# Pytorch의 전처리와 증강

```
import torch
from torchvision.transforms import v2

transform = v2.Compose([
    v2.ToTensor(),  # 리스트 → Tensor (C,H,W 형태로 자동 변환)
    v2.ConvertImageDtype(torch.float32),  # float32 변환
    v2.Normalize(mean=[0.0,0.0,0.0], std=[255.0,255.0,255.0])  # 0~255 → 0~1 스케일링
])
```

부성순강사(bakpak@empas.com)

# 이미지 전처리함수– 이미지처리 이해 (이미지는 0~255, 0~1 사이값이 같은 색상임)

```
1  import numpy as np
2  data= [
3          [[0,0,0],[0,0,255]],
4          [[255,0,0],[255,255,255]]
5        ]
6  data=np.array(data)
7  print(data.shape)
8
9
10 data_max1=data/255
11 data_max1
```

✓ 0.0s

```
(2, 2, 3)
```

1

```
array([[[0., 0., 0.],
        [0., 0., 1.]],

       [[1., 0., 0.],
        [1., 1., 1.]]])
```

2

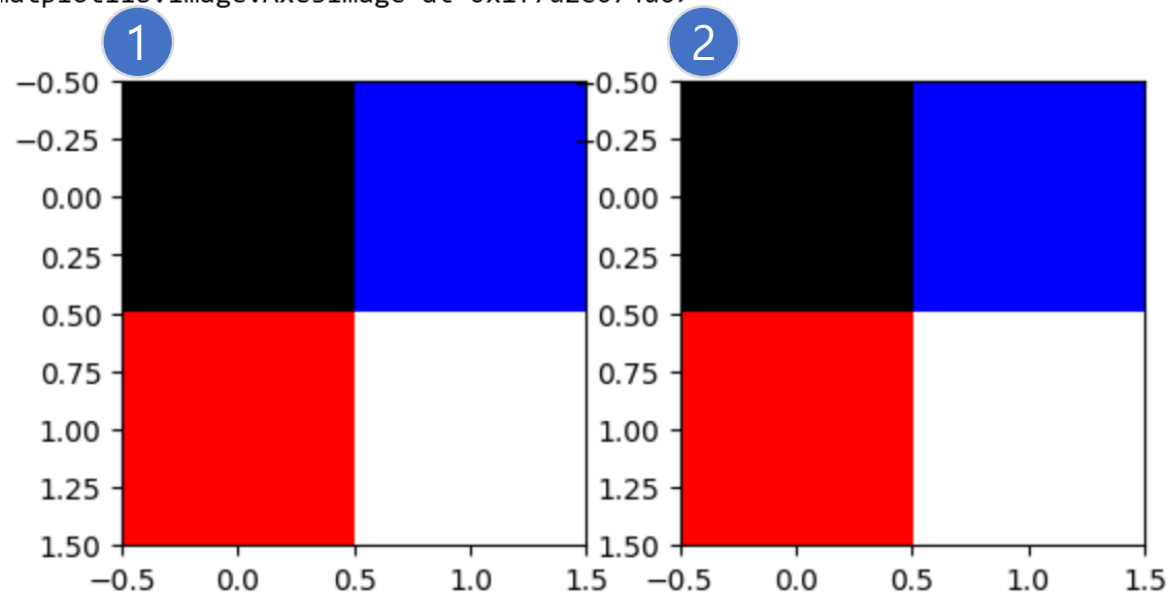
| 0,0,0 | 0,0,1 |
|---|---|
| 1,0,0 | 1,1,1 |

```
1  import matplotlib.pyplot as plt
2  plt.subplot(1,2,1)
3  plt.imshow(data)
4
5  plt.subplot(1,2,2)
6  plt.imshow(data_max1)
```

1 2

✓ 0.0s

```
matplotlib.image.AxesImage at 0x1f7d2e074d0>
```




부성순 bakpak@empas.com

# 이미지 전처리함수– 이미지처리 이해 (이미지는 0~255, 0~1 사이값이 같은 색상임)

```python
########################################
## 이미지 데이터를 받아서 /255를 하는 함수
########################################
data= [
        [[0,0,0],[0,0,255]],
        [[255,0,0],[255,255,255]]
       ]


def Matrix(x):
    return np.array(x)

def Scale(x):
    return x/255

data=Matrix(data)
print(data)
data=Scale(data)
data
```

✓ 0.0s

1

```
[[[  0   0   0]
  [  0   0 255]]

 [[255   0   0]
  [255 255 255]]]
```

2

```
array([[[0., 0., 0.],
        [0., 0., 1.]],

       [[1., 0., 0.],
        [1., 1., 1.]]])
```

```python
########################################
## 이미지 데이터를 받아서 /255를 하는 함수
########################################
import numpy as np

data = [
    [[0,0,0],[0,0,255]],
    [[255,0,0],[255,255,255]]
]

class transform_my():
    def Matrix(self, x):
        return np.array(x)

    def Scale(self, x):
        return x / 255

trans = transform_my()
data = trans.Matrix(data)
print("변환 후 (배열):\n", data)

data = trans.Scale(data)
print("\n정규화 후 (0~1 범위):\n", data)
```

부성순 bakpak@empas.com

# 이미지 전처리함수– 이미지처리 이해 (이미지는 0~255, 0~1 사이값이 같은 색상임)

```python
########################################
## 이미지 데이터를 받아서 /255를 하는 함수
########################################
import numpy as np

data = [
    [[0,0,0],[0,0,255]],
    [[255,0,0],[255,255,255]]
]

class transform_my():
    def Matrix(self, x):
        return np.array(x)

    def Scale(self, x):
        return x / 255

trans = transform_my()
data = trans.Matrix(data)
print("변환 후 (배열):\n", data)

data = trans.Scale(data)
print("\n정규화 후 (0~1 범위):\n", data)
```

1

2

```python
data = [
    [[0,0,0],[0,0,255]],
    [[255,0,0],[255,255,255]]
]

class transform_my():
    def Matrix(self, x):
        return np.array(x)

    def Scale(self, x):
        return x / 255

def myCompose(x):
    print(len(x))
    print(x)

trans = transform_my()
myCompose([
        trans.Matrix,
        trans.Scale
        ])
```

0.0s

```
<bound method transform_my.Matrix of <__main__.trar
```

함수객체만 생성됨
이 객체를 호출하면됨

**고민, 이 1,2,번을 한 개의 함수로 하려면**

**즉. 함수한개에 class의 Matrix, Scale를 넣음**

# 다음 내용은 pytorch의 Compose함수를 직접 구현한것임. 꼭 알아야 하는 내용은 아님

```python
import numpy as np

# 원본 데이터 (2x2x3 RGB 배열)
data = [
    [[0,0,0],[0,0,255]],
    [[255,0,0],[255,255,255]]
]

# ========================================
#  개별 변환 클래스 정의
# ========================================
class transform_my:
    def Matrix(self, x):
        return np.array(x)

    def Scale(self, x):
        return x / 255.0

# ========================================
#  Compose 기능 추가
# ========================================
class Compose:
    def __init__(self, transforms):
        self.transforms = transforms   # 변환 리스트 저장

    def __call__(self, x):
        for t in self.transforms:
            x = t(x)   # 순서대로 변환 적용
        return x


# ========================================
#  변환 파이프라인 구성
# ========================================
trans = transform_my()

# Compose 안에 변환을 순서대로 전달
my_pipeline = Compose([
    trans.Matrix,
    trans.Scale
])

# ========================================
#  실행
# ========================================
result = my_pipeline(data)

print("입력 형태:", type(data))
print("출력 타입:", type(result))
print("\n정규화 결과:\n", result)

```

Call 함수에 의해서
실행되는것만 이해

부성순 bakpak@empas.com

# 파이썬 기초 메서드

파이썬의 **"특별 메서드(Special Method)"**
**또는 "매직 메서드(Magic Method)"**

이름 앞뒤에 밑줄 두 개 __ 이름 __ 의 형식으로 되어 있음

| 메서드 이름 | 의미 |
|---|---|
| __ init __ | 객체 초기화 (constructor) |
| __ del __ | 객체 소멸 시 실행 |
| __ str __ | 객체를 문자열로 표현할 때 (print) |
| __ len __ | len()함수가 호출될 때 |
| __ add __ | '+' 연산자 사용 시 실행 |
| __ call __ | 객체를 함수처럼 호출할 때 |

__call__ 매직메서드 실습

```python
class AddAndGreet:
    def __init__(self, greeting="Hello"):
        self.greeting = greeting

    def __call__(self, num1, num2):
        # 객체를 호출하면 이 코드가 실행됨
        result = num1 + num2
        print(f"{self.greeting}! {num1} + {num2} = {result}")
        return result

# 사용
adder = AddAndGreet("Welcome")
# adder(5, 3)은 adder.__call__(5, 3)을 자동 실행
total = adder(5, 3)
```

```
Welcome! 5 + 3 = 8
```

부성순 bakpak@empas.com

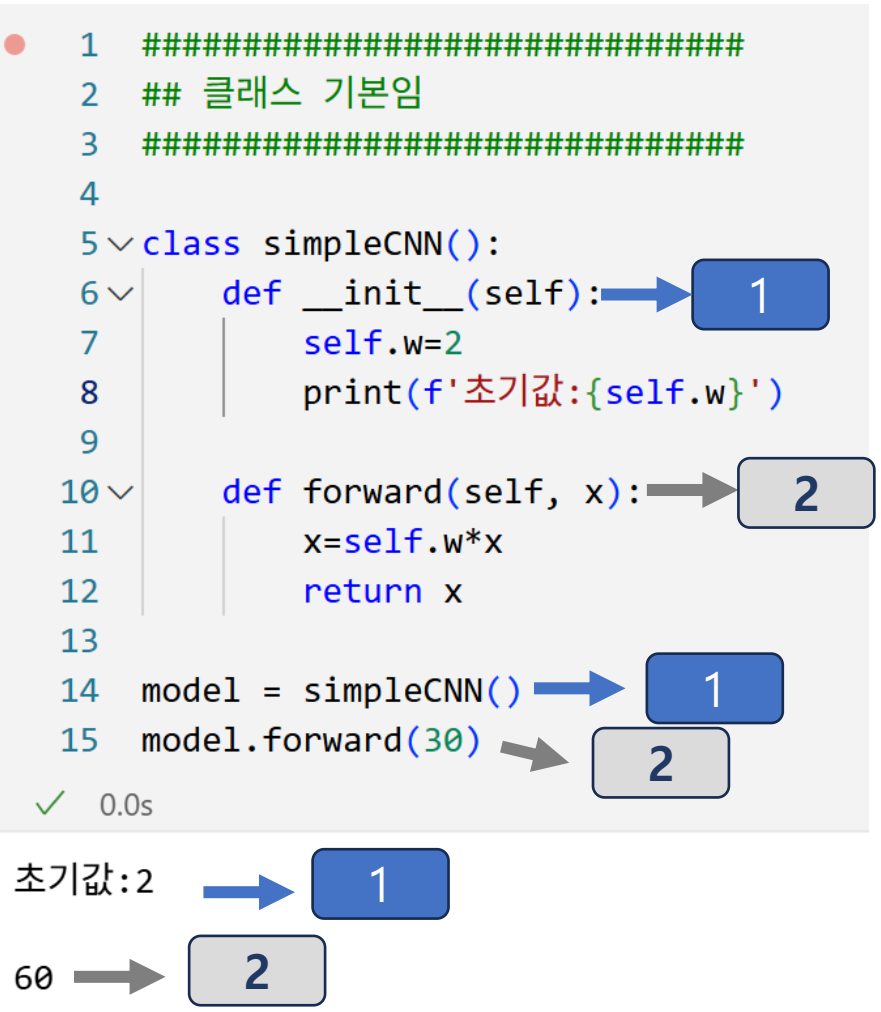
# 기초 클래스 작성하기

```python
################################
## 클래스 기본임
################################

class simpleCNN():
    def __init__(self):
        self.w=2
        print(f'초기값:{self.w}')

    def forward(self, x):
        x=self.w*x
        return x

model = simpleCNN()
model.forward(30)
```

✓ 0.0s

초기값:2

60




## 생성자

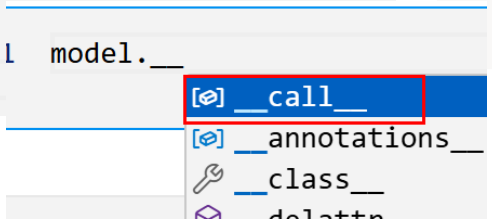
- "파이썬의 생성자(Constructor)"는 **클래스에서 객체가 만들어질 때 자동으로 실행되는 특별한 메서드**를 말합니다.
- **"파이썬이 객체를 처음 세팅할 때 부르는 자동 설정 함수 이름**

## 기본 개념

- 생성자의 이름은 항상 `__init__` 입니다.
- 주로 객체의 초기 상태(멤버 변수 값)를 설정할 때 사용합니다.

# Forward가 자동실행되게 하기

```python
import torch
import torch.nn as nn

# 원본 데이터 (2x2x3 RGB 배열)
data = [
    [[0,0,0],[0,0,255]],
    [[255,0,0],[255,255,255]]
]

# 리스트를 torch.Tensor로 변환
x = torch.tensor(data, dtype=torch.float32)

class SimpleCNN(nn.Module):  # torch.nn.Module 상속
    def __init__(self):
        super(SimpleCNN, self).__init__()  # 부모 클래스 초기화
        self.w = 2

    def forward(self, x):
        x = self.w * x
        return x

model=SimpleCNN()
model(x)
```




## PyTorch nn.Module의 __call__과 forward 자동 호출

PyTorch에서 모델을 정의할 때 torch.nn.Module 클래스를 상속받아 사용합니다.
사용자가 모델의 순전파(Forward Pass) 로직을 forward(self, x) 메서드 내에 정의하는 것은 핵심입니다

실제로 모델을 실행할 때는 model.forward(data) 대신
output = model(data)와 같이 일반 함수처럼 호출합니다.
이는 Python의 특별 메서드인 **call** 덕분에 가능한 것입니다.

개발자는 복잡한 프레임워크 내부 로직에 신경 쓸 필요 없이 forward 메서드에
모델의 계산 과정만 집중적으로 정의할 수 있습니다.
__call__이 이 forward를 안전하고 효율적으로 감싸서 실행해 주는 것입니다.

- **__call__의 역할**
  nn.Module은 내부적으로 이 **call** 메서드를 구현하고 있습니다.
  모델 인스턴스를 () 괄호를 사용해 호출하면, 파이썬은 자동으로 해당 객체의
  **call** 메서드를 실행합니다.
- **forward의 자동 호출**
  nn.Module의 **call** 메서드는 단순히 데이터를 전달하는 것 이상의
  역할을 수행합니다. 내부적으로 Hook(갈고리) 함수 실행, 상태 관리(예: 훈련/평가 모드),
  그리고 가장 중요한 자동 미분 그래프(autograd graph)를 위한 준비 등의
  필수 작업을 처리합니다. 이러한 준비가 모두 완료된 후,
  __call__은 최종적으로 사용자가 정의한 forward(x) 메서드를 호출하여 순전파 계산을 실행

부성순 bakpak@empas.com

# 02. CNN데이터 증강_Cifar10 .ipynb

## 데이터 전처리 (Data Pre-processing)

- 모델이 데이터를 **효율적**이고 **정확**하게 처리할 수 있도록 데이터의 품질을 높이고 형식을 통일합니다.
- 데이터의 크기, 형식, 값의 범위를 **표준화**하거나 **정규화**합니다.
- 데이터의 **본질적인 정보는 유지**하되, 모델 학습에 방해되는 요소를 제거하거나 범위를 조정합니다.
- 모델 학습 전 **반드시 수행**해야 하는 단계입니다.

## 데이터 증강 (Data Augmentation)

- 데이터셋의 **크기를 늘리고** 변동성을 부여하여 모델의 **과적합을 방지**하고 **일반화 성능**을 향상시킵니다.
- 기존 데이터를 기반으로 무작위적인 **변형된 가상 데이터**를 생성합니다.
- 이미지의 위치, 크기, 방향, 밝기 등의 **시각적 특징을 변화**시킵니다.
- 모델 성능 개선을 위한 **선택적 단계**입니다.
- **훈련 데이터**에 대해서만 모델 학습 중 **실시간** 또는 **훈련 이전**에 적용됩니다.

```python
# 실험 1: 기본 정규화
transform_no_aug = transforms.Compose([
    transforms.ToTensor(),
    # CIFAR-10 평균과 표준편차
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

trainset_no_aug = torchvision.datasets.CIFAR10(root='./data', train=True,
                                               download=True, transform=transform_no_aug)
trainloader_no_aug = DataLoader(trainset_no_aug, batch_size=64, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform_no_aug)
testloader = DataLoader(testset, batch_size=64, shuffle=False, num_workers=2)
```

```python
# 실험 2: 데이터 증강 적용
# 증강은 train에만 적용합니다.
transform_with_aug = transforms.Compose([
    # 데이터 증강 기법 추가
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    # 기본 정규화
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))
])

trainset_with_aug = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                 download=True, transform=transform_with_aug)
trainloader_with_aug = DataLoader(trainset_with_aug, batch_size=64, shuffle=True, num_workers=2)
```

부성순 bakpak@empas.com

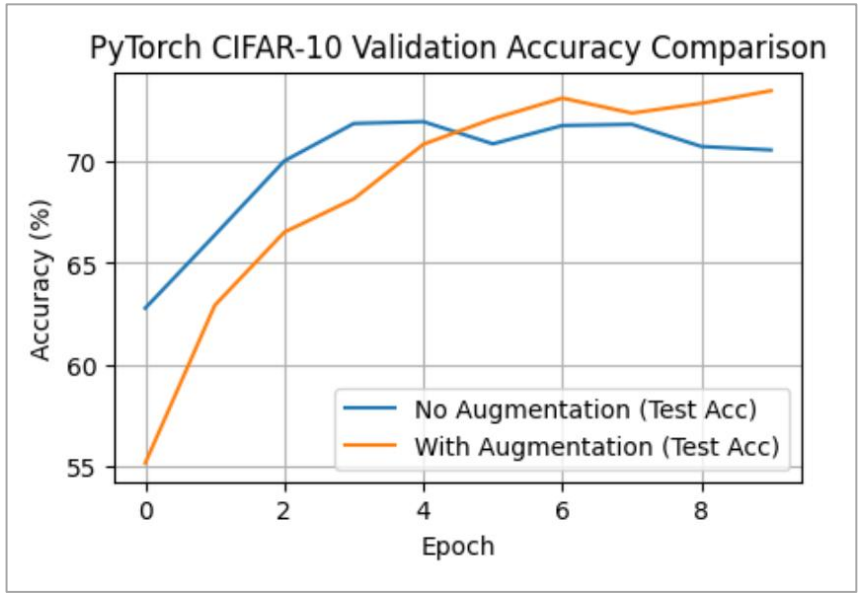
# 02. CNN데이터 증강_Cifar10 .ipynb

```
# 실험 설정
EPOCHS = 10

# --- 실험 1: 증강 없음 ---
print("Wn###  실험 1: 데이터 증강 없이 훈련 ###")
model_no_aug = SimpleCNN().to(device)
val_acc_no_aug, final_acc_no_aug = train_model(model_no_aug, trainloader_no_aug, testloader, epochs=EPOCHS)


# --- 실험 2: 증강 적용 ---
print("Wn###  실험 2: 데이터 증강 적용하여 훈련 ###")
model_with_aug = SimpleCNN().to(device)
val_acc_with_aug, final_acc_with_aug = train_model(model_with_aug, trainloader_with_aug, testloader, epochs=EPOCHS)
```

```
###  실험 1: 데이터 증강 없이 훈련 ###
Epoch [1/10], Loss: 1.3019, Test Acc: 62.79%
Epoch [2/10], Loss: 0.9329, Test Acc: 66.38%
Epoch [3/10], Loss: 0.7785, Test Acc: 70.02%
Epoch [4/10], Loss: 0.6621, Test Acc: 71.86%
Epoch [5/10], Loss: 0.5627, Test Acc: 71.95%
Epoch [6/10], Loss: 0.4718, Test Acc: 70.86%
Epoch [7/10], Loss: 0.3904, Test Acc: 71.76%
Epoch [8/10], Loss: 0.3146, Test Acc: 71.82%
Epoch [9/10], Loss: 0.2545, Test Acc: 70.73%
Epoch [10/10], Loss: 0.2046, Test Acc: 70.56%
```

```
###  실험 2: 데이터 증강 적용하여 훈련 ###
Epoch [1/10], Loss: 1.5349, Test Acc: 55.18%
Epoch [2/10], Loss: 1.1996, Test Acc: 62.93%
Epoch [3/10], Loss: 1.0724, Test Acc: 66.52%
Epoch [4/10], Loss: 0.9941, Test Acc: 68.16%
Epoch [5/10], Loss: 0.9395, Test Acc: 70.84%
Epoch [6/10], Loss: 0.8943, Test Acc: 72.09%
Epoch [7/10], Loss: 0.8612, Test Acc: 73.11%
Epoch [8/10], Loss: 0.8325, Test Acc: 72.37%
Epoch [9/10], Loss: 0.8147, Test Acc: 72.85%
Epoch [10/10], Loss: 0.7876, Test Acc: 73.47%
```

과적합




부성순 bakpak@empas.com