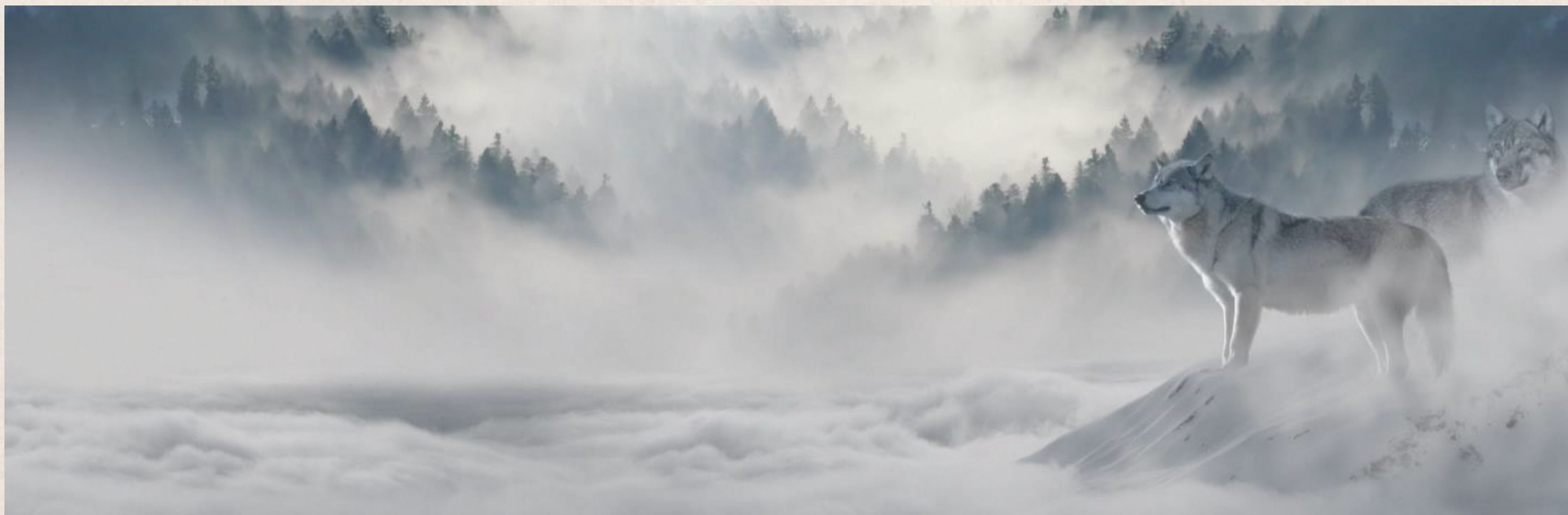


# CNN\_fashion100





# import

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

tensorflow` 라이브러리에서 `keras` 모듈을 가져옵니다. (keras 모듈 사용)

`keras` 모듈에서 `layers` 모듈을 가져옵니다 (layers모듈 사용)



# gpu설정

```
%tensorflow_version 2.x

import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
    raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

Colab only includes TensorFlow 2.x; %tensorflow\_version has no effect.  
Found GPU at: /device:GPU:0



## 데이터 + 정규화

```
fashion_mnist = tf.keras.datasets.fashion_mnist  
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()  
  
train_images, test_images = train_images / 255.0, test_images / 255.0
```

fashion 100 데이터 불러 오기



모델 예시

```
with tf.device('/device:GPU:0'):
    model = tf.keras.models.Sequential([

        tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32, input_shape=(28,28), kernel_initializer='he_uniform', activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32, kernel_initializer='he_uniform', activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPool2D(pool_size=(2,2)),
        tf.keras.layers.Dropout(0.25),

        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

batch_normalization_9 (Batch Normalization)	(None, 28, 28, 32)	128
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 32)	0
dropout_6 (Dropout)	(None, 14, 14, 32)	0
conv2d_10 (Conv2D)	(None, 12, 12, 64)	18496
batch_normalization_10 (Batch Normalization)	(None, 12, 12, 64)	256
conv2d_11 (Conv2D)	(None, 10, 10, 64)	36928
batch_normalization_11 (Batch Normalization)	(None, 10, 10, 64)	256
max_pooling2d_5 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_7 (Dropout)	(None, 5, 5, 64)	0
flatten_2 (Flatten)	(None, 1600)	0
dense_4 (Dense)	(None, 512)	819712
dropout_8 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 100)	51300
=====		
Total params: 937,348		
Trainable params: 936,964		
Non-trainable params: 384		



# Conv2D (필터)

```
tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, input_shape=(28,28,1), 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)),
```

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel		
0	-1	0
-1	5	-1
0	-1	0

114				

1. 입력 데이터는 3\*3 크기를 가지는 필터 (filter)라는 작은 윈도우와 곱해집니다

Conv2D 함수는 이러한 필터를 학습하여 최적의 필터를 찾아내는 과정도 포함하고 있습니다. 이를 통해 입력 데이터에서 가장 중요한 특징을 추출할 수 있게 됩니다.



# Conv2D (패딩)

```
tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, input_shape=(28,28,1), 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)),
```

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel		
0	-1	0
-1	5	-1
0	-1	0

114				

padding이 same인 경우, 출력 특징맵의 크기가 입력 특징맵의 크기와 상관없이 동일하게 유지됩니다.

3 <sub>0</sub>	3 <sub>1</sub>	2 <sub>2</sub>	1	0
0 <sub>2</sub>	0 <sub>2</sub>	1 <sub>0</sub>	3	1
3 <sub>0</sub>	1 <sub>1</sub>	2 <sub>2</sub>	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

padding이 valid인 경우 출력 특징맵의 크기는 입력 특징맵의 크기와 필터의 크기에 따라 결정됩니다.

스트라이드를 통해 조절 가능



## Conv2D (activation(relu))

```
tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, input_shape=(28,28,1), 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)),
```

**ReLU**  
 $\max(0, x)$



ReLU 함수는 입력값이 0보다 클 때는 선형 함수처럼 작동하지만, 입력값이 0 이하일 때는 항상 0을 출력함

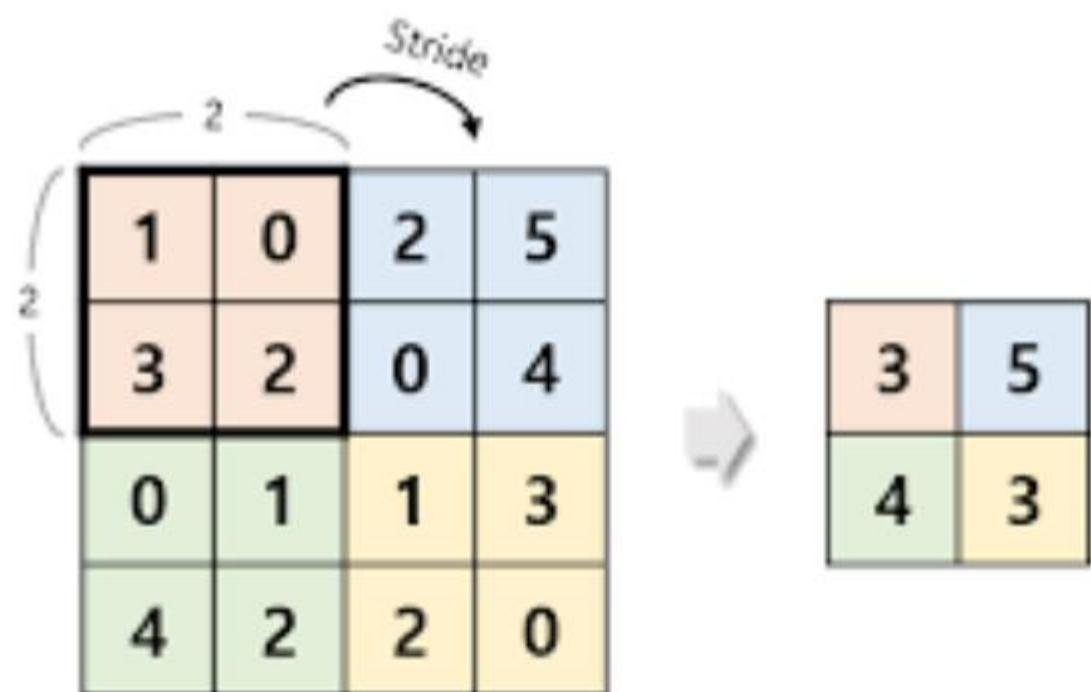
ReLU 함수는 계산 비용이 매우 적게 듦

죽은 뉴런(dead neuron) 현상이 발생할 수 있습니다. 이러한 현상이 발생하면 해당 뉴런은 학습 과정에서 어떠한 역할도 하지 못하게 됨



## Conv2D (maxpool2D)

```
tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, input_shape=(28,28,1), 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)),
```



Max Pooling

MaxPool2D 레이어는 입력으로 받은 64개의 28x28 크기의 특징 맵(feature map)에서 2x2 크기의 윈도우(window)를 이동시켜가며 각 윈도우에서 가장 큰 값을 출력으로 반환

입력 특징 맵보다 크기가 반으로 줄어들게 되는데, 이를 통해 계산량을 줄이고, 과적합(overfitting)을 방지하고, 불필요한 정보를 걸러낼 수 있음



## Conv2D (가중치)

```
tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, input_shape=(28,28,1), 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.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)),
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 64)	640
conv2d_1 (Conv2D)	(None, 28, 28, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0

$$64 * (3 * 3 + 1) = 640 \text{ [+1은 편향 값]}$$

$$64 * (3 * 3 * 64 + 1) = 36928$$

$$128 * (3 * 3 * 64 + 1) = 73856$$

$$256 * (3 * 3 * 128 + 1) = 295168$$

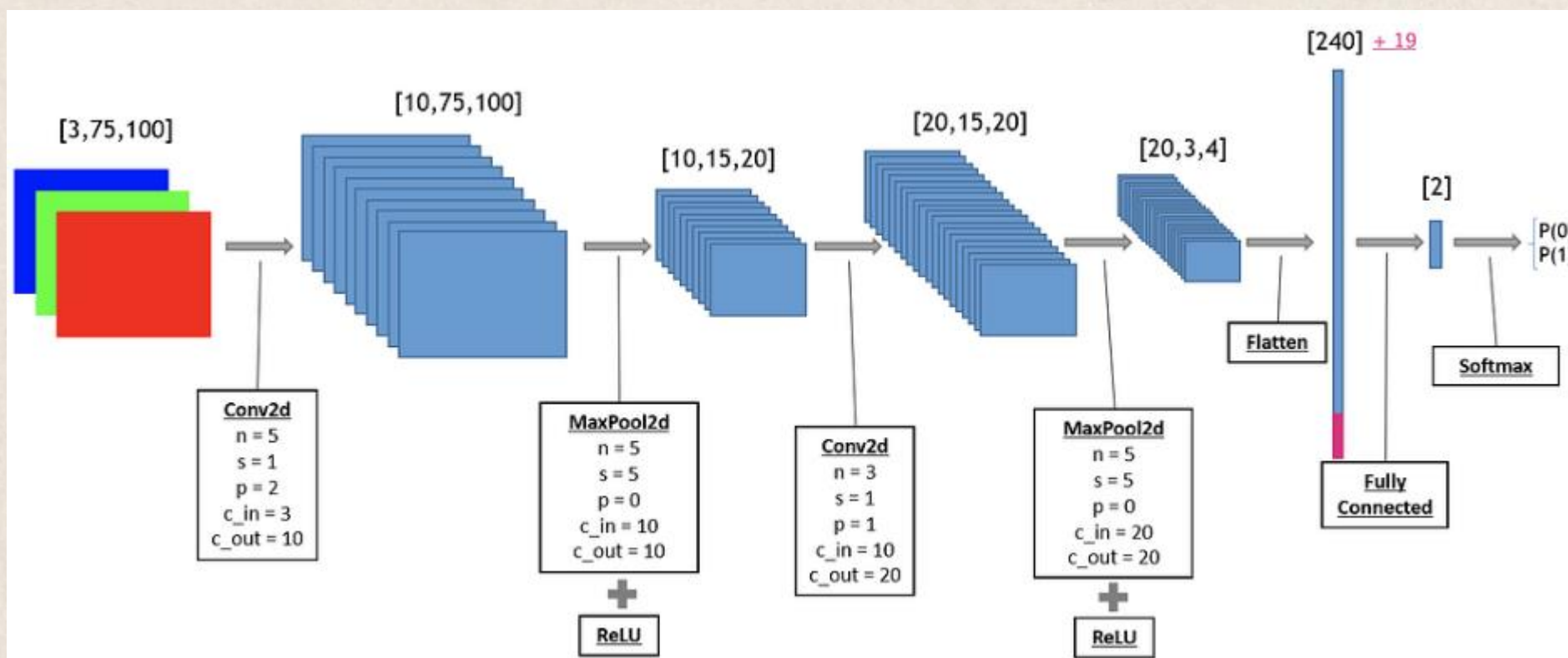
편향은 각 필터마다 하나씩 존재하며, 각 필터가 추출한 특징에 대해 일정한 값(예: 0)을 더해주는 역할을 합니다. 이를 통해, 각 필터가 이미지에서 추출한 특징이 양수와 음수 모두를 포함할 수 있도록 하며, 딥러닝 모델의 성능을 향상시킵니다.



# Flatten

```
tf.keras.layers.Flatten(),  
tf.keras.layers.Dense(units=512, activation='relu'),  
tf.keras.layers.Dropout(0.5),  
tf.keras.layers.Dense(units=256, activation='relu'),  
tf.keras.layers.Dropout(0.5),  
tf.keras.layers.Dense(units=10, activation='softmax')
```

`tf.keras.layers.Flatten()` 은 입력으로 들어온 다차원 배열을 1차원 배열로 평탄화(flatten)하는 역할을 합니다. 따라서 `(None, 7, 7, 256)` 크기의 출력 특징 맵을  
`tf.keras.layers.Flatten()` 레이어의 입력으로 사용하면 `(None, 7\*7\*256)` 크기의 1차원 배열이 출력됩니다.



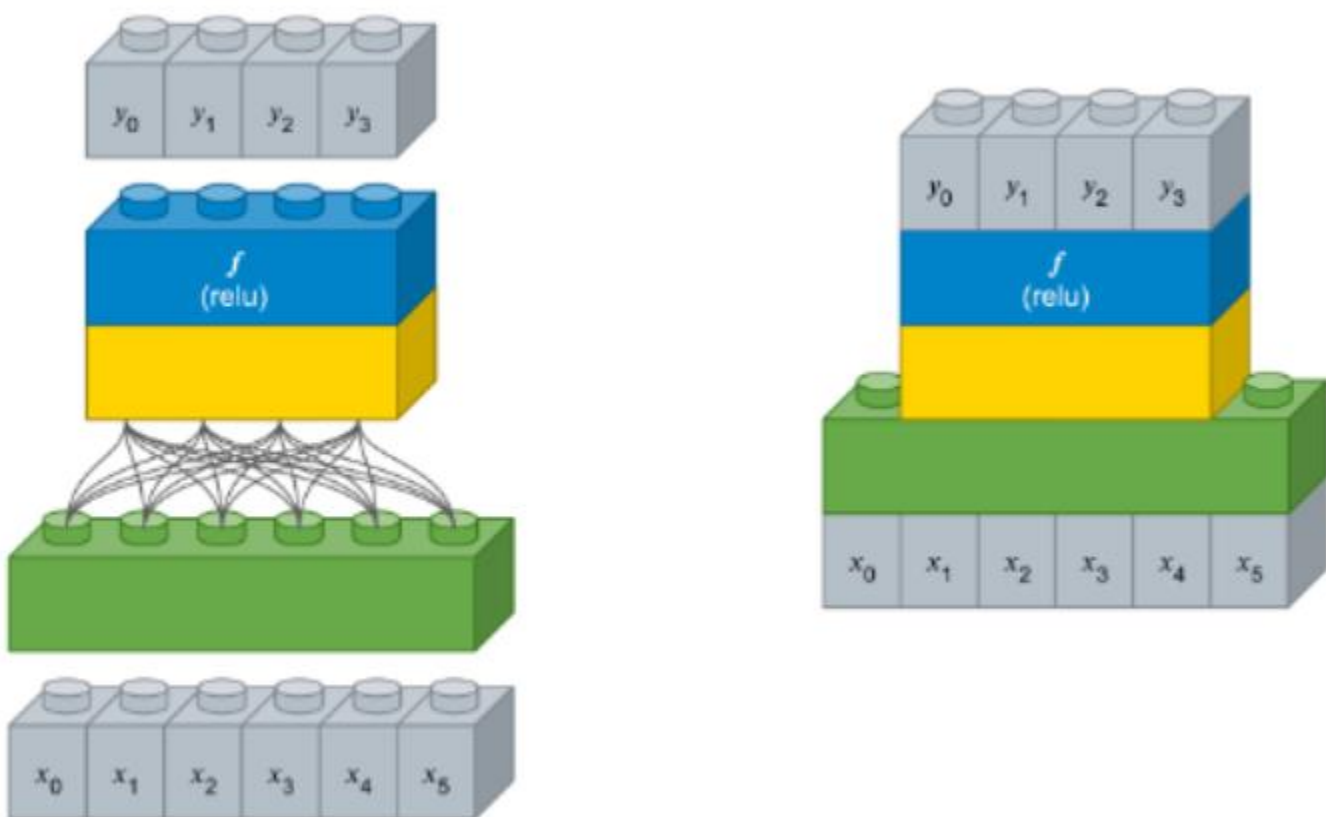


# Dense

```
tf.keras.layers.Flatten(),  
tf.keras.layers.Dense(units=512, activation='relu'),  
tf.keras.layers.Dropout(0.5),  
tf.keras.layers.Dense(units=256, activation='relu'),  
tf.keras.layers.Dropout(0.5),  
tf.keras.layers.Dense(units=10, activation='softmax')
```

`tf.keras.layers.Dense`는 Fully Connected Layer (완전 연결 레이어)의 일종으로, 입력 데이터와 가중치 행렬을 행렬곱(dot product)하고 편향(bias)을 더한 후, 활성화 함수(activation function)를 적용하는 레이어입니다.

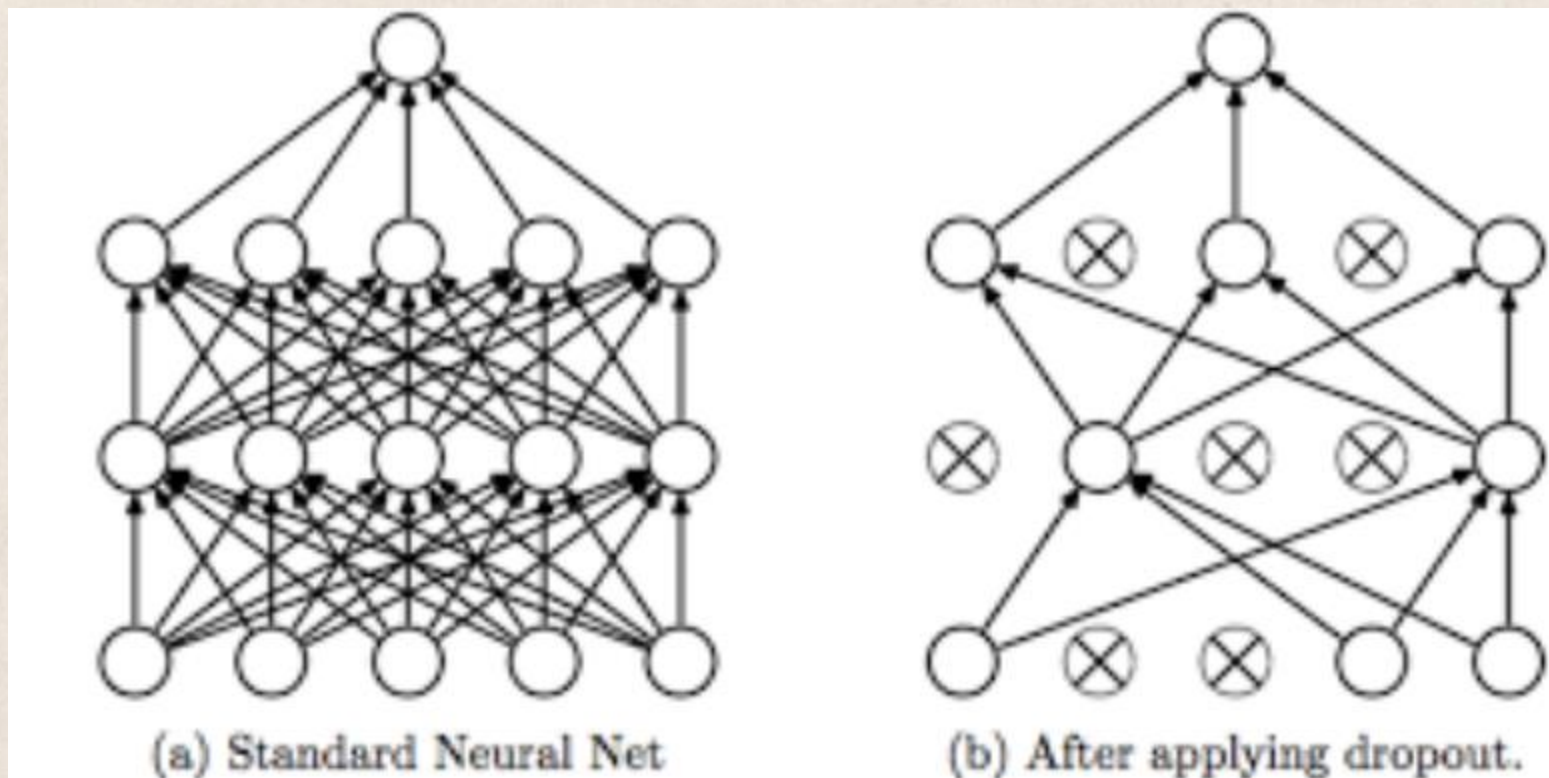
`Dense` 레이어의 인자로 출력 유닛 수 (output units), 활성화 함수(activation function), 입력 데이터의 형태(input shape) 등이 있습니다. 출력 유닛 수는 레이어가 출력하는 벡터의 크기를 결정하며, 활성화 함수는 레이어의 출력값을 변환하는 함수로, 비선형성을 추가하여 모델의 표현력을 증가시킵니다.





# Dropout

```
tf.keras.layers.Flatten(),  
tf.keras.layers.Dense(units=512, activation='relu'),  
tf.keras.layers.Dropout(0.5),  
tf.keras.layers.Dense(units=256, activation='relu'),  
tf.keras.layers.Dropout(0.5),  
tf.keras.layers.Dense(units=10, activation='softmax')
```



Dropout`은 딥러닝에서 과적합(overfitting)을 방지하기 위한 regularization 기법 중 하나입니다. `Dropout`은 레이어에 적용되며, 입력값의 일부를 랜덤하게 0으로 만듭니다. 이를 통해 특정 뉴런이 특정 입력값에 의존하는 것을 방지하고, 레이어의 복잡도를 줄여 일반화 성능을 향상시킵니다.



# categorical\_crossentropy

```
opt = tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.9, nesterov=True)

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

`categorical_crossentropy`는 일반적으로 다중 클래스 분류 문제에서 사용되는 손실 함수로써 이 손실 함수는 모델이 예측한 클래스 확률 분포와 실제 라벨의 확률 분포 사이의 차이를 계산합니다. 따라서 모델이 더 정확한 예측을 할 때 더 낮은 손실 값을 가지게 됩니다.

각 클래스의 확률을 [0.1, 0.3, 0.05, 0.02, 0.01, 0.01, 0.1, 0.1, 0.2, 0.01]로 예측하고, 실제 라벨이 2번 클래스라면 실제 라벨의 확률 분포는 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]이 됩니다. 따라서 2번 클래스의 손실 값은  $\log(0.05)$ 가 되고, 모든 클래스의 손실 값을 더한 후 최종 손실 값을 계산합니다.



# model.fit

```
history = model.fit(train_images, train_labels, epochs=5, validation_data = (test_images, test_labels))
```

model.fit() 함수는 모델을 학습시키는 함수입니다. 주어진 데이터셋을 사용하여 모델의 가중치를 업데이트 하고, 손실 함수 값을 최소화하는 방향으로 학습을 진행합니다.

batch\_size는 한 번에 학습할 데이터의 개수이며,  
epochs는 전체 데이터셋을 몇 번 반복해서 학습할지를 설정  
validation\_data`는 모델의 성능을 검증하기 위해 사용하는 데이터셋



# epoch

```
history = model.fit(train_images, train_labels, epochs=5, validation_data = (test_images, test_labels))  
  
loss, accuracy = model.evaluate(test_images, test_labels)  
print(loss, accuracy)
```

```
-----  
Epoch 1/10  
600/600 [=====] - 941s 2s/step - loss: 0.1897 - accuracy: 0.9416 - val_loss: 0.0452 - val_accuracy: 0.9852  
Epoch 2/10  
600/600 [=====] - 935s 2s/step - loss: 0.0554 - accuracy: 0.9851 - val_loss: 0.0275 - val_accuracy: 0.9910  
Epoch 3/10  
600/600 [=====] - 923s 2s/step - loss: 0.0422 - accuracy: 0.9882 - val_loss: 0.0262 - val_accuracy: 0.9924  
Epoch 4/10  
600/600 [=====] - 929s 2s/step - loss: 0.0319 - accuracy: 0.9910 - val_loss: 0.0200 - val_accuracy: 0.9950  
Epoch 5/10  
600/600 [=====] - 929s 2s/step - loss: 0.0257 - accuracy: 0.9928 - val_loss: 0.0215 - val_accuracy: 0.9934  
Epoch 6/10  
600/600 [=====] - 932s 2s/step - loss: 0.0246 - accuracy: 0.9929 - val_loss: 0.0208 - val_accuracy: 0.9940  
Epoch 7/10  
600/600 [=====] - 935s 2s/step - loss: 0.0194 - accuracy: 0.9944 - val_loss: 0.0234 - val_accuracy: 0.9941  
Epoch 8/10  
600/600 [=====] - 930s 2s/step - loss: 0.0172 - accuracy: 0.9953 - val_loss: 0.0207 - val_accuracy: 0.9945  
Epoch 9/10  
600/600 [=====] - 925s 2s/step - loss: 0.0153 - accuracy: 0.9952 - val_loss: 0.0221 - val_accuracy: 0.9938  
Epoch 10/10  
600/600 [=====] - 931s 2s/step - loss: 0.0154 - accuracy: 0.9956 - val_loss: 0.0237 - val_accuracy: 0.9942  
313/313 [=====] - 41s 128ms/step - loss: 0.0237 - accuracy: 0.9942  
최종 예측 성공률(%): 99.41999912261963
```

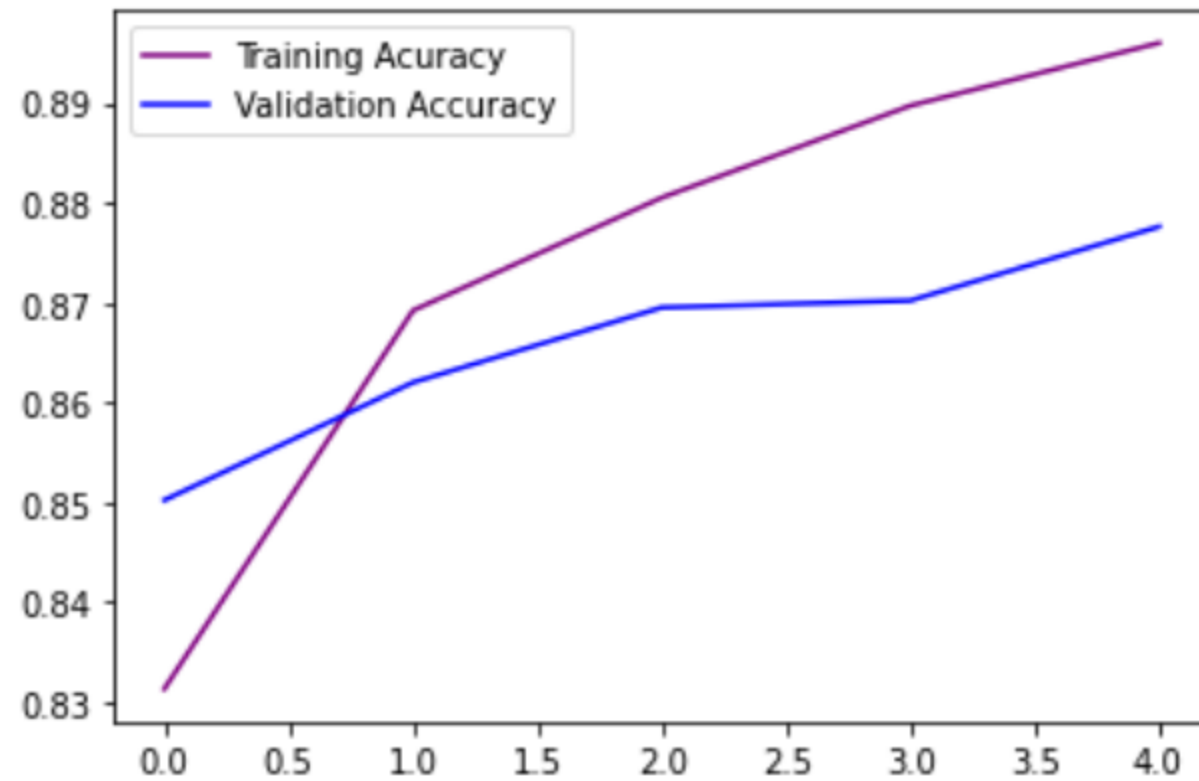
batch\_size = 100, iteration = 600, epoch = 10



# 정확도

```
acc = history.history['accuracy']  
val_acc = history.history['val_accuracy']  
  
plt.figure()  
plt.plot(acc,color = 'purple',label = 'Training Acuracy')  
plt.plot(val_acc,color = 'blue',label = 'Validation Accuracy')  
plt.legend()
```

<matplotlib.legend.Legend at 0x7f5d30227490>

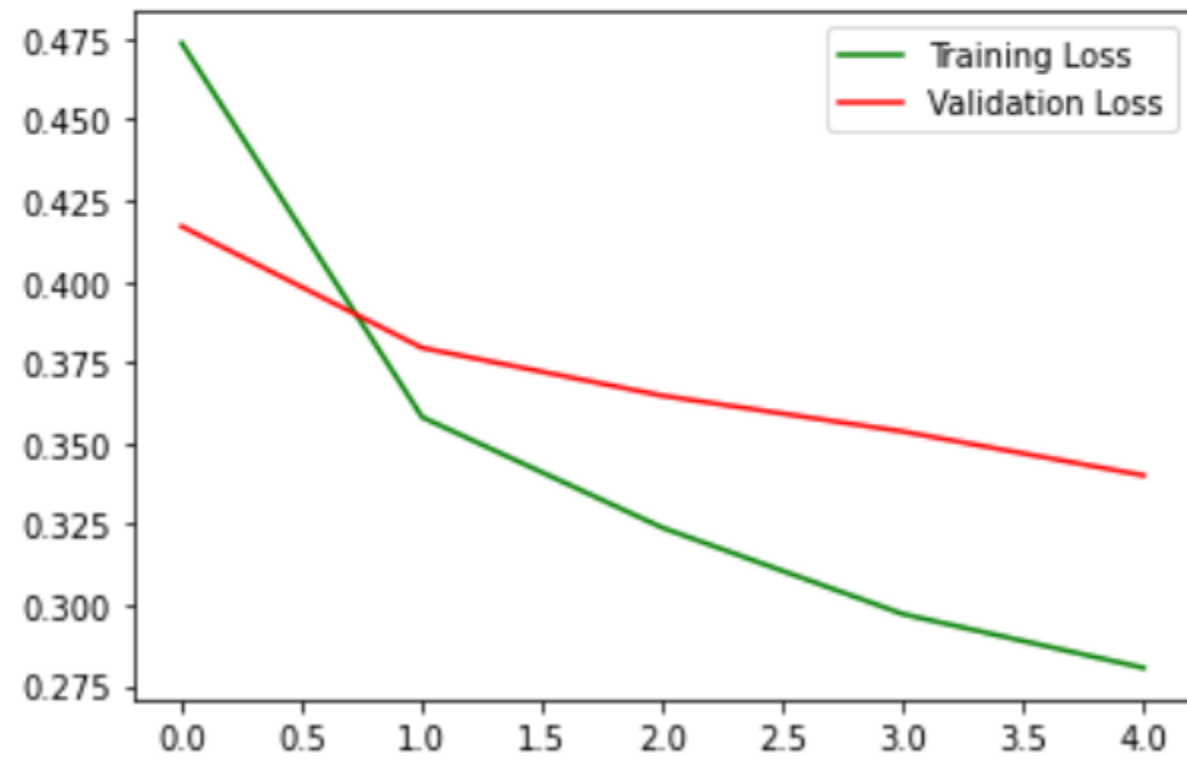




# 손실률

```
loss = history.history['loss']  
val_loss = history.history['val_loss']  
  
plt.figure()  
plt.plot(loss,color = 'green',label = 'Training Loss')  
plt.plot(val_loss,color = 'red',label = 'Validation Loss')  
plt.legend()
```

<matplotlib.legend.Legend at 0x7f5dcc635ac0>





## 테스트 진행(경로 설정)

```
root = '/content/drive/MyDrive/Colab Notebooks/GITHUB/colab_ML/ML_CNN/dataset/fashion/shoes2.jpeg'
img = cv2.imread(root, cv2.IMREAD_GRAYSCALE)
img = cv2.resize(255-img, (28, 28), interpolation=cv2.INTER_NEAREST)
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)/255
imgRGB = imgRGB.reshape(-1, 28, 28)

print(np.argmax(model.predict(imgRGB)[0]))
```

40개의 이미지 생성후 경로 저장



## 테스트 진행

```
0 0
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 19ms/step
9 9
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 19ms/step
8 9
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 22ms/step
9 9
1/1 [=====] - 0s 19ms/step
accuracy = 0.6
false_number = [5, 9, 11, 12, 13, 14, 15, 17, 18, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 33, 36, 38]
```

60% 정도의 정확성을 가짐