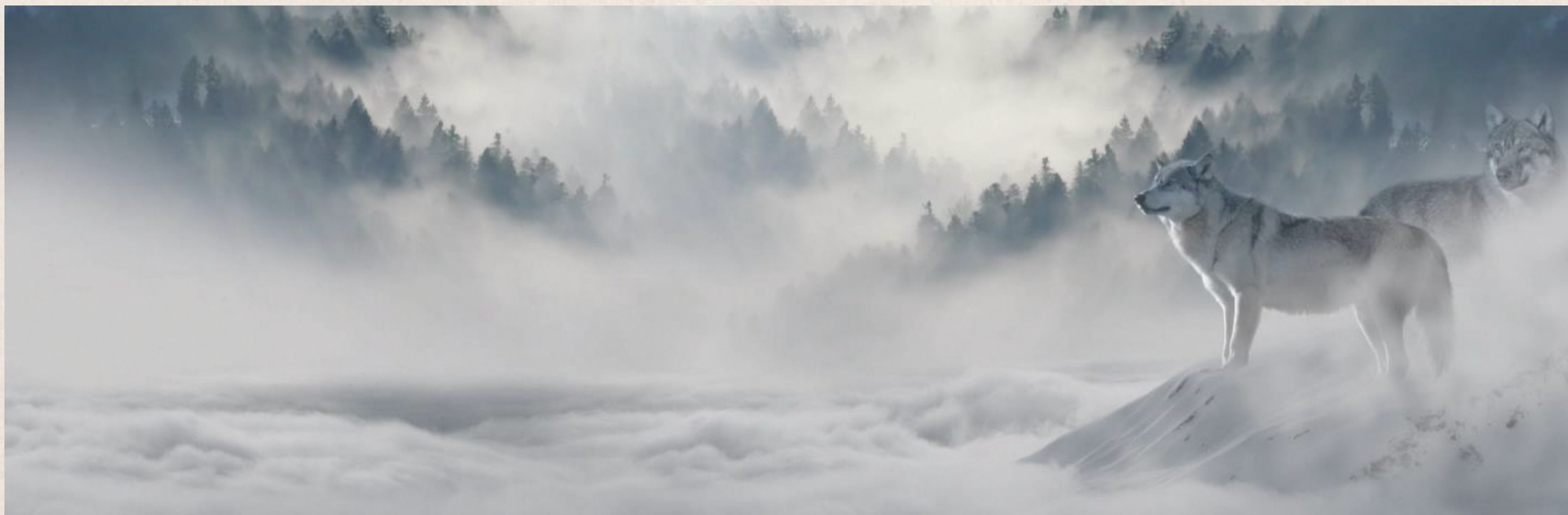


CNN_CIFAR100



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)

CIFAR 100 클래스 설정

```
# CIFAR-100 클래스
CIFAR100_CLASSES = sorted(['beaver', 'dolphin', 'otter', 'seal', 'whale', # aquatic mammals
                           'aquarium', 'fish', 'flatfish', 'ray', 'shark', 'trout', # fish
                           'orchids', 'poppies', 'roses', 'sunflowers', 'tulips', # flowers
                           'bottles', 'bowls', 'cans', 'cups', 'plates', # food containers
                           'apples', 'mushrooms', 'oranges', 'pears', 'sweet peppers', # fruit and vegetables
                           'clock', 'computer', 'keyboard', 'lamp', 'telephone', 'television', # household electrical devices
                           'bed', 'chair', 'couch', 'table', 'wardrobe', # household furniture
                           'bee', 'beetle', 'butterfly', 'caterpillar', 'cockroach', # insects
                           'bear', 'leopard', 'lion', 'tiger', 'wolf', # large carnivores
                           'bridge', 'castle', 'house', 'road', 'skyscraper', # large man-made outdoor things
                           'cloud', 'forest', 'mountain', 'plain', 'sea', # large natural outdoor scenes
                           'camel', 'cattle', 'chimpanzee', 'elephant', 'kangaroo', # large omnivores and herbivores
                           'fox', 'porcupine', 'possum', 'raccoon', 'skunk', # medium-sized mammals
                           'crab', 'lobster', 'snail', 'spider', 'worm', # non-insect invertebrates
                           'baby', 'boy', 'girl', 'man', 'woman', # people
                           'crocodile', 'dinosaur', 'lizard', 'snake', 'turtle', # reptiles
                           'hamster', 'mouse', 'rabbit', 'shrew', 'squirrel', # small mammals
                           'maple', 'oak', 'palm', 'pine', 'willow', # trees
                           'bicycle', 'bus', 'motorcycle', 'pickup truck', 'train', # vehicles 1
                           'lawn-mower', 'rocket', 'streetcar', 'tank', 'tractor' # vehicles 2
                           ])
```


데이터 + 정규화

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar100.load_data()
```

```
# Normalizing data
```

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

cifar 100 데이터 불러 오기

모델 예시

```
with tf.device('/device:GPU:0'):
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(kernel_size=(3,3), filters=32, input_shape=(32,32,3), 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.Conv2D(kernel_size=(3,3), filters=64, kernel_initializer='he_uniform', activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(kernel_size=(3,3), filters=64, 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.Flatten(),
        tf.keras.layers.Dense(units=512, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(units=100, 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 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	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

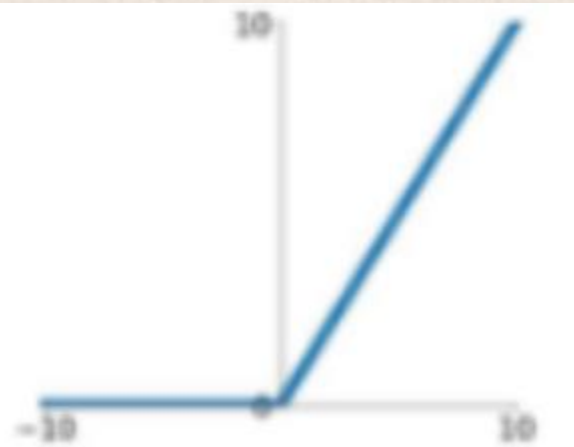
padding이 valid인 경우 출력 특징맵의 크기는 입력 특징맵의 크기와 필터의 크기에 따라 결정됩니다.

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
 $\max(0, x)$



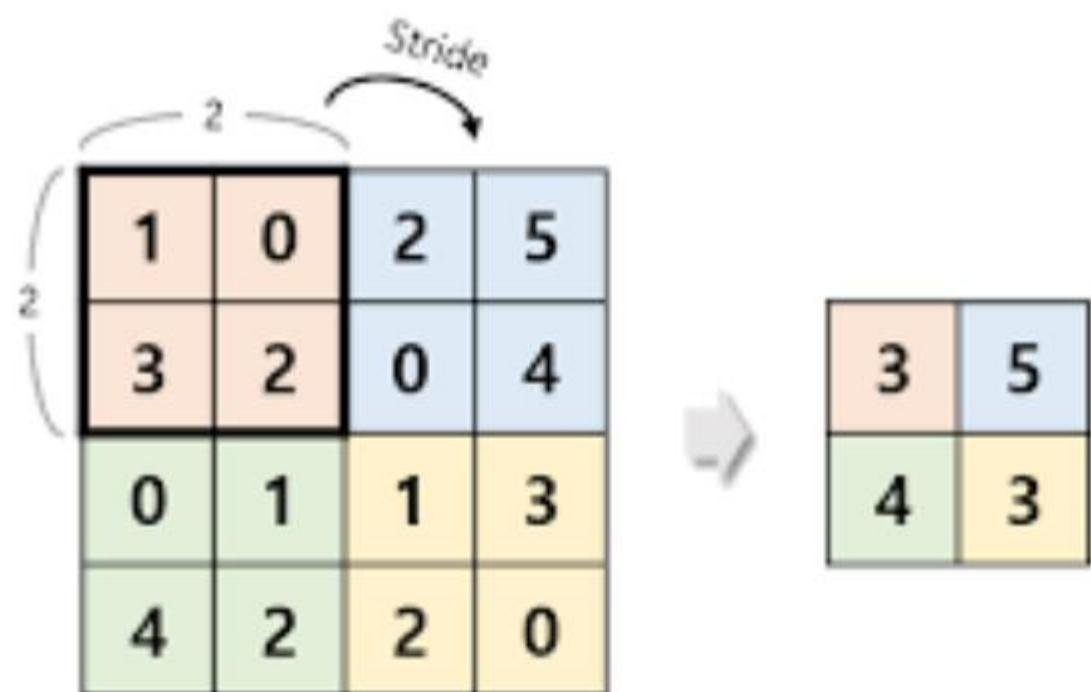
ReLU 함수는 입력값이 0보다 클 때는 선형 함수처럼 작동하지만, 입력값이 0 이하일 때는 항상 0을 출력함

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

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

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

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

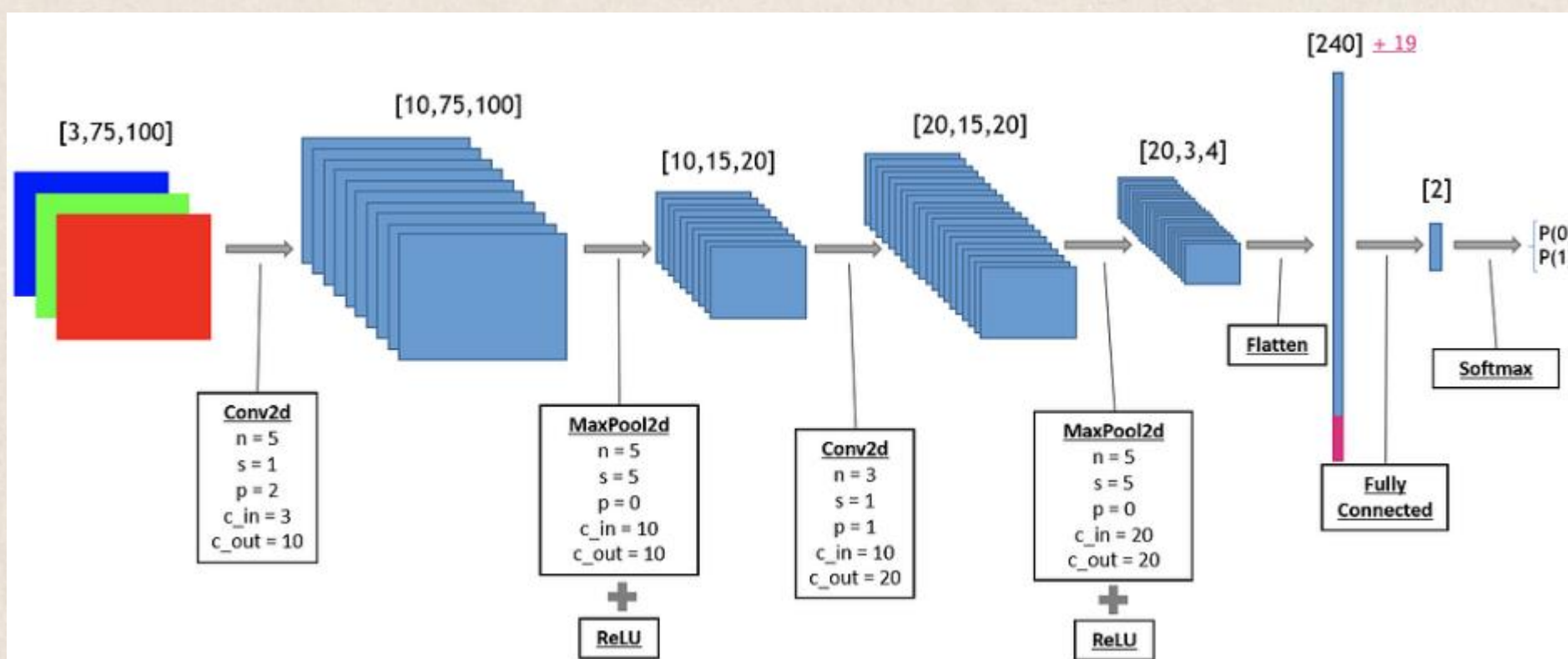
$$256 * (3 * 3 * 128 + 1) = 295168$$

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.

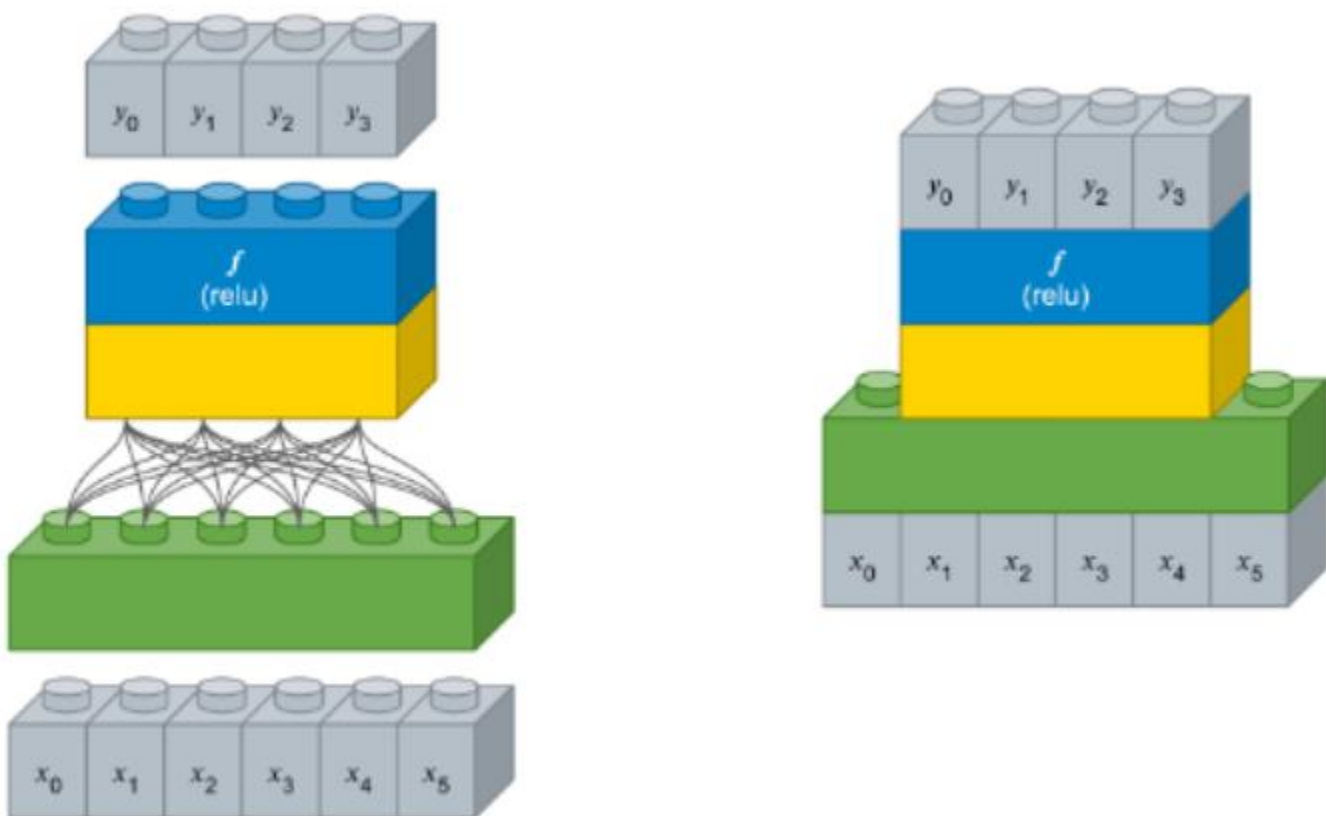


Dense

```
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')
```

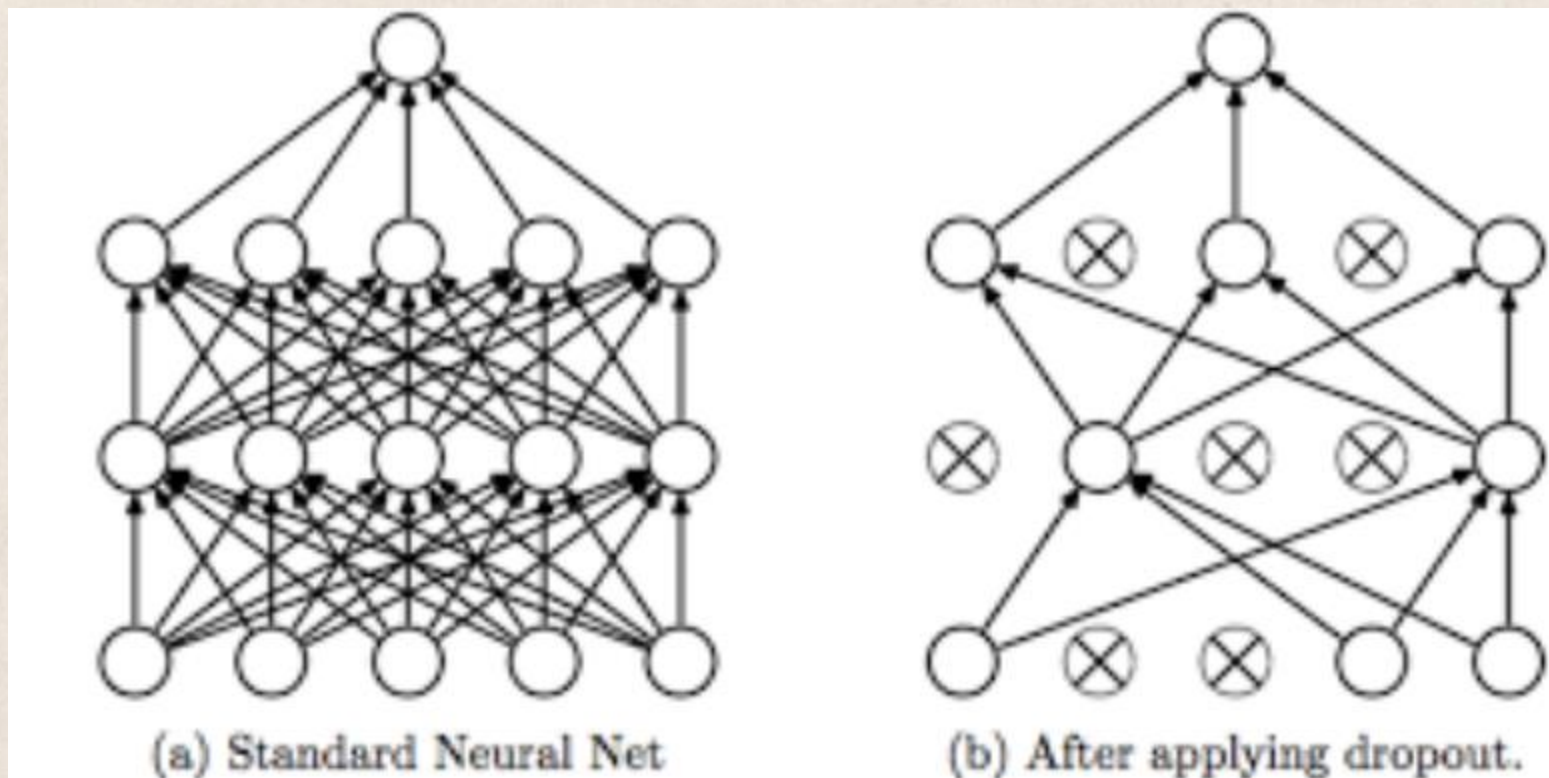
`tf.keras.layers.Dense``는 Fully Connected Layer (완전 연결 레이어)의 일종으로, 입력 데이터와 가중치 행렬을 행렬곱(dot product)하고 편향(bias)을 더한 후, 활성화 함수(activation function)를 적용하는 레이어입니다.

`Dense`` 레이어의 인자로써 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.

학습률

```
initial_lr = 0.01  
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(  
    initial_lr,  
    decay_steps=10000,  
    decay_rate=0.96,  
    staircase=True)
```

학습률 설정, 후 갱신률 설정

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.

Adam

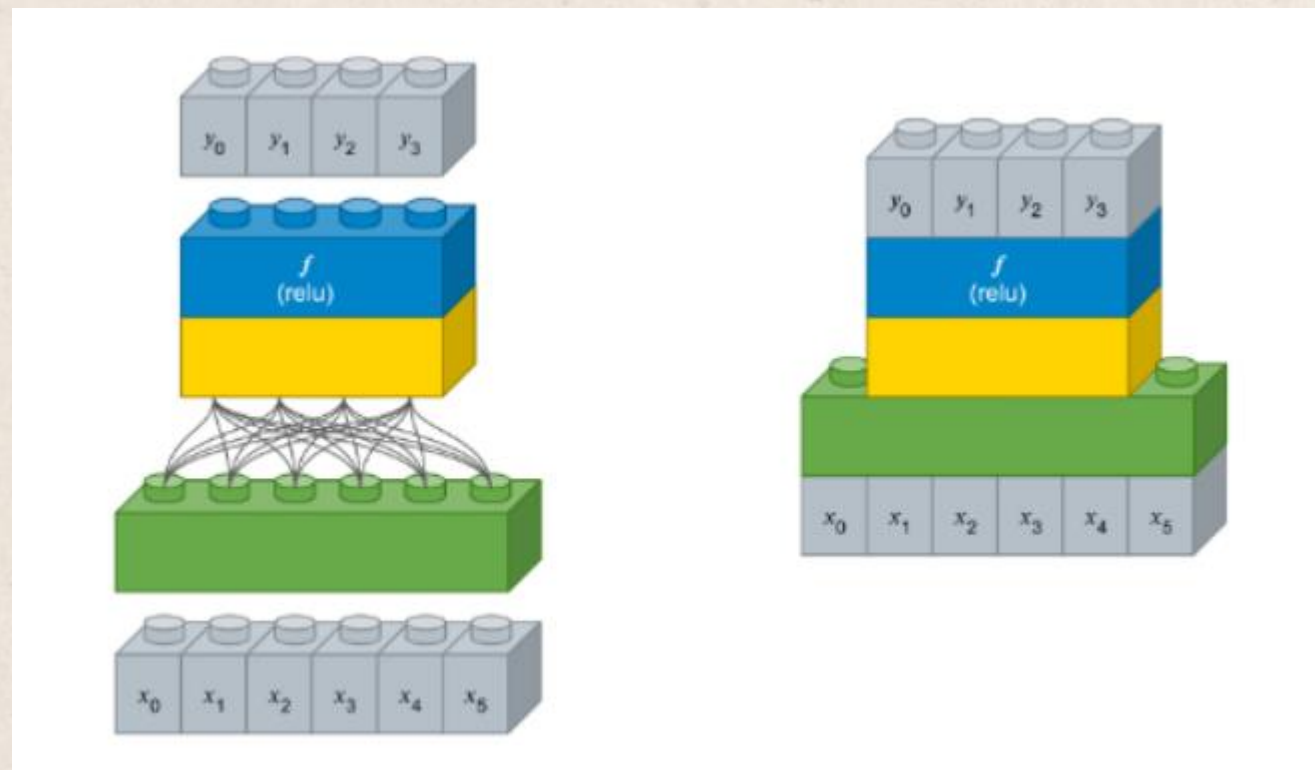
```
model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])  
model.summary()
```

이해가... 안됩니다 다음주 안에 추가하겠습니다.



summary(가중치)

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



가중치

dense_3 : 512(9216+1)

dense_4 : 256(512+1)

dense_5 : 10(256+1)

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
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		

model.fit

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

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

```
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
```

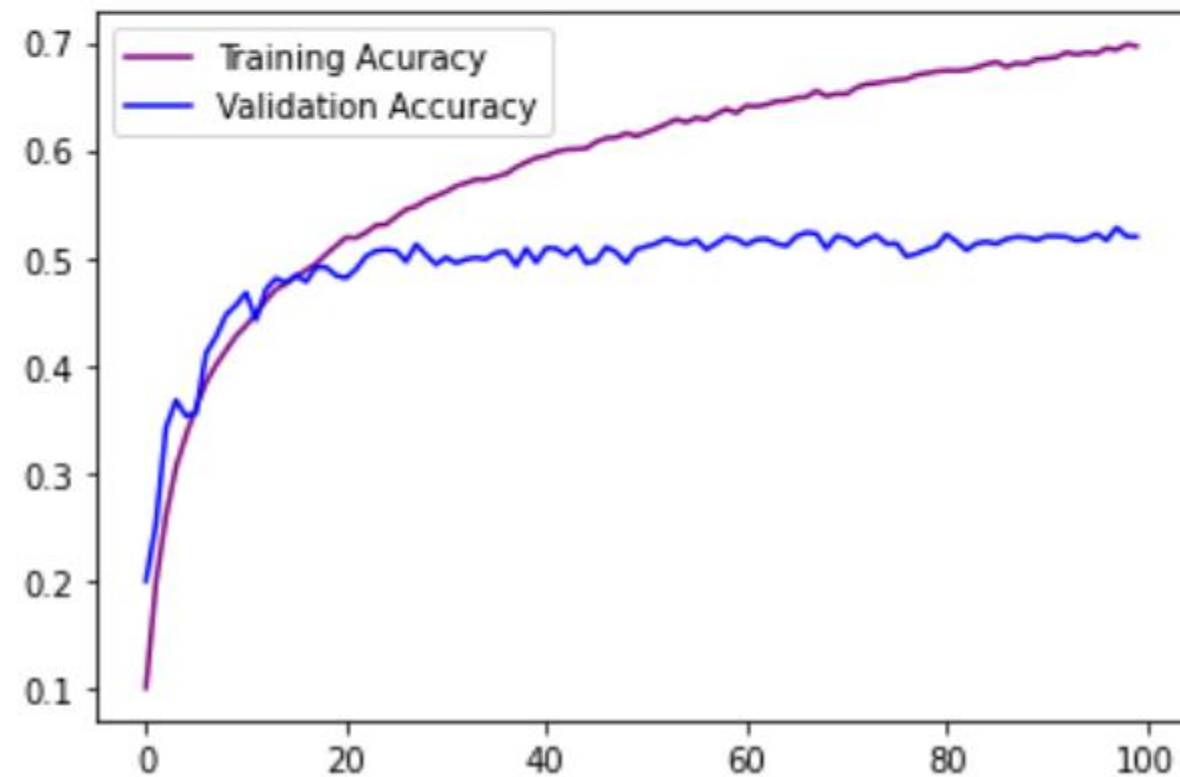
batch_size = 100, iteration = 600, epoch = 10

정확도

```
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()
```

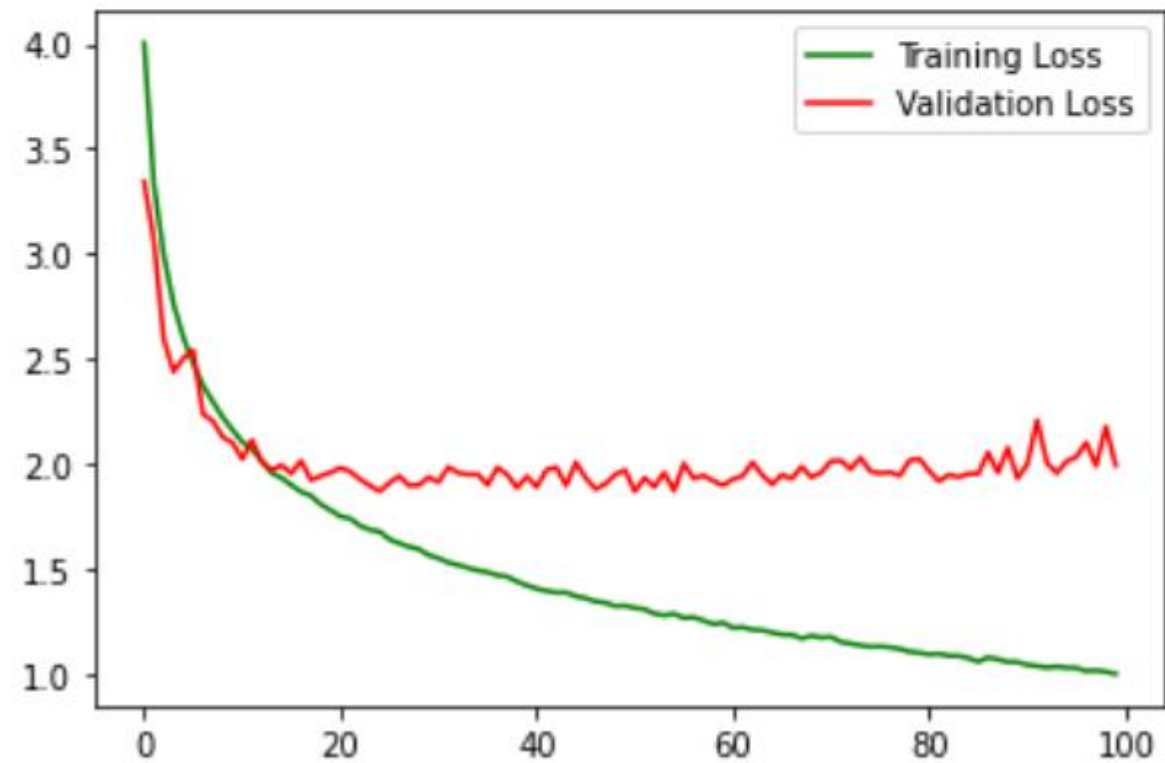
<matplotlib.legend.Legend at 0x7fceb9d5e3a0>



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 0x7fce9af51d30>



테스트 진행(경로 설정)

```
img_path_list = []  
for i in range(1,41):  
    root_dir = '/content/drive/MyDrive/Colab Notebooks/GITHUB/colab_ML/ML_CNN/dataset/cifar100/'  
    root_dir += str(i)  
    root_dir += '.jpg'  
    img_path_list.append(root_dir)  
  
print(img_path_list)
```

40개의 이미지 생성후 경로 저장

테스트 진행(경로 설정)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

count = 0
false_number=[]
for i in range(len(img_path_list)):
    img = cv2.imread(img_path_list[i])
    img = cv2.resize(img,(32,32),interpolation=cv2.INTER_NEAREST)
    imgRGB = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)/255
    imgRGB = imgRGB.reshape(-1,32,32,3)

    print(CIFAR100_CLASSES[(np.argmax((model.predict(imgRGB)>0.5).astype("int32")))]])
```

```
1/1 [=====] - 0s 22ms/step
apples hamster
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 24ms/step
leopard tree
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 32ms/step
snake tree
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 31ms/step
oranges oranges
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 41ms/step
oranges oranges
1/1 [=====] - 0s 30ms/step
0.15
```

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

예시

```
def draw_img(x) :  
    plt.figure()  
    plt.imshow(x)  
    plt.show()
```

```
idx = 4  
draw_img(x_test[idx])  
print(CIFAR100_CLASSES[y_test[idx][0]])
```

