In [3]:

```
# TensorFlow and tf.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
import tensorflow

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

In [4]:

```
(x_train, y_train), (x_test, y_test) = tensorflow.keras.datasets.fashion_mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
#x_train과 x_test가 0~255사이의 숫자인데, 이걸 0~1 사이의 숫자로 바꿔주는 것.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 2us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 2s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/
t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 1s 0us/step
```

In [11]:

```
x_train.shape
#28x28의 픽셀, 총 60000장.
```

Out[11]:

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

In [13]:

```
x_train[0].shape
```

Out[13]:

```
(28, 28)
```

In [16]:

```
y_train[0]
```

Out[16]:

```
9
```

In [5]:

```
fashion_mnist = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
#총 10개의 카테고리.

plt.figure(figsize = (100,100))
#전체 틀 사이즈의 크기. 크게 보기 위해서 100x100으로 했을 뿐이다.

for i in range(100): #총 100개 정도를 plot 해보자. i = 0부터 99까지
    plt.subplot(10,10,i+1) #row와 column을 10x10으로 만들어두고, 1부터 하나씩 인덱스를 매기게 된다.
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i], cmap = plt.cm.binary) #실질적으로 plot해주는 함수
    #x_train[0] = 28x28의 그림이 나올 것.

    plt.title(fashion_mnist[y_train[i]], fontsize = 100)
    #y_train의 숫자값을 표시해줄 수도 있지만 그걸 위에서 미리 지정한 글자로 표시해주는 것.

plt.show()
```

In [18]:

```
x_train.shape
```

Out[18]:

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

In [6]:

```
img_rows, img_cols = x_train.shape[1:]
#x_train.shape가 (60000, 28, 28)이므로 [1:]은 1번째부터 끝까지. 즉, (28, 28)이다.

model = Sequential()
model.add(Flatten(input_shape=(img_rows, img_cols)))
#mnist에서와 마찬가지로 28x28,짜리를 flatten해준다.

model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
WARNING:tensorflow:From C:₩Users₩Hyunseo Choi₩Anaconda3₩lib₩site-packages₩tensorfl
ow₩python₩ops₩resource_variable_ops.py:435: colocate_with (from tensorflow.python.
framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [7]:

```
model.summary()

#parameter number는 화살표의 숫자.
#이 모델이 어떻게 이렇게 생기게 되는지를 알아야 한다.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

In [8]:

```
Adam = tensorflow.keras.optimizers.Adam
model.compile(optimizer = Adam(lr = 0.0005), loss='sparse_categorical_crossentropy', metrics=['a
ccuracy'])
#lr = 0.005가 default일 것. 그러나 여기서는 0.0005로 설정해 두었다.
```

In [9]:

```
print(model.input_shape)
print(model.output_shape)
```

```
(None, 28, 28)
(None, 10)
```

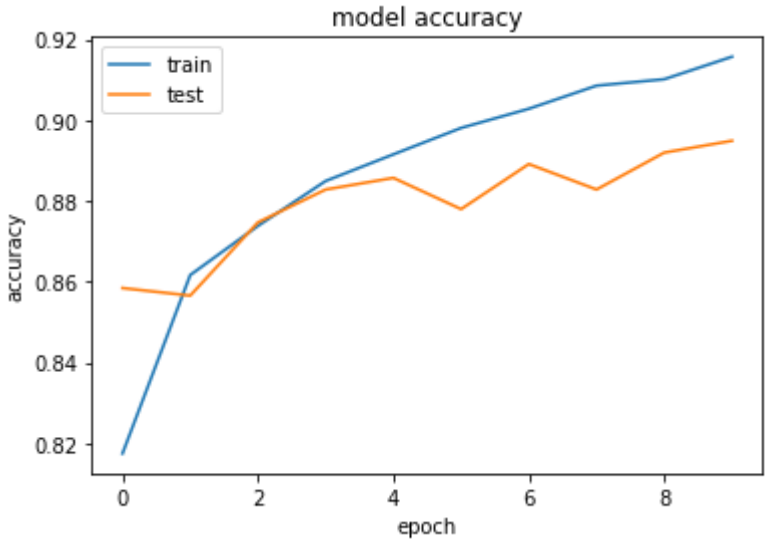
In [10]:

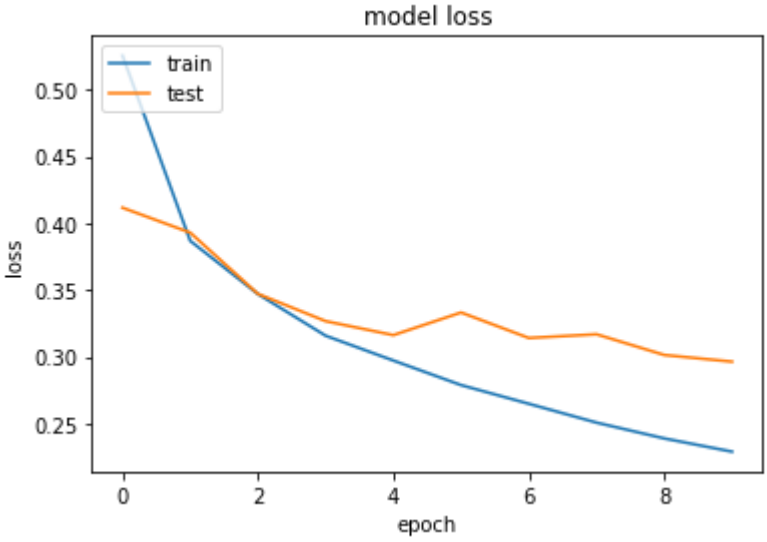
```
history = model.fit(x_train, y_train, epochs=10, validation_split = 0.33)
#계산되어 나온 값을 history라는 변수에 저장할 것이다.
#epoch = 10, 데이터를 10번 돌려본다.
#validation_split = 0.33, 67%만큼을 train set으로 돌릴 것이고, 나머지 33%을 test set으로 돌릴 것이다.
```

```
Train on 40199 samples, validate on 19801 samples
Epoch 1/10
40199/40199 [==============================] - 17s 414us/sample - loss: 0.5254 - a
cc: 0.8175 - val_loss: 0.4116 - val_acc: 0.8584
Epoch 2/10
40199/40199 [==============================] - 16s 400us/sample - loss: 0.3868 - a
cc: 0.8617 - val_loss: 0.3929 - val_acc: 0.8566
Epoch 3/10
40199/40199 [==============================] - 18s 440us/sample - loss: 0.3470 - a
cc: 0.8738 - val_loss: 0.3471 - val_acc: 0.8747
Epoch 4/10
40199/40199 [==============================] - 18s 460us/sample - loss: 0.3158 - a
cc: 0.8849 - val_loss: 0.3267 - val_acc: 0.8828
Epoch 5/10
40199/40199 [==============================] - 14s 345us/sample - loss: 0.2972 - a
cc: 0.8915 - val_loss: 0.3162 - val_acc: 0.8857
Epoch 6/10
40199/40199 [==============================] - 14s 354us/sample - loss: 0.2788 - a
cc: 0.8981 - val_loss: 0.3332 - val_acc: 0.8780
Epoch 7/10
40199/40199 [==============================] - 15s 366us/sample - loss: 0.2648 - a
cc: 0.9028 - val_loss: 0.3141 - val_acc: 0.8891
Epoch 8/10
40199/40199 [==============================] - 15s 362us/sample - loss: 0.2507 - a
cc: 0.9086 - val_loss: 0.3168 - val_acc: 0.8828
Epoch 9/10
40199/40199 [==============================] - 14s 360us/sample - loss: 0.2389 - a
cc: 0.9101 - val_loss: 0.3014 - val_acc: 0.8920
Epoch 10/10
40199/40199 [==============================] - 14s 352us/sample - loss: 0.2290 - a
cc: 0.9157 - val_loss: 0.2964 - val_acc: 0.8949
```

In [19]:

```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





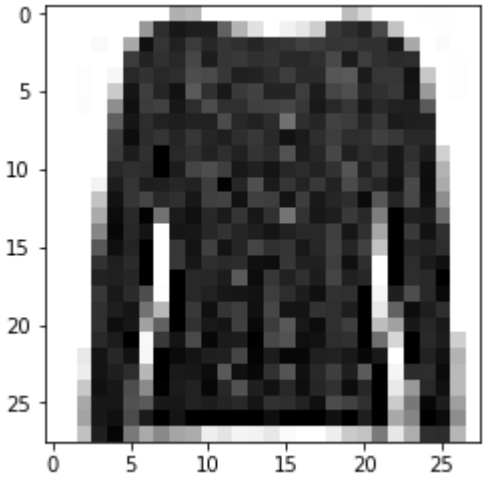
In [ ]:

```
#accuracy
#mnist는 너무 빨리 99%에 도달하기 때문에 fashion_mnist를 대안으로 많이 사용한다.
#일반화가 중요한데, 일반화가 되지 않은 경우 over fit의 상황이 생길 수 있다.
#epoch = 3의 상황에서 train과 test의 fit이 모두 정상적이다. 그러나 epoch = 4가 되는 순간 overfit
이 일어나게 된다.
#epoch = 4의 model을 미리 저장해뒀다면 이 모델을 사용하는 것이 가장 이상적일 것이다.

#loss
#epoch = 4의 상황에서 새로운 data에 대해서는 loss가 올라감을 볼 수 있다. 이런 상황이 over fit. g
eneralize가 되지 않았음을 알 수 있다.
```

In [20]:

```
testID = 89
fashion_mnist = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
plt.imshow(x_test[testID], cmap = plt.cm.binary)
out = model.predict(x_test[testID].reshape(1,img_rows, img_cols))
#predict함수. 이 과정을 inference라고 한다. 입력값만 주면 알아서 계산해서 결과값으로 보여줄 것.
#test set에서 하나를 가져와본다. 89번째 test 항목을 가져와서 reshape를 한다.
#input_shape가 (None, 28, 28)이었는데, 이 상황에서 28x28인 그림 한 장임을 나타내주기 위해서 (1,
 img_rows, img_cols)로 reshape해준다.

print(fashion_mnist[np.argmax(out)])
#out의 결과값들 중 몇 번째가 가장 큰 값인가? 2를 뱉을 것이고, fashion_mnist[2]는 Pullover일 것.
```

```
Pullover
```



In [21]:

```
#pullover?

out
#총 10개의 값. e-08 = 10의 -8승.
#e-01이 가장 작은 값. 72퍼센트가 3번째로 인식을 했다. 따라서 Pullover임을 도출한다.
```

Out[21]:

```
array([[1.9300172e-01, 5.4048115e-07, 7.2976071e-01, 2.9032328e-05,
        1.3627815e-03, 9.8983563e-08, 7.5483717e-02, 7.1278219e-09,
        3.6150101e-04, 1.7685327e-08]], dtype=float32)
```

In [22]:

```
np.sum(out) #총 10개의 값을 모두 더하면 1이 나온다.
```

Out[22]:

1.0

In [23]:

```
model.save_weights('fashion_mnist_weights.h5')
#중간중간의 모델을 save할 때 이렇게 쓴다. h5파일로 저장하게 된다.
```

In [24]:

```
model.load_weights('fashion_mnist_weights.h5')
#다시 불러오고 싶다면 load를 하는데, 그 weight를 불러오게 된다.
```

In [ ]:

```

```