```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt

fashion_mnist = keras.datasets.fashion_mnist
(X_train_full,y_train_full),(X_test,y_test)=fashion_mnist.load_data()

X_train_full.shape
X_train_full.dtype

X_valid, X_train = X_train_full[:5000]/255.0,X_train_full[5000:]/255.0
y_valid,y_train = y_train_full[:5000],y_train_full[5000:]
X_test = X_test /255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

패션 MNIST 데이터셋을 적재한다.

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

```
[3] class_names[y_train[0]]
```

```
'Coat'
```

```
[4] plt.imshow(X_train[0],cmap="binary")
    plt.axis('off')
    plt.show()
```

이미지를 확인해본다.

```
n_rows=4
n_cols =10
plt.figure(figsize=(n_cols*1.2, n_rows*1.2))
for row in range(n_rows):
  for col in range(n_cols):
    index = n_cols*row +col
    plt.subplot(n_rows,n_cols,index+1)
    plt.imshow(X_train[index], cmap = "binary")
    plt.axis('off')
    plt.title(class_names[y_train[index]], fontsize =12)
plt.subplots_adjust(wspace = 0.2, hspace = 0.5)
plt.show()
```

4행 10열로 훈련 이미지 40개를 출력한다.

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
```

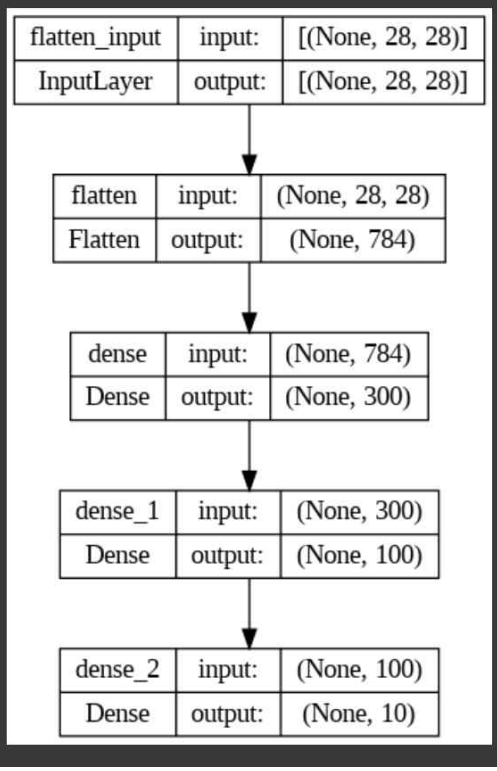
```
[7] model.add(keras.layers.Dense(300, activation="relu"))
    model.add(keras.layers.Dense(100, activation="relu"))
    model.add(keras.layers.Dense(10, activation="softmax"))
```

```
[8] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266610 (1.02 MB)
Trainable params: 266610 (1.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Sequencial API 로 모델을 만든 후 은닉층과 출력층을 추가하고 model.summary() 로 보여준다.

```
keras.utils.plot_model(model, "my_fashion_mnist_model.png", show_shapes=True)
```



Keras 모델의 구조를 시각화한다.

```
[10] model.layers
     for i in range(4):
       print(model.layers[i].name)

     flatten
     dense
     dense_1
     dense_2
```

Keras 모델에서 사용되는 모든 레이어의 목록을 반환한다.

```
[11] hidden1 = model.layers[1]

     weights,biases = hidden1.get_weights()
```

모델의 두 번째 레이어를 추출하여 hidden1 변수에 저장한 후 hidden1에서 가중치와 편향을 추출하여 weights 와 biases 변수에 각각 저장한다.

```
[13] weights.shape

     (784, 300)

[14] biases.shape

     (300,)
```

가중치와 편향의 형태를 각각 출력한다.

```
[15] weights

array([[ 0.02196094, -0.01604879,  0.05912909, ...,  0.07435071,
         0.04544255,  0.00582179],
       [-0.01522749,  0.04079878, -0.06916433, ...,  0.00088407,
         0.01567963,  0.03441385],
       [ 0.04541522,  0.05755201, -0.03514511, ...,  0.0281071 ,
        -0.01258096,  0.02456383],
       ...,
       [-0.07360154,  0.02332964, -0.00932786, ..., -0.0287984 ,
         0.02989011,  0.02534214],
       [ 0.06294772,  0.04587   , -0.0346186 , ..., -0.01705816,
        -0.05546772,  0.03612237],
       [-0.03427893,  0.06364237, -0.04471117, ..., -0.06345218,
         0.07302211, -0.06511087]], dtype=float32)
```

가중치를 출력한다.

```
[16] model.compile(loss="sparse_categorical_crossentropy",
                   optimizer = "sgd",
                   metrics = ["accuracy"])
```

모델을 컴파일한다.

```
history = model.fit(X_train,y_train,epochs = 30,
                    validation_data = (X_valid, y_valid))

Epoch 1/30
1719/1719 [==============================] - 17s 9ms/step - loss: 0.7189 - accuracy: 0.7649 - val_loss: 0.5134 - val_accuracy: 0.8280
Epoch 2/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.4887 - accuracy: 0.8288 - val_loss: 0.4820 - val_accuracy: 0.8328
Epoch 3/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.4463 - accuracy: 0.8435 - val_loss: 0.4223 - val_accuracy: 0.8530
Epoch 4/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.4199 - accuracy: 0.8511 - val_loss: 0.4015 - val_accuracy: 0.8668
Epoch 5/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3982 - accuracy: 0.8602 - val_loss: 0.3846 - val_accuracy: 0.8694
Epoch 6/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3811 - accuracy: 0.8645 - val_loss: 0.3711 - val_accuracy: 0.8730
Epoch 7/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.3679 - accuracy: 0.8695 - val_loss: 0.3684 - val_accuracy: 0.8722
Epoch 8/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.3559 - accuracy: 0.8734 - val_loss: 0.3559 - val_accuracy: 0.8738
Epoch 9/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3451 - accuracy: 0.8771 - val_loss: 0.3592 - val_accuracy: 0.8750
Epoch 10/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.3360 - accuracy: 0.8811 - val_loss: 0.3658 - val_accuracy: 0.8692
Epoch 11/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.3275 - accuracy: 0.8837 - val_loss: 0.3689 - val_accuracy: 0.8636
Epoch 12/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3197 - accuracy: 0.8853 - val_loss: 0.3320 - val_accuracy: 0.8822
Epoch 13/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.3125 - accuracy: 0.8880 - val_loss: 0.3260 - val_accuracy: 0.8816
Epoch 14/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3059 - accuracy: 0.8897 - val_loss: 0.3266 - val_accuracy: 0.8844
Epoch 15/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2984 - accuracy: 0.8927 - val_loss: 0.3392 - val_accuracy: 0.8802
Epoch 16/30
1719/1719 [==============================] - 8s 4ms/step - loss: 0.2927 - accuracy: 0.8941 - val_loss: 0.3259 - val_accuracy: 0.8790
Epoch 17/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2868 - accuracy: 0.8969 - val_loss: 0.3197 - val_accuracy: 0.8818
Epoch 18/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2808 - accuracy: 0.8992 - val_loss: 0.3330 - val_accuracy: 0.8772
Epoch 19/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2762 - accuracy: 0.9007 - val_loss: 0.3089 - val_accuracy: 0.8888
Epoch 20/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.2701 - accuracy: 0.9033 - val_loss: 0.3335 - val_accuracy: 0.8804
Epoch 21/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2660 - accuracy: 0.9039 - val_loss: 0.3201 - val_accuracy: 0.8832
Epoch 22/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.2600 - accuracy: 0.9058 - val_loss: 0.2997 - val_accuracy: 0.8898
Epoch 23/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2567 - accuracy: 0.9076 - val_loss: 0.3170 - val_accuracy: 0.8914
Epoch 24/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2520 - accuracy: 0.9089 - val_loss: 0.3133 - val_accuracy: 0.8876
Epoch 25/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2474 - accuracy: 0.9098 - val_loss: 0.2996 - val_accuracy: 0.8914
Epoch 26/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2423 - accuracy: 0.9134 - val_loss: 0.2964 - val_accuracy: 0.8944
Epoch 27/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2382 - accuracy: 0.9133 - val_loss: 0.3140 - val_accuracy: 0.8908
Epoch 28/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2350 - accuracy: 0.9157 - val_loss: 0.3009 - val_accuracy: 0.8920
Epoch 29/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2312 - accuracy: 0.9166 - val_loss: 0.3033 - val_accuracy: 0.8908
Epoch 30/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2280 - accuracy: 0.9182 - val_loss: 0.2924 - val_accuracy: 0.8912
```

주어진 학습 데이터와 검증 데이터에 대해 모델을 30번의 epoch 동안 학습시킨다.
최종 훈련 세트 정확도는 91.82% 가 나왔고, 최종 검증 세트 정확도는 89.12% 가 나왔다.

```
help(model.fit)
```

```
Help on method fit in module keras.src.engine.training:

fit(x=None, y=None, batch_size=None, epochs=1, verbose='auto', callbacks=None, validation_split=0.0, val
    Trains the model for a fixed number of epochs (dataset iterations).

    Args:
        x: Input data. It could be:
          - A Numpy array (or array-like), or a list of arrays
            (in case the model has multiple inputs).
          - A TensorFlow tensor, or a list of tensors
            (in case the model has multiple inputs).
          - A dict mapping input names to the corresponding array/tensors,
            if the model has named inputs.
          - A `tf.data` dataset. Should return a tuple
            of either `(inputs, targets)` or
            `(inputs, targets, sample_weights)`.
          - A generator or `keras.utils.Sequence` returning `(inputs,
            targets)` or `(inputs, targets, sample_weights)`.
          - A `tf.keras.utils.experimental.DatasetCreator`, which wraps a
            callable that takes a single argument of type
            `tf.distribute.InputContext`, and returns a `tf.data.Dataset`.
```

Keras의 fit 메서드에 대한 도움말을 확인한다.

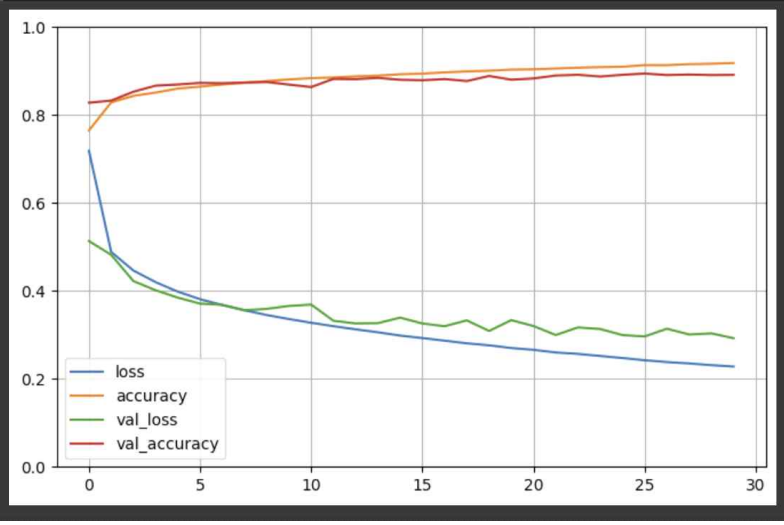
```
[19] history.params

{'verbose': 1, 'epochs': 30, 'steps': 1719}

print(history.epoch)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]

[21] history.history.keys()

dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

history.params 코드는 학습에 사용된 매개 변수들의 값들을 나타낸다.
그 후 각 epoch의 인덱스를 출력하고 학습 이력에 저장된 지표들의 이름을 출력한다.

```
history.history['accuracy']

[0.7648909091949463,
 0.828781843185424 8,
 0.8435090780258179,
 0.8510727286338806,
 0.8602181673049927,
 0.8645272850990295,
 0.869490921497345,
 0.8733817934989929,
 0.877109110355377 2,
 0.8811091184616089,
 0.8837454319000244,
 0.8852909207344055,
 0.8880000114440918,
 0.8896727561950684,
 0.8927090764045715,
 0.8941272497177124,
 0.8969454765319824,
 0.8992000222206116,
 0.9007090926170349,
 0.9032545685768127,
 0.9039090871810913,
 0.9058363437652588,
 0.9075999855995178,
 0.9089272618293762,
 0.9097636342048645,
 0.913418173789978,
 0.913345456123352,
 0.9157272577285767,
 0.9165818095207214,
 0.918218195438385]
```

history 객체에서 학습 중에 각 epoch에서의 정확도를 나타내는 리스트 또는 배열을 가져온다.

```
import pandas as pd

pd.DataFrame(history.history).plot(figsize =(8,5))
plt.grid(True)
plt.gca().set_ylim(0,1)
plt.show()
```



pandas를 사용하여 history 객체에 저장된 지표들을 데이터 프레임으로 변환한 후 시각화한다.

```
model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3289 - accuracy: 0.8822
[0.3289021849632263, 0.8822000026702881]
```

테스트 세트로 최종 일반화 오차를 추정한다.
최종 테스트 세트 정확도는 88.22%이다.

```
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_proba.round(2)
```

```
1/1 [==============================] - 0s 104ms/step
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
       [0.  , 0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```

모델을 사용하여 새로운 데이터에 대한 예측을 수행하고, 각 클래스에 속할 확률을 출력한다.

```
[26] plt.figure(figsize=(7.2,2.4))
     for index,image in enumerate(X_new):
       plt.subplot(1,3,index+1)
       plt.imshow(image, cmap = "binary")
       plt.axis('off')
       plt.title(class_names[y_test[index]],fontsize=12)
     plt.subplots_adjust(wspace=0.2, hspace=0.5)
     plt.show()
```

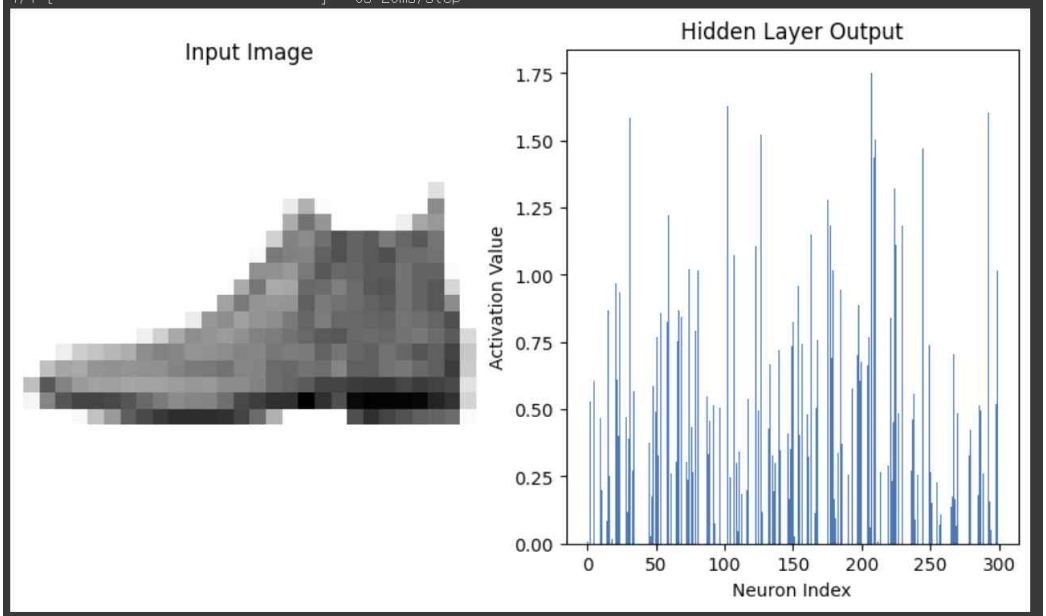
예측한 그림을 확인한다.

```
hidden_layer_output_model = keras.models.Model(inputs = model.input, outputs = model.layers[1].output)
```

```
def visualize_hidden_layer_output(image_index):
    image = X_test[image_index]
    hidden_layer_output = hidden_layer_output_model.predict(image.reshape(1,28,28))
    plt.figure(figsize = (10,5))
    plt.subplot(1,2,1)
    plt.imshow(image, cmap = "binary")
    plt.title("Input Image")
    plt.axis('off')

    plt.subplot(1,2,2)
    plt.bar(range(len(hidden_layer_output.flatten())), hidden_layer_output.flatten())
    plt.title("Hidden Layer Output")
    plt.xlabel("Neuron Index")
    plt.ylabel("Activation Value")

    plt.show()

    print("Actual Label:", class_names[y_test[image_index]])
    predicted_class_index = tf.argmax(model.predict(image.reshape(1, 28, 28)), axis=1).numpy()[0]
    print("Predicted Label:", class_names[predicted_class_index])
image_index = 0
visualize_hidden_layer_output(image_index)
```

은닉층의 출력을 시각화하고, 해당 이미지의 실제 레이블을 함께 표시하는 코드이다.

```
1/1 [==============================] - 0s 20ms/step
```




```
Actual Label: Ankle boot
1/1 [==============================] - 0s 21ms/step
Predicted Label: Ankle boot
```

코드 수행 결과이다.