]).numpy())
12
13 print(len(cces))
14 print(cces[:10])
```

10000

[0.009457429, 0.00055393134, 1.072883e-06, 1.072883e-06, 0.030104673, 1.072883e-06, 8.928377e-05, 8.2966224e-05, 9.5367386e-07, 2.6940936e-05]

# Numpy의 argsort()

## 정렬을 위한 첨자 알기

정렬의 첨자를 알아내고 이를 사용하여 정렬하는 방법

```
In [91]: 1 import numpy as np
          2 a = np.random.randint(0, 20, 10)
          3 a
Out[91]: array([ 8,  8, 16, 13, 12,  7, 15,  2, 12,  9])
          0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

```
In [92]: 1 np.argsort(a) #오름 차순(순방향) 정렬의 첨자 값
```

```
Out[92]: array([7, 5, 0, 1, 9, 4, 8, 3, 6, 2], dtype=int64)
```

```
In [93]: 1 np.argsort(-a) #내림 차순(역방향) 정렬의 첨자 값
```

```
Out[93]: array([2, 6, 3, 4, 8, 9, 0, 1, 5, 7], dtype=int64)
```

```
In [94]: 1 # 가장 큰 값
          2 print(a[np.argsort(a)[-1]])
          3 print(a[np.argsort(-a)[0]])
```

```
16
16
```

```
In [96]: 1 # 오름 차 순으로 값 출력
          2 for _ in range(len(a)):
          3     print(a[np.argsort(a)[_]], end = ' ')
```

```
2 7 8 8 9 12 12 13 15 16
```

```
In [95]: 1 # 가장 작은 값
          2 print(a[np.argsort(a)[0]])
          3 print(a[np.argsort(-a)[-1]])
```

```
2
2
```

```
In [97]: 1 # 내림 차 순으로 값 출력
          2 for _ in range(len(a)):
          3     print(a[np.argsort(-a)[_]], end = ' ')
```

```
16 15 13 12 12 9 8 8 7 2
```

# 손실 값이 가장 큰 순으로 첨자 알아 내기

## 예측 막대 그래프에서 손실 값(CCE) 쓰기

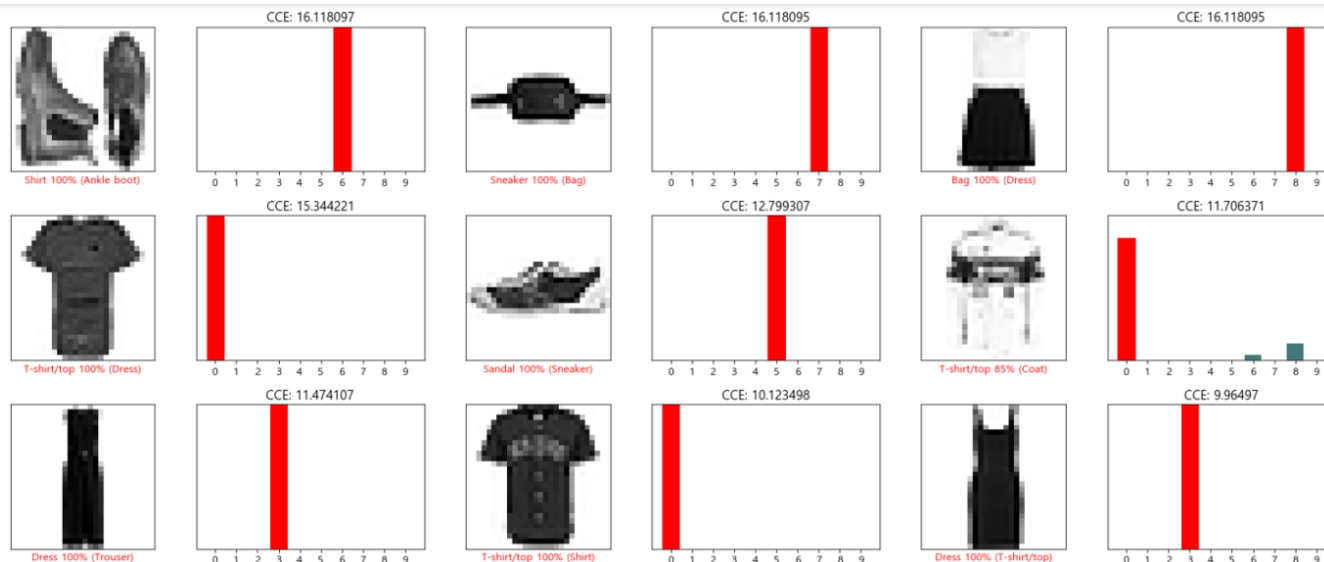
```
In [33]: 1 # 예측(원핫인코딩) 값을 막대그래프로 그리기(정답이면 파란색, 오답이면 빨간색)
          2 # X 레이블에 손실 값 CCE를 출력
          3 def plot_value_cce_array(i, predictions_array, true_label):
          4     plot_value_array(i, predictions_array, true_label)
          5     plt.title('CCE: ' + str(get_loss(true_label[i], predictions_array[i]).numpy()))
```

```
In [31]: 1 # 손실값이 큰 순으로 첨자를 저장
          2 arycces = np.array(cces)
          3 lst_arg = np.argsort(-arycces)
          4
          5 for i in range(20):
          6     print(cces[lst_arg[i]], end = ' ')
          7 print()
```

```
16.118097 16.118095 16.118095 15.344221 12.799307 11.706371 11.474107 10.123498 9.96497 9.9409685 9.673493 9.5
33807 9.21783 9.201383 9.154008 9.144351 8.754425 8.66029 8.639321 8.58075
```

# 손실 값이 가장 큰 30 개 그리기

```
In [103]: 1 # 손실 값이 큰 순으로 30개를 출력
2 count = 0
3 rows, cols = 10, 6
4 plt.figure(figsize=(22, 30))
5
6 for i in range(30):
7     count += 1
8     plt.subplot(rows, cols, count)
9     plot_image(lst_arg[i], pred_result, test_labels, test_images)
10    count += 1
11    plt.subplot(rows, cols, count)
12    plot_value_cce_array(lst_arg[i], pred_result, test_labels)
13
14 plt.tight_layout()
15 plt.show()
```



# 손실 값이 가장 큰 15 개

