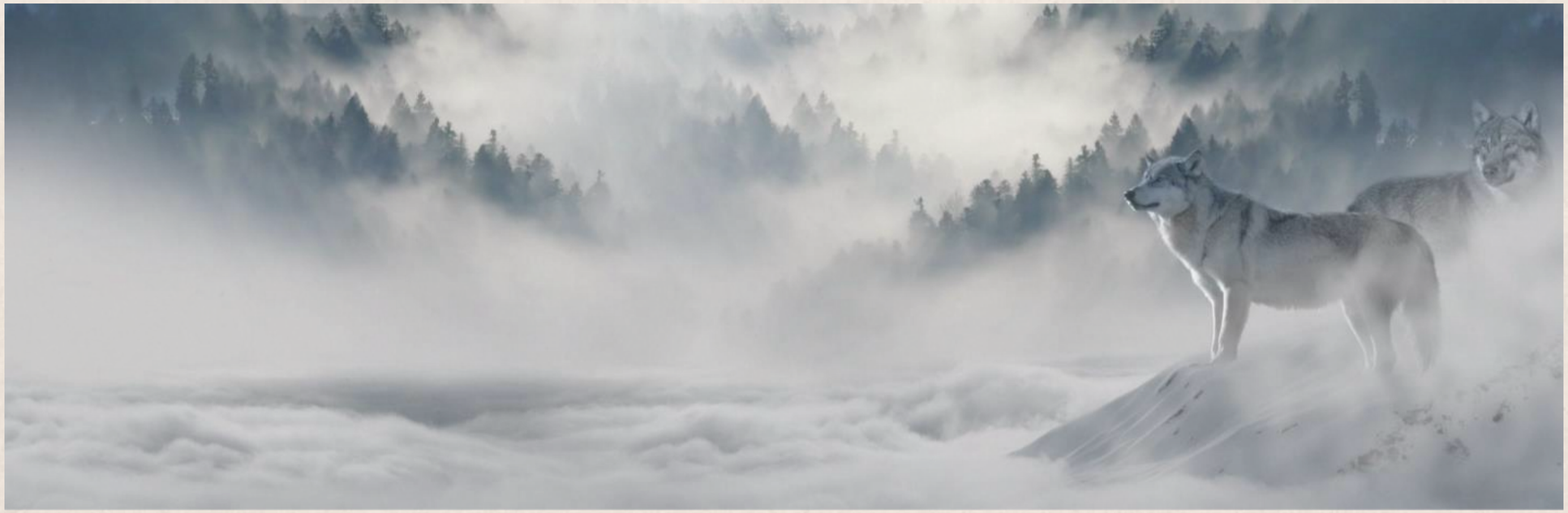
CNN_MNIST



경북대학교

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)

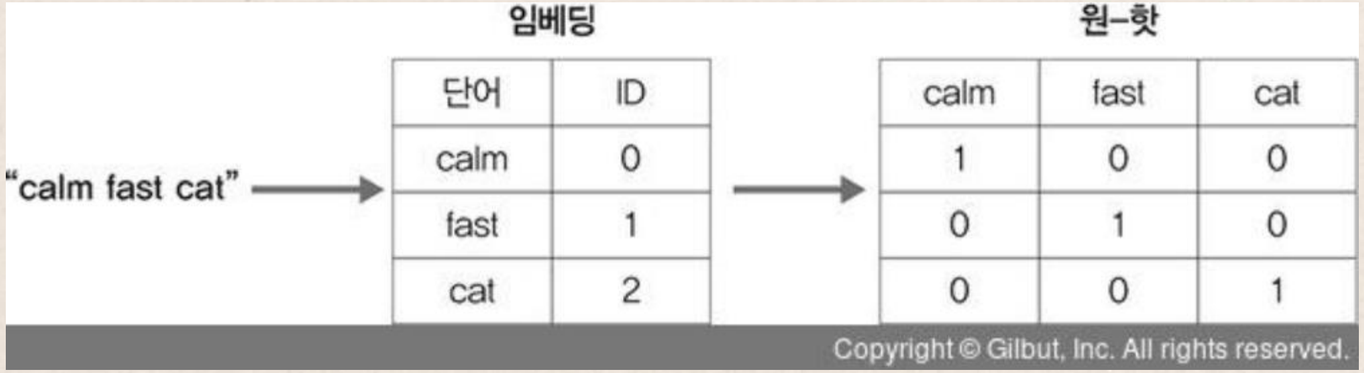
정규화

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# 각종 파라메터의 영향을 보기 위해 랜덤값 고정
tf.random.set_seed(1234)

# Normalizing data
x_train, x_test = x_train / 255.0, x_test / 255.0

# (60000, 28, 28) => (60000, 28, 28, 1)로 reshape
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

1. mnist 데이터 셋을 가지고 옵니다.
2. 학습결과의 비교를 위해 난수의 값 고정
3. 훈련 데이터를 0~1 사이의 값으로 만듬
4. ConV2D사용을 위해 적절한 input shape으로 reshape

one-hot 인코딩

```
# One-hot 인코딩
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```



to_categorical 함수는 정수 형태의 레이블 데이터를 원핫 인코딩 형태로 변환 해 줍니다.

형식 : to_categorical(y, num_classes=None, dtype='float32')

즉, `y` 값이 `i`일 때, `i`번째 비트만 1이고 나머지는 0인 형태로 변환 (범주형 데이터를 수치형 데이터로)

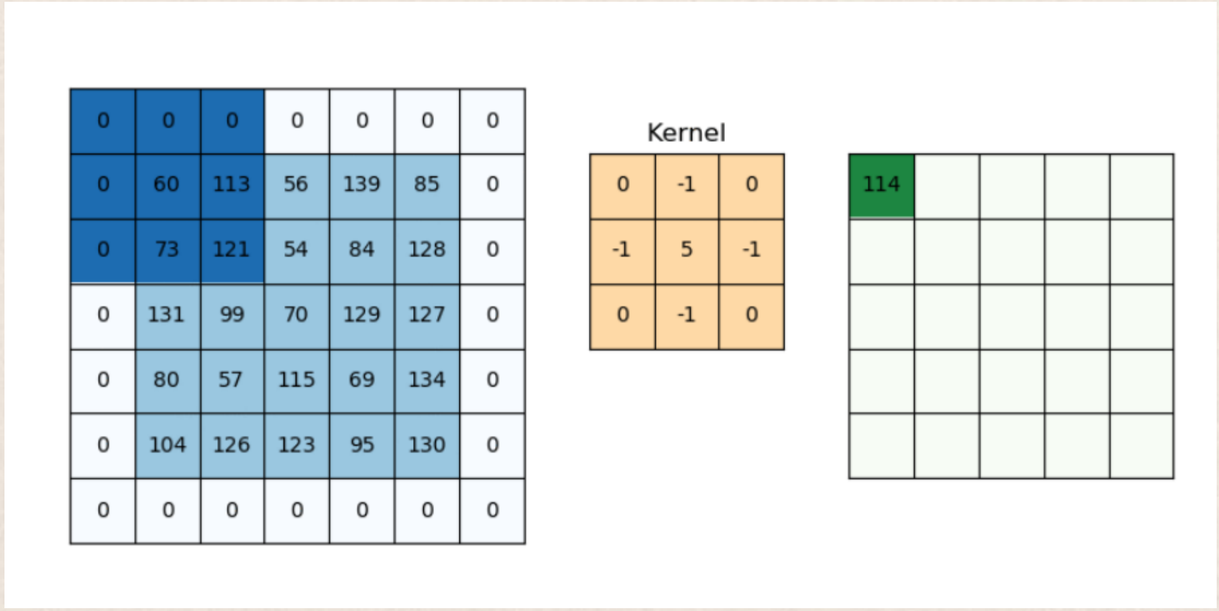
모델 예시

```
model = tf.keras.Sequential([
    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)),

    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')
])
```

```
Model: "sequential"
_________________________________________________________________
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 (MaxPooling  (None, 6, 6, 256)         0
2D)
flatten (Flatten)            (None, 9216)              0
dense_3 (Dense)              (None, 512)               4719104
dropout (Dropout)            (None, 512)               0
dense_4 (Dense)              (None, 256)               131328
dropout_1 (Dropout)          (None, 256)               0
dense_5 (Dense)              (None, 10)                2570
=================================================================
Total params: 5,259,594
Trainable params: 5,259,594
Non-trainable params: 0
```

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)),
```

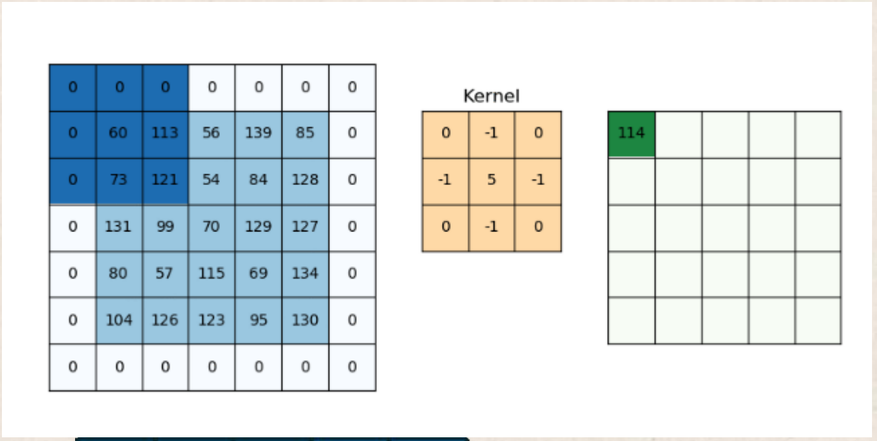


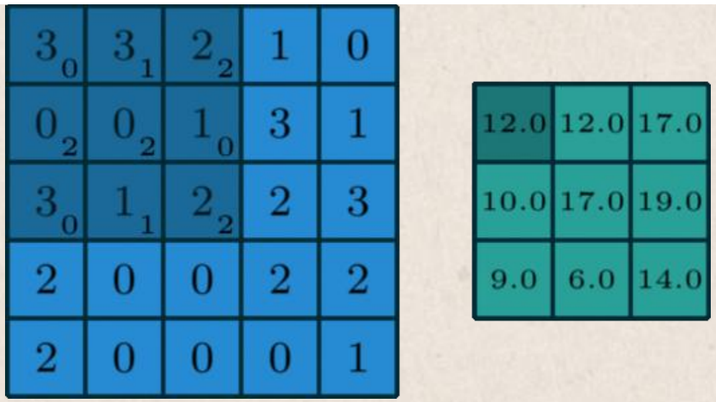
1. 입력 데이터는 3*3 크기를 가지는 필터(filter)라는 작은 윈도우와 곱해집니다

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)),
```





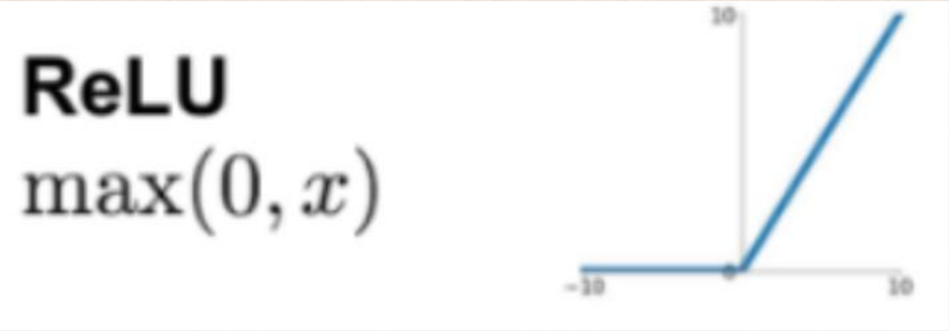
padding이 same인 경우, 출력 특징맵의 크기가 입력 특징맵의 크기와 상관없이 동일하게 유지됩니다.

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)),
```



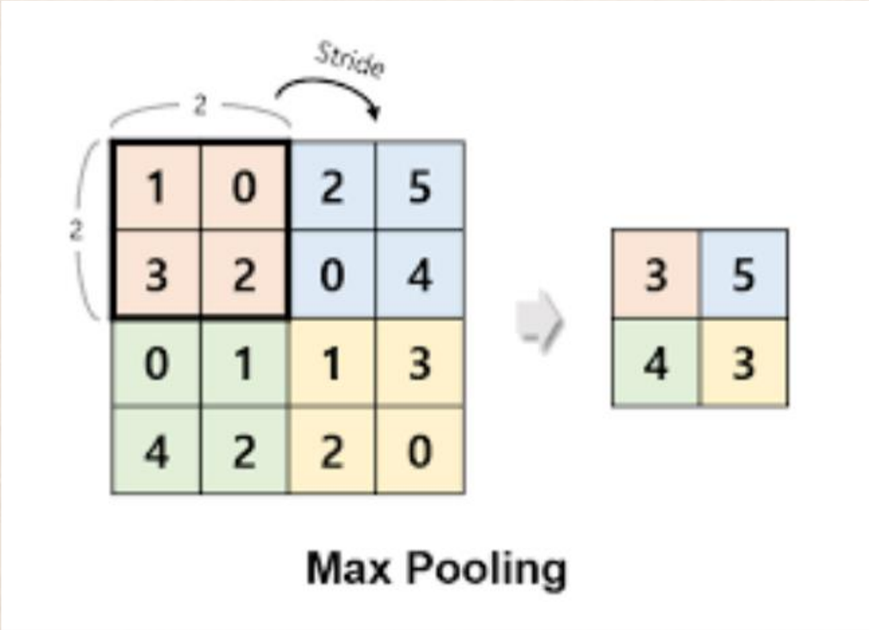
ReLU 함수는 입력값이 0보다 클 때는 선형 함수처럼 작동하지만, 입력값이 0 이하일 때는 항상 0을 출력함

ReLU 함수는 계산 비용이 매우 적게 듬

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)),
```



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)),
```

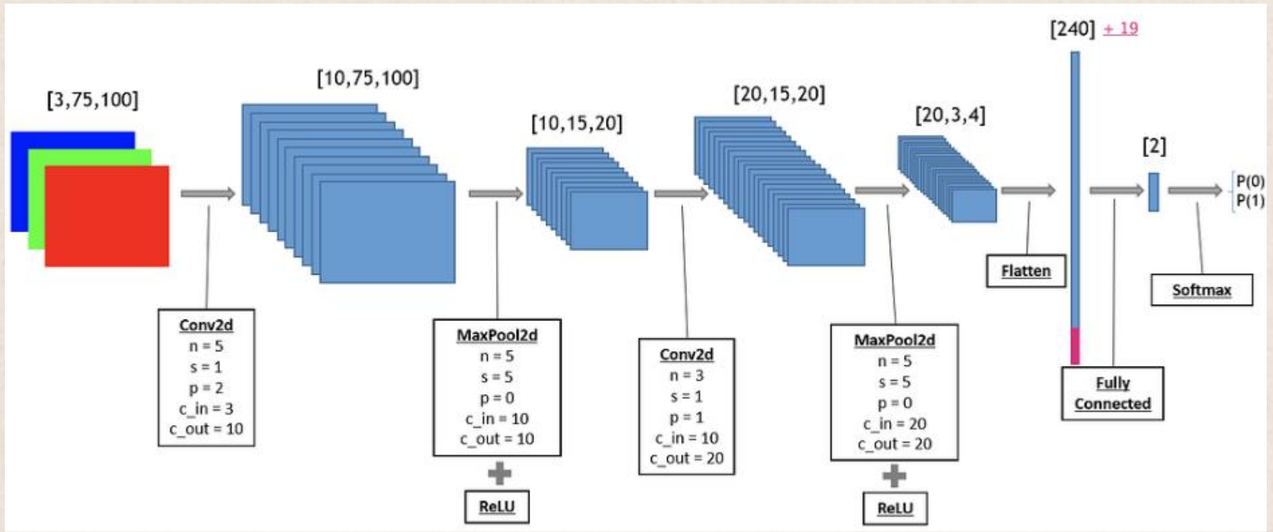
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 64)	640
conv2d_1 (Conv2D)	(None, 28, 28, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_3 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)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

64*(3*3+1) = 640 [+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

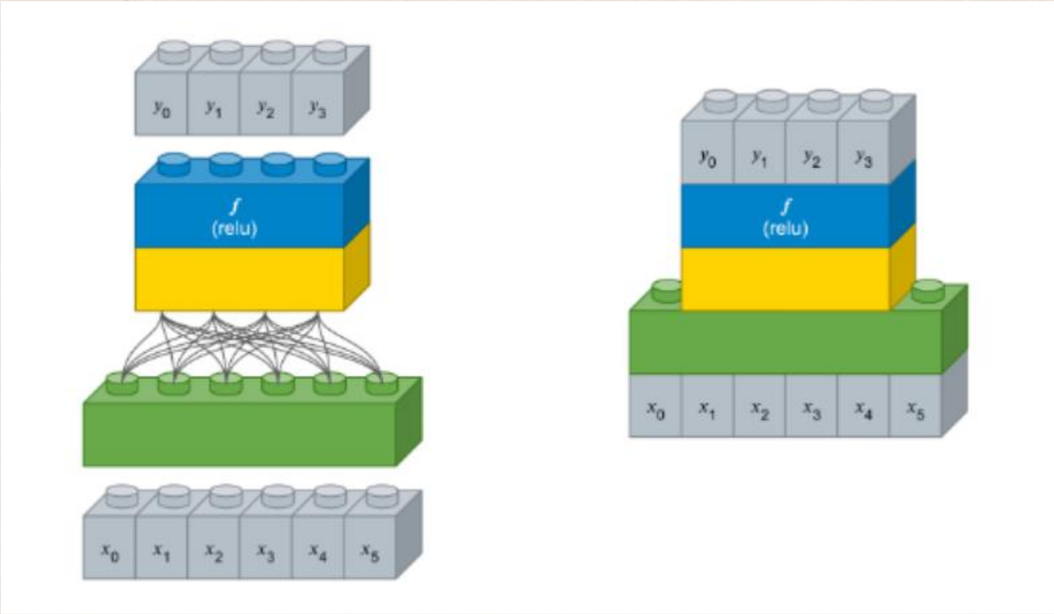
```
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(units=512, activation='relu'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(units=256, activation='relu'),
tf.keras.layers.Dropout(0.5),
tf.keras.layers.Dense(units=10, activation='softmax')
```



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')
```

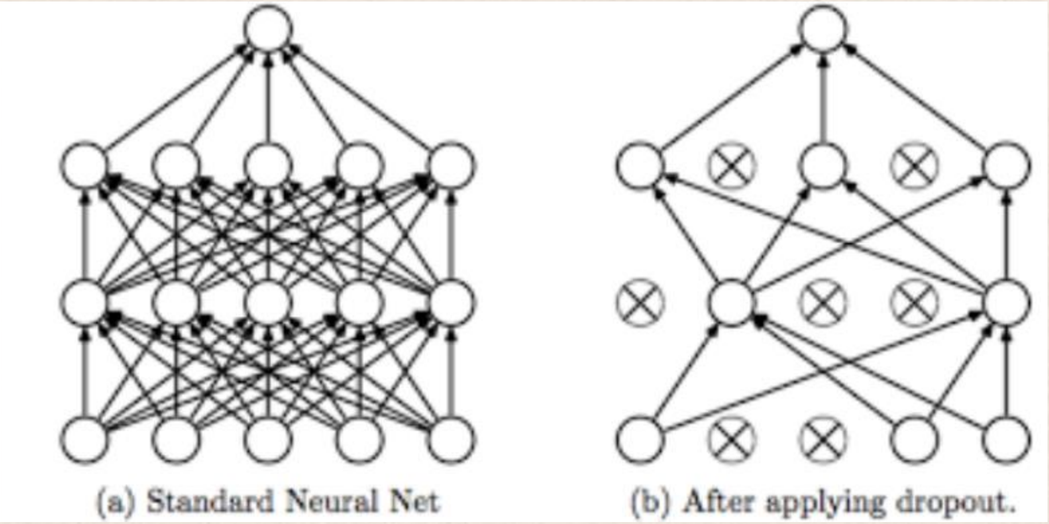


tf.keras.layers.Dense`는 Fully Connected Layer (완전 연결 레이어)의 일종으로, 입력 데이터와 가중치 행렬을 행렬곱(dot product)하고 편향(bias)을 더한 후, 활성화 함수(activation function)를 적용하는 레이어입니다.

Dense` 레이어의 인자로는 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')
```



(a) Standard Neural Net (b) After applying dropout.

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.

softmax는 출력층의 뉴런들이 각 클래스에 속할 확률을 나타내도록 만들어주는 함수입니다.
softmax함수를 사용하면 이들 확률 값이 항상 0과 1 사이이며, 전체 확률의 합이 1이 됩니다

model.compile

```
model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

model.compile()은 모델을 학습하기 위한 설정을 정의하는 메서드입니다.

loss : 손실 함수(loss function)는 모델이 예측한 값과 실제 값 사이의 차이를 계산하는 함수입니다.

optimizer : 옵티마이저(optimizer)는 가중치를 업데이트하는 알고리즘입니다.(`lr`은 학습률(learning rate)로, 가중치 업데이트 시 얼마나 크게 업데이트할 지를 결정)

metrics : 평가 지표(metrics)는 모델의 성능을 평가하는 지표로, 정확도(accuracy), 정밀도(precision), 재현율(recall) 등이 있습니

categorical_crosentropy

```
model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

categorical_crossentropy는 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.

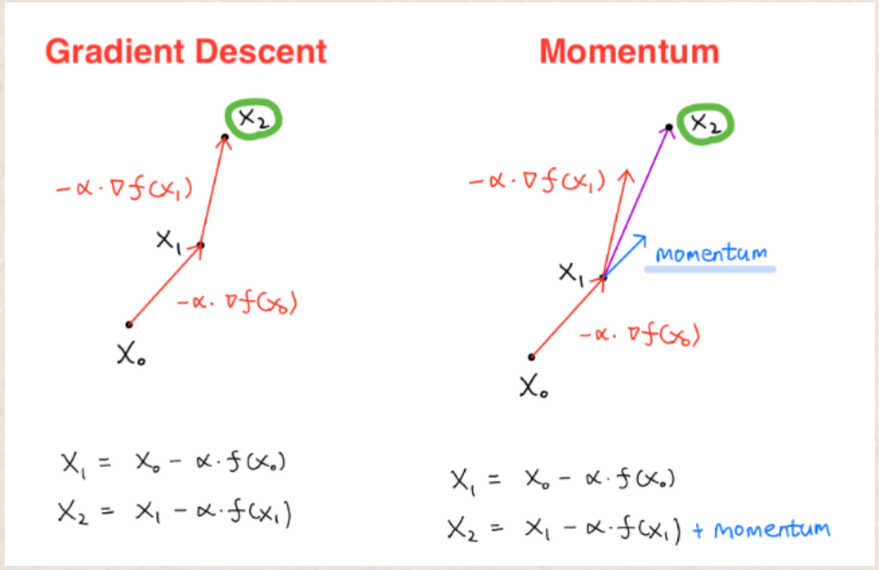
각 클래스의 확률을 [0.1, 0.3, 0.05, 0.02, 0.01, 0.01, 0.1, 0.1, 0.2, 0.01]로 예측하고, 실제 라벨이 2번 클래스라면 실제 라벨의 확률 분포는 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]이 됩니다. 따라서 2번 클래스의 손실 값은 log(0.05)가 되고, 모든 클래스의 손실 값을 더한 후 최종 손실 값을 계산합니다.

Adam

```
model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

RMSProp

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{v_t}} g_{t-1}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(g_{t-1})^2$$
$$v_1 = (g_0)^2$$



$$m_{t+1} \leftarrow \beta_1 m_t + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta)$$ Momentum

$$v_{t+1} \leftarrow \beta_2 v_t + (1-\beta_2)\nabla_\theta \mathcal{L}(\theta)^2$$ RMS Prop

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$ RMS Prop + Momentum

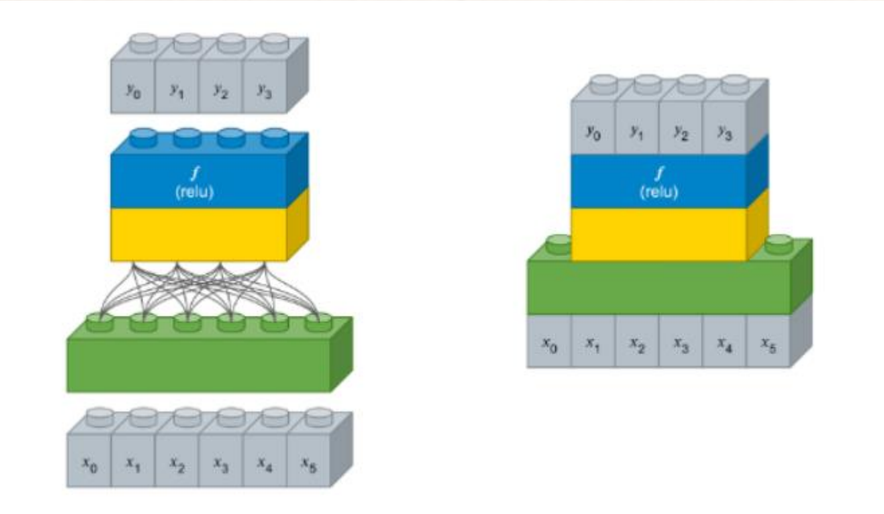
RMSProp + Momentum개념이 합쳐진 옵티마이저, 여러 파라미터들을 만질 수 있다는 장점

현시점까지의 계산되었던 모든 Gradient의 제곱합에 반비례하는 학습률, Global Optimum근처에서의 이동량 감소는 좋은 경우이지만, Global Optimum근처까지 도달하지 못했는데도 이동량이 줄어드는 것은 큰 단점입니다.
다음의 RMSprop는 이런점을 보완, 과거 Gradient의 값들과 현재 gradient값들의 비율을 하이퍼 파라미터로 조절

모멘텀은 step에서의 gradientη과 이제까지의(과거) 이동한 gradient의 exponential average를 더해줍니다.

summary(가중치)

```
model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```




```
_________________________________________________________________
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 (MaxPooling  (None, 6, 6, 256)         0
2D)
flatten (Flatten)            (None, 9216)              0
dense_3 (Dense)              (None, 512)               4719104
dropout (Dropout)            (None, 512)               0
dense_4 (Dense)              (None, 256)               131328
dropout_1 (Dropout)          (None, 256)               0
dense_5 (Dense)              (None, 10)                2570
=================================================================
Total params: 5,259,594
Trainable params: 5,259,594
Non-trainable params: 0
```

가중치
dense_3 : 512(9216+1)
dense_4 : 256(512+1)
dense_5 : 10(256+1)

model.fit

```
model.fit(x_train, y_train, batch_size=100, epochs=10, validation_data=(x_test, y_test))

result = model.evaluate(x_test, y_test)
print("최종 예측 성공률(%): ", result[1]*100)
```

model.fit() 함수는 모델을 학습시키는 함수입니다. 주어진 데이터셋을 사용하여 모델의 가중치를 업데이트하고, 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(x_train, y_train, batch_size=100, epochs=10, validation_data=(x_test, y_test))

result = model.evaluate(x_test, y_test)
print("최종 예측 성공률(%): ", result[1]*100)
```

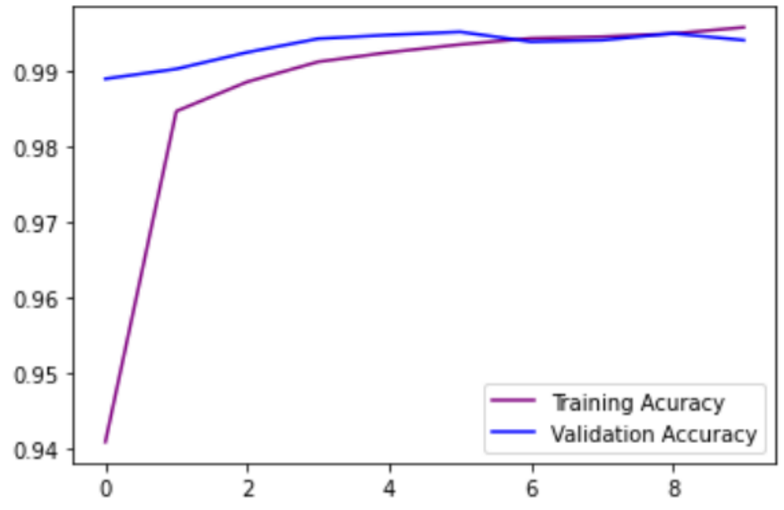
```
----------------------------------------------------------------
Epoch 1/10
600/600 [==============================] - 941s 2s/step - loss: 0.1897 - accuracy: 0.9416 - val_loss: 0.0452 - val_accuracy: 0.9852
Epoch 2/10
600/600 [==============================] - 935s 2s/step - loss: 0.0554 - accuracy: 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

accuracy graph

```
from PIL import Image
import matplotlib.pyplot as plt
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()
```

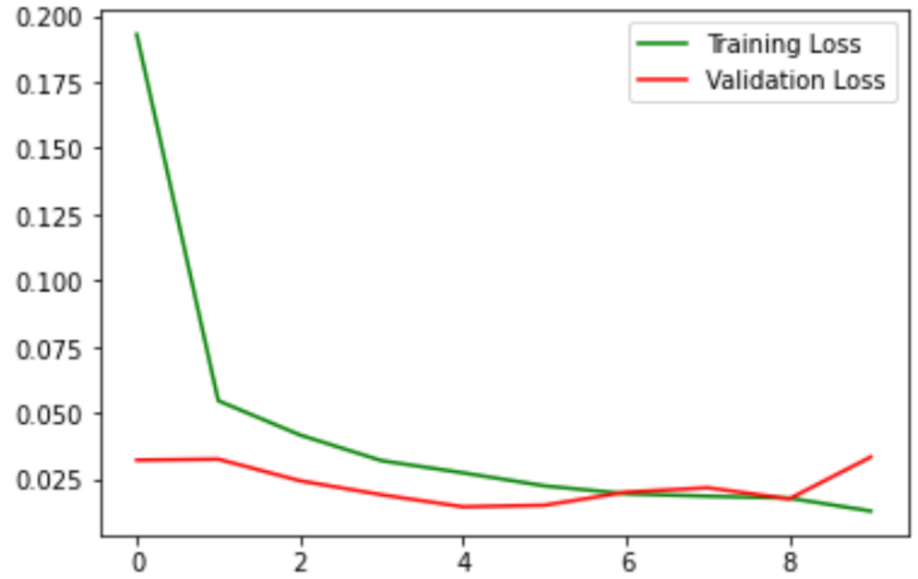
```
<matplotlib.legend.Legend at 0x7f903192e8e0>
```



loss graph

```
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 0x7f903005f9d0>
```



테스트 진행(경로 설정)

```
img_path_list = []
for i in range(1,10):
  root_dir = '/usr/colab/testex/'
  root_dir += str(i)
  root_dir += '.png'
  img_path_list.append(root_dir)

for i in range(10,31):
  root_dir = '/usr/colab/testex/'
  root_dir += str(i)
  root_dir += '.PNG'
  img_path_list.append(root_dir)
```

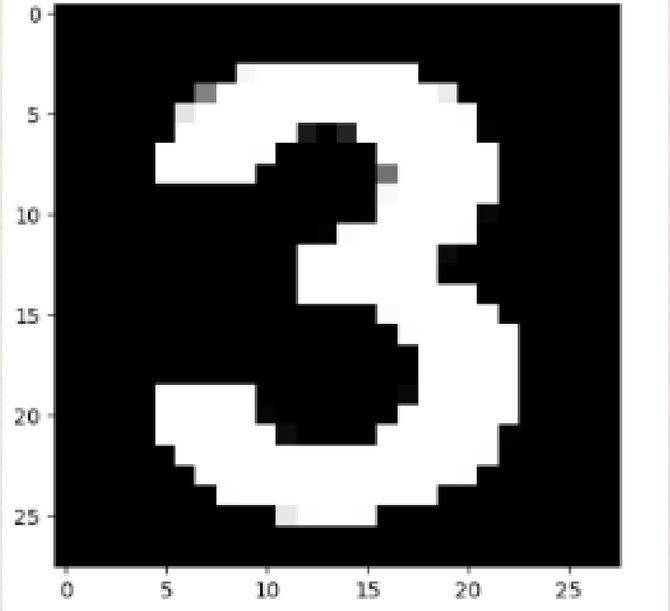
30개의 이미지 생성후 경로 저장

테스트 진행(경로 설정)

```
count = 0
false_number=[]
for i in range(len(img_path_list)):
  img = cv2.imread(img_path_list[i], cv2.IMREAD_GRAYSCALE)
  img = cv2.resize(255-img,(28,28))
  test_num = img.reshape((-1,28,28,1))

  print(np.argmax((model.predict(test_num)>0.5).astype("int32")), a[i])
  if (np.argmax((model.predict(test_num)>0.5).astype("int32"))==a[i] ):
    count+=1
  else:
    false_number.append(i)
print(f'accuracy = {count/30}')
print(f'false_number = {false_number}')
```

```
plt.imshow(img,'gray')
```

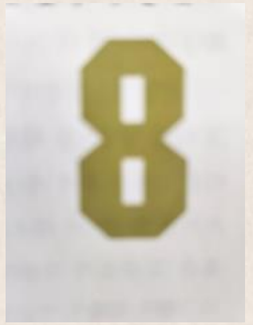
```
<matplotlib.image.AxesImage at 0x7ffb0e0fc880>
```



이미지 불러온후 입력 데이터와 형식을 똑같게 변환, 그 후 정확도 계산

테스트 진행(결과)

```
1/1 [==============================] - 0s 29ms/step
5 5
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
3 3
1/1 [==============================] - 0s 31ms/step
accuracy = 0.8666666666666667
false_number = [1, 3, 10, 23]
```



테스트셋에서는 정확도가 86%로 나옴