MLP실습

라인바이라인해석하기

훈련 돌려보기

훈련 여러번 10번 100번 돌려서 실습후 남기기

```
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

tf.keras API사용

```
fashion_mnist = tf.keras.datasets.fashion_mnist

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

학습하고 훈련할 데이터로 Fashion MNIST데이터 사용→임폴트

로드데이터를 사용해 학습에 사용할 train세트,테스트에 사용할 test세트로 네개의 Numpy배열이 반환

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

클래스의 이름으로 사용할 별도의 변수선언(하나씩 레이블에 매핑)

```
train_images.shape //훈련세트는 60000개의 이미지와 28x28픽셀
len(train_labels)//훈현세트는60,000개의 레이블
```

```
train_labels//훈련 각 레이블은 0~9사이의 정수
test_images.shape//테스트세트는 10,000개의 이미지와 28x28픽셀
len(test_labels)//테스트세트는 10,000개의 이미지에 대한 레이블
```

데이터의 구조파악

데이터 전처리하기

```
plt.figure() //figure생성하기
plt.imshow(train_images[0])//훈련세트의 첫번쨰 이미지 표시
plt.colorbar() //matplotlib그림에 컬러바 표시
plt.grid(False) //grid off
plt.show()// 생성된 모든 figure보여주기
```

```
train_images = train_images / 255.0

test_images = test_images / 255.0
```

신경망 모델에 주입하기전에 값을 0~1사이로 조정

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)//row=5,column=5,index=i+1의 여러개의 그래프 나타내기
    plt.xticks([])//x축간격표시
    plt.yticks([])//y축 간격표시
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary) //binary컬러의 cmap사용
    plt.xlabel(class_names[train_labels[i]]) //x축에 레이블 표사
plt.show()
```

훈련세트의 처음25개이미지와 클래스이름 출력(데이터포맷이 올바른지 확인)

모델구성

신경만모델을 만들기위해서 모델의 층을 구성하고 모델을 컴파일

신경망의 기본빌딩 블록은 레이어

딥러닝은 간단한 층을 연결하여 구성

층의 가중치는 훈련을 거치면서 학습

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),//첫번쨰층,2차원배열->1차원배열로 데이터변환
    tf.keras.layers.Dense(128, activation='relu'),//노드 128개,rectified linear unit정류선형유닛
    tf.keras.layers.Dense(10)//노드10개(10개의 클래스에 속할확률 반환->합은1)
])
```

활성화 함수의 종류https://yeomko.tistory.com/39

모델컴파일

```
model.compile(optimizer='adam', //모델의 업데이트 방식
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), //손실함수로 모델의 정확도 측정
              metrics=['accuracy']) //모니터링시 사용하기 위해 정확도 방식사용,
                            //Calculates how often predictions equal labels.
```

최적화 방식의 종류https://onevision.tistory.com/entry/Optimizer-의-종류와-특성-Momentum-RMSProp-Adam

---

모델 훈련

1. 훈련데이터를 모델에 주입
2. 모델이 이미지와 레이블을 매핑하는 방법 학습
3. 테스트 세트에대한 모델의 예측
4. 예측이 테스트배열의 레이블과 일치하는지 확인

```
model.fit(train_images, train_labels, epochs=10)
```

model.fit메서드인자(입력데이터,라벨값,학습반복횟수)

훈련되면서 매epochs마다 손실과 정확도 지표출력

정확도평가

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
//verbose = 상세한 로깅을 출력할지 말지 결정
print('\nTest accuracy:', test_acc)
```

evaluate함수를 사용해 모델의 최종적인 acuuracy와 loss를 알수있다

loss = 예측값과 실제값이 차이나는 정도

테스트 세트의 정확도와 훈련 정확도의 차이는 과대적합 때문

예측하기

훈련된 모델에 대한 일부이미지 예측하기

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
predictions[0] //첫번째 예측확인, 나오는 숫자배열은 모델의 신뢰도를 나타냄

np.argmax(predictions[0])//가장 높은 신뢰도를 가진 레이블
test_labels[0]//가장 높은 신뢰도 가 맞는지 테스트 레이블에서 확인
```

10개의 예측을 모두 그래프로 표현

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

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:   //예측값과 실제값이 맞으면 파란색 아니면 빨간색
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)//이미지 x 축에 예측레이블,유사도%,실제레이블,색 표시

def plot_value_array(i, predictions_array, true_label): //그래프
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))//x축 간격10
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red') //예측레이블은 빨간색바
  thisplot[true_label].set_color('blue') //실제레이블은 파란색바
```

이미지 예측 출력

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 5   //행열 설정
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows)) //1열당 두개의 figure
for i in range(num_images):     //이미지 개수만큼 반복
  plt.subplot(num_rows, 2*num_cols, 2*i+1)     //이미지
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)      //그래프
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

훈련된 모델 사용하기

훈련된 모델을 사용해 이미지에 대한 예측을 만들기

```
# Grab an image from the test dataset.
img = test_images[1]
print(img.shape)

# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0)) //2차원배열로 예측 만들기
print(img.shape)
```

```
predictions_single = probability_model.predict(img)
print(predictions_single)

plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
plt.show()

np.argmax(predictions_single[0]) //예상과 같이 예측
```

훈련 횟수에 따른 차이점 실습

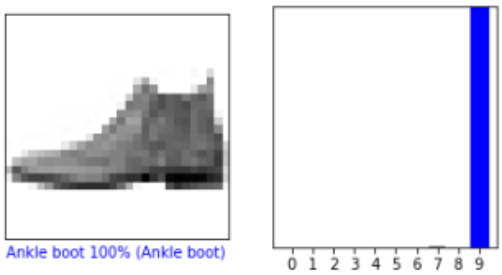
10회 훈련

```
Epoch 1/10
1875/1875 [==============================] - 7s 2ms/step - loss: 0.4973 - accuracy: 0.8251
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3719 - accuracy: 0.8647
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3322 - accuracy: 0.8795
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3112 - accuracy: 0.8861
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2944 - accuracy: 0.8920
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2786 - accuracy: 0.8984
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2665 - accuracy: 0.9016
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2573 - accuracy: 0.9049
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2472 - accuracy: 0.9085
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2387 - accuracy: 0.9109
<keras.callbacks.HiCNNy at 0x7f4ca0376990>
```

```
313/313 - 1s - loss: 0.3455 - accuracy: 0.8772 - 913ms/epoch - 3ms/step
CNN
Test accuracy: 0.8772000074386597
```

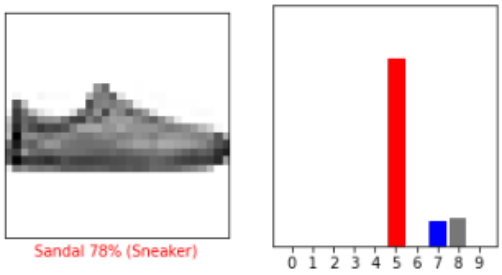
```
[18]  predictions[0]

array([7.0787526e-10, 3.2261610e-10, 4.1849124e-10, 1.9443384e-11,
       1.8463362e-09, 2.0672122e-04, 9.8970609e-10, 4.2741103e-03,
       9.2530777e-10, 9.9551916e-01], dtype=float32)
```

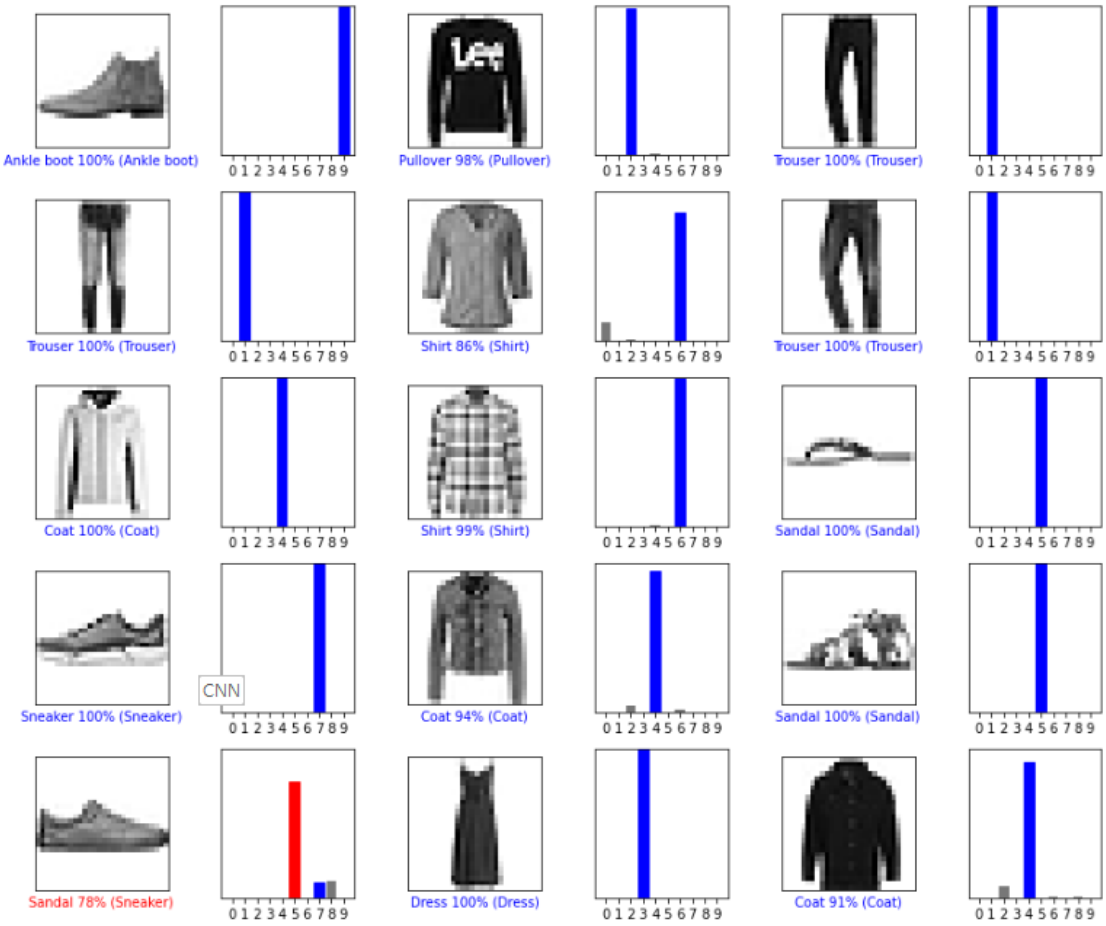
```
[22] i = 0
     plt.figure(figsize=(6,3))
     plt.subplot(1,2,1)
     plot_image(i, predictions[i], test_labels, test_images)
     plt.subplot(1,2,2)
     plot_value_array(i, predictions[i],  test_labels)
     plt.show()
```




```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

CNN




Ankle boot 100% (Ankle boot)
Pullover 98% (Pullover)
Trouser 100% (Trouser)
Trouser 100% (Trouser)
Shirt 86% (Shirt)
Trouser 100% (Trouser)
Coat 100% (Coat)
Shirt 99% (Shirt)
Sandal 100% (Sandal)
Sneaker 100% (Sneaker)
CNN
Coat 94% (Coat)
Sandal 100% (Sandal)
Sandal 78% (Sneaker)
Dress 100% (Dress)
Coat 91% (Coat)
0 1 2 3 4 5 6 7 8 9

100회 훈련

```
Epoch 90/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0562 - accuracy: 0.9795
Epoch 91/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0564 - accuracy: 0.9793
Epoch 92/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0535 - accuracy: 0.9804
Epoch 93/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0555 - accuracy: 0.9789
Epoch 94/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0531 - accuracy: 0.9804
Epoch 95/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0532 - accuracy: 0.9802
Epoch 96/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0525 - accuracy: 0.9804
Epoch 97/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0536 - accuracy: 0.9802
Epoch 98/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0524 - accuracy: 0.9804
Epoch 99/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0510 - accuracy: 0.9805
Epoch 100/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0520 - accuracy: 0.9807
<keras.callbacks.History at 0x7f2350128090>
```

모델이 훈련되면서 손실과 정확도 지표가 출력됩니다. 이 모델은 훈련 세트에서 약 0.88(88%) 정도의 정확도를 달성합니다.

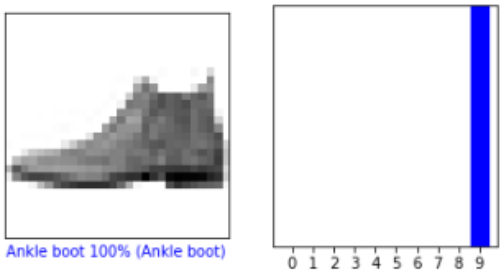
▸ 정확도 평가 CNN

다음으로, 모델이 테스트 데이터세트에서 작동하는 방식을 비교합니다.

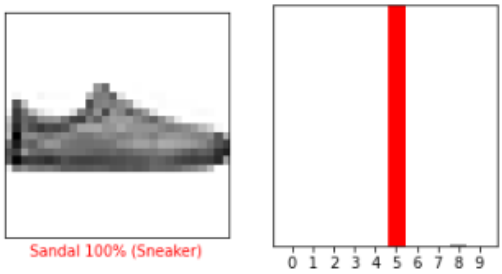
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('#nTest accuracy:', test_acc)
```

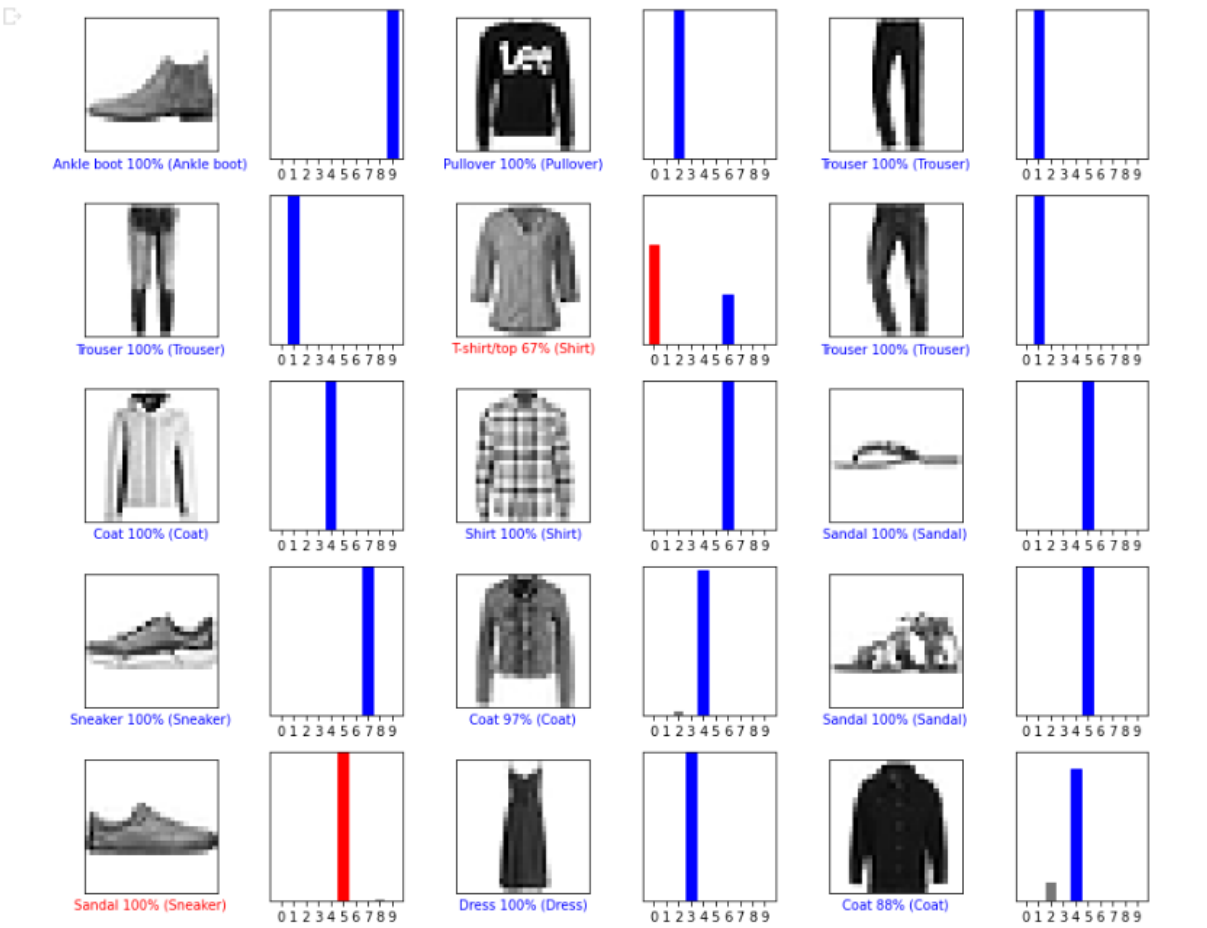
```
313/313 - 1s - loss: 0.8062 - accuracy: 0.8853 - 745ms/epoch - 2ms/step

Test accuracy: 0.8852999806404114
```

```
[23] i = 0
     plt.figure(figsize=(6,3))
     plt.subplot(1,2,1)
     plot_image(i, predictions[i], test_labels, test_images)
     plt.subplot(1,2,2)
     plot_value_array(i, predictions[i],  test_labels)
     plt.show()
```




```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```





100회 추가 훈련(200)

```
Epoch 90/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0269 - accuracy: 0.9905
Epoch 91/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0231 - accuracy: 0.9915
Epoch 92/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0259 - accuracy: 0.9905
Epoch 93/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0296 - accuracy: 0.9898
Epoch 94/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0208 - accuracy: 0.9927
Epoch 95/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0254 - accuracy: 0.9906
Epoch 96/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0243 - accuracy: 0.9916
Epoch 97/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0298 - accuracy: 0.9901
Epoch 98/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0212 - accuracy: 0.9922
Epoch 99/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0222 - accuracy: 0.9924
Epoch 100/100
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0256 - accuracy: 0.9912
<keras.callbacks.History at 0x7f22c816eed0>
```

모델이 훈련되면서 손실과 정확도 지표가 출력됩니다. 이 모델은 훈련 세트에서 약 0.88(88%) 정도의 정확도

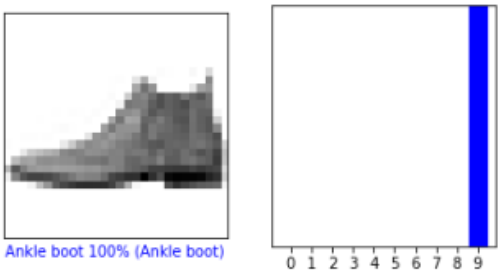
정확도 평가

다음으로, 모델이 테스트 데이터세트에서 작동하는 방식을 비교합니다.

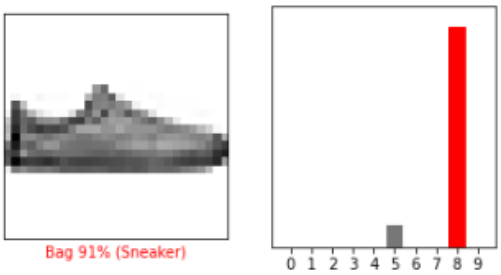
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('#nTest accuracy:', test_acc)
```

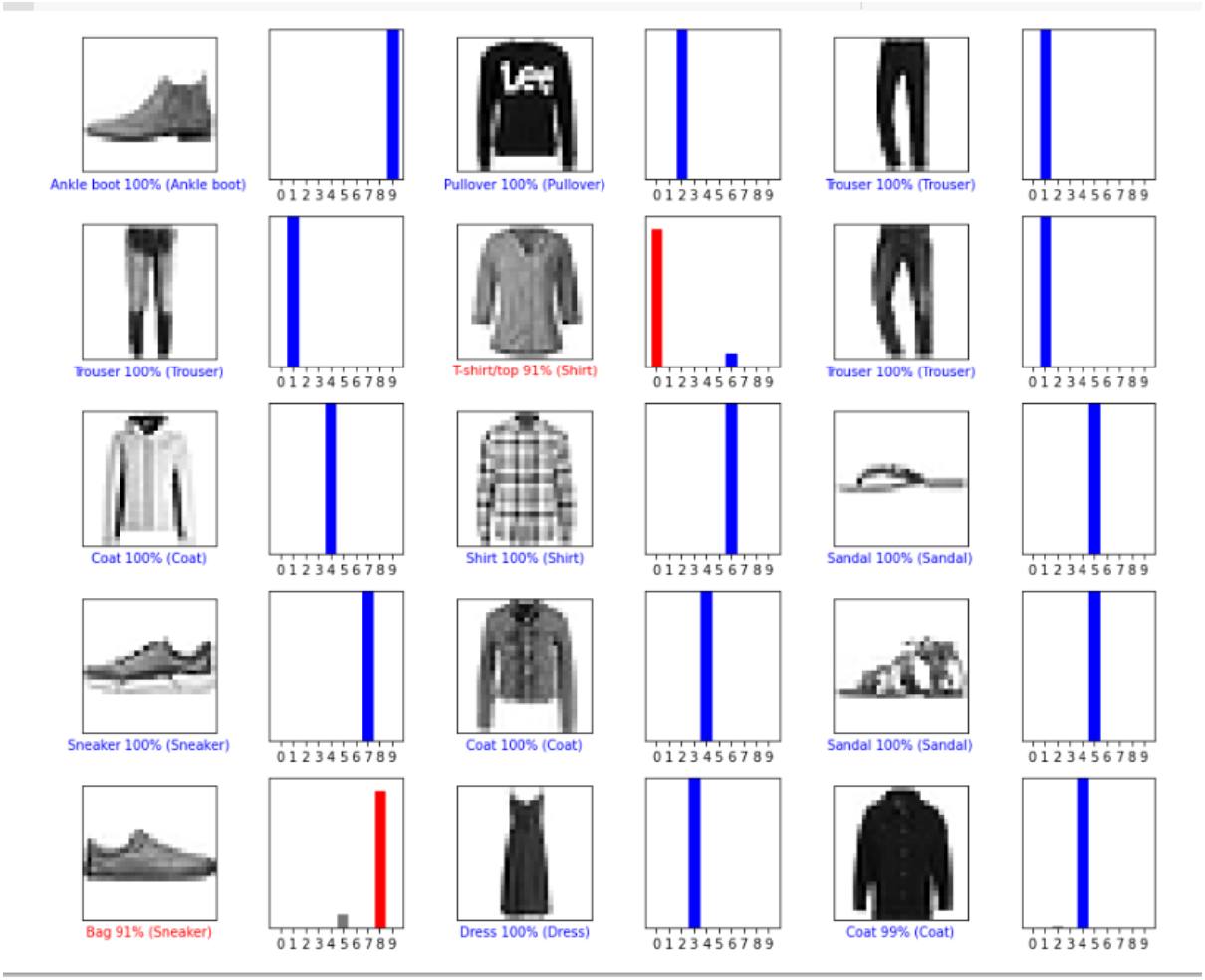
```
313/313 - 1s - loss: 1.3226 - accuracy: 0.8854 - 521ms/epoch - 2ms/step

Test accuracy: 0.8853999972343445
```

```
[40] i = 0
     plt.figure(figsize=(6,3))
     plt.subplot(1,2,1)
     plot_image(i, predictions[i], test_labels, test_images)
     plt.subplot(1,2,2)
     plot_value_array(i, predictions[i],  test_labels)
     plt.show()
```




```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```




Ankle boot 100% (Ankle boot)
Pullover 100% (Pullover)
Trouser 100% (Trouser)
Trouser 100% (Trouser)
T-shirt/top 91% (Shirt)
Trouser 100% (Trouser)
Coat 100% (Coat)
Shirt 100% (Shirt)
Sandal 100% (Sandal)
Sneaker 100% (Sneaker)
Coat 100% (Coat)
Sandal 100% (Sandal)
Bag 91% (Sneaker)
Dress 100% (Dress)
Coat 99% (Coat)
0 1 2 3 4 5 6 7 8 9

1000회 추가훈련

```
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0122 - accuracy: 0.9974
Epoch 990/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0078 - accuracy: 0.9982
Epoch 991/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0056 - accuracy: 0.9986
Epoch 992/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0090 - accuracy: 0.9980
Epoch 993/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0039 - accuracy: 0.9990
Epoch 994/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0132 - accuracy: 0.9974
Epoch 995/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0056 - accuracy: 0.9984
Epoch 996/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0069 - accuracy: 0.9983
Epoch 997/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0098 - accuracy: 0.9978
Epoch 998/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0102 - accuracy: 0.9976
Epoch 999/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0025 - accuracy: 0.9991
Epoch 1000/1000
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0074 - accuracy: 0.9982
<keras.callbacks.History at 0x7f22b47c4050>
```

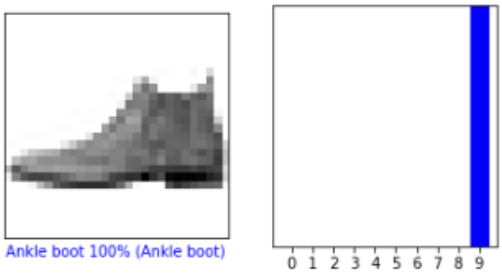
모델이 훈련되면서 손실과 정확도 지표가 출력됩니다. 이 모델은 훈련 세트에서 약 0.88(88%) 정도의 정확도를

정확도 평가

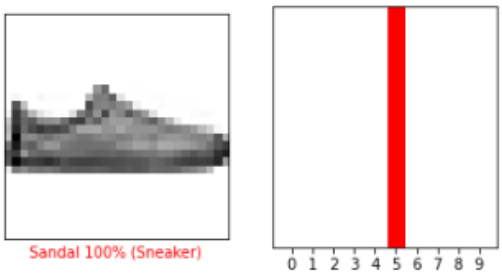
다음으로, 모델이 테스트 데이터세트에서 작동하는 방식을 비교합니다.

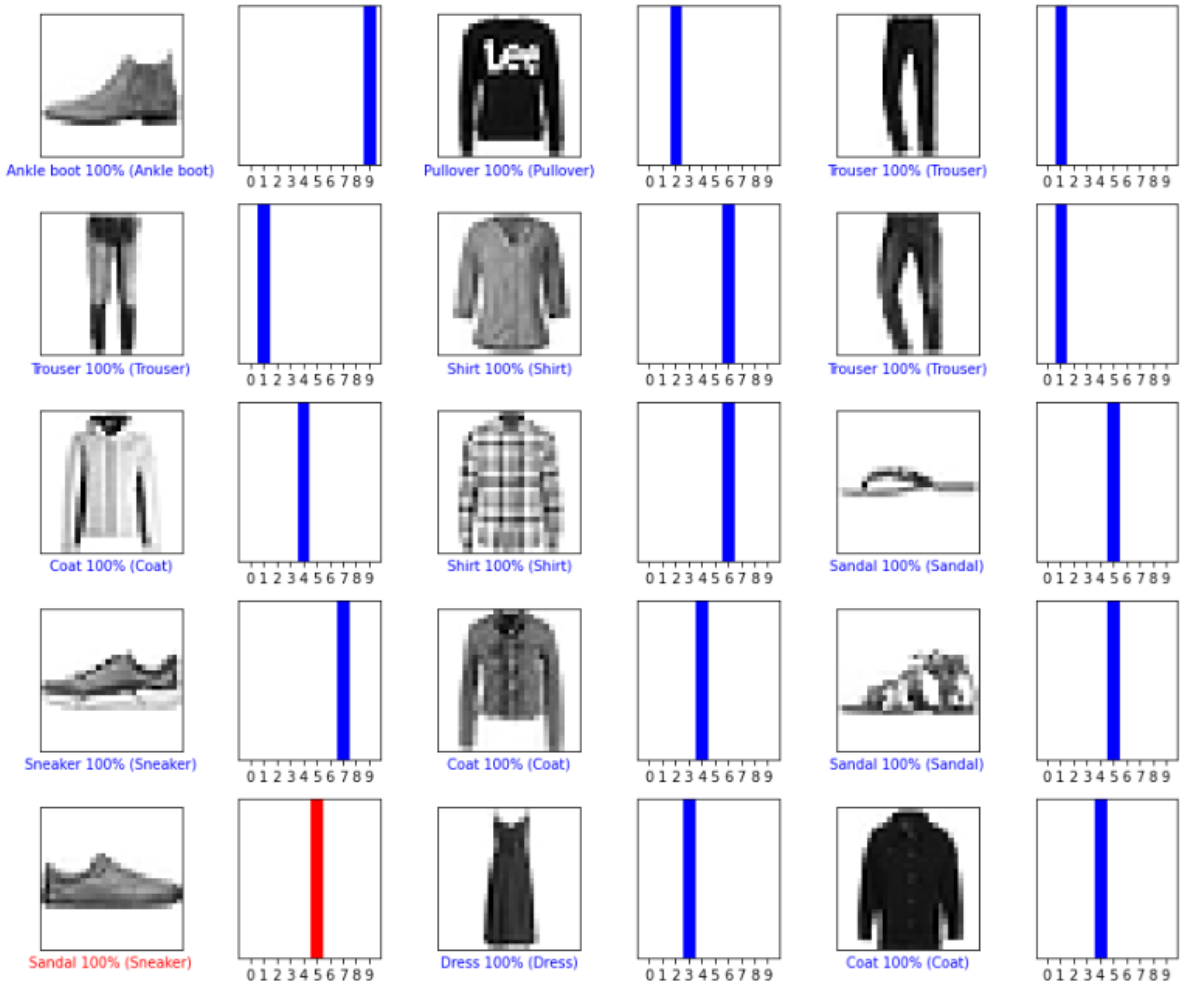
```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('#nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 3.7758 - accuracy: 0.8852 - 510ms/epoch - 2ms/step

Test accuracy: 0.885200023651123
```




```
i = 12
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels
plt.show()
```





훈련횟수를 늘릴수록 확실하게 신뢰도 점수가 올라가고 예측이 맞을 확률도 증가합니다. 훈련할때의 accuracy도0.99%까지 올라간것을 확인했습니다.

하지만 여전히 신뢰도 점수가 높지만 잘못예측한 경우가 있고,

100회째 훈련에서 셔츠를 티셔츠라고 잘못 예측한것과

200회 훈련째 이미지 12번 이미지를 가방이라고 예측한점이 잘 이해가 가지 않습니다.