

## 6.3 Fashion MNIST 데이터셋에 적용하기

# 데이터 로드와 정규화

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()

train_X = train_X / 255.0
test_X = test_X / 255.0
```

# Conv2D 레이어를 위한 모양 변형

- Conv2D 레이어로 컨볼루션 연산을 수행
  - 이미지는 보통 채널을 가짐
    - 컬러 이미지는 RGB의 3채널, 흑백 이미지는 1채널
  - Conv2D레이어는 채널을 가진 형태의 데이터를 받도록 기본적으로 설정
  - 채널을 갖도록 데이터의 Shape를 변형
  - Fashion MNIST데이터를 구성하는 흑백 이미지는 1개의 채널을 갖기 때문에 reshape() 함수를 사용해 데이터의 가장 뒤쪽에 채널 차원을 추가

```
# reshape 이전
print(train_X.shape, test_X.shape)

train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

# reshape 이후
print(train_X.shape, test_X.shape)
```

```
(60000, 28, 28) (10000, 28, 28)
(60000, 28, 28, 1) (10000, 28, 28, 1)
```

# 데이터 확인 시각화

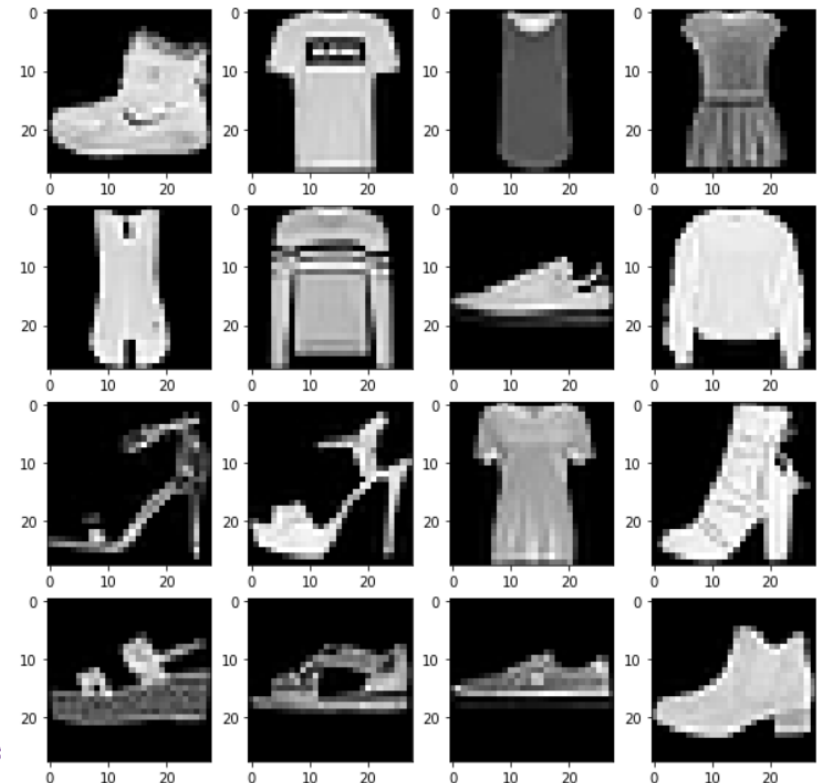
```
import matplotlib.pyplot as plt

# 전체 그래프의 크기를 width = 10, height = 10으로 지정합니다.
plt.figure(figsize=(10, 10))
for c in range(16):
    # 4행 4열로 지정한 그리드에서 c+1번째의 칸에 그래프를 그립니다. 1~16번째 칸을 채우게 됩니다.
    plt.subplot(4,4,c+1)
    plt.imshow(train_X[c].reshape(28,28), cmap='gray')

plt.show()

# 훈련 데이터이 1~16번째 까지의 라벨 프린트합니다.
print(train_Y[:16])
```

라벨	범주
0	티셔츠/상의
1	바지
2	스웨터
3	드레스
4	코트
5	샌들
6	셔츠
7	운동화
8	가방
9	부츠



[9 0 0 3 0 2 7 2 5 5 0 9 5 5 7 9]

# 컨볼루션 신경망 모델 정의

## • 컨볼루션 신경망 모델

- 풀링 레이어 또는 드롭아웃 없이 정의된 모델

## • Conv2D의 인수

- kernel\_size
  - 필터 행렬의 크기, 수용 영역 (receptive field)
    - (높이, 너비)
- filters
  - 필터의 개수

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 16)	160
conv2d_1 (Conv2D)	(None, 24, 24, 32)	4640
conv2d_2 (Conv2D)	(None, 22, 22, 64)	18496
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3965056
dense_1 (Dense)	(None, 10)	1290
Total params: 3,989,642		
Trainable params: 3,989,642		
Non-trainable params: 0		

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1),
                           kernel_size=(3,3), filters=16),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.summary()
```

# GPU 사용 설정

- 구글 코랩에서는 무료로 GPU를 사용
  - [메뉴]-[런타임]-[런타임 유형 변경]-[하드웨어 가속기]-[GPU]로 지정
- GPU 확인
  - 구글 코랩에서 지원하는 GPU의 성능
    - Tesla K80을 사용

```
[ ] 1 !nvidia-smi
```



Tue Apr 21 06:56:00 2020

+-----+ <table><tr><td colspan="2">NVIDIA-SMI</td><td>440.64.00</td><td colspan="3" rowspan="2">Driver Version: 418.67</td><td colspan="4" rowspan="2">CUDA Version: 10.1</td></tr></table> +-----+										NVIDIA-SMI		440.64.00	Driver Version: 418.67			CUDA Version: 10.1						
NVIDIA-SMI		440.64.00	Driver Version: 418.67			CUDA Version: 10.1																
+-----+ <table><tr><th>GPU</th><th>Name</th><th>Persistence-M</th><th>Bus-Id</th><th>Disp.A</th><th>Volatile Uncorr. ECC</th></tr><tr><th>Fan</th><th>Temp</th><th>Perf</th><th>Pwr:Usage/Cap</th><th>Memory-Usage</th><th>GPU-Util</th><th>Compute M.</th></tr></table> +-----+										GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC																	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.																
0	Tesla K80	Off	00000000:00:04.0	Off	0																	
N/A	31C	P8	26W / 149W	0MiB / 11441MiB	0%	Default																

+-----+ <table><tr><td colspan="5">Processes:</td><td>GPU Memory</td></tr><tr><th>GPU</th><th>PID</th><th>Type</th><th>Process name</th><th></th><th>Usage</th></tr></table> +-----+						Processes:					GPU Memory	GPU	PID	Type	Process name		Usage
Processes:					GPU Memory												
GPU	PID	Type	Process name		Usage												
No running processes found																	

Processes:						GPU Memory
GPU	PID	Type	Process name		Usage	
No running processes found						

# 컨볼루션 신경망 모델 학습

```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

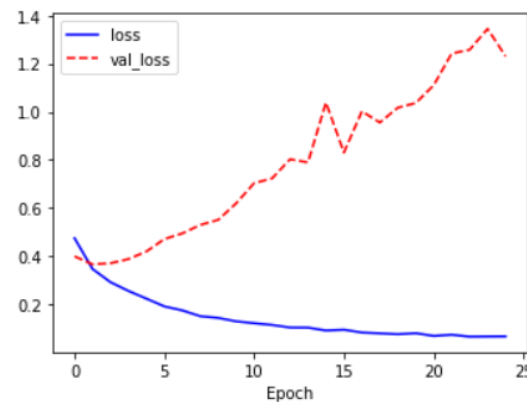
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
```

```
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()
```

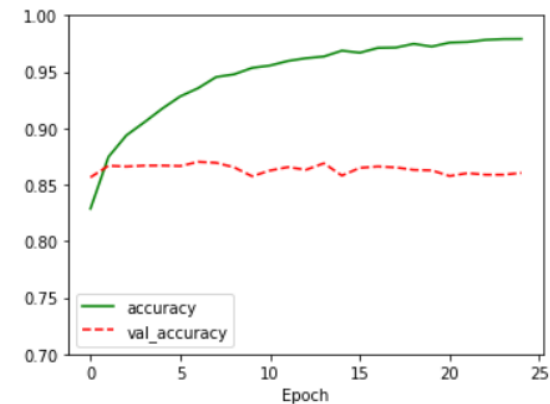
```
plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'r--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
```

```
plt.show()
```

```
model.evaluate(test_X, test_Y, verbose=0)
```



[1.3350350856781006, 0.855400025844574]



# 풀링 레이어, 드롭아웃 레이어 추가

## • 과적합을 줄이는 데 기여

- MaxPool2D(strides=(2,2)), Dropout(rate=0.3)
  - **strides**: 필터가 계산 과정에서 한 스텝마다 이동하는 크기
    - 기본값은 (1,1)이고, (2,2) 등으로 설정할 경우 한 칸씩 건너뛰면서 계산
  - **rate**: 제외할 뉴런의 비율
- Params의 개수
  - **기존 3,989,642에서 241,546으로 대폭 감소**
    - Flatten에 들어오는 Params의 개수가 기존 (None, 30976)에 비해 (None, 1152)

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64),
    tf.keras.layers.MaxPool2D(strides=(2,2)),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(rate=0.3),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

```
])
```

```
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.summary()
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
=====		
conv2d_9 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_4 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_10 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_5 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_11 (Conv2D)	(None, 3, 3, 128)	73856
flatten_3 (Flatten)	(None, 1152)	0
dense_6 (Dense)	(None, 128)	147584
dropout_2 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 10)	1290
=====		

Total params: 241,546  
Trainable params: 241,546  
Non-trainable params: 0



# 훈련과 시각화

```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

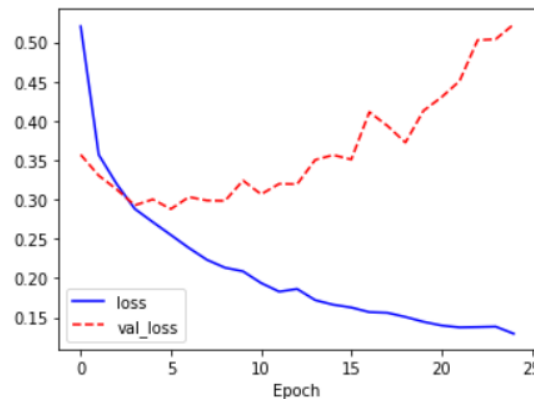
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
```

```
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()
```

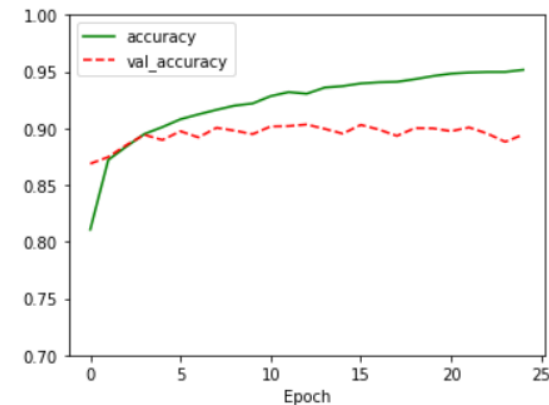
```
plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'r--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
```

```
plt.show()
```

```
model.evaluate(test_X, test_Y, verbose=0)
```



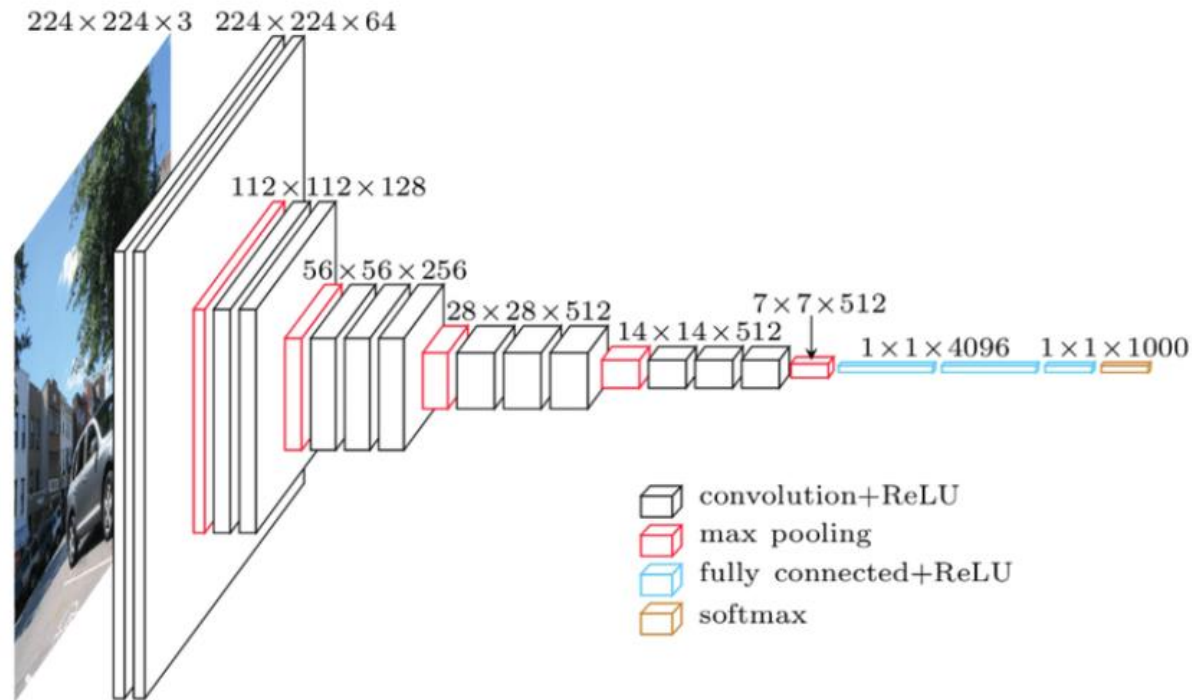
[0.5307855606079102, 0.8906999826431274]



CNN(합성곱)  
성능 높이기

# 많은 층 쌓기

## • VGGNet 컨볼루션 신경망 응용



VGGNet architecture [19]

# 데이터 불러오기 및 정규화

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()

train_X = train_X / 255.0
test_X = test_X / 255.0

# reshape 이전
print(train_X.shape, test_X.shape)

train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

# reshape 이후
print(train_X.shape, test_X.shape)

(60000, 28, 28) (10000, 28, 28)
(60000, 28, 28, 1) (10000, 28, 28, 1)
```

# CNN 모델 정의

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32,
                           padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128, padding='same', activation='relu')
    ,
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=256, padding='valid', activation='relu'
    ),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

# VGG-7

- 다음 패턴을 2차례 반복
  - 컨볼루션 레이어를 2개 겹치고 풀링 레이어를 1개 사용
  - 과적합을 방지의 드롭아웃 레이어
    - 풀링 레이어 다음에 드롭아웃 레이어 배치
- Flatten 층, 드롭아웃 레이어를 배치
  - 다음에 이어지는 3개의 Dense 레이어 사이
- VGG-7 레이어
  - 컨볼루션 레이어와 Dense 레이어의 개수

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 32)	320
conv2d_5 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
dropout_4 (Dropout)	(None, 14, 14, 64)	0
conv2d_6 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_7 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_5 (Dropout)	(None, 6, 6, 256)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_3 (Dense)	(None, 512)	4719104
dropout_6 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 256)	131328
dropout_7 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 10)	2570
Total params: 5,240,842		
Trainable params: 5,240,842		
Non-trainable params: 0		

# 학습과 시각화

```
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

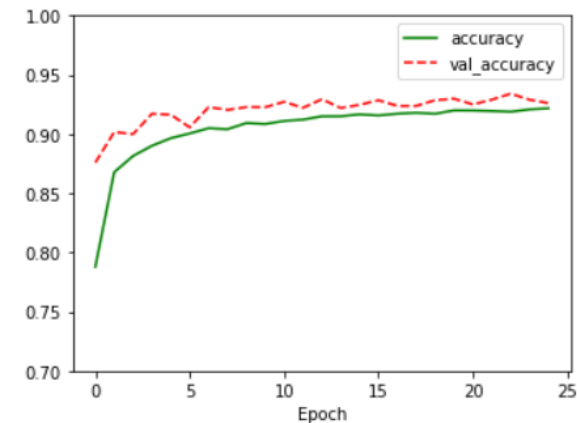
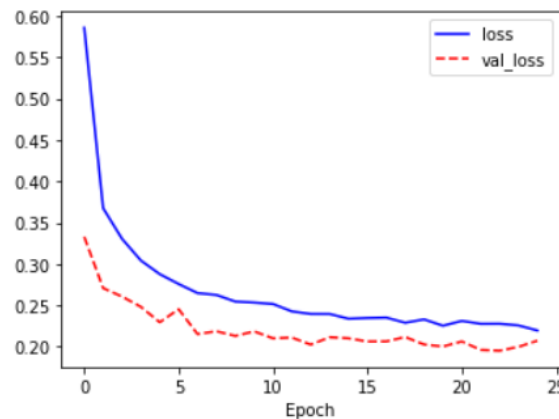
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
```

```
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'], 'r--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
```

```
plt.show()
```

```
model.evaluate(test_X, test_Y, verbose=0)
```



```
[0.23273345828056335, 0.9169999957084656]
```

# 이미지 보강

- 훈련데이터에 없는 이미지를 새롭게 만들어내서 훈련데이터를 보강하는 기법
- **ImageDataGenerator**
  - Tensorflow 이미지 보강 작업



```

from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np

image_generator = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.10,
    shear_range=0.5,
    width_shift_range=0.10,
    height_shift_range=0.10,
    horizontal_flip=True,
    vertical_flip=False)

augment_size = 100

x_augmented = image_generator.flow(np.tile(train_X[0].reshape(28*28), 100).reshape(-1, 28, 28, 1),
                                   np.zeros(augment_size), batch_size=augment_size, shuffle=False)

.next()[0]

```

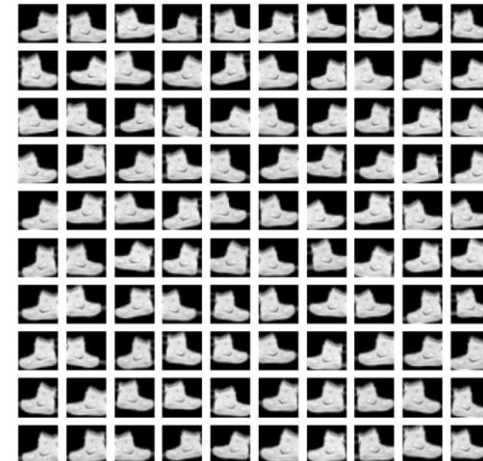
# 새롭게 생성된 이미지 표시

```

import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for c in range(100):
    plt.subplot(10, 10, c+1)
    plt.axis('off')
    plt.imshow(x_augmented[c].reshape(28, 28), cmap='gray')

plt.show()

```



## 3만개 이미지 보강

- 훈련 데이터의 50%인 30,000개의 이미지를 추가하기 위해 `augment_size=30000`로 설정
- 이미지를 변형할 원본 이미지를 찾기 위해 `np.random.randint()` 함수를 활용하여 0~59,999 범위의 정수 중에서 30,000개의 정수를 뽑았습니다. 이 때 정수는 중복 가능합니다.
- 만약, 중복을 원치 않으면, `np.random.ranint()` 대신에 `np.random.choice()` 함수를 사용하고 `replace` 인수를 `False`로 설정
- `copy()` 함수를 사용하여 원본은 보전
- `image_generator.flow()` 함수를 사용
  - 30,000개의 새로운 이미지를 생성
- `np.concatenate()`
  - 훈련 데이터에 보강 이미지를 추가

# 이미지 보강 소스

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(train_X, train_Y), (test_X, test_Y) = fashion_mnist.load_data()

train_X = train_X / 255.0
test_X = test_X / 255.0

# reshape 이전
print(train_X.shape, test_X.shape)

train_X = train_X.reshape(-1, 28, 28, 1)
test_X = test_X.reshape(-1, 28, 28, 1)

# reshape 이후
print(train_X.shape, test_X.shape)

image_generator = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.10,
    shear_range=0.5,
    width_shift_range=0.10,
    height_shift_range=0.10,
    horizontal_flip=True,
    vertical_flip=False)

augment_size = 30000

randidx = np.random.randint(train_X.shape[0], size=augment_size)
x_augmented = train_X[randidx].copy()
y_augmented = train_Y[randidx].copy()
x_augmented = image_generator.flow(x_augmented, np.zeros(augment_size), batch_size=augment_size, shuffle=False).next()[0]

# 원래 데이터인 x_train에 이미지 보강된 x_augmented를 추가합니다.
train_X = np.concatenate((train_X, x_augmented))
train_Y = np.concatenate((train_Y, y_augmented))

print(train_X.shape)
```

# 모델 정의

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(input_shape=(28,28,1), kernel_size=(3,3), filters=32,
                           padding='same', activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, padding='same', activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=128, padding='same', activation='relu'),

    tf.keras.layers.Conv2D(kernel_size=(3,3), filters=256, padding='valid', activation='relu'),

    tf.keras.layers.MaxPool2D(pool_size=(2,2)),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=256, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(units=10, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# model.summary()
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)
```

# 훈련 시각화

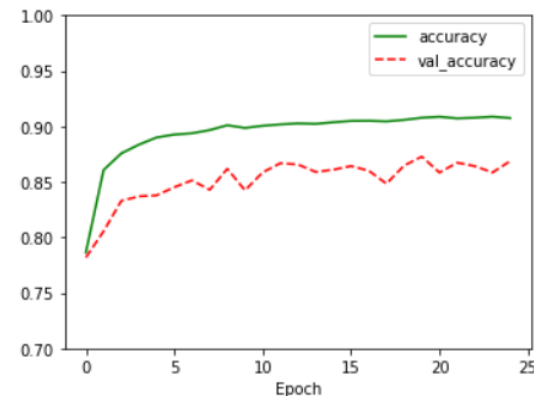
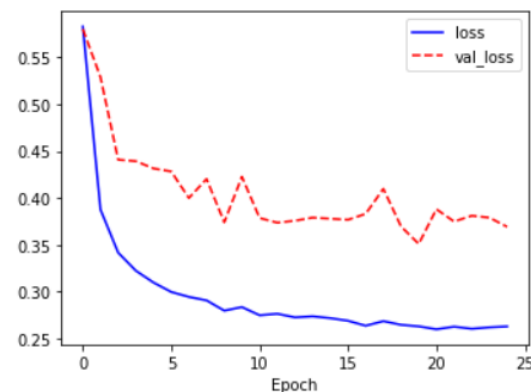
```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

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'], 'r--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()

model.evaluate(test_X, test_Y, verbose=0)
```

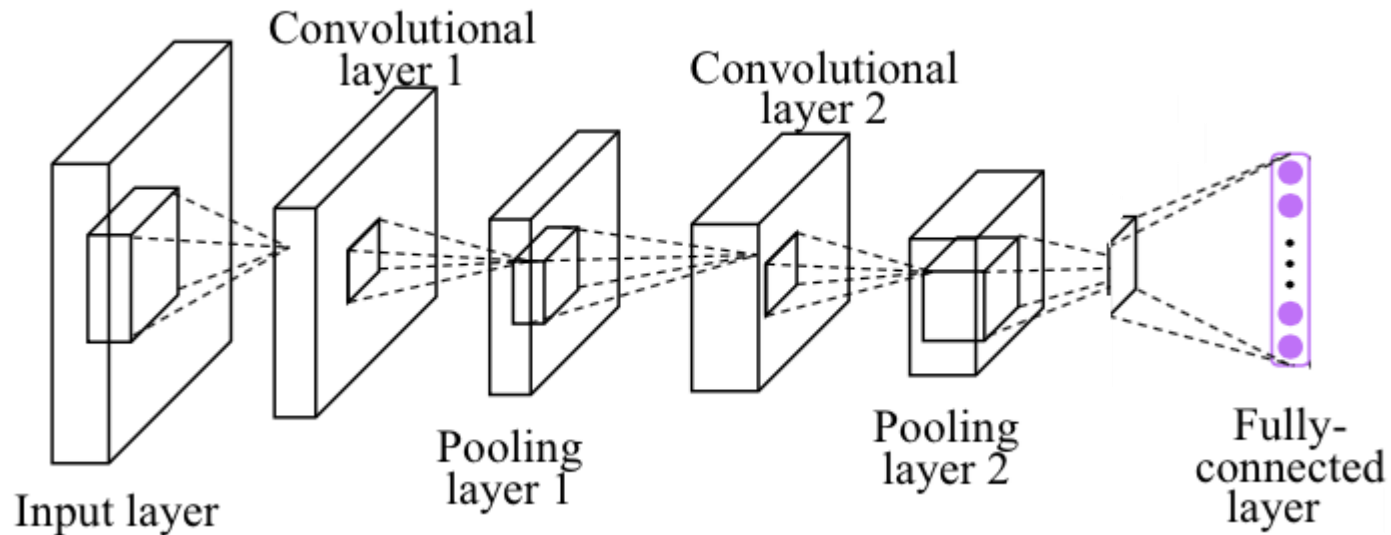


[0.2110530138015747, 0.9251000285148621]

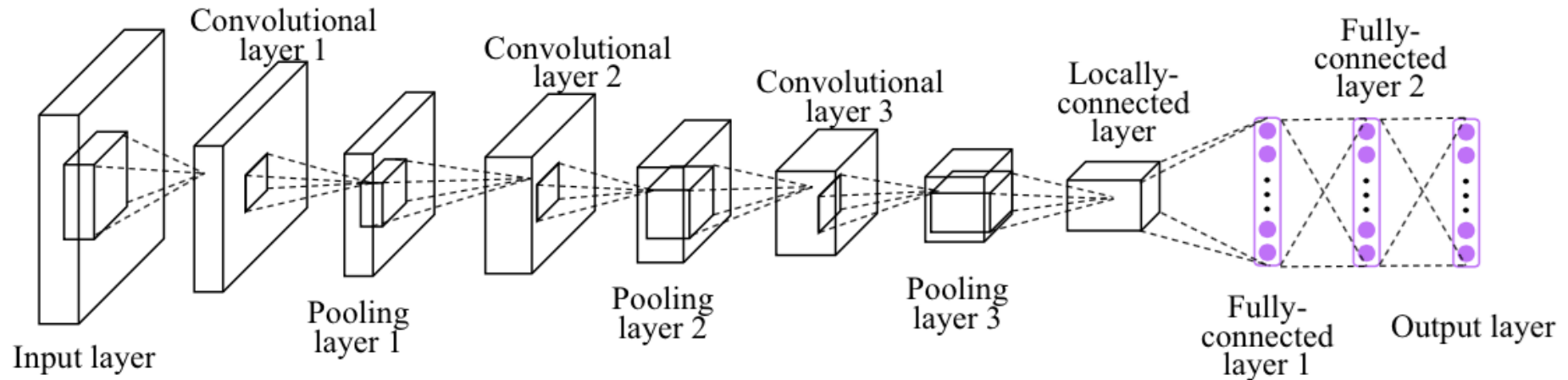
# CNN(합성곱) 텐서플로 구현

# MNIST simple CNN

- 정확도 98.6%



# Deep CNN





파라미터( $W$ 와  $b$ ) 수 계산

# W와 b의 수 계산

- 컨볼루션 파라미터 수 =  $\text{커널사이즈}^2 * \text{커널수} * \text{채널(색상)} + \text{커널수(bias)}$
- 일반 완전 연결층 파라미터 수 =  $(\text{입력수} + 1) * \text{출력수}$

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 26, 26, 32)	320 ← $(3*3 * 32 * 1) + 32$
max_pooling2d_14 (MaxPooling)	(None, 13, 13, 32)	0
flatten_14 (Flatten)	(None, 5408)	0
dense_31 (Dense)	(None, 128)	692352 ← $(5408 + 1) * 128$
dense_32 (Dense)	(None, 10)	1290

=====  
 Total params: 693,962  
 Trainable params: 693,962  
 Non-trainable params: 0

```

# 2. 모델 구성하기
model = Sequential()
    #컨볼루션 층
model.add(Conv2D(32, input_shape=(28,28,1), kernel_size=(3, 3), activation='relu'))
    #맥스풀링
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
    #완전 연결층
model.add(Dense(128, activation='relu'))
    #출력층
model.add(Dense(10, activation='softmax'))
  
```