

14주차(2/2)

# ML Open Framework PyTorch

파이썬으로 배우는 기계학습

한동대학교  
김영섭 교수

# 기계학습 오픈 프레임워크: 파이토치

---

- 학습 목표
  - 파이토치 (**PyTorch**)를 이해한다.
  - **PyTorch**를 이용하여 **MNIST** 데이터를 분석한다.
  - **PyTorch**를 이용하여 합성곱 신경망(**CNN**)을 구현한다.
- 학습 내용
  - 기계학습을 위한 오픈 프레임워크 - **PyTorch**
  - **PyTorch**를 이용한 **MNIST** 데이터 처리
  - **PyTorch**를 이용한 합성곱 신경망(**CNN**) 구현
  - 기계학습 모델 옰로(**YOLO**)
  - 기계학습 모델 갠(**GAN**)

# 1. 기계학습 오픈 프레임워크: 파이토치

---

- 간결하고 구현이 빠르다.
- **Define by Run** 방식
- 파이썬 라이브러리와 높은 호환성
- **NumPy** 와 비슷한 **Tensor** 연산이 **GPU**로도 가능
- 페이스북 인공지능 연구팀에 의해 개발



## 2. 파이토치 Tensor: 행렬 생성

---

```
1 x = torch.empty(5, 3)
2 print(x)
```

```
tensor([[ 0.0000e+00,  2.5244e-29,  0.0000e+00],
        [ 2.5244e-29, -6.7749e-03,  4.5843e-41],
        [-6.7974e-03,  4.5843e-41, -6.7974e-03],
        [ 4.5843e-41,  1.5755e-19,  4.2039e-45],
        [ 0.0000e+00,  0.0000e+00,  1.7751e+28]])
```

## 2. 파이토치 Tensor: 행렬 생성

```
1 x = torch.empty(5, 3)
2 print(x)
```

```
tensor([[ 0.0000e+00,  2.5244e-29,  0.0000e+00],
        [ 2.5244e-29, -6.7749e-03,  4.5843e-41],
        [-6.7974e-03,  4.5843e-41, -6.7974e-03],
        [ 4.5843e-41,  1.5755e-19,  4.2039e-45],
        [ 0.0000e+00,  0.0000e+00,  1.7751e+28]])
```

```
1 x = torch.rand(5, 3)
2 print(x)
```

```
tensor([[ 0.0359,  0.8183,  0.4344],
        [ 0.1910,  0.5686,  0.2170],
        [ 0.8663,  0.7237,  0.3810],
        [ 0.0831,  0.6760,  0.4928],
        [ 0.1605,  0.3736,  0.1245]])
```

## 2. 파이토치 Tensor: 행렬 생성

```
1 x = torch.empty(5, 3)
2 print(x)
```

```
tensor([[ 0.0000e+00,  2.5244e-29,  0.0000e+00],
        [ 2.5244e-29, -6.7749e-03,  4.5843e-41],
        [-6.7974e-03,  4.5843e-41, -6.7974e-03],
        [ 4.5843e-41,  1.5755e-19,  4.2039e-45],
        [ 0.0000e+00,  0.0000e+00,  1.7751e+28]])
```

```
1 x = torch.rand(5, 3)
2 print(x)
```

```
tensor([[ 0.0359,  0.8183,  0.4344],
        [ 0.1910,  0.5686,  0.2170],
        [ 0.8663,  0.7237,  0.3810],
        [ 0.0831,  0.6760,  0.4928],
        [ 0.1605,  0.3736,  0.1245]])
```

```
1 x = torch.zeros(5, 3,
2                 dtype=torch.long)
3 print(x)
```

```
tensor([[ 0,  0,  0],
        [ 0,  0,  0],
        [ 0,  0,  0],
        [ 0,  0,  0],
        [ 0,  0,  0]])
```

## 2. 파이토치 Tensor: 행렬 생성

```
1 x = torch.empty(5, 3)
2 print(x)
```

```
tensor([[ 0.0000e+00,  2.5244e-29,  0.0000e+00],
        [ 2.5244e-29, -6.7749e-03,  4.5843e-41],
        [-6.7974e-03,  4.5843e-41, -6.7974e-03],
        [ 4.5843e-41,  1.5755e-19,  4.2039e-45],
        [ 0.0000e+00,  0.0000e+00,  1.7751e+28]])
```

```
1 x = torch.rand(5, 3)
2 print(x)
```

```
tensor([[ 0.0359,  0.8183,  0.4344],
        [ 0.1910,  0.5686,  0.2170],
        [ 0.8663,  0.7237,  0.3810],
        [ 0.0831,  0.6760,  0.4928],
        [ 0.1605,  0.3736,  0.1245]])
```

```
1 x = torch.zeros(5, 3,
2                 dtype=torch.long)
3 print(x)
```

```
tensor([[ 0,  0,  0],
        [ 0,  0,  0],
        [ 0,  0,  0],
        [ 0,  0,  0],
        [ 0,  0,  0]])
```

```
1 x = torch.tensor([5.5, 3])
2 print(x)
```

```
tensor([ 5.5000,  3.0000])
```



## 2. 파이토치 Tensor: 넘파이 변환(Bridge)

- NumPy → Torch Tensor

- Torch Tensor → NumPy

```
1 import numpy as np
2 npa = np.ones(5)
3 → trh = torch.from_numpy(npa)
4 np.add(trh, 1, out=npa)
5 print(npa)
6 print(trh)
```



## 2. 파이토치 Tensor: 넘파이 변환(Bridge)

### ■ NumPy → Torch Tensor

### ■ Torch Tensor → NumPy

```
1 import numpy as np
2 npa = np.ones(5)
3 trh = torch.from_numpy(npa)
4 ➡ np.add(trh, 1, out=npa)
5 print(npa)
6 print(trh)
```



```
[2. 2. 2. 2. 2.]
tensor([ 2.,  2.,  2.,  2.,  2.], dtype=torch.float64)
```

## 2. 파이토치 Tensor: 넘파이 변환(Bridge)

### ■ NumPy → Torch Tensor

```
1 import numpy as np
2 npa = np.ones(5)
3 trh = torch.from_numpy(npa)
4 np.add(trh, 1, out=npa)
5 print(npa)
6 print(trh)
```



```
[2. 2. 2. 2. 2.]
tensor([ 2.,  2.,  2.,  2.,  2.], dtype=torch.float64)
```

### ■ Torch Tensor → NumPy

```
1 import numpy as np
2 trh = torch.ones(5)
3 → npa = trh.numpy()
4 print(trh, '/', npa)
5 trh.add_(1)
6 print(trh, '/', npa)
```

## 2. 파이토치 Tensor: 넘파이 변환(Bridge)

### ■ NumPy → Torch Tensor

```
1 import numpy as np
2 npa = np.ones(5)
3 trh = torch.from_numpy(npa)
4 np.add(trh, 1, out=npa)
5 print(npa)
6 print(trh)
```



```
[2. 2. 2. 2. 2.]
tensor([ 2.,  2.,  2.,  2.,  2.], dtype=torch.float64)
```

### ■ Torch Tensor → NumPy

```
1 import numpy as np
2 trh = torch.ones(5)
3 npa = trh.numpy()
4 print(trh, '/', npa)
→ 5 trh.add_(1)
6 print(trh, '/', npa)
```




```
tensor([ 1.,  1.,  1.,  1.,  1.]) / [1. 1. 1. 1. 1.]
tensor([ 2.,  2.,  2.,  2.,  2.]) / [2. 2. 2. 2. 2.]
```

### 3. CNN 신경망 구현: 생성자

```
1 class CNN(nn.Module):
2     def __init__(self):
3         super(CNN, self).__init__()
4         self.conv1 = nn.Conv2d(1, 16, kernel_size=2)
5         self.conv2 = nn.Conv2d(16, 32, kernel_size=2)
6         self.conv3 = nn.Conv2d(32, 64, kernel_size=2)
7         self.dropout = nn.Dropout(0.2)
8         self.fc1 = nn.Linear(256, 10)
9
10    def forward(self, x):
11        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
12        x = self.dropout(x)
13        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
14        x = self.dropout(x)
15        x = F.max_pool2d(F.relu(self.conv3(x)), 2)
16        x = x.view(-1, 256)
17        x = self.fc1(x)
18        return F.log_softmax(x, dim=1)
```

### 3. CNN 신경망 구현: 생성자

- 생성자: `__init__()`
- 합성곱: **`Conv2d()`**




```
1 class CNN(nn.Module):
2     def __init__(self):
3         super(CNN, self).__init__()
4         self.conv1 = nn.Conv2d(1, 16, kernel_size=2)
5         self.conv2 = nn.Conv2d(16, 32, kernel_size=2)
6         self.conv3 = nn.Conv2d(32, 64, kernel_size=2)
7         self.dropout = nn.Dropout(0.2)
8         self.fc1 = nn.Linear(256, 10)
9
10    def forward(self, x):
11        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
12        x = self.dropout(x)
13        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
14        x = self.dropout(x)
15        x = F.max_pool2d(F.relu(self.conv3(x)), 2)
16        x = x.view(-1, 256)
17        x = self.fc1(x)
18        return F.log_softmax(x, dim=1)
```

### 3. CNN 신경망 구현: 생성자

- 출력층: **Linear()**

```
1 class CNN(nn.Module):
2     def __init__(self):
3         super(CNN, self).__init__()
4         self.conv1 = nn.Conv2d(1, 16, kernel_size=2)
5         self.conv2 = nn.Conv2d(16, 32, kernel_size=2)
6         self.conv3 = nn.Conv2d(32, 64, kernel_size=2)
7         self.dropout = nn.Dropout(0.2)
8         self.fc1 = nn.Linear(256, 10)
9
10    def forward(self, x):
11        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
12        x = self.dropout(x)
13        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
14        x = self.dropout(x)
15        x = F.max_pool2d(F.relu(self.conv3(x)), 2)
16        x = x.view(-1, 256)
17        x = self.fc1(x)
18        return F.log_softmax(x, dim=1)
```



### 3. CNN 신경망 구현: 순전파

```
1 class CNN(nn.Module):
2     def __init__(self):
3         super(CNN, self).__init__()
4         self.conv1 = nn.Conv2d(1, 16, kernel_size=2)
5         self.conv2 = nn.Conv2d(16, 32, kernel_size=2)
6         self.conv3 = nn.Conv2d(32, 64, kernel_size=2)
7         self.dropout = nn.Dropout(0.2)
8         self.fc1 = nn.Linear(256, 10)
9
10    def forward(self, x):
11        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
12        x = self.dropout(x)
13        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
14        x = self.dropout(x)
15        x = F.max_pool2d(F.relu(self.conv3(x)), 2)
16        x = x.view(-1, 256)
17        x = self.fc1(x)
18        return F.log_softmax(x, dim=1)
```



### 3. CNN 신경망 구현: 모델 모양

```
CNN(  
    (conv1): Conv2d(1, 16, kernel_size=(2, 2), stride=(1, 1))  
    (conv2): Conv2d(16, 32, kernel_size=(2, 2), stride=(1, 1))  
    (conv3): Conv2d(32, 64, kernel_size=(2, 2), stride=(1, 1))  
    (dropout): Dropout(p=0.2, inplace=False)  
    (fc1): Linear(in_features=256, out_features=10, bias=True)  
)
```

```
1 class CNN(nn.Module):  
2     def __init__(self):  
3         super(CNN, self).__init__()  
4         self.conv1 = nn.Conv2d(1, 16, kernel_size=2)  
5         self.conv2 = nn.Conv2d(16, 32, kernel_size=2)  
6         self.conv3 = nn.Conv2d(32, 64, kernel_size=2)  
7         self.dropout = nn.Dropout(0.2)  
8         self.fc1 = nn.Linear(256, 10)  
9  
10    def forward(self, x):  
11        x = F.max_pool2d(F.relu(self.conv1(x)), 2)  
12        x = self.dropout(x)  
13        x = F.max_pool2d(F.relu(self.conv2(x)), 2)  
14        x = self.dropout(x)  
15        x = F.max_pool2d(F.relu(self.conv3(x)), 2)  
16        x = x.view(-1, 256)  
17        x = self.fc1(x)  
18        return F.log_softmax(x, dim=1)
```

```
1 cnn = CNN()  
2 # 손실 함수와 옵티마이저 정의  
3 criterion = torch.nn.CrossEntropyLoss()  
4 optimizer = torch.optim.Adam(cnn.parameters(), lr=0.001)  
5  
6 print(cnn)
```

### 3. CNN 신경망 구현: 손실함수

- 그외 다양한 손실함수
- **nn.MSELoss**
- **nn.NLLLoss**
- **nn.PoisonNLLLoss**
- **nn.KLDivLoss**
- ...

```
1 cnn = CNN()
2 # 손실 함수와 옵티마이저 정의
3 criterion = torch.nn.CrossEntropyLoss()
4 optimizer = torch.optim.Adam(cnn.parameters(), lr=0.001)
5
6 print(cnn)
```

### 3. CNN 신경망 구현: 학습

- `train()`

```
1 epochs = 10
2 cnn.train()
3 for epoch in range(epochs):
4     avg_loss = 0
5
6     for X, Y in data_loader:
7
8         optimizer.zero_grad()
9         y_hat = cnn(X)
10        loss = criterion(y_hat, Y)
11        loss.backward()
12        optimizer.step()
13
14        avg_loss += loss.item() / len(data_loader)
15
16    print('|Epoch: {:3d}| loss = {:.7f}'.format(epoch + 1, avg_loss))
```

### 3. CNN 신경망 구현: 평가

- `eval()`
- `no_grad()`

```
1 epochs = 10
2 cnn.train()
3 for epoch in range(epochs):
4     avg_loss = 0
5
6     for X, Y in data_loader:
```

```
1 cnn.eval()
2 with torch.no_grad():
3     X_test = mnist_test.data.view(len(mnist_test), 1, 28, 28).float()
4     Y_test = mnist_test.targets
5
6     prediction = cnn(X_test)
7     correct_prediction = torch.argmax(prediction, 1) == Y_test
8     accuracy = correct_prediction.float().mean()
9     print('Accuracy:', accuracy.item())
```

### 3. CNN 신경망 구현: 평가

```
1 epochs = 10
2 cnn.train()
3 for epoch in range(epochs):
4     avg_loss = 0
5
6     for X, Y in data_loader:
```

```
1 cnn.eval()
2 with torch.no_grad():
3     X_test = mnist_test.data.view(len(mnist_test), 1, 28, 28).float()
4     Y_test = mnist_test.targets
5
6     prediction = cnn(X_test)
7     correct_prediction = torch.argmax(prediction, 1) == Y_test
8     accuracy = correct_prediction.float().mean()
```

Accuracy: 0.9607999920845032

## 4. 기계학습 모델: YOLO

---

- 실시간 물체인식에 사용
- **YOLO: You Only Look Once**
- 1초에 45장 이미지 분석 (45 fps)



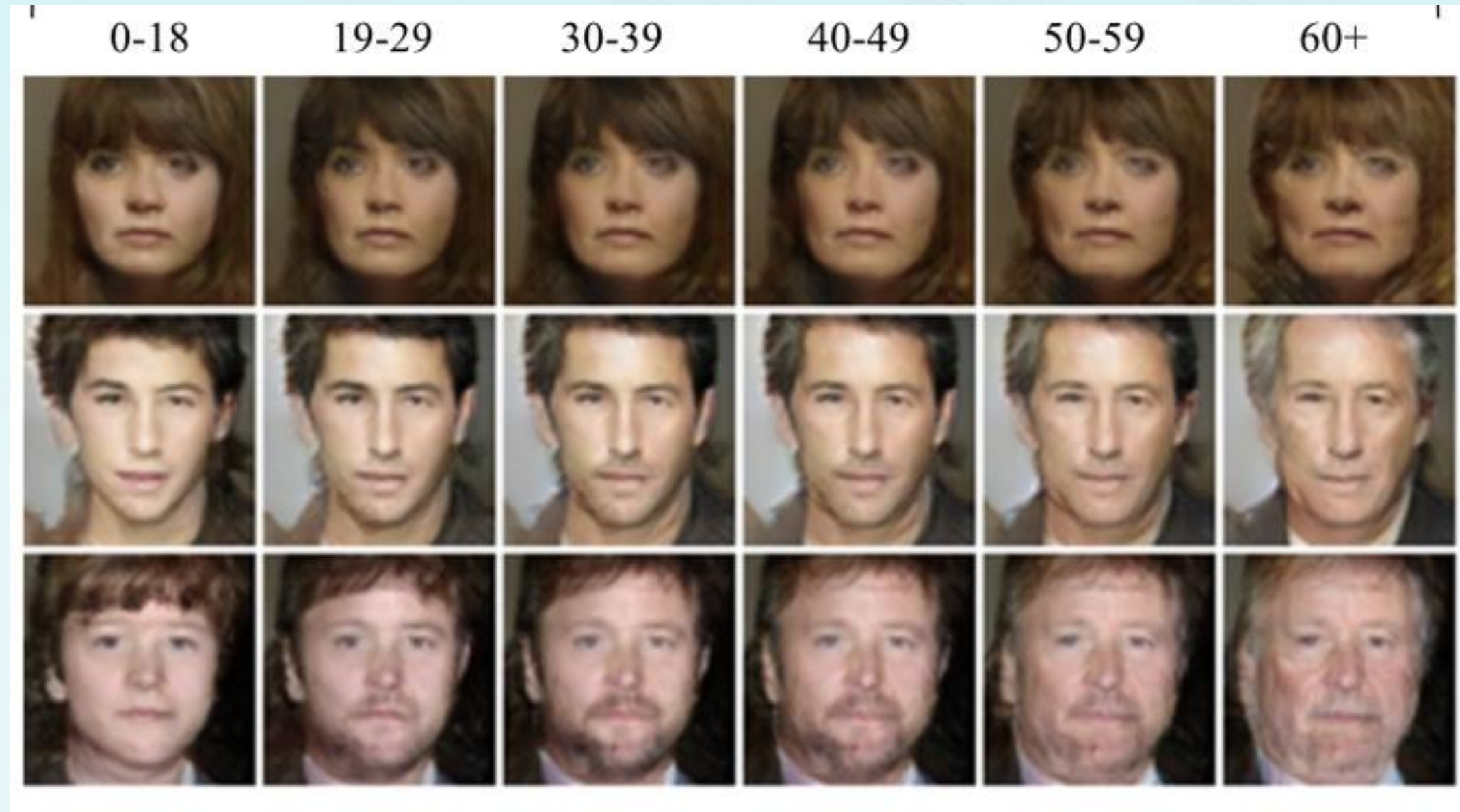
## 4. 기계학습 모델: GAN – 생성적 적대 신경망

---

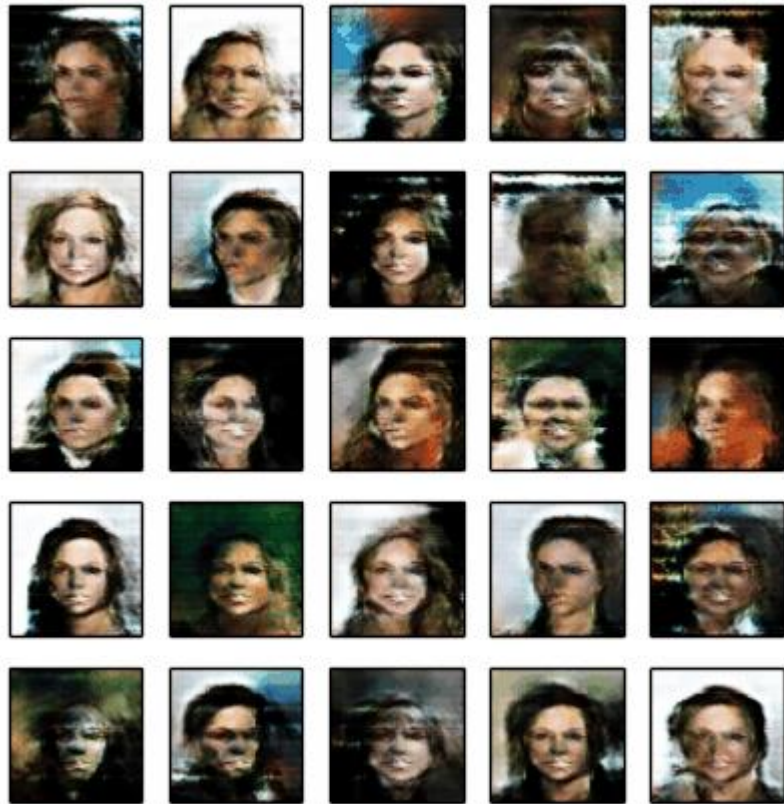
- **Generative Adversarial Networks**
- 이미지 직접 생성
- 예 : 현재 얼굴 사진 학습하여 미래 얼굴 이미지 생성



## 4. 기계학습 모델: GAN – 생성적 적대 신경망



## 4. 기계학습 모델: GAN – 생성적 적대 신경망



Epoch 1

## 4. 기계학습 모델: GAN

---

# 오픈 프레임워크 - 파이토치

---

- 학습 내용
  - 기계학습을 위한 오픈 프레임워크 **PyTorch**
  - **CNN**을 이용한 **MNIST** 데이터셋 분석
  - 기계학습 모델 **YOLO**
  - 기계학습 모델 **GAN**

# ML Open Framework PyTorch

파이썬으로 배우는 기계학습

한동대학교  
김영섭 교수