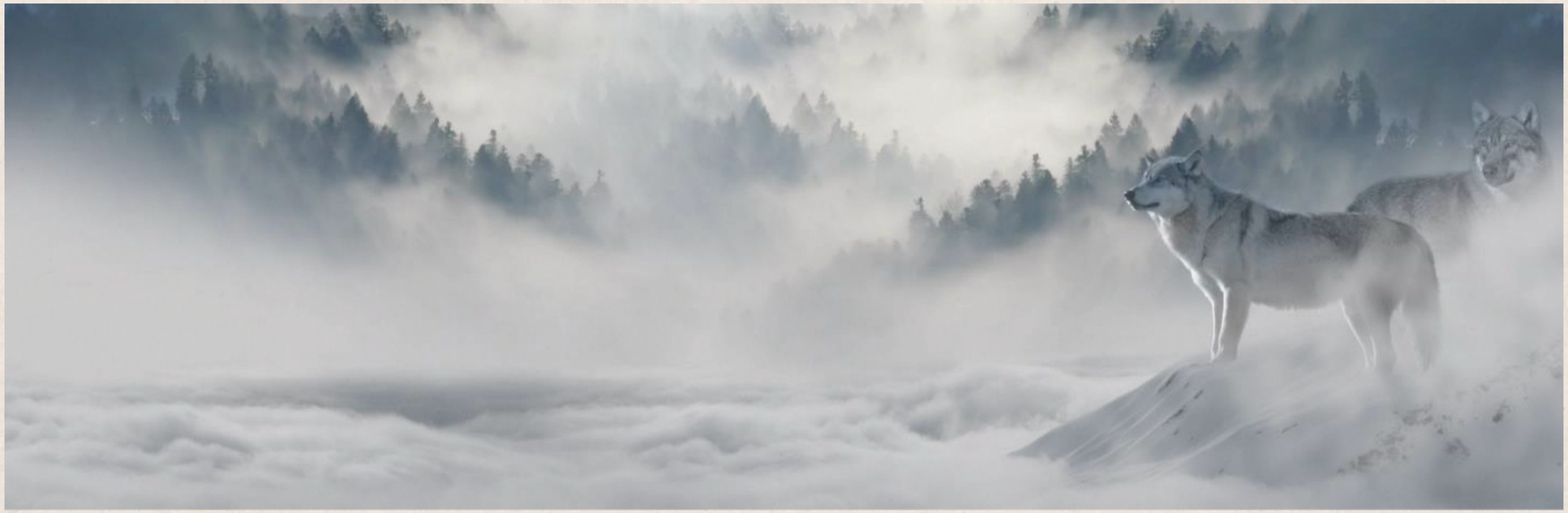
CNN_pytorch_cifar100



경북대학교

GPU 사용 설정

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
print(device + " is available")
```

CUDA가 있으면 device = 'cuda'로 설정해서 언제든 gpu를 사용할 수 있게 만들어줌

데이터셋 로드

```
cifar_train = torchvision.datasets.CIFAR100(root='./',
                                            train=True,
                                            transform=transforms.ToTensor(),
                                            download=True)
cifar_test = torchvision.datasets.CIFAR100(root='./',
                                           train=False,
                                           transform=transforms.ToTensor(),
                                           download=True)
# Load CIFAR-100 dataset
train_dataloader = torch.utils.data.DataLoader(dataset=cifar_train,
                                               batch_size=batch_size,
                                               shuffle=True,
                                               drop_last=True)
test_dataloader = torch.utils.data.DataLoader(dataset=cifar_test,
                                              batch_size=batch_size,
                                              shuffle=True,
                                              drop_last=True)
```

root = cifar 데이터셋을 저장할 경로

train : True로 설정 시 trainset, False로 설정시 test set을 불러옴

download = True로 설정시 인터넷에서 데이터셋 다운로드

transform = 데이터셋을 불러온 후 어떤 전처리를 할지 설정

여기서는 ToTensor()함수를 사용하여 0~255를 0~1로 변환

import

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

torch : pytorch의 기본 패키지, tensor 연산 및 딥러닝 모델 구현에 필요한 함수들을 제공

torch.nn : 신경망 모델을 구현하기 위한 클래스와 함수들이 들어있는 모듈

torch.nn.functional : 신경망 모델에서 자주 사용하는 함수

torch.optim : 경사하강법등 최적화 알고리즘 제공

torchvision : 이미지 및 비디오 처리를 위한 패키지

torchvision.transforms : 이미지 변환을 위한 함수

데이터 불러오기

```
# Load CIFAR-100 dataset
train_dataloader = torch.utils.data.DataLoader(dataset=cifar_train,
                                               batch_size=batch_size,
                                               shuffle=True,
                                               drop_last=True)
test_dataloader = torch.utils.data.DataLoader(dataset=cifar_test,
                                              batch_size=batch_size,
                                              shuffle=True,
                                              drop_last=True)
```

torch.utils.data.DataLoader = 미리 정의된 batch size로 train_set과 test_set을 load함

root = cifar 데이터셋을 저장할 경로

train : True로 설정 시 trainset, False로 설정시 test set을 불러옴

download = True로 설정시 인터넷에서 데이터셋 다운로드

transform = 데이터셋을 불러온 후 어떤 전처리를 할지 설정

여기서는 ToTensor()함수를 사용하여 0~255를 0~1로 변환

기본 파라미터 값 설정

```
# CIFAR-100 데이터셋 불러오기
learning_rate = 0.001
training_epochs = 20
batch_size = 128
```

학습률 : 알고리즘에서 가중치를 업데이트 할때 사용되는 스칼라 값. 학습률이 작을수록 가중치 업데이트가 느리게 이루어짐

배치사이즈 : 학습데이터를 학습할때 미니 배치의 크기

num_classes : 분류하고자 하는 클래스 개수

epochs : 전체 학습 데이터셋을 몇번 반복할 것인지

ConvNet class __init__

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.layer1 = torch.nn.Sequential(
        torch.nn.Conv2d(3, 32, (3,3), padding='same'),
        torch.nn.ReLU(),
        torch.nn.BatchNorm2d(32),
        torch.nn.MaxPool2d((2,2)),
        torch.nn.Dropout(0.5)
    )
    self.layer2 = torch.nn.Sequential(
        torch.nn.Conv2d(32, 64, (3,3), padding='same'),
        torch.nn.ReLU(),
        torch.nn.BatchNorm2d(64),
        torch.nn.MaxPool2d((2,2)),
        torch.nn.Dropout(0.5)
    )
```

1. `in_channels` (int): 입력 이미지의 채널 수
2. `out_channels` (int): 출력 이미지의 채널 수 (필터의 개수)
3. `kernel_size` (int 또는 Tuple[int, int]): 컨볼루션 커널의 크기
4. `stride` (int 또는 Tuple[int, int]): 컨볼루션 연산 시 필터의 이동 간격
5. `padding` (int 또는 Tuple[int, int]): 입력 이미지 주변에 추가할 패딩 크기
6. `dilation` (int 또는 Tuple[int, int]): 컨볼루션 커널 내부의 간격
7. `groups` (int): 입력 채널을 나누어 여러 그룹으로 컨볼루션 연산을 수행할 때 그룹 수
8. `bias` (bool): 편향(bias) 사용 여부

class ConvNet은 nn.Module 클래스를 상속받아 정의됨, nn.Module 클래스는 pytorch에서 제공하는 모든 신경망 모듈의 기본 클래스

ConV2d 함수를 다섯번 작성, maxpool 작성, Linear 3개작성, dropout작

출력 크기 계산 공식

$$O = \frac{I - K + 2P}{S} + 1$$

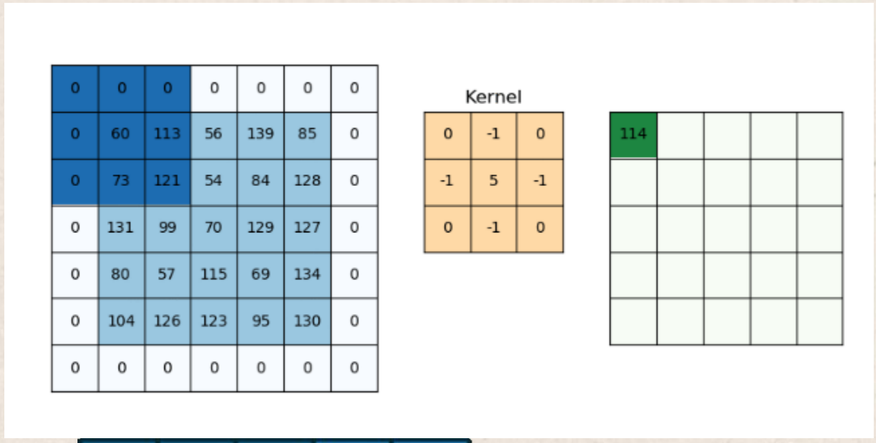
1. 입력 이미지의 크기를 W x W 라 가정합니다.
2. 필터(커널)의 크기를 K x K 라 가정합니다.
3. 패딩의 크기를 P 라 가정합니다.
4. 스트라이드 값(필터를 이동시키는 간격)을 S 라 가정합니다.
5. 합성곱 연산을 수행하면 필터와 입력 이미지가 겹치는 부분에서 곱셈한 결과를 모두 더한 값이 출력 이미지의 한 픽셀에 대응합니다.
6. 입력 이미지의 한 픽셀을 중심으로 필터를 이동시키는 경우, 이동할 수 있는 범위는 (W-K+1) x (W-K+1) 입니다.
7. 하지만 이동할 수 있는 범위가 출력 이미지의 크기와 일치하지 않을 수 있습니다. 따라서, 이동할 수 있는 범위에서 출력 이미지의 크기에 해당하는 부분만 잘라내어 사용합니다.
8. 출력 이미지의 크기를 H x H 라 가정합니다.
9. 이 때, 이동할 수 있는 범위에서 출력 이미지의 크기에 해당하는 부분은 (H-1) x (H-1) 이므로, (W-K+1) x (W-K+1) = (H-1) x (H-1) 입니다.
10. 이를 정리하면, H = ((W-K+1) - 1) / S + 1 이 됩니다.
11. 하지만 이 공식에 패딩을 추가할 수도 있습니다. 패딩을 P만큼 추가하면, 입력 이미지의 크기는 (W+2P) x (W+2P) 가 됩니다.
12. 이 경우, 출력 이미지의 크기를 H x H 라 가정하면, (W+2P-K+1) x (W+2P-K+1) = (H-1) x (H-1) 입니다.
13. 이를 정리하면, H = ((W+2P-K+1) - 1) / S + 1 이 됩니다.
14. 따라서, ((W-K+2P)/S)+1 공식이 유도됩니다.

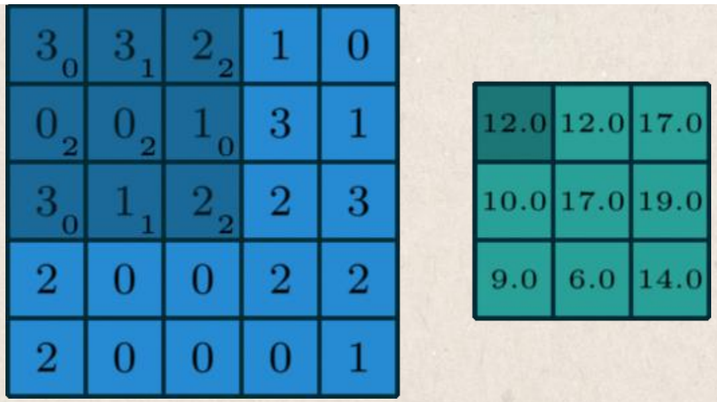
```
self.conv1 = nn.Conv2d(1, 10, kernel_size=5) # input channel = 1, filter = 10, kernel size = 5, zero padding = 0, stribe = 1
# ((W-K+2P)/S)+1 공식으로 인해 ((28-5+0)/1)+1=24 -> 24x24로 변환
```

```
self.conv2 = nn.Conv2d(10, 20, kernel_size=5) # input channel = 1, filter = 10, kernel size = 5, zero padding = 0, stribe = 1
# ((12-5+0)/1)+1=8 -> 8x8로 변환
```

출력 크기 = ((입력 크기 - 필터 크기 + 2*패딩) / 스트라이드) + 1

Conv2D (패딩)





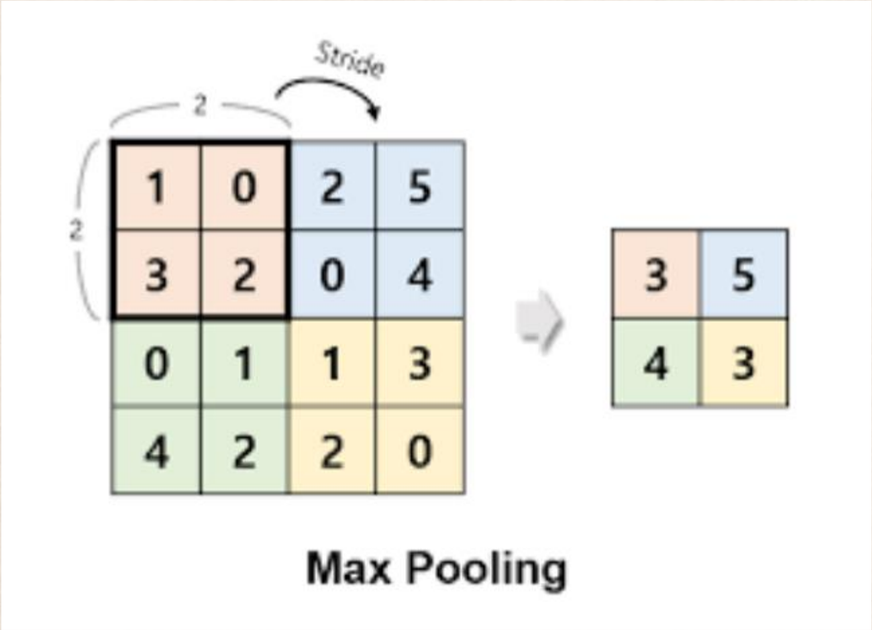
padding이 same인 경우, 출력 특징맵의 크기가 입력 특징맵의 크기와 상관없이 동일하게 유지됩니다.

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

스트라이드를 통해 조절 가능

Conv2D (maxpool2D)

```
x = F.relu(self.mp(self.conv1(x)))
x = F.relu(self.mp(self.conv2(x)))
```

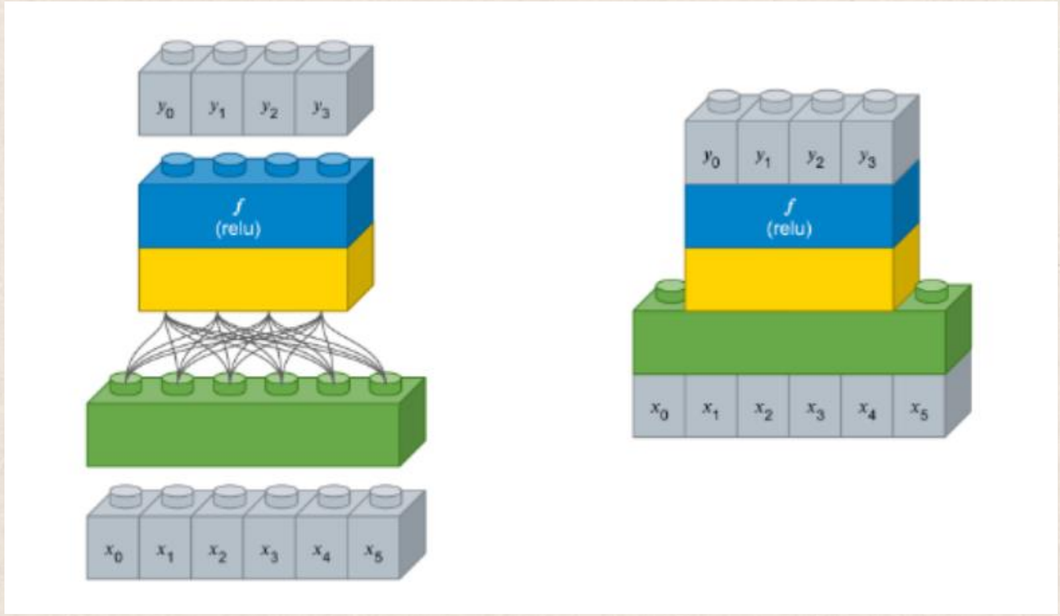


MaxPool2D 레이어는 입력으로 받은 64개의 28x28 크기의 특징 맵(feature map)에서 2x2 크기의 윈도우(window)를 이동시켜가며 각 윈도우에서 가장 큰 값을 출력으로 반환

입력 특징 맵보다 크기가 반으로 줄어들게 되는데, 이를 통해 계산량을 줄이고, 과적합(overfitting)을 방지하고, 불필요한 정보를 걸러낼 수 있음

Dense>>Linear

```
self.drop2D = nn.Dropout2d(p=0.25, inplace=False) # 랜덤하
self.mp = nn.MaxPool2d(2)  # 오버피팅을 방지하고, 연산에
self.fc1 = nn.Linear(320,100) # 4x4x20 vector로 flat한 것
self.fc2 = nn.Linear(100,10) # 100개의 출력을 10개의 출력
```



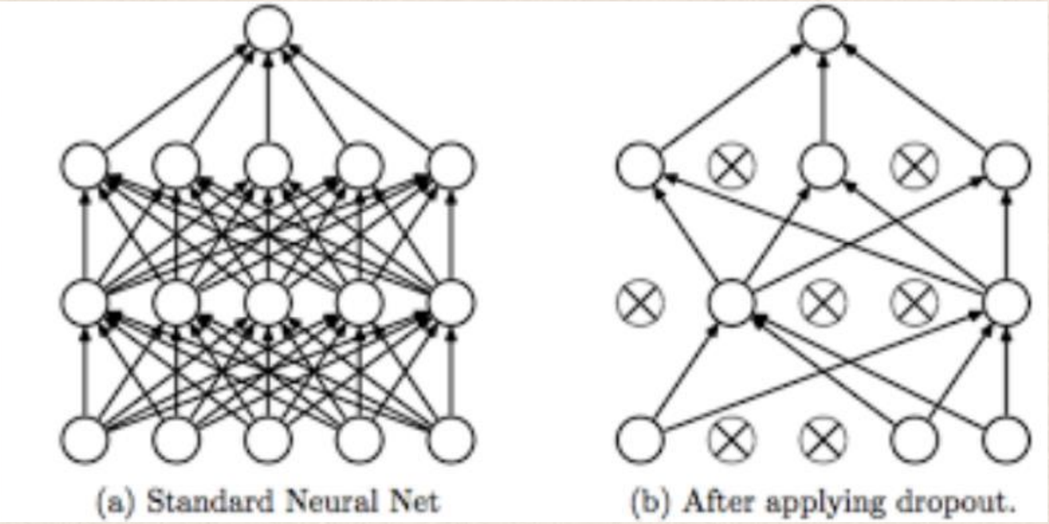
nn.Linear` 함수는 PyTorch에서 완전 연결(fully connected) 레이어를 정의하는데 사용됩니다. 이 함수는 입력과 출력의 차원을 인자로 받아 가중치 행렬을 만들고, 입력에 가중치를 곱하고 편향을 더하여 출력을 계산합니다.

예를 들어, `nn.Linear(10, 5)`는 10차원의 입력을 5차원의 출력으로 변환하는 완전 연결 레이어를 정의합니다. 이 레이어의 가중치 행렬은 크기가 (5, 10)이며, 편향 벡터의 크기는 (5,)입니다.

==`Dense(5, input_shape=(10,))

Dropout

```
self.drop2D = nn.Dropout2d(p=0.25, inplace=False) #
```



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.

def forward

```
def forward(self, x):
      x = F.relu(self.mp(self.conv1(x))) # convolution layer 1번에 relu를 씌우고 maxpool, 결과값은 12x12x10
      x = F.relu(self.mp(self.conv2(x))) # convolution layer 2번에 relu를 씌우고 maxpool, 결과값은 4x4x20
      x = self.drop2D(x)
      x = x.view(x.size(0), -1) # flat
      x = self.fc1(x) # fc1 레이어에 삽입
      x = self.fc2(x) # fc2 레이어에 삽입
      return F.log_softmax(x) # fully-connected layer에 넣고 logsoftmax 적용
```

x = F.relu(self.mp(self.conv1(x))): 입력 데이터 `x`를 첫 번째 합성곱 레이어 `self.conv1`을 통해 통과시키고, 그 결과에 ReLU 함수를 적용하고, 맥스 풀링 레이어 `self.mp`를 적용합니다

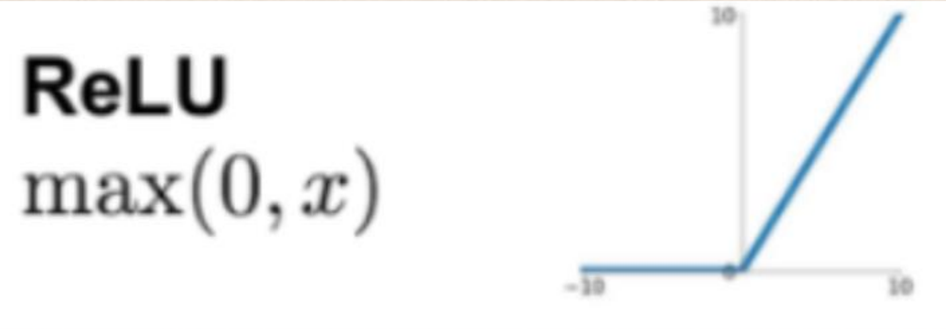
`x = self.drop2D(x)`: 출력값 `x`에 2D 드롭아웃 레이어 `self.drop2D`를 적용합니다.

x = x.view(x.size(0), -1)`: 출력값 `x`를 2차원으로 변환합니다. 이는 fully connected 레이어에 입력으로 넣기 위한 과정(Flatten)

`x = self.fc1(x)`: 2차원으로 변환된 출력값 `x`를 첫 번째 fully connected 레이어 `self.fc1`에 입력으로 넣습니다
return F.log_softmax(x)`: 마지막으로 fully connected 레이어를 통과한 결과값 `x`에 로그 소프트맥스 함수를 적용한 결과를 반환합니다.

Conv2D (activation(relu)

```
def forward(self, x):
      x = F.relu(self.mp(self.conv1(x))) # convolution layer 1번에 relu를 씌우고 maxpool, 결과값은 12x12x10
      x = F.relu(self.mp(self.conv2(x))) # convolution layer 2번에 relu를 씌우고 maxpool, 결과값은 4x4x20
      x = self.drop2D(x)
      x = x.view(x.size(0), -1) # flat
      x = self.fc1(x) # fc1 레이어에 삽입
      x = self.fc2(x) # fc2 레이어에 삽입
      return F.log_softmax(x) # fully-connected layer에 넣고 logsoftmax 적용
```



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

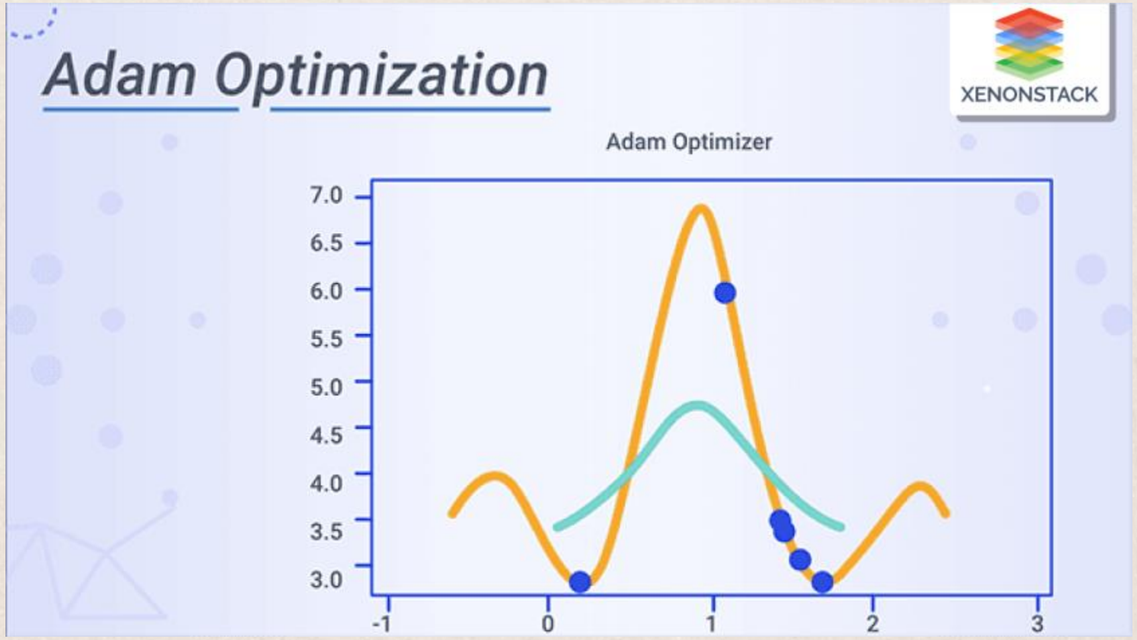
ReLU 함수는 계산 비용이 매우 적게 듬

죽은 뉴런(dead neuron) 현상이 발생할 수 있습니다. 이러한 현상이 발생하면 해당 뉴런은 학습 과정에서 어떠한 역할도 하지 못하게 됨

Adam

```
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
```

이해가... 안됩니당 다음주 안에 추가하겠습니다.



모델 학습 설정

```
model = ConvNet().to(device) # CNN instance 생성
# Cost Function과 Optimizer 선택
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
```

model = ConvNet().to(device)`: `ConvNet` 클래스를 인스턴스화하여 `model` 변수에 저장합니다. 이때 `to(device)` 메서드를 사용하여 모델이 GPU에서 실행되도록 설정합니다.

criterion = nn.CrossEntropyLoss().to(device)`: 크로스 엔트로피 손실 함수 `nn.CrossEntropyLoss()`를 인스턴스화하여 `criterion` 변수에 저장

optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)`: Adam 옵티마이저를 인스턴스화하여 `optimizer` 변수에 저장합니다. model.parameters()`를 사용하여 모델의 학습 가능한 매개변수들을 전달하고, `lr` 매개변수를 통해 학습률을 설정합니다.

모델 학습 설정(test)

```python
def test(dataloader, model, criterion):
  size = len(dataloader.dataset)
  num_batches = len(dataloader)
  model.eval()
  test_loss, test_acc = 0, 0
  with torch.no_grad():
    for X, y in dataloader:
      X, y = X.to(device), y.to(device)
      pred = model(X)
      test_loss += criterion(pred, y).item()
      test_acc += (pred.argmax(1) == y).type(torch.float).sum().item()

  test_loss /= num_batches
  test_acc /= size

  print(f"TEST: \n Accuracy: {(100*test_acc):>.1f}%, Avg loss: {test_loss:>8f} \n")

  return test_loss, test_acc
```

dataset읽어오는 모듈,모델,손실함수
모델을 평가모드로 설정한 후 no_grad()를 사용하여 그래디언트 계산을 비활성화 합니다. (모델의 파라미터를 업데이트 하는 과정에서 계산 생략 (평가할때는 필요 X))

그 후 dataloader에서 배치데이터를 읽어오면서
손실과 정확도를 반환합니다.

X = 입력데이터(batch), y = 정답데이터

모델 학습 설정(train)

```
def train(dataloader, model, criterion, optimizer):
  size = len(dataloader.dataset)
  train_accuracy, train_loss = 0, 0
  model.train()
  for batch, (X, y) in enumerate(dataloader, 0):

    X, y = X.to(device), y.to(device)
    optimizer.zero_grad()
    pred = model(X)
    loss = criterion(pred, y)
    loss.backward()
    optimizer.step()

    train_accuracy += (pred.argmax(1) == y).type(torch.float).sum().item()
    train_loss += loss.item()

    if batch % 100 == 0:
      current = (batch+1) * len(X)

      print(f"loss: {train_loss/current:>7f} accuracy: {train_accuracy/curre

  return train_loss/len(dataloader), train_accuracy/size
```

dataset읽어오는 모듈,모델,손실함수,최적화 함

size변수에 전체 데이터셋의 크기 저장. 그 후 초기화, 그리고 모델을 학습모드로 설정.

입력데이터와 정답데이터를 분배한 후 gpu로 설정, optimizer 기울기 초기화. 그 후 예측 시작 순전파, 역전파 그 후 파라미터 업데이트,

정확도와 손실도 계산 후 배치수가 100의 배수일때마다 손실과 정확도 출력

모델 훈련 후 저장

```
epochs = 10
training_loss, training_accuracy = [], []
validation_loss, validation_accuracy = [], []
for i in range(epochs):
  print("Epoch {}₩n----------------------------".format(i+1))
  train_loss, train_accuracy = train(train_loader, model, criterion, optimizer)
  val_loss, val_accuracy = test(test_loader, model, criterion)

  training_loss.append(train_loss)
  training_accuracy.append(train_accuracy)
  validation_loss.append(val_loss)
  validation_accuracy.append(val_accuracy)
history = [training_loss,training_accuracy,validation_loss, validation_accuracy]
PATH_TO_TRAINED_MODEL = 'model.pth'
torch.save(model.state_dict(), PATH_TO_TRAINED_MODEL)
```

epochs 변수설정
train_loss와 train_accuracy
val_loss와 val_accuracy list에 저장 후
history로 묶음

model 저장!!

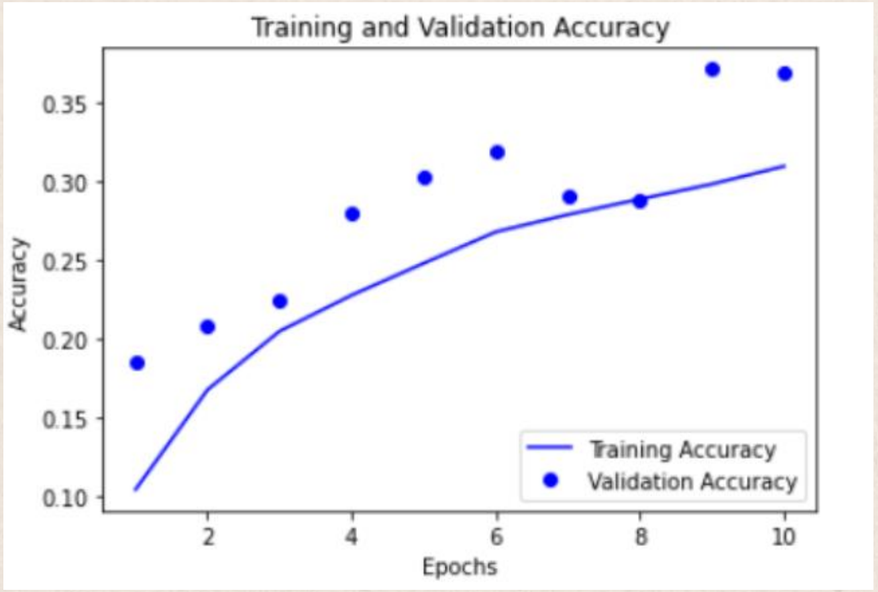
epoch

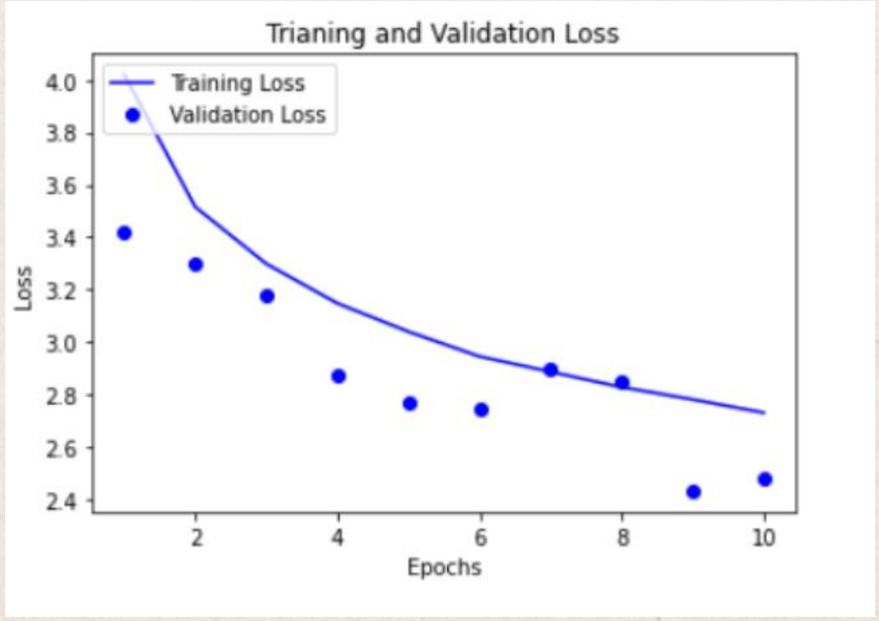
```
Epoch 9
----------------------
loss:0.021712 accuracy: 0.281250 [  128/50000]
loss:0.021682 accuracy: 0.298035 [12928/50000]
loss:0.021737 accuracy: 0.297613 [25728/50000]
loss:0.021741 accuracy: 0.298588 [38528/50000]
TEST:
 Accuracy: 37.2%, Avg loss: 2.432543

Epoch 10
----------------------
loss:0.019443 accuracy: 0.375000 [  128/50000]
loss:0.021296 accuracy: 0.311572 [12928/50000]
loss:0.021346 accuracy: 0.308030 [25728/50000]
loss:0.021330 accuracy: 0.308970 [38528/50000]
TEST:
 Accuracy: 37.0%, Avg loss: 2.477928
```

\epoch = 10, Accuracy : 37%, Avg loss : 2.477

accuracy, loss graph





Conv2D (가중치)

```
def forward(self, x):
      x = F.relu(self.mp(self.conv1(x))) # convolution layer 1번에 relu를 씌우고 maxpool, 결과값은 12x12x10
      x = F.relu(self.mp(self.conv2(x))) # convolution layer 2번에 relu를 씌우고 maxpool, 결과값은 4x4x20
      x = self.drop2D(x)
      x = x.view(x.size(0), -1) # flat
      x = self.fc1(x) # fc1 레이어에 삽입
      x = self.fc2(x) # fc2 레이어에 삽입
      return F.log_softmax(x) # fully-connected layer에 넣고 logsoftmax 적용
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 10, 24, 24]             260
         MaxPool2d-2          [-1, 10, 12, 12]               0
            Conv2d-3           [-1, 20, 8, 8]           5,020
         MaxPool2d-4           [-1, 20, 4, 4]               0
         Dropout2d-5           [-1, 20, 4, 4]               0
            Linear-6                [-1, 100]          32,100
            Linear-7                 [-1, 10]           1,010
================================================================
Total params: 38,390
Trainable params: 38,390
Non-trainable params: 0
----------------------------------------------------------------
```

필터 크기 5*5개
출력 필터 개수 10개
따라서 10*5*5 + 10(편향) = 260

입력체널의 수는 10, 출력 채널의 수는 20
10*20*5*5 + 20(편향) = 5020

입력채널의 수 20*4*4 = 320, 출력 채널 = 100
320*100 +100(편향) = 32100

테스트 진행(경로 설정)

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


print(img_path_list)
```

40개의 이미지 생성후 경로 저장

테스트 진행(실제 데터)

```
# Load the model from file
model = Net()
PATH_TO_TRAINED_MODEL = 'model.pth'
model.load_state_dict(torch.load(PATH_TO_TRAINED_MODEL, map_location=torch.device('cpu')))
```

저장했던 모듈 불러온 후 평가모드로 전환(gpu 다써서 cpu로 전환)

테스트 진행(실제 데터)

```
[92] a=['apples','apples','telephone','telephone','subway','subway','tea','tea','elephant','elephant','sunflower','sunflower','

100] 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 devi
                                '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
                               ])
```

class 설정

테스트 진행(실제 데터)

```python
import torch
from PIL import Image
from torchvision import transforms

# Load the image
for i in range(len(img_path_list)):
  image_path = img_path_list[i]
  image = Image.open(image_path)

  # Define the transformation to apply to the image
  transform = transforms.Compose([
      transforms.Resize((32, 32)), # Resize the image to fit the input shape of your model
      transforms.ToTensor(), # Convert the image to a PyTorch tensor
      transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]) # Normalize the image
  ])

  # Apply the transformation to the image
  image_tensor = transform(image)

  # Reshape the tensor to add a batch dimension
  image_tensor = image_tensor.unsqueeze(0)

  # Load your trained model
  model = Net().to(device)
  model.load_state_dict(torch.load("model.pth"))

  # Pass the image through the model to get the predicted class probabilities
  with torch.no_grad():
      model.eval()
      output = model(image_tensor.to(device))
      probabilities = torch.softmax(output, dim=1)

  # Get the predicted class
  predicted_class = torch.argmax(probabilities, dim=1).item()

  print("Predicted class:", CIFAR100_CLASSES[predicted_class], a[i])
```

```
Predicted class: worm apples
Predicted class: worm apples
Predicted class: lamp telephone
Predicted class: worm telephone
Predicted class: telephone subway
Predicted class: worm subway
Predicted class: chair tea
Predicted class: worm tea
Predicted class: worm elephant
Predicted class: worm elephant
Predicted class: lamp sunflower
Predicted class: lamp sunflower
Predicted class: worm house
Predicted class: chair house
Predicted class: worm sea
Predicted class: worm sea
Predicted class: lamp cycle
Predicted class: worm cycle
Predicted class: worm flower
Predicted class: worm flower
Predicted class: lamp tv
Predicted class: worm tv
Predicted class: worm sneak
Predicted class: chair sneak
Predicted class: kangaroo mountain
Predicted class: worm mountain
Predicted class: worm people
Predicted class: chair people
Predicted class: worm skunk
Predicted class: worm skunk
Predicted class: worm butterfly
Predicted class: worm butterfly
Predicted class: lamp chair
Predicted class: oranges chair
Predicted class: worm hamster
Predicted class: chair hamster
Predicted class: worm tree
Predicted class: worm tree
Predicted class: oranges oranges
Predicted class: chair oranges
```

예상값과 정답, 그리고 output출력
정확도 : 약 5%정도