AI Vision 검사기 프로젝트 실습

1. 학습용 데이터 구축

1.1 훈련 데이터와 테스트 데이터

머신러닝의 지도학습에서는 취급하는 데이터를 학습에 사용하는 훈련(training) 데이터와 학습된 모델의 정밀도(precision) 평가에 사용하는 테스트(test) 데이터로 나누어 사용합니다. 머신러닝 모델의 평가에는 학습에 사용되지 않은 테스트 데이터를 사용해야 합니다. 데이터를 분리하는 방법 중에 홀드아웃 방법과 k-분할 교차검증과 학습 데이터 구축 시 고려 사항에 대해 살펴보겠습니다.

1.2 홀드아웃 방법

홀드아웃 방법은 주어진 데이터셋을 훈력 데이터와 테스트 데이터 두 가지로 분할하는 방법입니다. scikit-learn 라이브러리의 train_test_split() 함수를 사용하여 홀드아웃 방법을 구현할 수 있습니다.

< 홀드아웃 실습 >

```
from sklearn import datasets
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test: ", X_test.shape)
```

1.3 k-분할 교차검증

k-분할 교차검증은 모델 평가 검증 방법의 하나입니다. 비복원 추출을 이용하여 훈련 데이터셋을 k개로 분할한 다음 k-1개의 데이터는 학습 데이터셋으로 사용하고, 나머지 1개를 모델 테스트에 사용하는 방법입니다. 따라서 k회 학습과 평가를 반복하여 얻은 k개 성능 평가의 평균을 구해 평균 성능을 산출합니다.
홀드아웃 방법보다 k배의 연산이 필요한 단점과 안정되고 정확한 모델 평가가 가능한 장점이 있습니다.

< k-분할 교차검증 실습 >

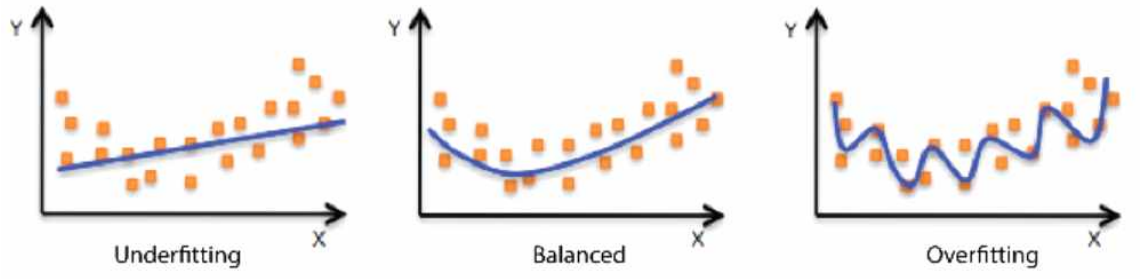
```
from sklearn import svm, datasets, model_selection

iris = datasets.load_iris()
X = iris.data
y = iris.target
svc = svm.SVC(C=1, kernel='rbf', gamma=0.001)
scores = model_selection.cross_val_score(svc, X, y, cv=5)

print(scores)
print("평균 점수", scores.mean())
```

1-4. 과적합과 회피 방법

데이터의 패턴에서 기준을 만들 때, 편중된 데이터가 사용되어 올바른 기준을 만들지 못한 경우를 데이터를 과하게 학습한 상태라 하며, 이를 과적합(overfitting)이라 부릅니다.

과적합을 피하기 위해 딥러닝에서는 학습 시 무작위로 일부뉴런을 없애는 드롭아웃 기법을 사용하며, 그 밖의 방법으로 편향된 데이터를 없애는 정규화(regularization, normalization)가 있습니다.

반대로 데이터를 제대로 학습하지 못한 상태를 과소적합(underfitting)이라고 합니다. 과적합 모델은 분산이 크고, 과소적합 모델은 편향이 크다고 할 수 있습니다.

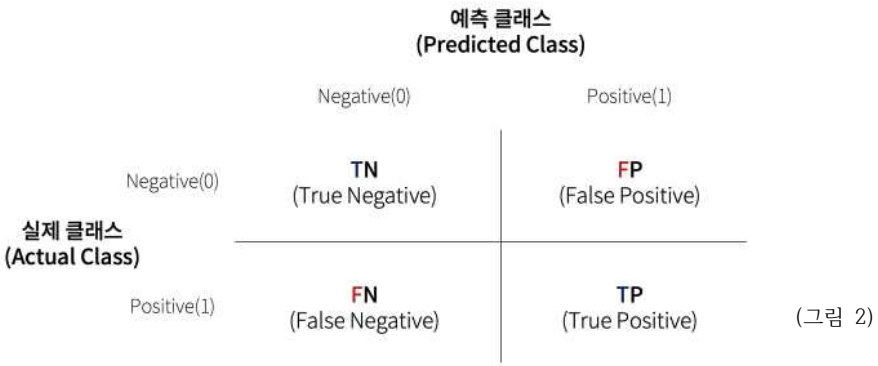


(그림 1)

1-5. 성능평가지표와 PR곡선

성능평가지표 중에 혼동 행렬(confusion matrix)은 각 데스트 데이터에 대한 모델의 예측 결과 다음의 4가지 관점에서 분류하여 정리한 표입니다.

- 참 양성(True Positive): 양성 클래스로 예측되었고 결과도 양성 클래스인 개수
- 참 음성(True Negative): 음성 클래스로 예측되었고 결과도 음성 클래스인 개수
- 거짓 양성(False Positive): 양성 클래스로 예측되었지만, 결과는 음성 클래스인 개수
- 거짓 음성(False Negative): 음성 클래스로 예측되었지만, 결과는 양성 클래스인 개수



(그림 2)

참 양성과 참 음성은 머신러닝 모델이 정답을 도출한 것이며, 거짓 음성과 거짓 양성을 오답을 도출한 것을 보여줍니다.

< 혼동 행렬 구현 실습 >

```
import numpy
from sklearn.metrics import confusion_matrix

y_true = [0,0,0,1,1,1]
y_pred = [1,0,0,1,1,1]
confmat = confusion_matrix(y_true, y_pred)

print(confmat)
```

혼동 행렬을 바탕으로 산출 가능한 성능평가지표로 정확도(Accuracy)가 있습니다. 정확도는 모든 경우 중에 진단 결과가 맞은 비율로 다음과 같이 계산할 수 있습니다.

$$\text{정확도} = \frac{TP + TN}{FP + FN + TP + TN}$$

데이터가 한쪽으로 치우쳐 있는 경우에는 정확도는 문제가 발생하기 때문에 머신러닝에서는 적합률(precision, 정밀도), 재현류(recall), F값(F-measure)을 성능지표로 평가하는 경우가 많습니다.

적합률은 양성으로 예측된 데이터 중 실제로 양성인 것의 비율로 다음과 같습니다.

$$\text{적합률} = \frac{TP}{TP + FP}$$

재현률은 실제 양성 데이터 중에 양성으로 예측된 것의 비율로 다음과 같습니다.

$$\text{재현율} = \frac{TP}{TP + FN}$$

F값은 적합률과 재현율의 조화 평균으로 다음과 같습니다.

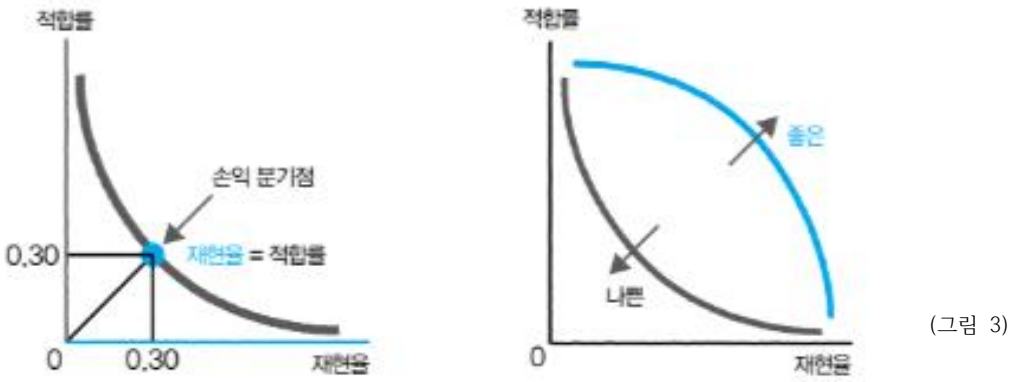
$$F\text{값} = 2 \times \frac{\text{적합률} \times \text{재현율}}{\text{적합률} + \text{재현율}}$$

적합률, 재현율, F값 모두 0에서 1 사이 값을 가지며, 1에 가까울수록 성능이 좋은 것을 의미합니다.

< 성능평가지표 구현 실습 >

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score, f1_score

y_true = [0,0,0,1,1,1]
y_pred = [1,0,0,1,1,1]

print("Precision: %.3f" % precision_score(y_true, y_pred))
print("Recall: %.3f" % recall_score(y_true, y_pred))
print("F1: %.3f" % f1_score(y_true, y_pred))
```

적합률과 재현율은 상충관계(trade-off)에 있는 값으로, PR 곡선(precision-recall curve)은 가로축을 재현율, 세로축을 적합률로 한 그래프입니다. PR 곡선에서 적합률과 재현율이 일치하는 지점을 손익분기점(break-even point, BEP)라고 합니다.



(그림 3)

2. 파이썬 라이브러리 활용 데이터 전처리

2.1 딥러닝을 위한 파이썬 기본 문법

(1) 리스트(List)

리스트형은 수치, 문자열 등의 데이터를 한꺼번에 저장할 수 있는 자료형으로, [요소1, 요소2, ...]처럼 기술합니다. 리스트에 저장되는 값 각각을 요소(element) 또는 객체(object)라고 합니다.

< 리스트 자료형 실습 >

```
a = ["red", "green", "blue"]

print(a)

print(type(a))
```

리스트형은 저장되어 있는 요소가 다른 자료형이어도 되고, 변수를 리스트 안에 저장할 수도 있습니다.

< 리스트 자료형 실습 >

```
n = 5

a = [3, n, "사과"]

print(a)
```

리스트의 요소로 리스트형을 저장하여 중첩된 구조를 만들 수 있습니다.

< 리스트 자료형 실습 >

```
subject_name_1 = "국어"
subject_score_1 = 97
subject_name_2 = "영어"
subject_score_2 = 80
subject_name_3 = "수학"
subject_score_3 = 95

scores = [[subject_name_1, subject_score_1], [subject_name_2, subject_score_2], [subject_name_3, subject_score_3]]

print(scores)
```

리스트 요소에는 0, 1, 2, ... 순서로 인덱스 번호가 할당되어 있습니다. 주의할 점은 0부터 인덱스 번호가 할당된다는 것입니다. 또한 리스트의 가장 마지막 요소의 인덱스 번호는 -1, 끝에서 두 번째 요소의 인덱스 번호는 -2처럼 뒤에서부터 순서대로 번호를 지정할 수도 있습니다.

리스트에서 새로운 리스트를 추출할 수 있는데, 이를 슬라이스라고 합니다. 작성법은 '리스트[start:end]' 형식으로, 인덱스 번호 start부터 end-1 까지의 리스트를 추출합니다.

< 리스트 자료형 실습 >

```
hangul = ["가", "나", "다", "라", "마", "바", "사", "아"]

print(hangul[1])
print(hangul[-2])
print(hangul[1:5])
print(hangul[1:-5])
print(hangul[:5])
print(hangul[5:])
print(hangul[0:20])
```

리스트 요소를 갱신할 때는 '리스트[인덱스 번호] = 값' 형식을 사용합니다. 슬라이스를 이용하여 갱신도 가능합니다. 리스트 요소를 추가할 때는 '리스트1 + 리스트2' 형식이나, '리스트명.append(추가할 요소)' 형식울 사용합니다.

리스트 요소를 삭제할 때는 'del 리스트[삭제할 요소의 인덱스 번호]' 형식을 사용하며, 슬라이스로 지정할 수도 있습니다.

< 리스트 자료형 실습 >

```
hangul = ["가", "나", "다", "라", "마", "바", "사", "아"]
hangul2 = ["자", "차", "카"]

print(hangul)

hangul= hangul + hangul2
print(hangul)

hangul.append("타")
print(hangul)

hangul[4:6] = ["A", "B"]
print(hangul)

del hangul[2]
print(hangul)

del hangul[4:6]
print(hangul)
```

리스트형을 이용할 때 주의할 점은 리스트 변수를 다른 변수에 대입(예; y = x) 한 뒤에 대입한 변수에서 값을 바꾸면 원래 리스트 변수의 값도 바뀌게 됩니다. 이를 방지하기 위해서는 y = x 대신에 y = x[:] 또는 y = list(x)로 표기하면 됩니다.

< 리스트 자료형 실습 >

```
hangul = ["가", "나", "다"]
hangul2 = hangul

print(hangul)

hangul2[1] = "하"
print(hangul)

hangul3 = hangul[:]
hangul3[1] = "너"
print(hangul)
```

(2) 딕셔너리(Dictionary)

딕셔너리형은 리스트형처럼 여러 데이터를 다룰 때 사용하며, 데이터가 키(key)와 값(value) 쌍으로 되어 있습니다. 값을 얻기 위해서는 인덱스 번호가 아니라 키를 통해서 얻습니다. 딕셔너리 만들 때는 '{키1: 값1, 키2: 값2, ...}' 형식을 사용합니다.

< 딕셔너리 자료형 실습 >

```
town = {"전라북도": "전주", "전라남도": "무안"}

print("전라북도의 중심 도시는 " + town["전라북도"] + "입니다")
```

딕셔너리 값을 갱신할 때는 '딕셔너리명["값을 갱신할 키"] = 값" 형식을, 요소를 추가할 때는 '딕셔너리명["추가할 키"] = 값" 형실을 사용합니다.
딕셔너리 요소를 삭제할 때는 'del 딕셔너리명["삭제할 키"]' 형식을 사용합니다.

< 딕셔너리 자료형 실습 >

```
town = {"전라북도": "전주", "전라남도": "무안"}

print(town)
town["전라남도"] = "나주"
print(town)

town["충청북도"] = "충주"
print(town)

del town["전라남도"]
print(town)
```

(3) 튜플(Tuple)

튜플형도 리스트형처럼 여러 데이터를 다룰 때 사용하는데, 리스트형과 달리 요소를 수정하거나 추가, 삭제할 수 없습니다. 튜플을 만들 때는 '(값1, 값2,...)' 형식을 사용합니다.

< 튜플 자료형 실습 >

```
a = (1, 2, 3)

print(a)
print(type(a))

a[3] = 4

del a[1]

a[1] = 4
```

(4) 함수(Function)

함수는 입력값을 받아 어떤 일을 수행하고 그 결과물을 반환해 줍니다. 함수를 사용하면 동일한 처리를 여러 번 작성하지 않고 반복하여 사용할 수 있는 장점이 있습니다.

함수 작성법은 'def 함수명(매개변수명):'입니다. 매개변수는 함수에서 전달받을 입력값을 저장할 변수이며, 매개변수가 없을 수도 있습니다. 함수의 처리 범위는 들여쓰기로 구분합니다.

함수를 호출할 때는 '함수명(인수)' 형식을 사용합니다. 함수는 정의 후에 호출 가능합니다.

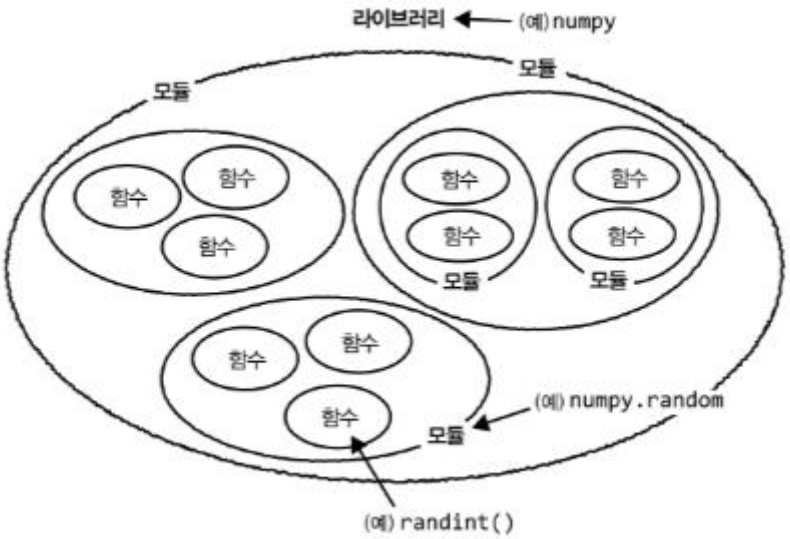
< 함수 실습 >

```
def greeting():
    print("안녕하세요!")

def cube_cal(n):
    print(n ** 3)

def cube_cal2(n):
    a = n ** 3
    return a

def intro(name, age):
    print("저는 " + name + "이고, 나이는 " + str(age) + "살 입니다.")

greeting()
cube_cal(3)
b = cube_cal2(4)
print(b)
intro("홍길동", 20)
```

(5) Numpy

Numpy(넘파이)는 파이썬으로 벡터나 행렬 계산을 빠르게 할수 있도록 특화된 기본 라이브러리입니다. 라이브러리는 외부에서 읽어 들이는 파이썬 코드 묶음으로, 여러 모듈들이 포함되어 있습니다. 모듈은 많은 함수가 통합되어 있는 것입니다.

Numpy 외에 Scipy, Pandas, scikit-learn, Matplotlib 등이 자주 이용되는 라이브러리들입니다.



(그림 4)

< Numpy 행렬 처리 실습 >

```
import numpy as np
import time
from numpy.random import rand

N = 200

matA = np.array(rand(N, N))
matB = np.array(rand(N, N))
matC = np.array([[0] * N for _ in range(N)]) # [0] 리스트를 N번 반복한 것을 range(N)만큼 반복함

print(matC.shape)

start = time.time()

for i in range(N):
    for j in range(N):
        for k in range(N):
            matC[i][j] = matA[i][k] * matB[k][j]

print("파이썬 기능만으로 계산한 결과: %.2f[초]" % float(time.time() - start))
```

< Numpy 행렬 처리 실습 >

```
import numpy as np
import time
from numpy.random import rand

N = 200

matA = np.array(rand(N, N))
matB = np.array(rand(N, N))
matC = np.array([[0] * N for _ in range(N)]) # [0] 리스트를 N번 반복한 것을 range(N)만큼 반복함

# print(matC.shape)
# print(matC[1][1])

start = time.time()

for i in range(N):
    for j in range(N):
        for k in range(N):
            matC[i][j] = matA[i][k] * matB[k][j]

print("파이썬 기능만으로 계산한 결과: %.2f[초]" % float(time.time() - start))

start = time.time()
matC = np.dot(matA, matB)

print("Numpy를 사용하여 계산한 결과: %.2f[초]" % float(time.time() - start))
```

Numpy에는 배열을 고속으로 처리하는 ndarray 클래스가 있습니다. ndarray를 생성하는 방법 중 하나는 Numpy.array() 함수를 이용하는 것입니다. ndaaray 클래스는 1차원 배열은 벡터(vector), 2차원 배열은 행렬(matrix), 3차원 배열 이상은 텐서(tensor)라고 부릅니다.

< ndarray 클래스 실습 >

```
import numpy as np

array_1D = np.array([1,2,3,4,5,6,7,8])
array_2D = np.array([[1,2,3,4],[5,6,7,8]])
array_3D = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])

print(array_1D.shape)
print(array_2D.shape)
print(array_3D.shape)
```

ndarray 배열도 리스트아 마찬가지로 다른 변수에 대입할 경우 해당 변수의 요소 값을 변경하면 원래 ndarray 배열의 요소 값도 변경됩니다. 이를 피하기 위해서는 copy() 메서드로 '변수 = 복사할배열명.copy()' 형식으로 사용해야 합니다.

< ndarray 클래스 실습 >

```
import numpy as np

array1 = np.array([1, 2, 3, 4, 5])
print(array1)

array2 = array1
array2[2] = 100
print(array1)
array1 = np.array([1, 2, 3, 4, 5])
print(array1)

array2 = array1.copy()
array2[2] = 100
print(array1)
```

ndarray의 슬라이스는 리스트와 달리 배열의 복사본이 아닌 view라는 것입니다. view는 원래 배열을 가리키는 것으로, ndaaray의 슬라이스를 복사본으로 만들려면 copy() 메서드를 사용해야 합니다.

< ndarray 클래스 실습 >

```
import numpy as np

array1 = np.arange(10)
list1 = [x for x in range(10)]
print(list1)

array2 = array1[:]
array2[0] = 100
print(array1)

list2 = list1[:]
list2[0] = 100
print(list1)

array1 = np.arange(10)
print(array1)

array2 = array1[:].copy()
array2[0] = 100
print(array1)
```

ndarray는 [] 안에 논리값(True, False) 배열을 사용하여 요소를 추출할 수 있는 부울 인덱스 참조를 지원합니다. '배열명[ndarray 논리값 배열]' 형식으로 표기하면 논리값 배열의 True에 해당하는 요소의 ndarray를 만들어 반환해 줍니다.

< ndarray 클래스 실습 >

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print(arr[np.array([False, True, True, False, True])])
print(arr[arr % 2 == 1])
```

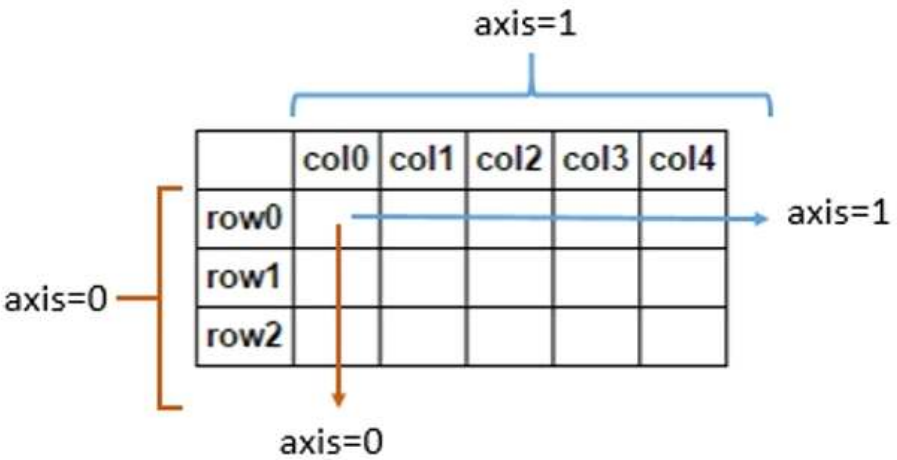
Numpy에는 ndarray 배열의 각 요소에 대한 연산 결과를 반환하는 함수인 범용 함수와 1차원 배열만을 대상으로 하는 수학의 집합연산을 수행하는 집합 함수를 지원합니다. 또한 random 모듈로 난수를 생성할 수 있습니다.

Numpy에서 2차원 배열은 'Numpy.array([리스트, 리스트])' 형식으로 만들 수 있습니다. ndarray 배열 내부에 있는 shape 변수를 통해 배열 각 차원의 요소 수를 알 수 있습니다. 그리고 'ndarray배열.reshape(a, b)' 형식을 사용하여 (a, b)와 같은 행렬로 변환할 수 있습니다.

< ndarray 클래스 실습 >

```
import numpy as np

arr = np.array([[1,2,3,4],[5,6,7,8]])
print(arr)
print(arr.shape)

print(arr.reshape(4,2))
```

Numpy 2차원 배열에서는 axis라는 개념을 사용합니다. axis는 좌표축과 같으며 아래 그림과 같이 행마다 처리하는 축이 axis=1, 열마다 처리하는 축이 axis=0으로 설정됩니다.



(그림 5)

< ndarray 클래스 실습 >

```
import numpy as np

arr = np.array([[1,2,3,4],[5,6,7,8]])

print(arr.sum())
print(arr.sum(axis=0))
print(arr.sum(axis=1))
```

ndarray 행렬에서 특정 순서로 행을 추출하기 위해 추출할 행의 순서를 나타내는 배열을 인덱스 참조로 전달할 수 있습니다. 이처럼 인덱스 참조로 인덱스 배열을 이용하는 방법을 팬시 인덱스 참조라 합니다.

< ndarray 클래스 실습 >

```
import numpy as np

arr = np.array([[1,2],[3,4],[5,6],[7,8]])
print(arr[[3, 0, 2]])
```

ndarray 행렬에서는 Numpy.transpose() 함수나 '.T'를 사용하여 행과 열을 바꾸는 전치가 가능합니다. 또한 리스트형과 마찬가지로 sort() 메서드로 요소를 정렬할 수 있습니다. Numpy.sort() 함수로 정렬할 경우 정렬된 배열의 인덱스를 반환합니다. argsort() 메서드는 정렬된 배열의 인덱스를 반환합니다.

< ndarray 클래스 실습 >

```
import numpy as np

arr = np.arange(12).reshape(2, 6)
print(arr)
print(np.transpose(arr))
print(arr.T)

arr = np.array([[1, 6, 9, 3], [7, 2, 4, 5]])
print(np.sort(arr))
print(arr.argsort())
arr.sort(0)
print(arr)
arr.sort(1)
print(arr)
```

(6) Pandas

Pandas(판다스)는 일반적인 데이터베이스에서 이뤄지는 작업을 수행할 수 있으며, 수치뿐만 아니라 이름과 주소 등 문자열 데이터도 쉽게 처리할 수 있습니다. Pandas에는 Series와 DataFrame의 두 가지 데이터 구조가 존재하며, 주로 사용되는 구조는 2차원 테이블로 나타내는 DataFrame입니다. 각 행과 열에는 라벨이 부여되어 있으며, 행의 라벨은 인덱스(index), 열의 라벨은 컬럼(column)이라 부릅니다. Series는 1차원 배열로 DataFrame의 행 또는 열로 볼 수 있으며, Series도 각 요소에 라벨이 붙어 있습니다.

< Series, DataFrame 실습 >

```
import numpy as np

arr = np.arange(12).reshape(2, 6)
print(arr)
print(np.transpose(arr))
print(arr.T)

arr = np.array([[1, 6, 9, 3], [7, 2, 4, 5]])
print(np.sort(arr))
print(arr.argsort())
arr.sort(0)
print(arr)
arr.sort(1)
print(arr)
```

DataFrame은 pd.DataFrame()에 Series를 전달하여 생성할 수도 있고, 딕셔너리형을 넣어 생성할 수도 있습니다.

< DataFrame 실습 >

```
import pandas as pd
index = ["서울", "부산", "전주", "익산"]
data1 = [2020, 2021, 2022, 2023]
data2 = [1, 3, 5, 7]
series1 = pd.Series(data1, index=index)
series2 = pd.Series(data2, index=index)

df = pd.DataFrame([series1, series2])
print(df)

df.index = [1, 2]
print(df)
```

DataFrame에는 'DataFrame형변수.append("Series형 데이터“, ignore_index=True)' 형식으로 새로운 행을 추가할 수 있습니다. 기존 DataFrame의 컬럼과 추가할 Series형 데이터의 인덱스가 일치하지 않으

면 새로운 컬럼으로 추가되고, 값이 존재하지 않는 요소는 NaN으로 채워집니다.
DataFrame에 'DataFrame형변수["새로운 컬럼명"]' 형식으로 Series 또는 리스트를 대입하여 열을 추가할 수 있습니다.

< DataFrame 실습 >

```
import pandas as pd
index = ["서울", "부산", "전주", "익산"]
data1 = [2020, 2021, 2022, 2023]
data2 = [1, 3, 5, 7]
series1 = pd.Series(data1, index=index)
series2 = pd.Series(data2, index=index)

df = pd.DataFrame([series1, series2])
print(df)

series3 = pd.Series(["특별시", "광역시", "시", "시", "시"], index=["서울", "부산", "전주", "익산", "군산
"])
df = df.append(series3, ignore_index=True)
print(df)

df["제주"] = [2025, 11, "시"]
print(df)
```

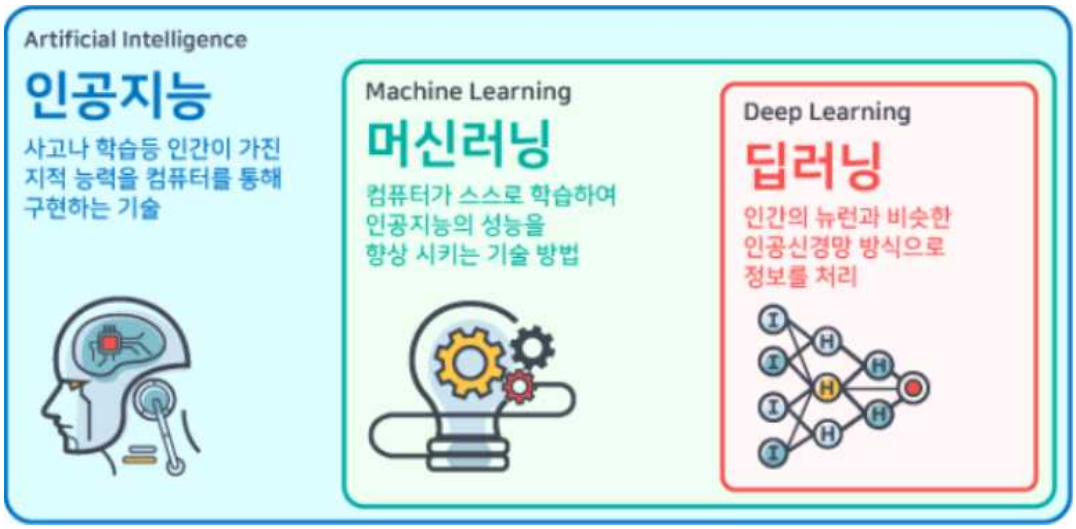
3. 딥러닝 모델 및 프레임워크 이해

3-1. 인공지능, 머신러닝, 딥러닝의 관계

인공지능(artificial intelligence)은 인간의 지능을 갖고 있는 기능을 갖춘 컴퓨터 시스템이며, 인간의 지능을 기계 등에 인공적으로 구현한 것입니다.

머신러닝(machine learning)은 클라우드 컴퓨터가 학습 모형을 기반으로 외부에서 주어진 데이터를 통해 스스로 학습하는 것을 말합니다. 빅데이터를 분석하고 가공해서 새로운 정보를 얻어 내거나 미래를 예측하는 기술을 말하며, 축적된 데이터를 토대로 상관관계와 특성을 찾아내고 결론을 도출합니다.

딥러닝(deep learning)은 컴퓨터가 여러 데이터를 이용해 마치 사람처럼 스스로 학습할 수 있게 인공신경망을 기반으로 구축한 기계 학습 기술이며, 축적된 데이터를 분석만 하지 않고 데이터를 학습까지 하는 기계 학습 능력을 활용하여 결론을 도출합니다.



(그림 6)

3-2. 머신러닝 학습 방식

지도 학습(Supervised Learning)은 데이터에 대한 학습데이터가 주어진 상태에서 컴퓨터를 학습시키는 방법입니다. 지도 학습의 예로, 영상 처리가 가장 대표적이라고 할 수 있습니다. 코끼리, 기린과 같은 학습데이터를 학습기에 넣어 이미지를 인식시켜 학습시켜야만 하기때문에 영상 처리에서는 학습데이터가 필수적입니다.

비지도 학습(Unsupervised Learning)은 데이터에 대한 학습데이터가 없는 상태에서 오직 입력데이터만 이용해서 컴퓨터를 학습시키는 방법입니다. 비지도 학습의 예로, 뉴스 기사 분류, DNA 분류, SNS 관계 분류 등 많은 분야에 응용됩니다. 입력된 데이터를 비슷한 그룹으로 묶어 예측하는 모델을 학습합니다. 비지도 학습은 예측이 목적이 아니라, 데이터의 구성 또는 특징을 밝히는 목적으로 사용되는 그룹핑 알고리즘입니다.

강화학습(Reinforcement Learning)은 행동에 대한 보상을 받으며 학습하여 어떤 환경 안에서 선택 가능한 행동들 중에 보상을 최대화하는 행동 또는 행동 순서를 선택하는 방법입니다. 강화 학습의 예로, 게임이 가장 대표적이라고 할 수 있습니다.

(그림 7)

3-3. 딥러닝 모델

딥러닝은 머신 러닝의 특정한 한 분야로서 연속된 층layer에서 점진적으로 의미 있는 표현을 배우는 데 강점이 있으며, 데이터로부터 표현을 학습하는 새로운 방식입니다.

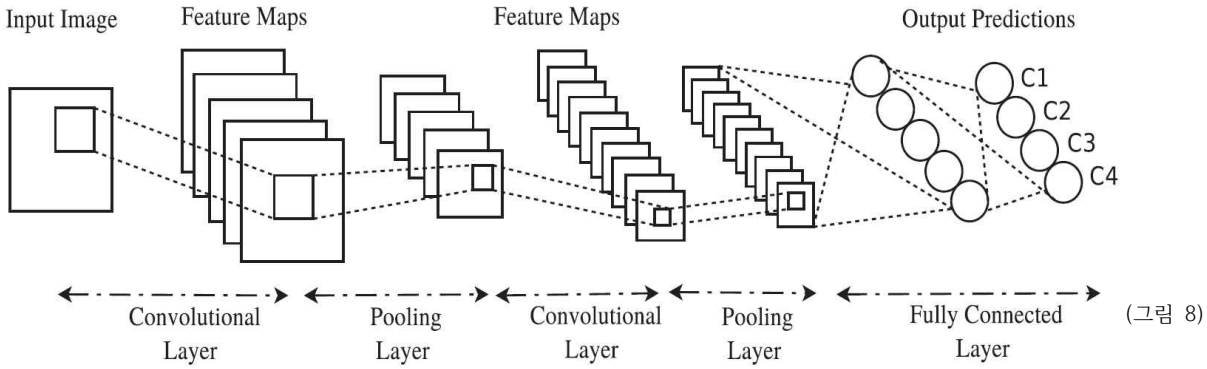
딥러닝의 딥deep이란 단어가 어떤 깊은 통찰을 얻을 수 있다는 것을 의미하지는 않습니다. 그냥 연속된 층으로 표현을 학습한다는 개념을 나타냅니다. 데이터로부터 모델을 만드는 데 얼마나 많은 층을 사용했는지가 그 모델의 깊이가 됩니다. 최근의 딥러닝 모델은 표현 학습을 위해 수십 개, 수백 개의 연속된 층을 가지고 있습니다. 이 층들을 모두 훈련 데이터에 노출해서 자동으로 학습시킵니다. 한편 다른 머신러닝 접근 방법은 1~2개의 데이터 표현 층을 학습하는 경향이 있습니다. 그래서 이런 방식을 얕은 학습(shallow learning)이라 부르기도 합니다

(1) 합성곱 신경망 모델

합성곱 신경망(CNN; Convolutional Neural Network)은 음성인식이나 이미지 인식에서 주로 사용되는 신경망의 하나로, 다차원 배열 데이터를 처리하도록 구성되어 있어, 컬러 이미지와 같은 다차원 배열 처리에 특화되어 있습니다.

일반적인 신경망이 이미지 데이터 전체를 하나의 데이터로 처리함으로써, 이미지의 위치가 조금 달라지거나 왜곡된 경우 성능이 저하되는 문제가 있습니다. 합성곱 신경망은 이미지를 하나의 데이터가 아닌 여러 개로 분할하여 처리함으로써, 이미자가 왜곡되더라도 이미지의 부분적 특성을 추출할 수 있어 성능을 유지할 수 있습니다.

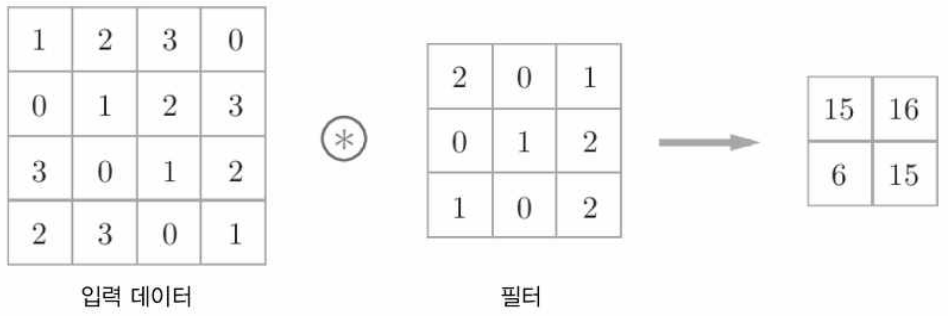
합성곱 신경망은 다음 그림과 같이 입력층, 합성곱층, 풀링층, 완전연결층, 출력층의 다섯 계층으로 구성되어 있습니다.



(그림 8)

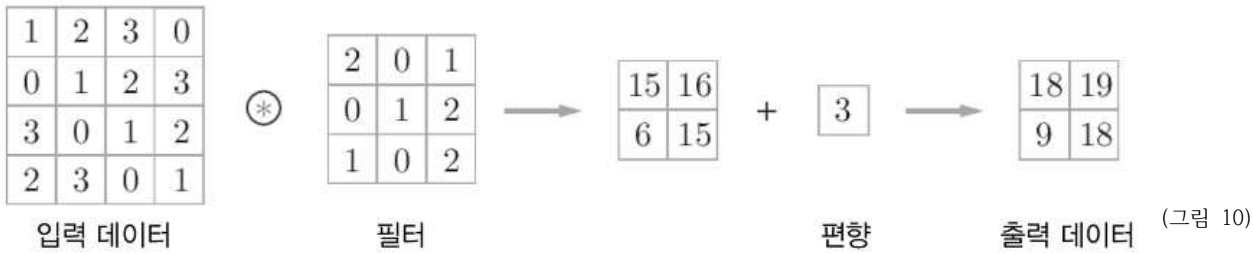
입력층(input layer)는 입력 이미지 데이터가 최초로 거치게 되는 계층입니다. 이미지는 높이, 너비, 채널의 값을 갖는 3차원 데이터로, 이미지가 그레이 스케일이면 채널은 1, 컬러이면 3 값을 가집니다.

합성곱층(convolutional layer)는 입력 데이터에서 특성을 추출하는 역할을 수행합니다. 입력 이미지에 대한 특성을 추출하기 위해 필터(또는 커널; kernel)을 이용합니다. 이미지의 모든 영역을 필터를 적용하여 특성을 추출한 결과물이 특성 맵(feature map)이 됩니다. 필터의 크기는 3x3, 5x5가 주로 사용되며, 필터를 적용하는 위치 간격인 스트라이드(stride)는 1 또는 다른 값을 적용할 수 있습니다. 아래 그림의 예는 3x3 필터에 스트라이드를 1로 주어 합성곱을 구한 것입니다. 컬러 이미지의 경우 RGB 각각에 서로 다른 필터를 적용하여 합성곱을 구한 후 결과를 더해 줍니다.



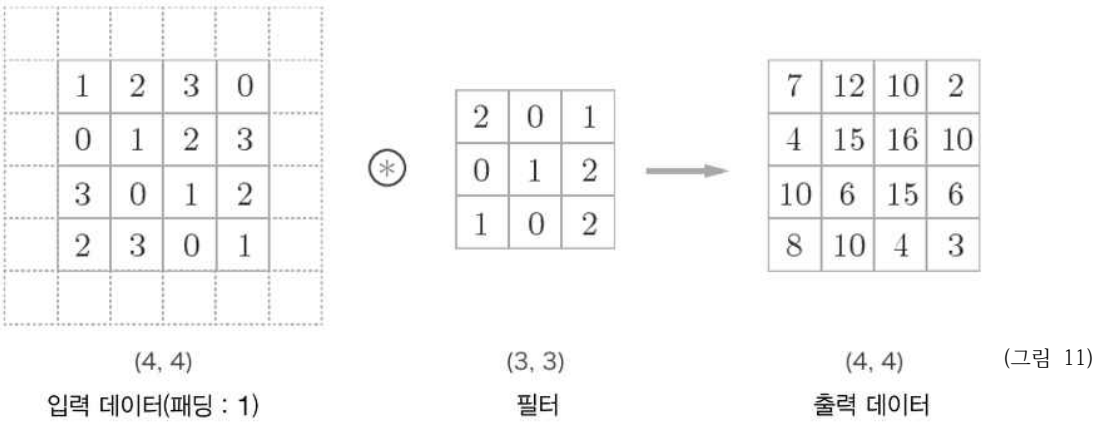
(그림 9)

완전연결 신경망에서는 가중치(weight) 매갭변수와 편향(bias)가 존재하는데, 합성곱 신경망에서는 필터의 매개변수가 완전연결 신경망의 가중치에 해당하며, 합성공 신경망을 통해 학습이 반복되면서 필터의 요소값이 매번 갱신됩니다. 편향은 아래 그림과 같이 필터를 적용한 후의 데이터에 더해지고 항상 하나만 존재합니다.



(그림 10)

다음 그림과 같이 출력의 크기를 조절하기 위하여 입력 데이터의 주변을 특정값(0 또는 1)으로 채우는 것을 패딩(padding)이라 합니다.

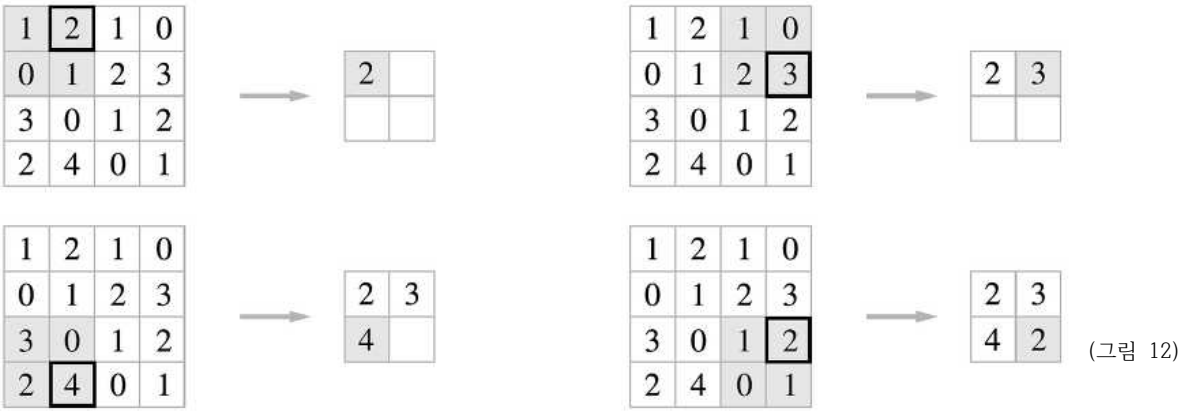


(그림 11)

풀링(Pooling)은 2차원 데이터의 가로 및 세로 방향의 공간을 줄이는 연산으로, 대상 영역의 최대값을 취

하는 최대 풀링(max pooling), 대상 영역의 평균을 취하는 평균 풀링(average pooling) 등이 있습니다. 다음 그림은 2x2 최대 풀링을 스트라이드 2로 처리하는 예입니다.

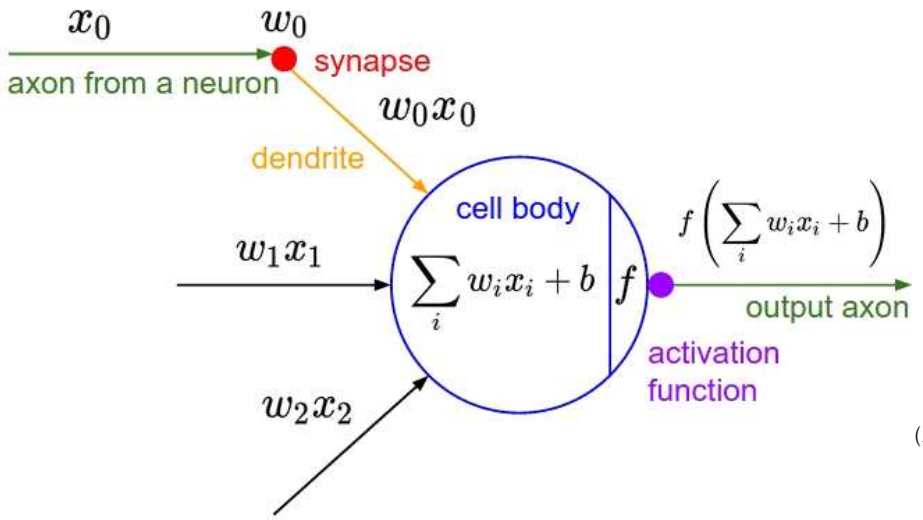
풀링 연산은 2차원 데이터의 크기를 줄이는 연산으로, 입력 데이터의 채널 수는 그대로 출력 데이터로 보내므로, 채널별로 계산을 수행합니다. 입력 데이터가 조금 변하더라도 풀링 계층에서 그 변화를 흡수하여 사라지게 하여 입력의 변화에 대한 영향을 줄여 줍니다.



(그림 12)

완전 연결 계층(Fully-connected layer)는 입력 데이터와 가중치가 1대 1로 대응하는 계층으로, 행렬의 내적(Affine)으로 표현되는 계층입니다. 완전 연결 계층의 문제점은 데이터의 형상이 무시된다는 것입니다. 입력 데이터가 다차원 형상일 때, 완전 연결 계층에 입력해주기 위해서는 이 다차원 데이터를 1차원으로 평탄화(Flattening)해 준 후에 입력해야만 합니다.

아래 그림과 같이 각 노드에 가중치를 곱하고 편향을 더한 값을 다음 계층으로 전달할 때 개입하는 활성화 함수(Activation function)는 그 값이 특정 조건을 만족하면 활성화 함수를 거친 출력값을 다음 노드로 보내 활성화하고, 그렇지 않으면 해당 노드를 비활성화한다.

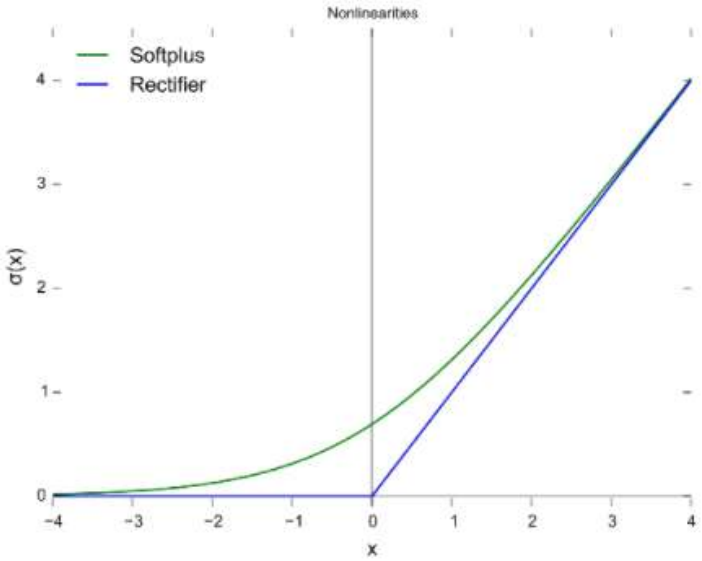


(그림 13)

활성화 함수에 따라 출력값이 달라지므로 적절한 활성화 함수를 선택하는 것이 중요하며, 다양한 활성화 함수 중에 합성곱 신경망에서 주로 사용되는 ReLU 함수오 Softmax 활성화 함수를 살펴 보겠습니다.

ReLU(Rectified Linear Unit) 함수는 입력값이 음수이면 0으로 비활성화하고, 입력값이 양수이면 해당 값을 그대로 출력하는 활성화 함수로, sigmoid에 비해 소실되는 기울기가 적어 학습에 용이하며, 비선형성을 이용해 사전훈련 없이도 신경망 훈련이 가능하며, 복잡한 데이터셋에서 상대적으로 빠르고 효과적인

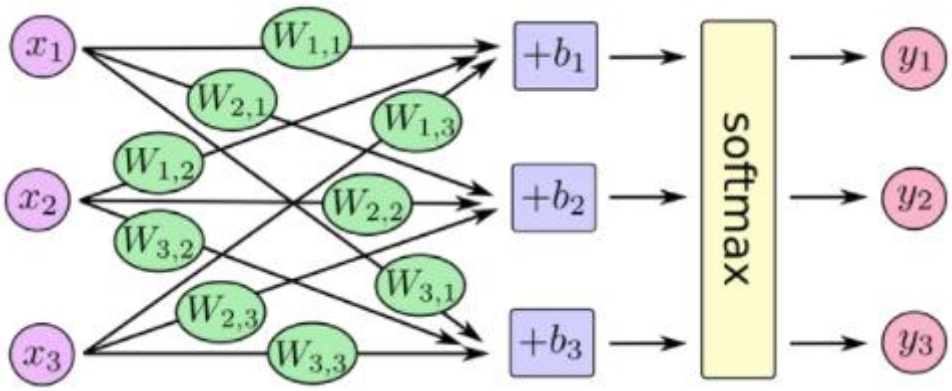
훈련이 가능해, 주로 컴퓨터 비전, 음성인식 분야에 사용됩니다.



(그림 14)

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

소프트맥스(Softmax) 함수는 다중 분류 문제 해결에 적용할 수 있는 함수로, 실수값을 가진 K개의 벡터를 입력하면, 각각의 요소들을 지수함수를 통해 정규화하여 0과 1 사이 값으로 출력됩니다. 다음 그림과 같이 소프트맥스 함수가 3개의 요소를 입력 받으면 각각의 요소가 정규화되어 y1, y2, y3로 출력됩니다. 이 때 3개의 확률값의 합은 1이 되도록 합니다. 소프트맥스 함수는 지수함수를 통한 정규화를 통하여 최대값을 보다 극대로 확장시켜 다른 요소들과 차이를 증폭시켜 출력해 줍니다.



(그림 15)

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

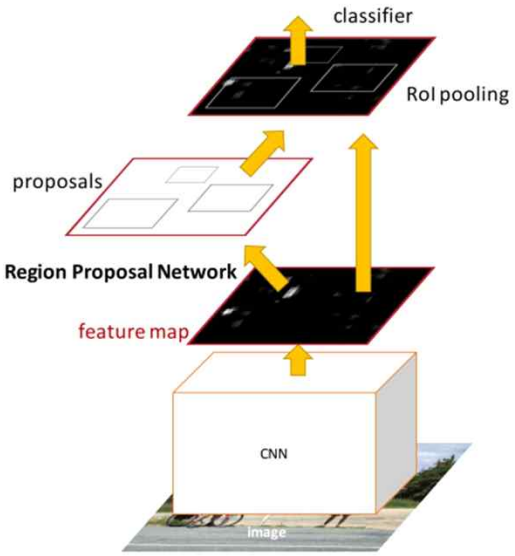
4. YOLO V5 환경 세팅 및 모델 아키텍쳐 분석

YOLO (You Only Look Once)는 '한 번에 보고 바로 처리를 하겠다'라는 처리 속도가 빠르다는 장점을 내세운 객체 탐지(Object Detection) 신경망 모델로, 객체 탐지를 위한 빠르고 가벼운 모델입니다.

첫 번째 버전은 2016 년에 출시되었으며 24개의 컨볼루션 레이어와 2 개의 완전연결 레이어 아키텍처가 뒤 따르고, 4 개의 다른 버전을 통해 개선되었습니다. YOLOv5는 Ultralitycs에 의해 오픈 소스가되었으며 전체 프로젝트는 Github에 공유되어 있습니다.

4-1. Faster R-CNN (Region with Convolutional Neural Nerwork) 구조

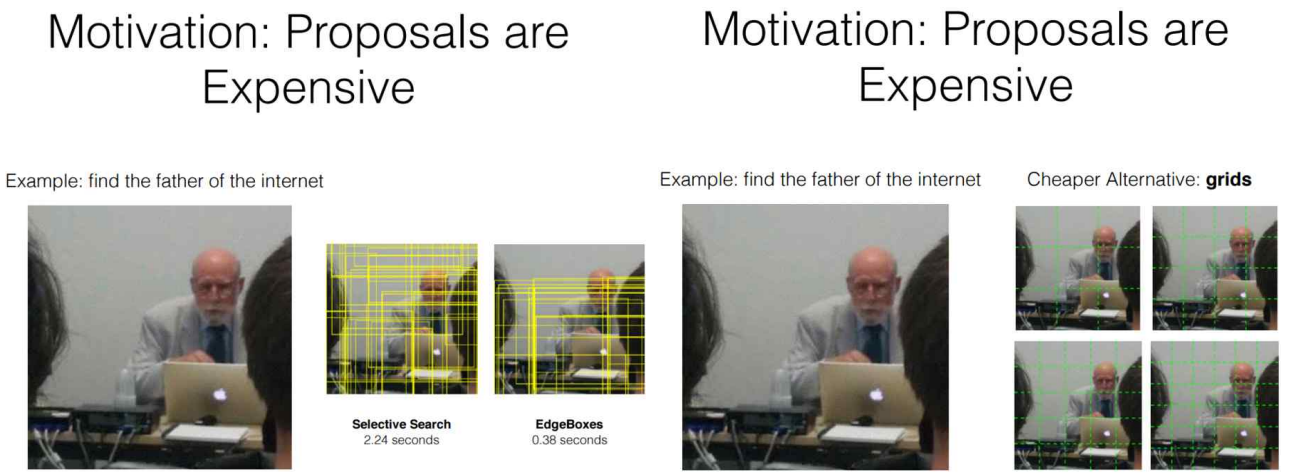
Region Proposal Network 가 영상에서 오브젝트가 있을 것 같은 영역(Region)들을 뽑아서 (여기에 어떤 클래스가 있는지 확인해보라고) 제안(proposal)하는 구조입니다. 학습 과정에서는 2,000개, 테스트 과정에서는 300개 정도의 후보 영역을 뽑기 때문에 느립니다.



(그림 16)

< Faster R-CNN 구조 >

기존의 R-CNN이 느린 이유는 Proposal 수도 많고, 그 과정에서의 오버헤드도 크기 때문입니다. 아래 그림은 영상에서 객체의 경계박스를 찾는 두 가지 방법인 Proposal 방식과 Grid 방식을 비교한 것입니다. Grid 방식은 grid cell의 개수가 곧 proposal의 수로 proposal을 구하는 과정에서 오버헤드가 전혀 없습니다. Grid cell에 객체가 있다는 보장은 없지만 그것은 다른 방식도 마찬가지이며, Yolo는 실시간성을 확보하기 위해 proposal 수가 적은 grid 방식을 발전시켜 왔습니다.



(그림 17)

4-2. Yolo 네트워크

Yolo는 네트워크의 최종 출력단에서 경계박스의 위치 찾기와 클래스 분류가 동시에 수행됩니다. 즉, 단 하나의 네트워크가 한 번에 특징도 추출하고, 경계박스도 만들고, 클래스도 분류하므로, 간단하고 빠릅니다.

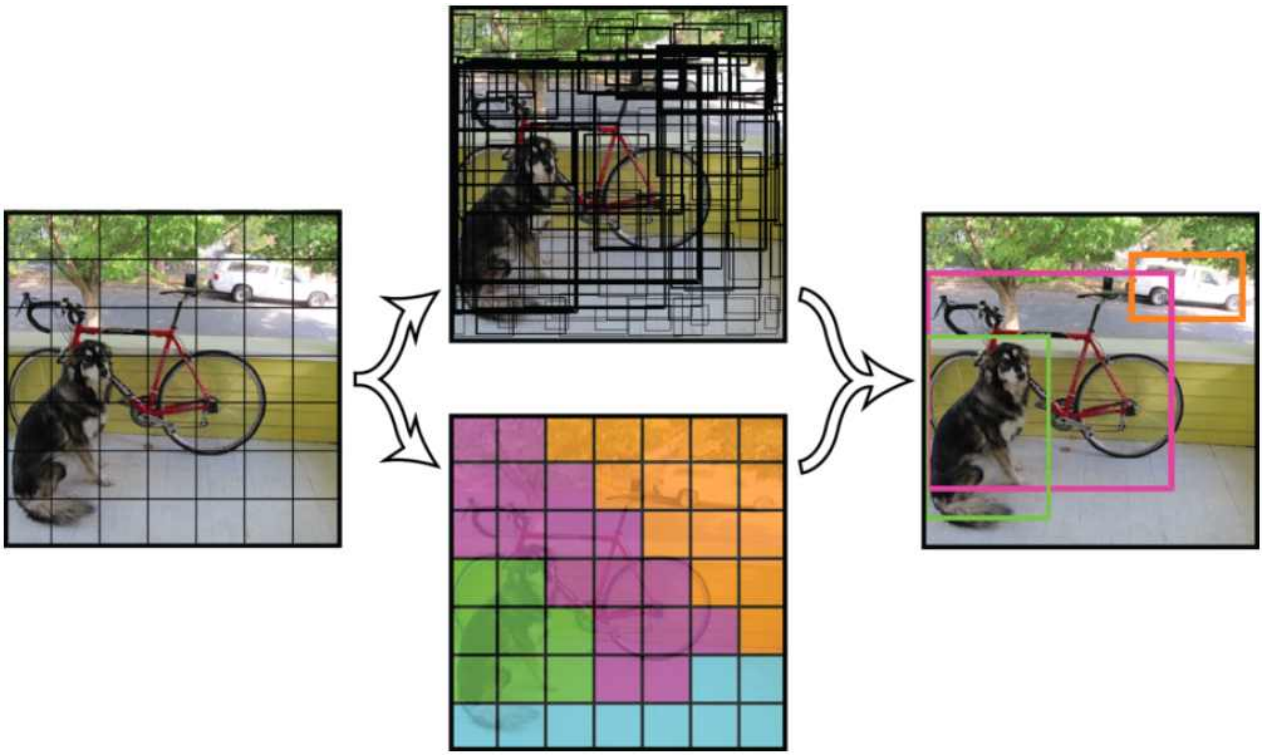
Yolo는 아래 좌측의 입력 영상이 네트워크를 통과하면 중앙의 2개 데이터를 얻게 되며, 이것이 네트워크의 최종 출력입니다. 이 안에는 수많은 경계박스들과 영상을 7x7 grid로 나눴을 때, 해당 grid cell 안에는 어떤 클래스가 있는지에 대한 정보가 인코딩 되어 있습니다.

가운데 위쪽은 경계 박스에 대한 정보로, 서로 다른 크기의 상당히 많은 경계박스들이 그려져 있어 복잡해 보입니다. 네트워크는 영상을 7x7 grid로 나누며, 각 grid에서 중심을 grid 안쪽으로 하면서 크기가 일정하지 않은 경계박스를 2개씩 생성합니다. Grid cell이 7x7 = 49 개이므로, 경계박스는 총 98개가 만들어 집니다.

경계박스 안쪽에 어떤 객체가 있을 것 같다고 확신(confidence score)할수록 박스를 굵게 그려줍니다. 굵은 경계박스들만 남기고 얇은 경계박스(어떤 객체도 없는 것 같다고 생각되는 것들)을 지웁니다. 남은 후보 경계 박스들을 NMS(Nom-maximal suppression 비-최대값 억제) 알고리즘를 이용해 선별하면 우측의 이미지처럼 3개만 남게 됩니다.

경계박스의 색깔은 클래스를 의미합니다. 중앙의 아래쪽에 이미지가 7x7 의 grid로 분할되어 총 49개의 grid cell이 있는데, 각 grid cell은 해당 영역에서 제안한(proposal) 경계박스 안의 객체가 어떤 클래스인지를 색깔로 표현하고 있습니다. 그러므로 최종적으로 남은 3개의 경계박스 안에 어떠한 클래스가 있는지 알 수 있어, 우측의 최종 결과를 얻게 됩니다.

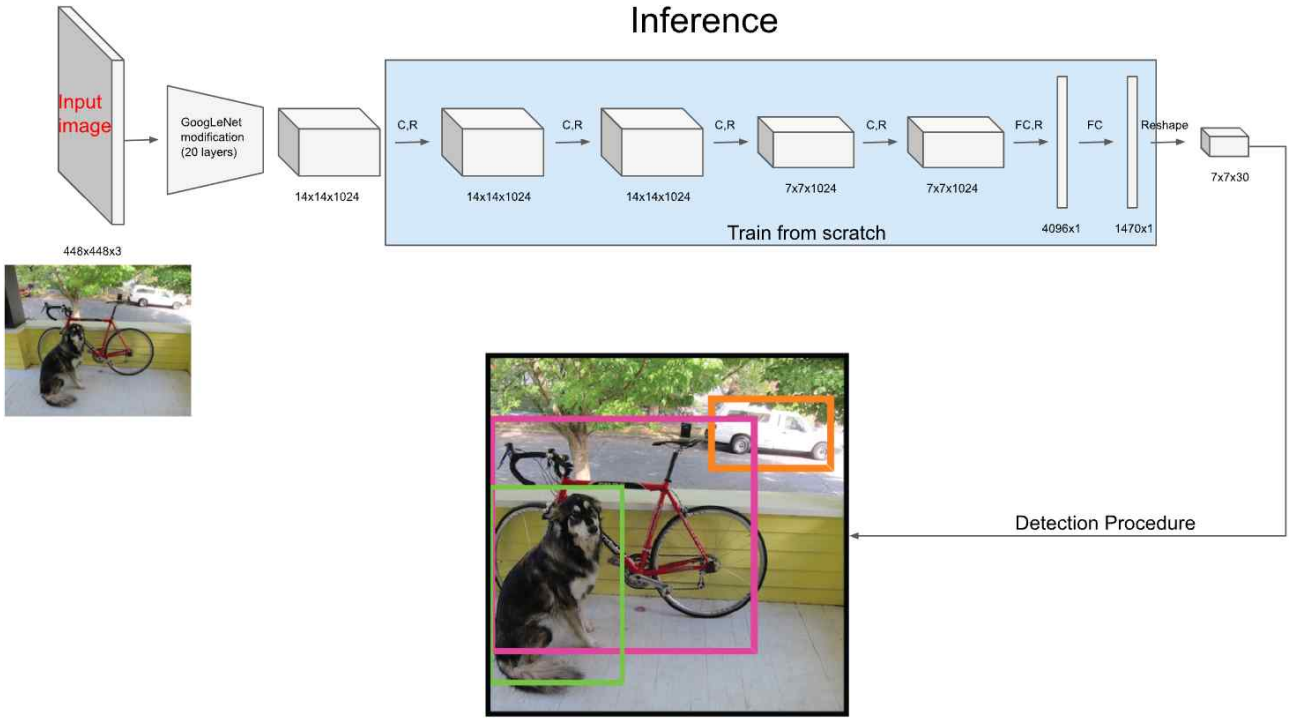
네트워크가 생성하는 경계박스의 숫자는 grid cell의 2배로, grid cell이 49개이므로 경계박스는 총 98개가 만들어 집니다. ROI 혹은 객체 후보라고 할 수 있는 이 경계박스들은 문턱값(threshold) 보다 작으면 지워 줍니다.



(그림 18)

4-3. Yolo version 1 네트워크 구조

네트워크 구조는 직선적으로, GoogleLeNet을 약간 변형시켜 특징 추출기로 사용했습니다. 그 다음, 컨볼루션 레이어 4회, 풀 커넥션 레이어 2회를 하고 사이즈를 7x7x30으로 조정하여 끝납니다. 이 마지막 특징 데이터 7x7x30이 바로 예측 결과이며, 이 안에 경계박스와 클래스 정보 등 모든 것이 들어 있습니다.



(그림 19)

4-4. Yolo 버전

Joseph Redmon의 YOLO 버전
버전 1: 통합 된 실시간 객체 감지(2016)
버전 2: 더 좋고, 더 빠르고, 더 강력함(2017)
버전 3: 점진적 개선(2018)
Joseph Redmon의 컴퓨터 비전 연구 중단 발표(2020.2)

Alexey Bochkovskiy의 YOLO 버전
버전 4: Darknet 기반 Tesla V100에서 65FPS의 실시간 속도. COCO 데이터 세트에서 43.5%의 AP값 획득

Glenn Jocher의 YOLO 버전(2020.5)
버전 5: Darknet의 포크가 아닌, PyTorch 구현. 모자이크 데이터 확대 및 자동 학습 경계 상자 앵커가 포함됨. Tesla P100에서, 이미지 당 0.007 초 객체 예측. 평균 140 FPS 주장.

4.5 Yolov5 사용 환경 구축

1) Yolov5 설치하기
Github repository에 있는 Yolov5를 git clone 명령으로 복사해서 설치합니다.

```
git clone https://github.com/ultralytics/yolov5.git
```

2) PyTorch 설치하기

아나콘다 가상환경을 만들어 주고 CUDA 버전에 맞는 PyTorch를 설치합니다.

```
!conda create -n yolov5 python=3.8
!conda activate yolov5
!conda install pytorch==1.7.1 torchvision==0.8.2 torchaudio==0.7.2 cudatoolkit=11.0 -c pythorch
```

3) Yolov5에 필요한 나머지 모듈들 설치하기

```
%cd yolov5
!pip install -r requirements.txt
```

4) 데이터 폴더 설정하기

data 폴더 안에 images와 labels 폴더를 만들고, 그 안에 train, valid 폴더를 만듭니다.
이러한 폴더 구조는 사용자가 임의로 변경하여 설정할 수 있습니다.

```
|--yolov5
|--data
    |--images
        |--train
        |--valid
    |--labels
        |--train
        |--valid
```

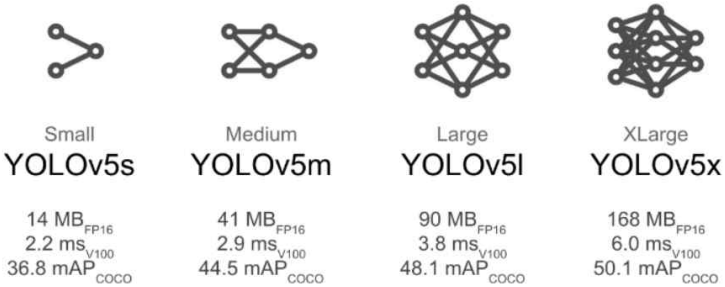
5) Dataset yaml 파일 만들기

아래 예의 dataset 관련 내용이 들어 있는 적당한 이름의 yaml 파일을 만든다. yaml 파일 안에는 ① training data 경로, ②validation data 경로, ③탐지할 class 개수, ④class 이름 리스트가 들어갑니다.

```
train: /data/images/train
val: /data/images/valid

nc: 1
names: ['wheat']
```

6) Yolov5 모델 선택하기

Yolov5 폴더 내의 model 폴더 안에는 yolov5s.yaml, yolov5l.yaml, yolov5m.yaml, yolov5x.yaml 파일이 들어 있는데, 이 파일들은 각각 Yolov5의 4가지 버전(s, m, l, x)의 모델 파일들입니다. s 버전은 성능이 제일 낮지만 FPS(frames per second)가 가장 높고, x 버전이 성능이 제일 높지만 FPS는 가장 낮습니다.



(그림 20)

FPS: Object detection 성능 지표의 하나로, 초당 탐지하는 frame 수를 의미합니다. FPS가 높을수록 영상 처리 속도가 빨라집니다. 실시간 영상처리를 위해서는 보통 30FPS가 요구됩니다.

7) Model yaml 파일 변경하기

필요에 따라 선택한 모델의 yaml 파일을 열어 nc(number of classes) 값을 변경해줍니다.

```
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

# Parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

8) 학습하기

다음과 같이 train.py 파일을 실행하여 학습을 실행합니다.

```
!python train.py --img 640 --batch 32 --epochs 50 --data '/data/dataset.yaml' --cfg ./models/yolov5s.yaml --weights yolov5s.pt --name test
```

yolov5 폴더 내의 runs/train 폴더에 exp2, exp3,... 형식의 exp 폴더가 생성되어 학습 결과가 저장됩니다.

인자값 설명

--img: input image size

--batch: batch size

--epochs: epoch 수

--data: dataset 관련 yaml 파일

--cfg: yolov5 모델 yaml 파일

--weights: 전이학습 사용시에 적용한 weight 파일로 --cfg에서 설정한 모델에 맞게 설정

--device: GPU/CPU

--name: 학습정보 저장에 사용할 파일 이름

--resume: 학습이 어떤 이유로 중간에 멈춘 경우에 가장 최근에 저장된 weight로부터 학습을 계속할 때 사용 (예; python train.py --resum runs/train/test/weight/last.pt), epoch도 중단 지점부터 다시 학습됨

5. YoloV5를 활용한 실시간 탐지 실습

5-1. colab 사용하기

구글 colalb(코랩)은 구글 colaboratory 서비스의 줄임말로 다음과 같은 특징이 있습니다.
- 별도의 파이썬 설치과정 없이 브라우저에서 python을 작성하고 실행 가능합니다.
- 데이터 분석에 사용되는 Tensorflow, Keras, matplotlib, scikit-learn, pandas 등과 같은 패키지가 기본적으로 설치되어 있습니다.
- Git과 연도이 가능하며, GPU를 무료로 사용할 수 있습니다.
- 구글 코랩은 클라우드 기반의 주피터 노트북 개발환경으로 주피터 노트북보다 좋은 기능을 제공합니다.
- 코랩은 구글 드라이브, 도커, 리눅스 , 구글 클라우드 등 기술로 이루어져 있습니다.

colab을 사용하기 위해서는 먼저, 구글에 로그인을 한 후, 웹브라우저로 아래 URL로 colab에 접속합니다.

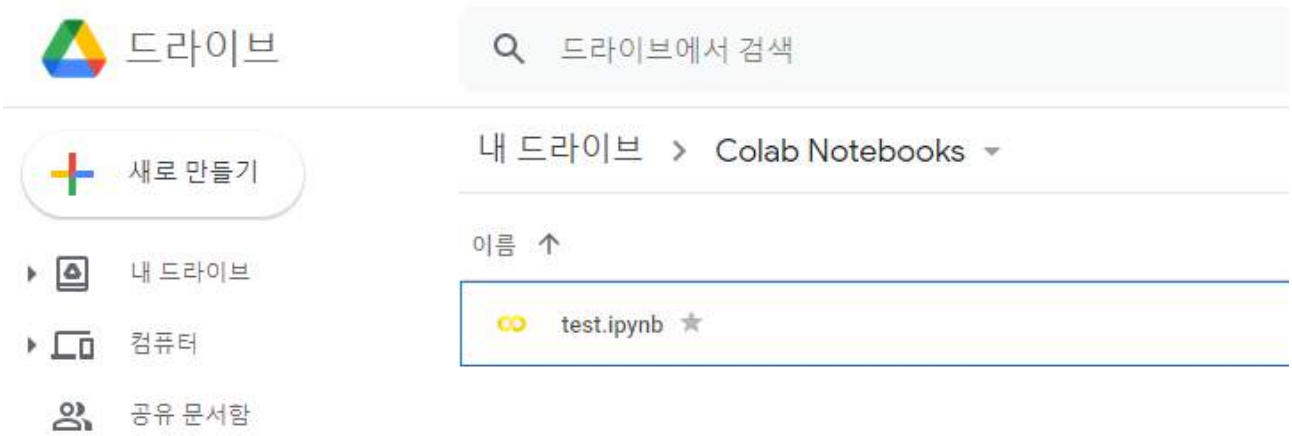
```
https://colab.research.google.com/notebooks/intro.ipynb
```

아래와 같은 colab 메인 페이지가 나타나며, 시작하기 아래의 화살표로 표시된 칸에 파이썬 코드를 입력하여 실행할 수 있습니다.

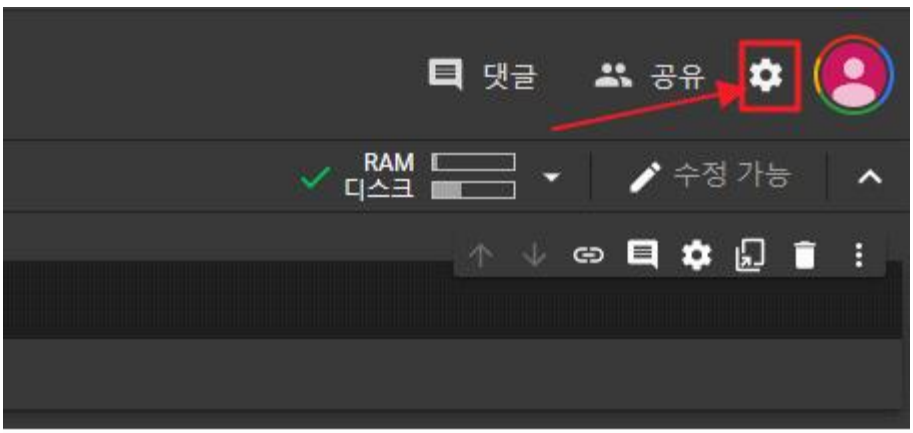
새로운 파일에서 파이썬 코드를 작성하기 위해서는 왼쪽 상단 파일 메뉴의 새노트 항목을 클릭합니다.

아래와 같이 새로운 코드를 입력할 수 있는 화면이 보이며, 이제 코드를 입력하여 실행할 수 있습니다.

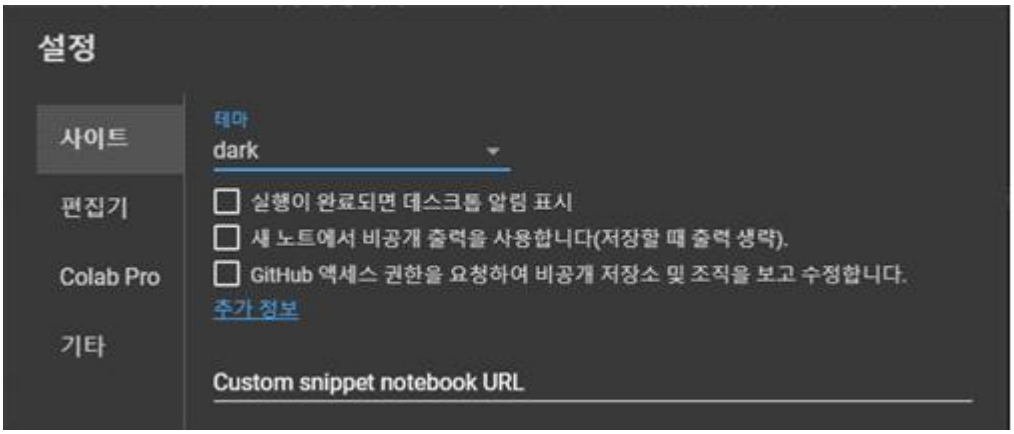
코드를 작성한 후, <Ctrl>+s 또는 파일 메뉴의 저장 항목을 클릭하면 구글 드라이브에 아래와 같이 Colab Notebooks 폴더 안에 파일이 저장됩니다.



colab 접속 화면에서 아래 그림과 같이 우측 상단의 설정열기 아이콘을 누르면 colab에 대한 설정을 할 수 있습니다.



설정하기에서 사이트, 편집기, Colab Pro, 기타에 대해 설정이 가능합니다.

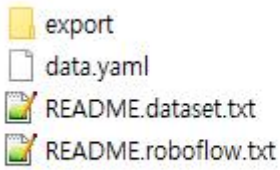


5-2. 학습 및 테스트용 데이터 가져오기

roboflow 사이트에서 실습에 사용할 데이터 파일을 가져와 압축을 해제합니다.

```
!curl -L "https://public.roboflow.ai/ds/WKkUorQ71T?key=wIBAdyawPa" > roboflow.zip; unzip roboflow.zip
```

압축 파일을 삭제하고 싶다면, rm roboflow..zip을 명령줄에 추가합니다.
content 폴더 안에 dataset 폴더를 만든 후, 다음과 같은 압축 해제 결과 폴더 및 파일들을 dataset 폴더 안으로 이동시킵니다. export 폴더 안에는 images와 labels 폴더가 들어 있습니다.



/content 폴더로 이동한 후, Yolov5를 설치합니다.

```
%cd /content
!git clone https://github.com/ultralytics/yolov5.git
```

yolov5 폴더로 이동한 후, 추가로 필요한 모듈들을 설치합니다.

```
%cd yolov5
!pip install -r requirements.txt
```

데이터셋 관련 설정이 들어 있는 data.yaml 파일의 내용을 cat 명령어로 확인합니다.

```
%cat /content/dataset/data.yaml

train:  /train/images
val:  /valid/images

nc:  1
names:  ['pistol']
```

train과 val의 설정 값이 dataset 폴더 안에 있는 images 폴더의 경로와 다른 것을 확인할 수 있으며, 추후에 수정을 할 것입니다.

루트로 이동 후에 glob 명령으로 학습 및 평가에 사용할 이미지 데이터 파일 리스트를 만듭니다.

```
%cd /
from glob import glob
img_list = glob('/content/dataset/export/images/*.jpg')
print(len(img_list))
```

전체 이미지 파일 리스트를 학습용과 평가용으로 분할한 이미지 파일 리스트를 만듭니다.

```
from sklearn.model_selection import train_test_split
train_img_list, val_img_list = train_test_split(img_list, test_size=0.2, random_state=100)
print(len(train_img_list), len(val_img_list))
```

분할한 이미지 파일 리스트의 내용을 텍스트 파일에 저장합니다.

```
with open('./content/dataset/train.txt', 'w') as f:
  f.write('\n'.join(train_img_list) + '\n')
with open('./content/dataset/val.txt', 'w') as f:
  f.write('\n'.join(val_img_list) + '\n')
```

데이터셋 관련 설정 파일을 열어 현재 설정에 맞게 내용을 수정합니다.

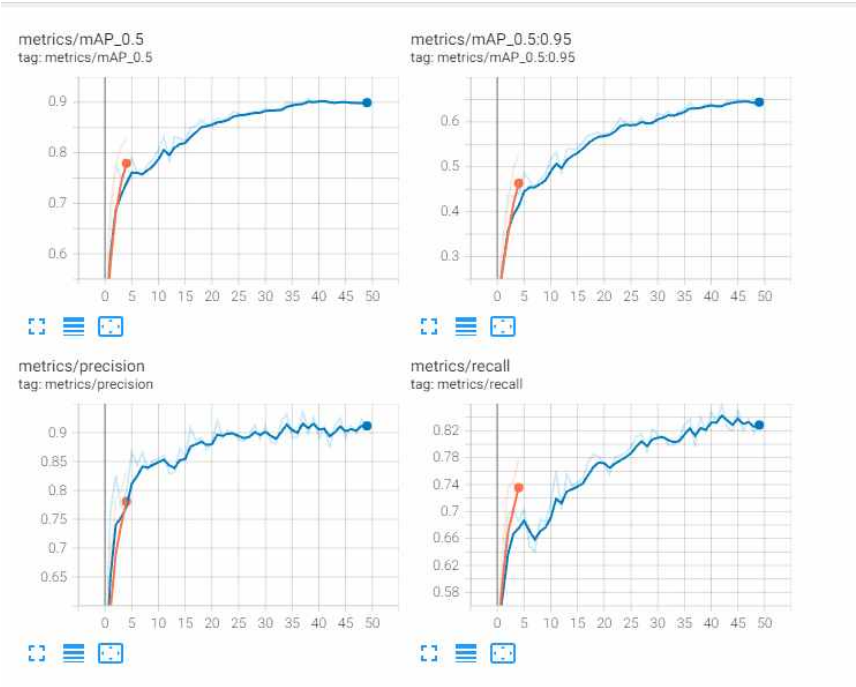
```
import yaml

with open('/content/dataset/data.yaml', 'r') as f:
  data = yaml.load(f, Loader=yaml.FullLoader)

print(data)

data['train'] = '/content/dataset/train.txt'
data['val'] = '/content/dataset/val.txt'

with open('/content/dataset/data.yaml', 'w') as f:
  yaml.dump(data, f)

print(data)
```

yolov5 폴더로 이동하여 학습을 실행합니다.

```
%cd /content/yolov5

!python train.py --img 416 --batch 16 --epochs 50 --data /content/dataset/data.yaml
--cfg ./models/yolov5s.yaml --weights yolov5s.pt --name gun_yolov5s_results
```

학습 실행 결과를 yolov5 폴더 내의 runs 폴더 안에서 확인할 수 있으며, 아래와 같이 tensorboard를 활용해서 확인할 수 있습니다.

```
%load_ext tensorboard
%tensorboard --logdir /content/yolov5/runs/
```



< Tensorboard 실행 결과 예 >

학습한 결과 모델을 사용하여 새로운 이미지에 대한 객체 탐지를 아래와 같이 할 수 있습니다.

```
from IPython.display import Image
import os

val_img_path = val_img_list[10]

!python detect.py --weights /content/yolov5/runs/train/gun_yolov5s_results/weights/best.pt
--img 416 --conf 0.5 --source "{val_img_path}"

Image(os.path.join('/content/yolov5/runs/detect/exp7', os.path.basename(val_img_path)))
```

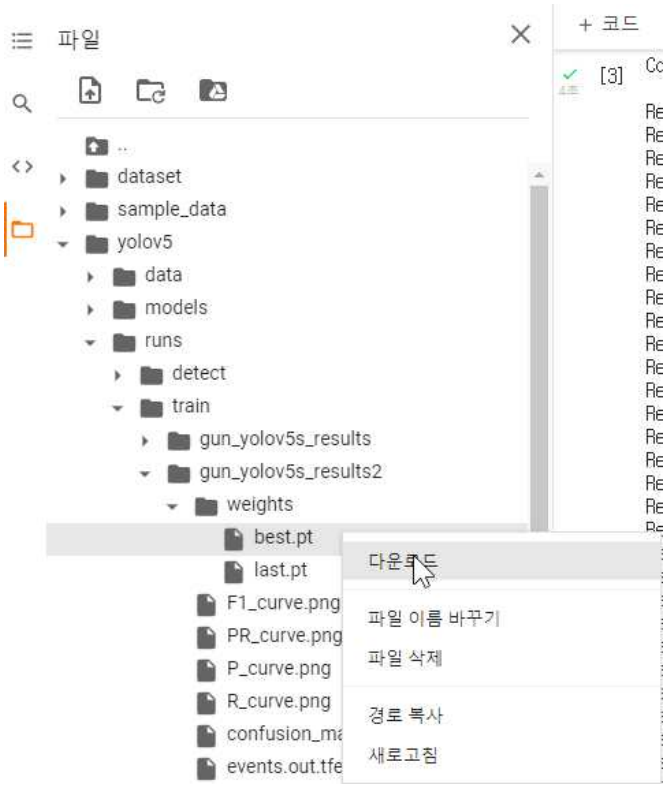
객체 탐지 실행 결과는 다음과 같습니다.

6. 실시간 검출 실습

실시간 검출을 확인해 보기 위해서 로컬 컴퓨터에 Git, Anaconda, Python, Yolov5를 설치합니다.

Git URL: https://git-scm.com/
아나콘다 URL: https://www.anaconda.com/products/individual
Python: 아나콘다 설치할 때 같이 설치합니다.
Yolov5: git clone https://github.com/ultralytics/yolov5.git

다음으로 실습에 적용하기 위해 colab을 통해 yolov5로 생성한 모델의 weights 파일을 내려 받아 yolov5 폴더에 넣습니다.



실시간 검출을 확인해 보기 위해, 인터넷에서 적당한 권총(pistol) 관련 MP4 형식의 동영상을 찾아 내려받기를 합니다.
내려받은 동영상 파일(예: pistol_test.mp4)을 yolov5 폴더로 이동시키고, 아래 명령을 수행하여 실시간 검출을 실행합니다.

```
from IPython.display import Image
import os

val_img_path = val_img_list[10]

!python detect.py --weights /content/yolov5/runs/train/gun_yolov5s_results/weights/best.pt
--img 416 --conf 0.5 --source "{val_img_path}"

Image(os.path.join('/content/yolov5/runs/detect/exp7', os.path.basename(val_img_path)))
```

결과 동영상 검출 결과를 아래와 같이 확인할 수 있으며, 결과 동영상 파일도 확인할 수 있습니다.

```
Speed: 1.9ms pre-process, 188.2ms inference, 0.8ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs₩detect₩exp4
```

<참조 그림>

(그림 1) https://kau-deeperent.tistory.com/104

(그림 2) https://ususok.tistory.com/294

(그림 3) 파이썬으로 배우는 딥러닝 교과서, 한빛미디어

(그림 4) 파이썬으로 배우는 딥러닝 교과서, 한빛미디어

(그림 5) https://suhwanc.tistory.com/86

(그림 6) https://hyeonjiwon.github.io/machine%20learning/ML-1/

(그림 7) https://m.blog.naver.com/k0sm0s1/221863569856

(그림 8) http://journal.auric.kr/kiee/XmlViewer/f387531

(그림 9~12) https://kolikim.tistory.com/53?category=733477

(그림 13) https://cs231n.github.io/neural-networks-1/

(그림 14,15) https://dsbook.tistory.com/59

(그림 16~19) https://blog.naver.com/sogangori/220993971883

(그림 20) https://lynnshin.tistory.com/47