Keras CNN 모델 구현

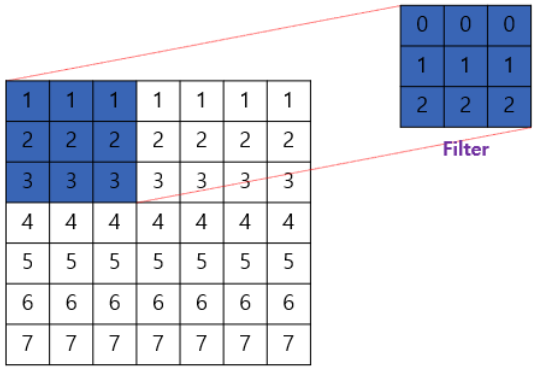
최석재 *lingua@naver.com*

Convolution Layer

- Convolution Layer의 특징
  - 입력 데이터의 형상을 유지
  - 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식
  - 복수의 필터로 이미지의 특징 추출 및 학습
  - 필터를 공유 가중치로 사용하기에 ANN에 비하여 가중치가 매우 적음

Convolution Layer

- Convolution Layer를 이용한 모델 구현



합성곱이 2차원 데이터에 대해 이루어지므로 Conv2D라고 한다

  - from keras import models
  - from keras import layers
  - import numpy as np
  - model = models.Sequential()

필터 개수 필터 크기(행, 열)

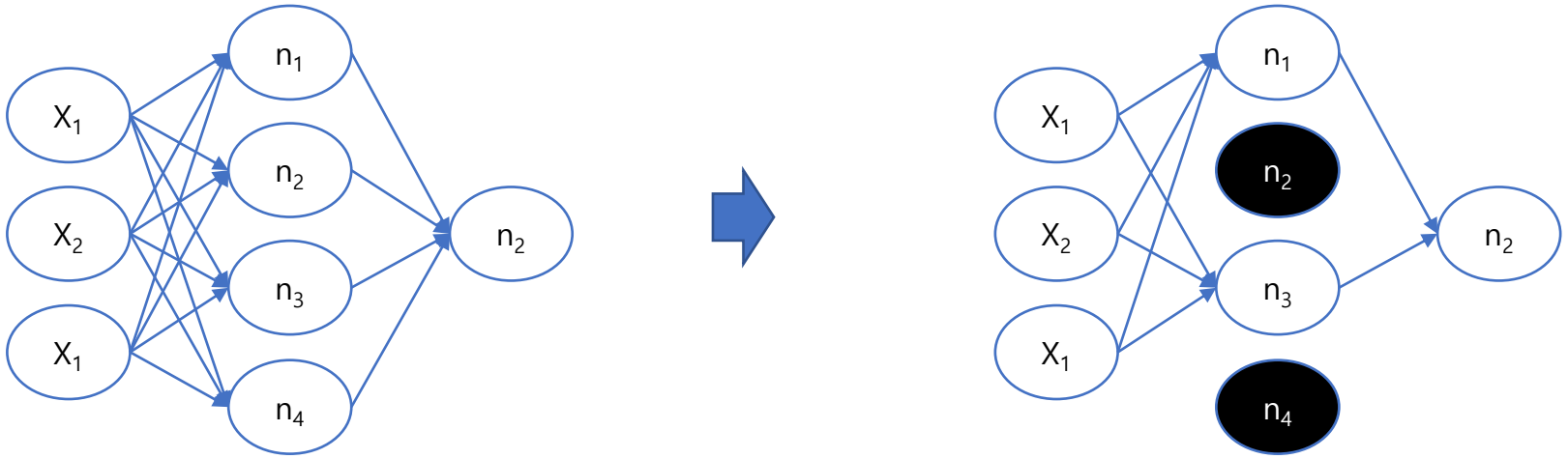
  - model.add(layers.Conv2D(**filters=32**, **kernel_size=(3, 3)**, input_shape=(28, 28, 1), activation='relu'))
  - model.add(layers.Conv2D(**filters=64**, **kernel_size=(3, 3)**, activation='relu'))

※ Conv2D층의 input_shape는 (image_height, image_width, image_channels)

※ strides 기본값은 (1, 1)

모델 개선 방법

Dropout



- layers.Dropout(0.25)

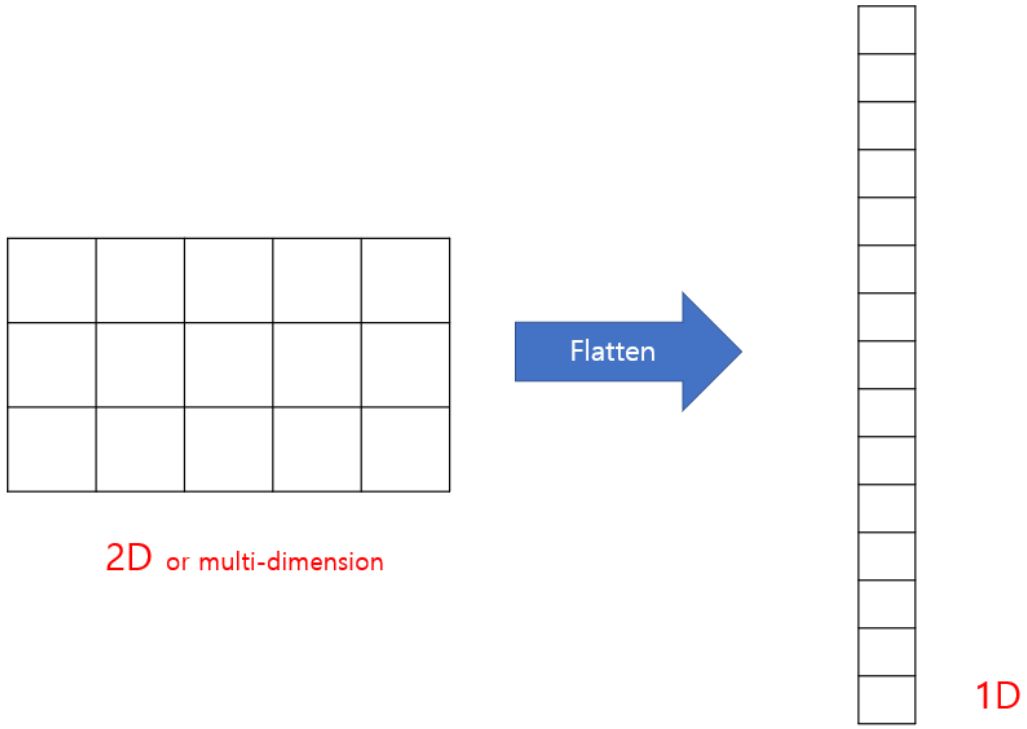
- Dropout은 노드의 일부분을 사용하지 않는 기법. 그 결과 과대적합을 줄일 수 있다
- 특정 노드의 가중치나 편향이 큰 값을 가지면 다른 노드는 큰 값을 갖는 노드에 의존(Co-adaptation 현상)하는데,
- 뉴런의 일부를 무작위하게 제거하면 이러한 불필요한 협업을 방지하게 된다
- 즉, 중요하지 않은 우연한 패턴은 깨뜨리며 과대적합을 줄인다
- 노드의 제거는 가중치를 0으로 만듦으로써 이루어진다

훈련 데이터를 학습하는 과정에서만 Dropout을 하고, Test 및 예측 시에는 다시 Dropout을 하지는 않는다
훈련 과정에서 사용된 이미 Dropout이 적용된 모델을 그대로 사용한다. Keras는 이를 자동으로 적용해준다

Flatten

- 이미지와 같이 다차원으로 구성된 데이터라도 최종 활성화 함수를 거쳐 결과를 출력하기 위해서는 각 이미지를 1차원 배열로 변환해야 한다
- 이를 수행하는 것이 Flatten()

- layers.Flatten()



Batch Normalization

$$y_i = \frac{x_i - E[X]}{\sqrt{Var[X] + \varepsilon}} * \gamma + \beta$$

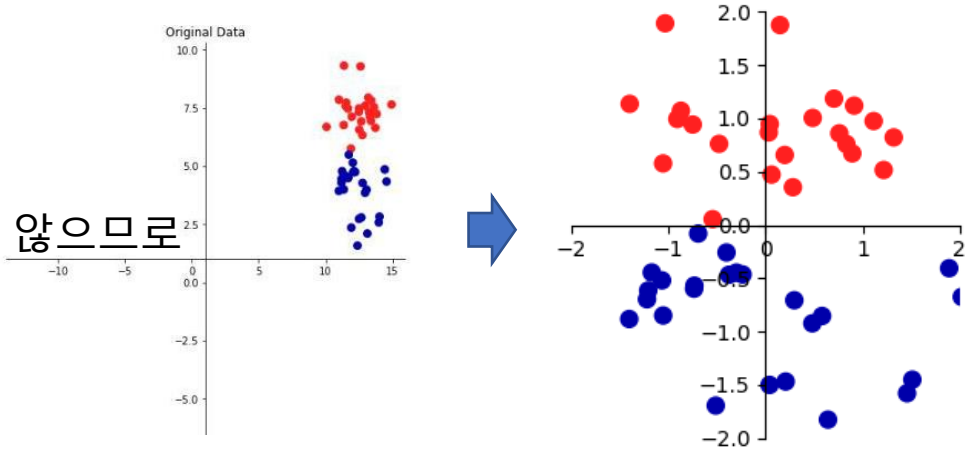
- 배치 정규화는 모델 학습 시 각 미니배치에서 평균과 분산을 조정하여 데이터의 분포를 표준화하는 기법
- 데이터의 평균이 0이 되도록 하고, 그 값을 다시 표준편차로 나누어서 분산이 1이 되도록 한다

※ 실제로는 0이 되지 않게 작은 ε값을 분모에 더하고,
높은 성능을 갖도록 학습 가능한 작은값 매개변수 γ와 β를 더한다

- 각 배치에서의 평균과 분산을 구하여 각각의 데이터에 적용하며,
- 그 결과로 배치 단위로 평균 0, 분산 1로 데이터의 값이 조정되어 데이터가 균일하게 된다

- 배치마다 데이터의 범위가 다르면 모델의 학습이 새로운 배치가 올 때마다 달라져 불안정해진다
- 균일하게 조정된 데이터로 학습된 모델은 속도, 성능, 안정성이 향상된다
- 기울기 소실 및 폭주 문제도 완화된다

- 활성화 함수 앞에 두어야 학습이 잘 되는 것으로 알려져 있으며,
- 배치 정규화 이후에 드롭아웃을 사용하면 분산 조정이 제대로 되지 않으므로 드롭아웃 이후에 배치 정규화를 사용해야 성능이 안정된다



Batch Normalization 설정

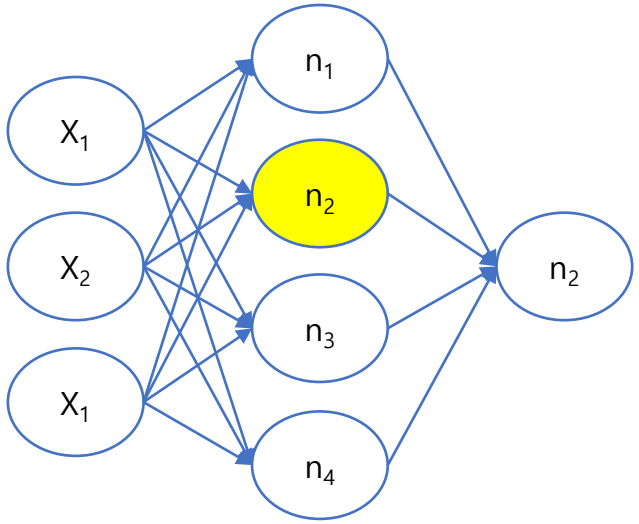
- model_bn = Sequential()

- model_bn.add(Dense(64, use_bias=False, input_shape=(max_words,)))
- **model_bn.add(BatchNormalization())**
- model_bn.add(Dense(32, use_bias=False))
- **model_bn.add(BatchNormalization())**
- model_bn.add(Dense(1, activation='sigmoid'))

Batch Normalization 앞에 편향과 활성화 함수를 사용하지 않는다

정규화(가중치 규제)

Weight Regularization



특이한 데이터가 입력이 되었고, 이 데이터를 학습하기 위하여
모델이 n2 노드에서 이를 학습하도록 가중치를 설정하려고 할 때,

특이한 데이터는 소수이므로 일반적인 범위의 가중치로는 n2 노드가 영향을 미치기 어렵다
따라서 n2에 관련된 가중치를 크게 올려서 전체 모델의 계산 결과로도 해당 데이터가
학습되게 하려고 한다

그러나 이렇게 되면 소수의 특이 데이터는 학습이 되겠지만,
다수의 일반적인 데이터가 제대로 학습되지 못하는 일이 발생한다

따라서, 일부 데이터를 포기하더라도 가중치가 일정한 수준을 넘지 않도록 하여
다수의 데이터가 학습되는 데 지장을 받지 않는 범위에서
소수의 특이한 데이터도 학습될 수 있도록 하는 지점을 찾아야 한다

L1 규제 $\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\alpha\sum_{n=1}^{N}|\theta_n|$

L2 규제 $\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2+\alpha\sum_{n=1}^{N}\theta_n^2$

※ θ : 가중치

규제 방식은 특정 배치에 나타난 특이한 데이터를 학습하기 위해
일단 일부 노드에 가중치를 올려주고, 그 모델의 손실값을 계산할 때
일반적인 손실 계산(왼쪽 항)에 추가 페널티(오른쪽 항)을 더한다

이와 같이 하면 가중치가 일정 정도 이상으로 커졌을 때
전체 손실값이 매우 커져 모델은 다음 배치 학습 때는
설정했던 가중치를 적극적으로 낮춰 전체 손실값이 높아지지 않게 한다

만약 일부 노드의 가중치를 높였는데도 전체 손실값이 그리 커지지 않았다면
다음 배치 학습 때 해당 노드의 가중치를 크게 바꿀 필요가 없어진다

- L2 규제는 제곱 연산이므로 1보다 작기만 하면 전체 비용이 낮게 나오도록 유도되는데, L1 규제는 0이 되어야만 확실하게 줄어드는 효과를 본다
- 그래서 L1 규제를 사용할 때는 많은 노드 가중치들이 0이 되는 현상이 나타난다 → 노드 삭제 효과 → 일반화에 기여하는 노드만 남는 장점

규제 설정

```
from keras.models import Sequential
from keras.layers import Dense
from keras import regularizers

# 기본 모델
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(max_words,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# l1 규제 모델
model_l1 = Sequential()
model_l1.add(Dense(64, kernel_regularizer=regularizers.l1(0.01), activation='relu', input_shape=(max_words,)))
model_l1.add(Dense(32, kernel_regularizer=regularizers.l1(0.01), activation='relu'))
model_l1.add(Dense(1, activation='sigmoid'))

# l2 규제 모델
model_l2 = Sequential()
model_l2.add(Dense(64, kernel_regularizer=regularizers.l2(0.01), activation='relu', input_shape=(max_words,)))
model_l2.add(Dense(32, kernel_regularizer=regularizers.l2(0.01), activation='relu'))
model_l2.add(Dense(1, activation='sigmoid'))
```

※ 규제 상수의 기본값은 0.01 이다
효과가 충분하지 않으면 0.001 ~ 0.01 사이의 값을 사용해본다

훈련 데이터를 학습하는 과정에서만 규제 상수를 적용하고, Test 및 예측 시에는 적용하지 않는다
훈련 과정에서 이미 규제 상수가 적용된 모델을 그대로 사용한다. Keras는 이를 자동으로 적용해준다

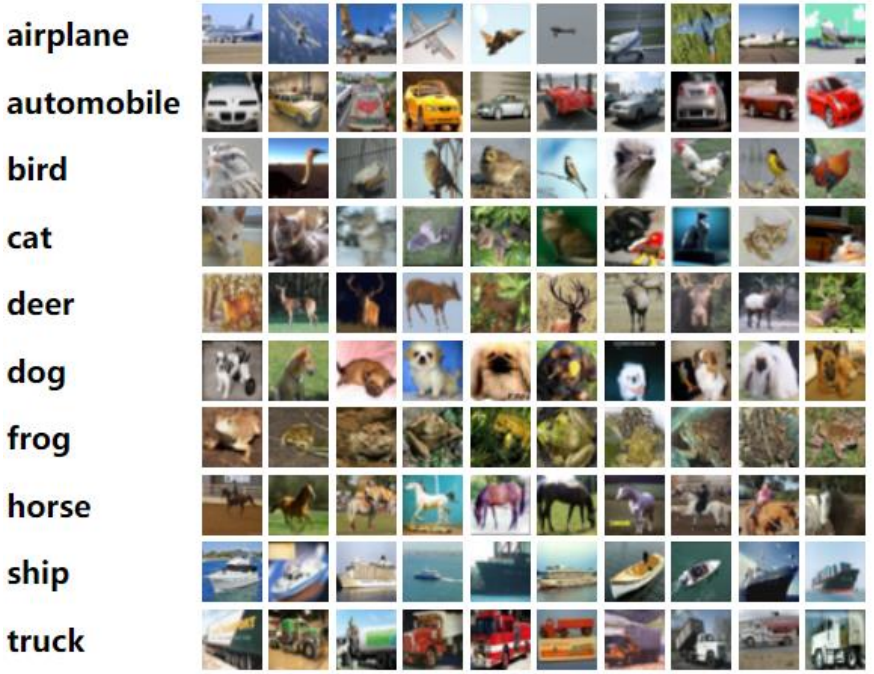
CNN 구현하기

CIFAR-10 이미지 데이터셋

- 케라스에 내장되어 있는 CIFAR-10 데이터셋은
- 10개 클래스, 32x32 사이즈의 60,000개 이미지로 구성되어 있음
- 50,000개는 훈련 이미지, 10,000개는 테스트 이미지로 구분

- 10개 클래스는 0~9번으로 인덱싱되어 있음

0: airplane
1: automobile
2: bird
3: cat
4: deer
5: dog
6: frog
7: horse
8: ship
9: truck



CNN 구현하기

※ 이번 실습은 훈련에 시간이 많이 소요되므로
CPU로 작성하고, 모델 훈련할 때는 GPU에서 실행할 수 있음

- CIFAR-10 데이터 불러오기

- from keras.datasets import cifar10
- (train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

- print(train_images.shape) # (50000, 32, 32, 3)
- print(test_images.shape) # (10000, 32, 32, 3)

- print(train_labels)
- print(test_labels)

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
(50000, 32, 32, 3)
(10000, 32, 32, 3)
[[6]
 [9]
 [9]
 ...
 [9]
 [1]
 [1]]
[[3]
 [8]
 [8]
 ...
 [5]
 [1]
 [7]]
```

CNN 구현하기

- 데이터 정규화

- train_images = train_images.astype('float32')/255

- test_images = test_images.astype('float32')/255

CNN 구현하기

- 종속변수를 범주형으로 변환
  - 또한, label을 범주형으로 만들어 분류 알고리즘에 적합한 형태가 되게 함

  - from tensorflow.keras.utils import to_categorical
  - train_labels = to_categorical(train_labels)
  - test_labels = to_categorical(test_labels)

  - import numpy as np
  - import sys
  - np.set_printoptions(threshold=sys.maxsize)
  - print(train_labels[:5])

```
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

CNN 구현하기

- Layer 쌓기
- from keras import models
- from keras import layers

```
model = models.Sequential()
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=2))
model.add(layers.Conv2D(filters=256, kernel_size=(3, 3), activation='relu'))
model.add(layers.Flatten())
```

Feature Extraction

```
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.25))
model.add(layers.Dense(10, activation='softmax'))
```

Classification

모델 요약

- model.summary()

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2  (None, 15, 15, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 conv2d_2 (Conv2D)           (None, 11, 11, 128)       73856

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 3, 3, 256)         295168

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 128)               295040

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 684746 (2.61 MB)
Trainable params: 684746 (2.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3x3x256=2304

Conv2D에서 각 필터는 각 채널에 대하여
합성곱을 수행하므로 필터의 수는 출력 텐서의 채널 수가 된다

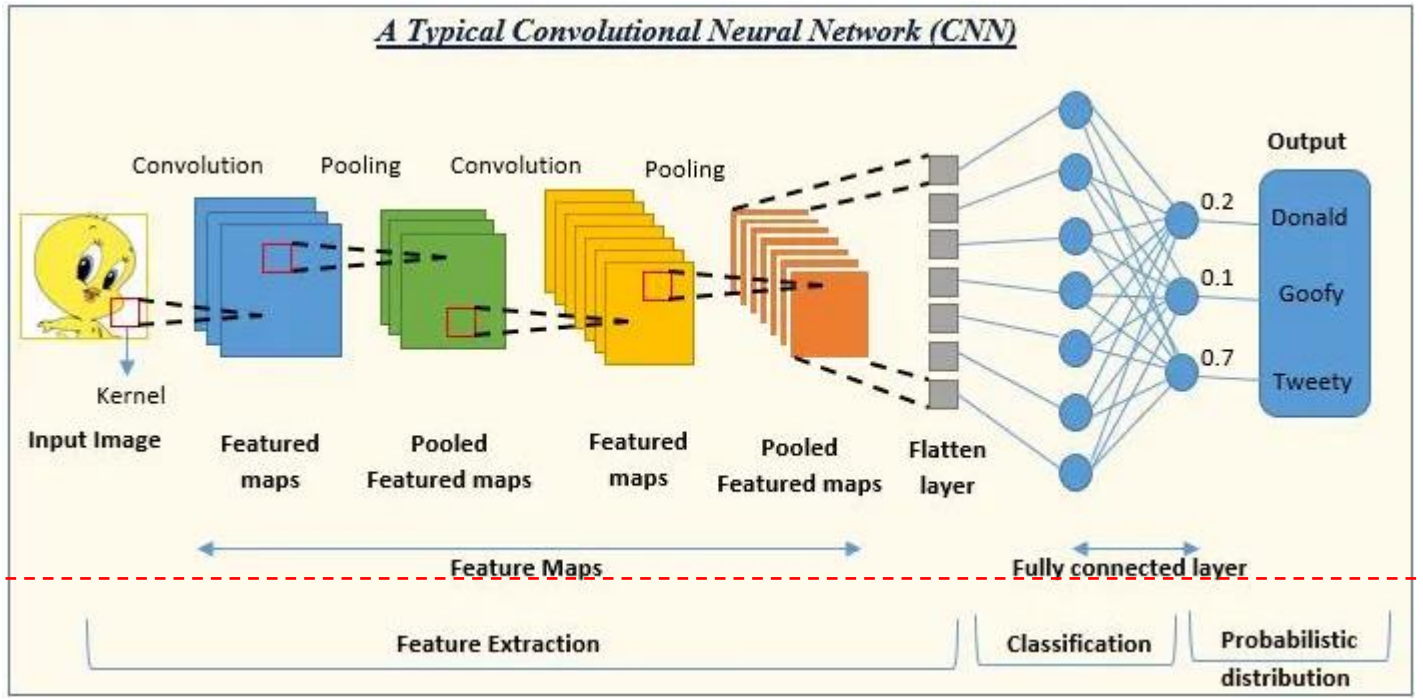
Conv2D 층과 MaxPooling2D 층은 4D를 받아 4D를 출력한다

완전연결층(Dense층)과 합성곱층(Conv2D층) 사이의 근본적인 차이는 지역패턴을 학습하느냐에 있다
Dense층은 입력특성 모두에 있는 패턴을 학습하지만, 합성곱층은 지역패턴을 학습한다

CNN 구현하기

- Compile model

  - model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])



CNN 구현하기

- 모델 훈련

- history = model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_data=(test_images, test_labels))

```
Epoch 1/10
391/391 [==============================] - 8s 7ms/step - loss: 1.8621 - acc: 0.3222 - val_loss: 1.4775 - val_acc: 0.4673
Epoch 2/10
391/391 [==============================] - 2s 5ms/step - loss: 1.4564 - acc: 0.4783 - val_loss: 1.3332 - val_acc: 0.5280
Epoch 3/10
391/391 [==============================] - 2s 5ms/step - loss: 1.2431 - acc: 0.5608 - val_loss: 1.1350 - val_acc: 0.5991
Epoch 4/10
391/391 [==============================] - 2s 5ms/step - loss: 1.0857 - acc: 0.6218 - val_loss: 1.0830 - val_acc: 0.6257
Epoch 5/10
391/391 [==============================] - 2s 5ms/step - loss: 0.9505 - acc: 0.6701 - val_loss: 0.9825 - val_acc: 0.6606
Epoch 6/10
391/391 [==============================] - 2s 5ms/step - loss: 0.8356 - acc: 0.7108 - val_loss: 0.9615 - val_acc: 0.6744
Epoch 7/10
391/391 [==============================] - 2s 5ms/step - loss: 0.7354 - acc: 0.7468 - val_loss: 0.9273 - val_acc: 0.6902
Epoch 8/10
391/391 [==============================] - 2s 5ms/step - loss: 0.6473 - acc: 0.7753 - val_loss: 0.8998 - val_acc: 0.7034
Epoch 9/10
391/391 [==============================] - 2s 5ms/step - loss: 0.5674 - acc: 0.8040 - val_loss: 0.8103 - val_acc: 0.7364
Epoch 10/10
391/391 [==============================] - 2s 5ms/step - loss: 0.4890 - acc: 0.8312 - val_loss: 0.8511 - val_acc: 0.7279
```

CNN 구현하기

- 예측 및 평가

  - test_loss, test_acc = model.evaluate(test_images, test_labels)
  - print('test_acc:', test_acc)

```
313/313 [==============================] - 1s 2ms/step - loss: 0.8511 - acc: 0.7278
test_acc: 0.7278000116348267
```

CNN 구현하기

- Accuracy & Loss 확인

  - acc = history.history['acc']
  - val_acc = history.history['val_acc']
  - loss = history.history['loss']
  - val_loss = history.history['val_loss']

  - print('Accuracy of each epoch:', acc)
  - epochs = range(1, len(acc) +1)

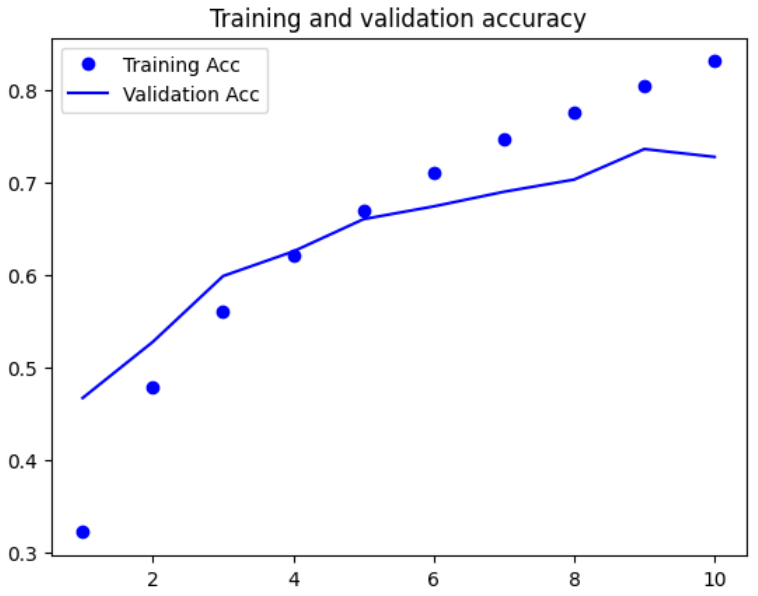
```
Accuracy of each epoch: [0.32221999764442444, 0.478300005197525, 0.5608199834823608, 0.6218000054359436, 0.6700800061225891, 0.7108200192451477, 0.7467799782752991, 0.7752599716186523, 0.8040199875831604, 0.8312000036239624]
```

CNN 구현하기

- 정확도 그래프

  - import matplotlib.pyplot as plt

  - plt.plot(epochs, acc, 'bo', label='Training Acc')
  - plt.plot(epochs, val_acc, 'b', label='Validation Acc')
  - plt.title('Training and validation accuracy')
  - plt.legend()

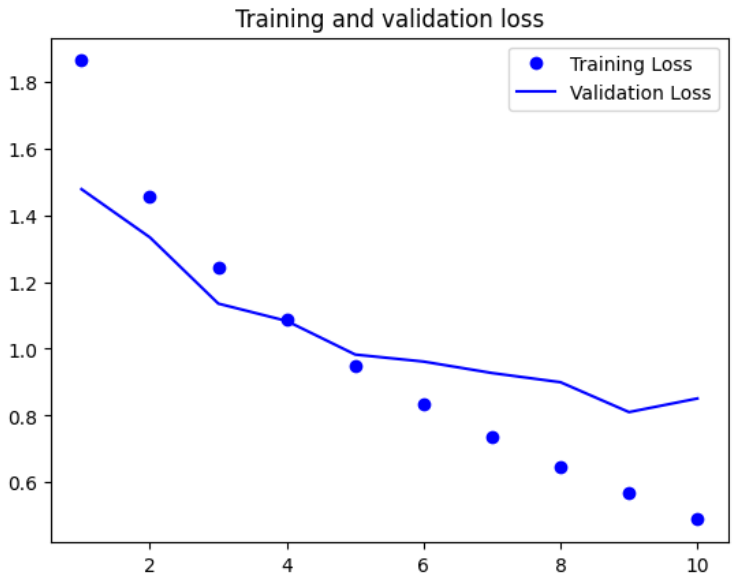


CNN 구현하기

- 손실값 그래프

  - plt.figure() # 새로운 그림을 그린다

  - plt.plot(epochs, loss, 'bo', label='Training Loss')
  - plt.plot(epochs, val_loss, 'b', label='Validation Loss')
  - plt.title('Training and validation loss')
  - plt.legend()



모델 저장

- 구글 드라이브 접속

- from google.colab import drive
- drive.mount('/content/gdrive')

모델 저장
- import os
- os.chdir("/content/gdrive/MyDrive/pytest_img/models")
- model.save("cifar10.h5")

```
# models 경로가 없으면 아래의 코드로 경로를 생성한다

import os

model_dir = '/content/gdrive/MyDrive/pytest_img/models/'

if not os.path.exists(model_dir):
    os.mkdir(model_dir)

os.chdir(model_dir)
```

모델 재사용

모델 불러오기

- from keras.models import load_model
- loaded_model = load_model("cifar10.h5")

데이터 예측하기

- predictions = loaded_model.predict(test_images)
- print(predictions.shape)

```
313/313 [==============================] - 1s 2ms/step
(10000, 10)
```

예측 결과 확인

앞의 10개만 확인한다

- print(predictions[:10])

```
[[8.05530138e-03 6.32284163e-03 2.42556632e-03 8.78391981e-01
  7.17100338e-05 8.20656568e-02 5.69600891e-03 2.56296434e-03
  9.48072225e-03 4.92728269e-03]
 [3.46870311e-02 2.92967842e-03 7.34436085e-08 1.79008936e-07
  3.42313378e-08 1.80460646e-10 5.52627277e-10 4.32094943e-10
  9.52584863e-01 9.79817100e-03]
 [4.50591557e-02 7.60095567e-02 1.14713330e-04 6.04064902e-04
  1.04754923e-04 7.38606468e-05 8.57499763e-05 1.02682156e-04
  7.53898621e-01 1.23946853e-01]
 [8.65727425e-01 3.22938594e-03 8.66796356e-03 2.88822595e-03
  1.05559807e-02 3.27327289e-05 1.38702098e-05 4.10336870e-05
  1.06560446e-01 2.28282344e-03]
 [6.85672103e-06 1.63899108e-06 2.15011332e-02 2.05022916e-02
  4.29437049e-02 3.62144201e-03 9.11273479e-01 4.48101946e-06
  1.43195895e-04 1.78548544e-06]
 [2.21758857e-02 6.85916469e-02 7.76629746e-02 1.32236585e-01
  5.29981032e-03 7.45883435e-02 5.67018986e-01 1.28051518e-02
  9.76814795e-03 2.98524573e-02]
 [1.75236096e-03 4.13765520e-01 1.23284338e-03 6.68811379e-03
  3.97609170e-07 1.01563768e-04 4.33019932e-06 5.02209732e-05
  5.35489344e-05 5.76351106e-01]
 [2.30636215e-04 9.98253927e-06 9.66002175e-04 2.68305914e-04
  5.00195347e-05 3.35250807e-05 9.98326957e-01 1.59328431e-06
  4.28399944e-05 7.02576144e-05]
 [1.68429390e-02 5.92071738e-05 3.82104754e-01 3.82798523e-01
  5.33602983e-02 1.07726373e-01 1.52958063e-02 4.05819491e-02
  7.82950257e-04 4.47207625e-04]
 [7.65154429e-04 9.01345789e-01 2.64635397e-04 1.16255775e-04
  8.86625094e-06 9.46384262e-06 1.72938671e-04 1.13002443e-06
  2.17757584e-03 9.51381177e-02]]
```

예측 결과 분류

앞의 10개만 확인한다
열을 기준으로 최댓값의 인덱스를 추출한다

- predicted_classes = np.argmax(predictions[:10], axis=1)
- print("예측 결과:", predicted_classes)

```
예측 결과: [3 8 8 0 6 6 9 6 3 1]
```

0: airplane
1: automobile
2: bird
3: cat
4: deer
5: dog
6: frog
7: horse
8: ship
9: truck