PyTorch Seminar

Day1. Training Code Structure

Duhyeon Kim / May 2025

PyTorch

PyTorch Seminar

Day1. Training Code Structure

Duhyeon Kim / May 2025

PyTorch

Contents

NN, ML algorithm, Training code structure

A. Convolutional Neural Networks

B. Back-Propagation

C. PyTorch Training Directory composition (UEGAN paper code)

D. Dataloader

E. Model Definition (nn.Module)

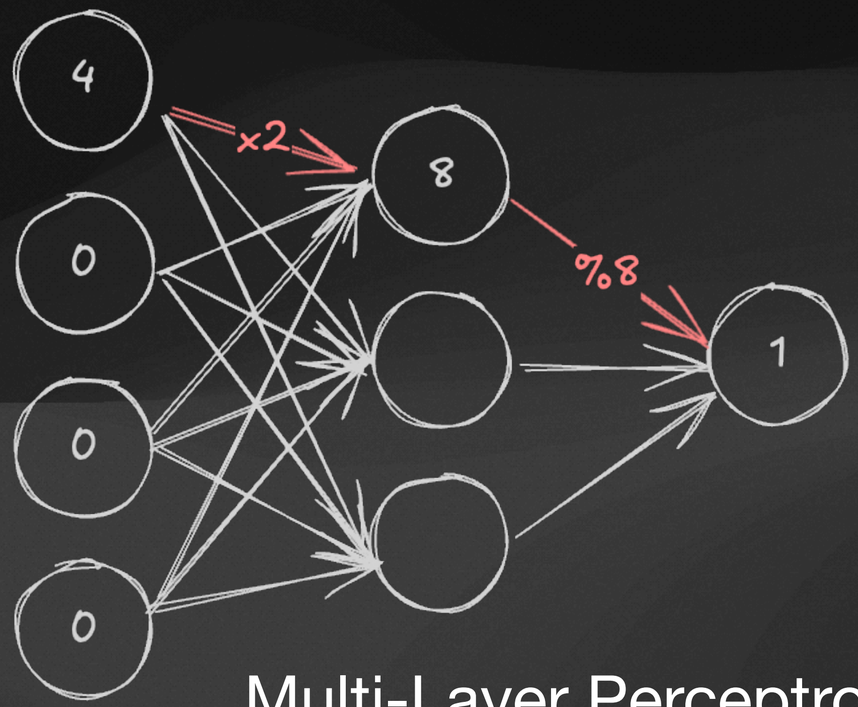
F. Loss function

G. Optimizer

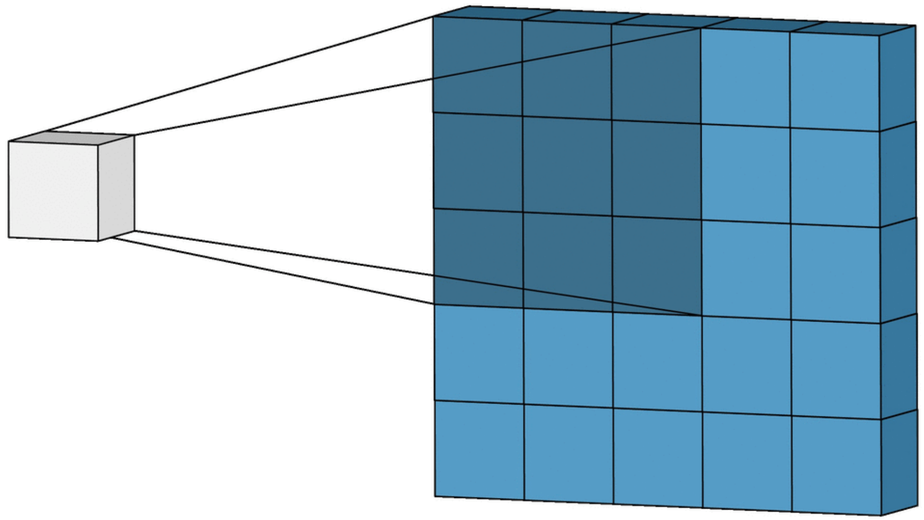
PyTorch

Neural Network Layers

Multi Layer Perceptron / Convolutional Neural Network



Multi-Layer Perceptrons



Convolutional Layer

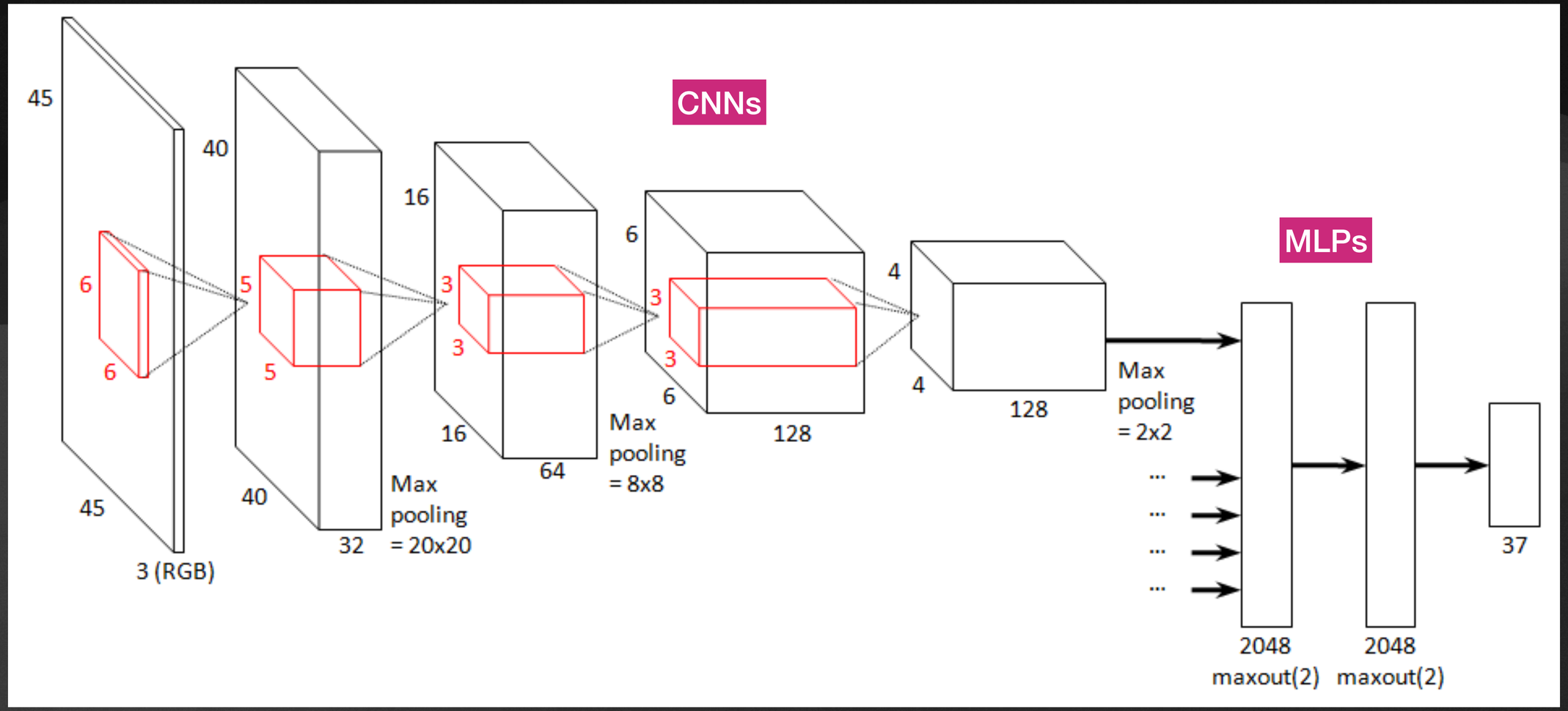
GEMM(General Matrix-Matrix Multiplication)

+ non-linear activation Function

PyTorch

Model (combi of NNs)

Simple Image classifier

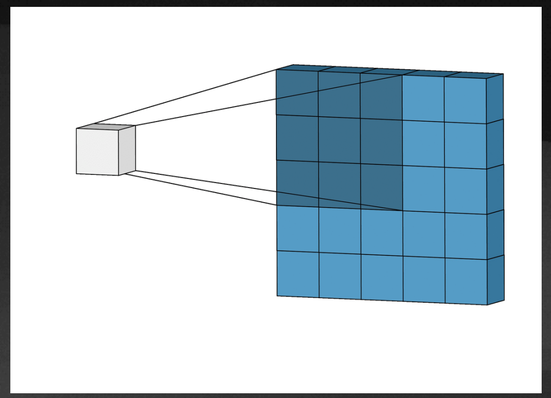


Backpropagation

Gradient Descent

Ground Truth (y)

Og



```
self.conv = nn.Conv2d(
    in_channels=1,
    out_channels=1,
    kernel_size=64,
    stride=1,
    padding=kernel_size // 2,
    bias=False
)
```

Model output

We need to **reduce** the **difference**

Backpropagation

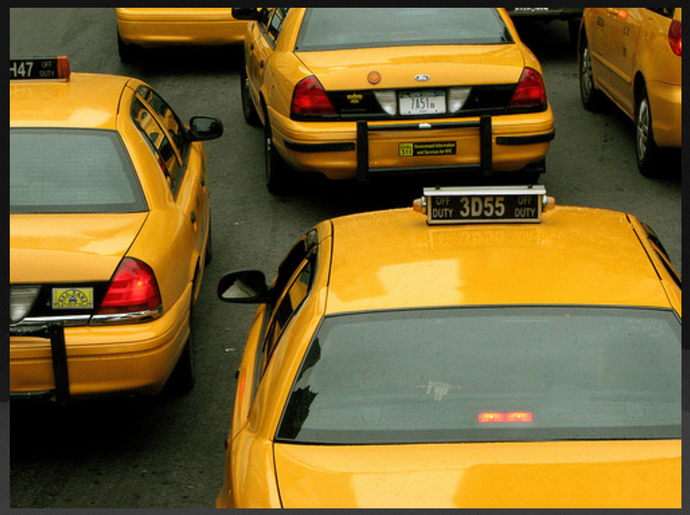
Gradient Descent

variable (y) variable (x) Fixed

$$\text{loss} = (\hat{y} - y)^2$$

🤔 How can we **Magnify the difference?**

Ground Truth (y)



Model output ($\hat{y}$)

We need to **reduce** the **difference**

Backpropagation

Simple Example

Difference

$$y = x^2$$

$x \rightarrow \boxed{x'}$ to reduce diff.

$$x' \leftarrow x - \alpha \frac{\partial L}{\partial x}$$

We need to **reduce** the **difference**

Backpropagation

Gradient Descent

variable (y) variable (x) Fixed

$$\text{loss} = (\hat{y} - y)^2$$

$$\theta = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \ldots\}$$

$$loss = (\hat{y} - y)^2$$

$$\hat{y} = f_\theta(x)$$

Combi of Neural Networks

Ground Truth (y)

Model output ($\hat{y}$)

$$\frac{\partial \text{loss}}{\partial \theta} = \frac{\partial \text{loss}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta}$$

Fixed

$$\frac{\partial \text{loss}}{\partial \hat{y}} = 2(\hat{y} - y) \cdot \frac{\partial f_\theta(x)}{\partial \theta}$$

Fixed

$$\theta := \theta - \alpha \cdot 2(\hat{y} - y) \cdot \frac{\partial f_\theta(x)}{\partial \theta}$$

Learning Rate

We need to **reduce** the **difference**

Backpropagation

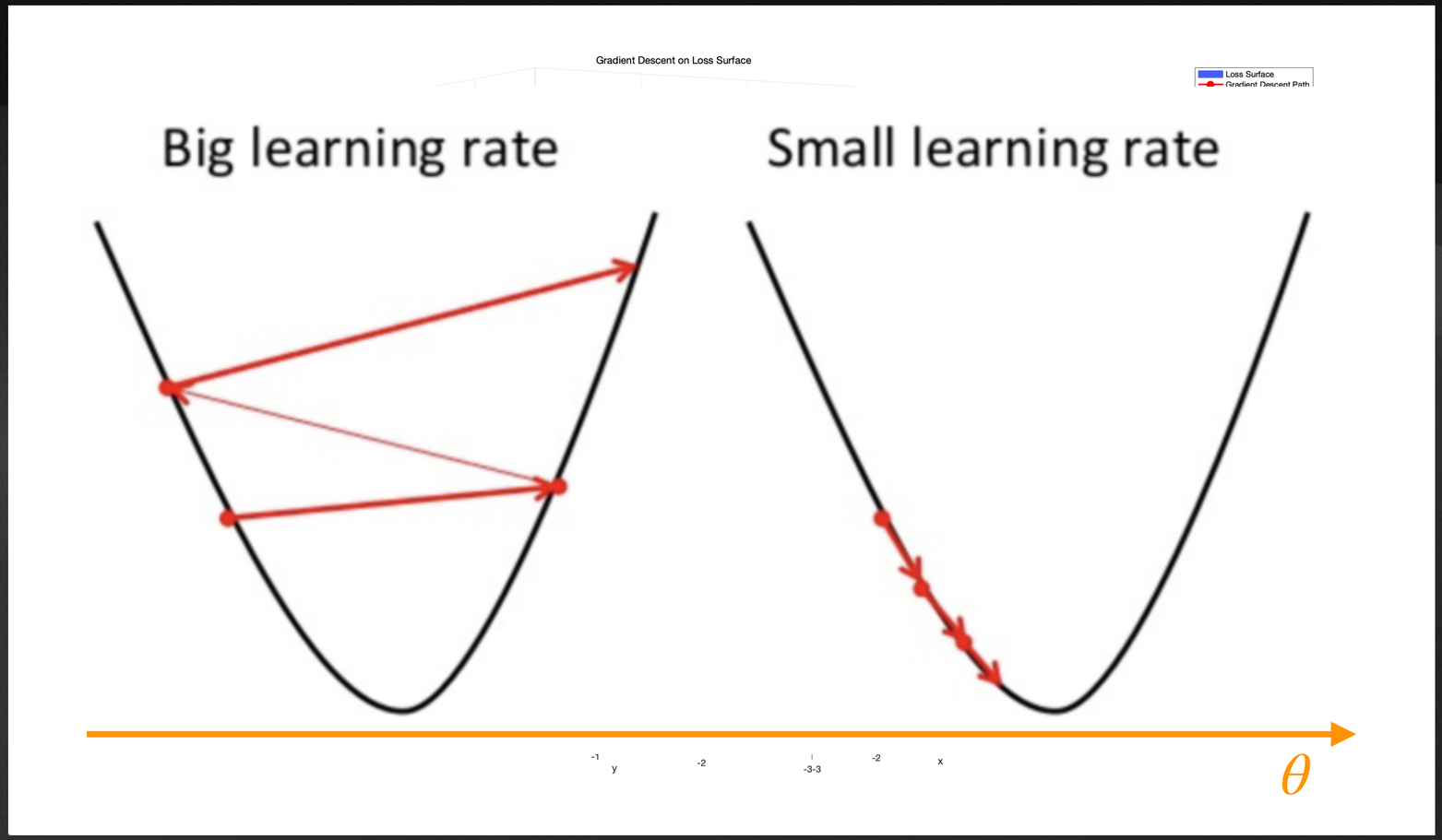
Gradient Descent

$$\theta := \theta - \alpha \cdot 2(\hat{y} - y) \cdot \frac{\partial f_\theta(x)}{\partial \theta}$$

Fixed

Learning Rate

$$\theta = \{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \ldots\}$$



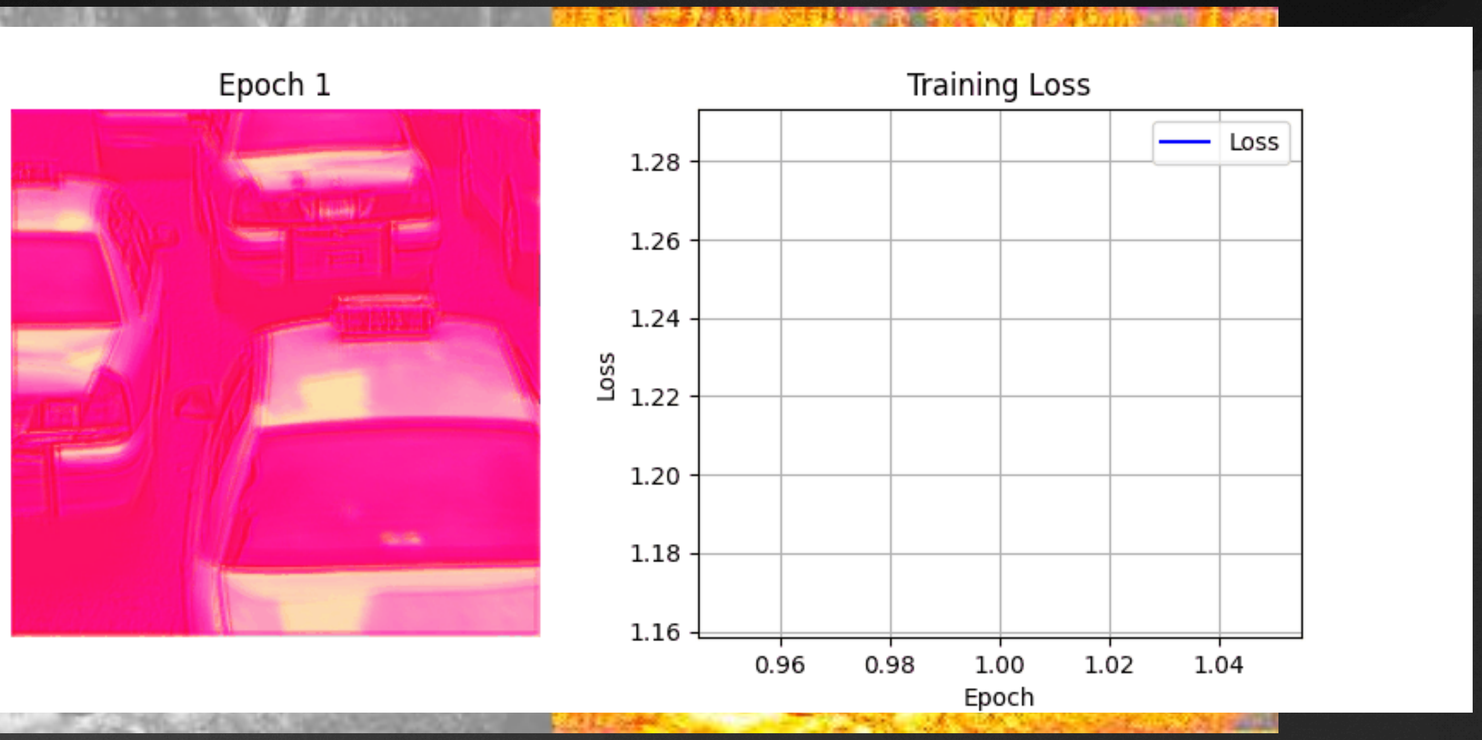
We need to **reduce** the **difference**

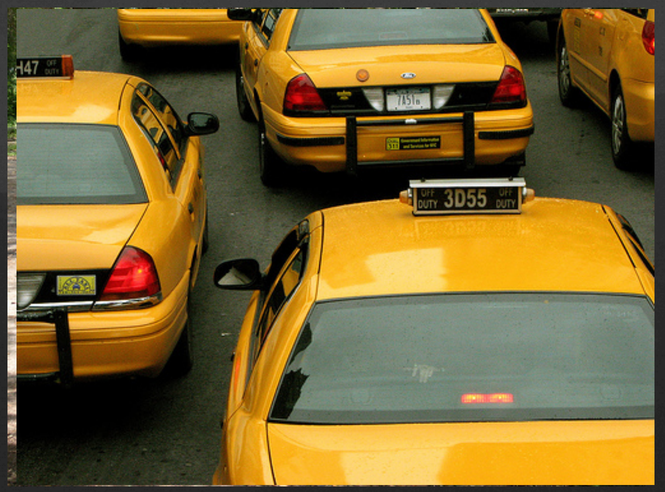
https://builtin.com/data-science/gradient-descent

Backpropagation

When optimized

But when Inferenced to new image?!



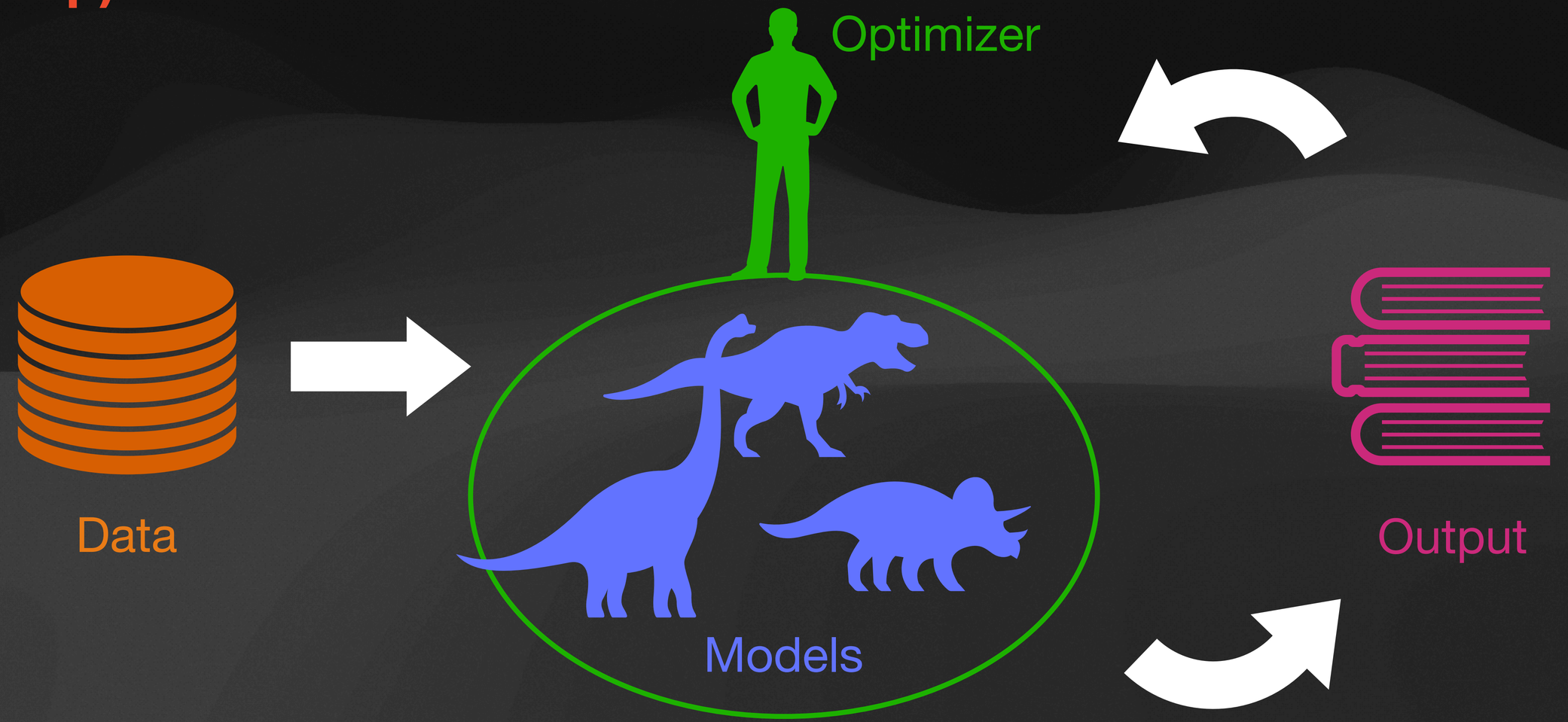


To have General Representation

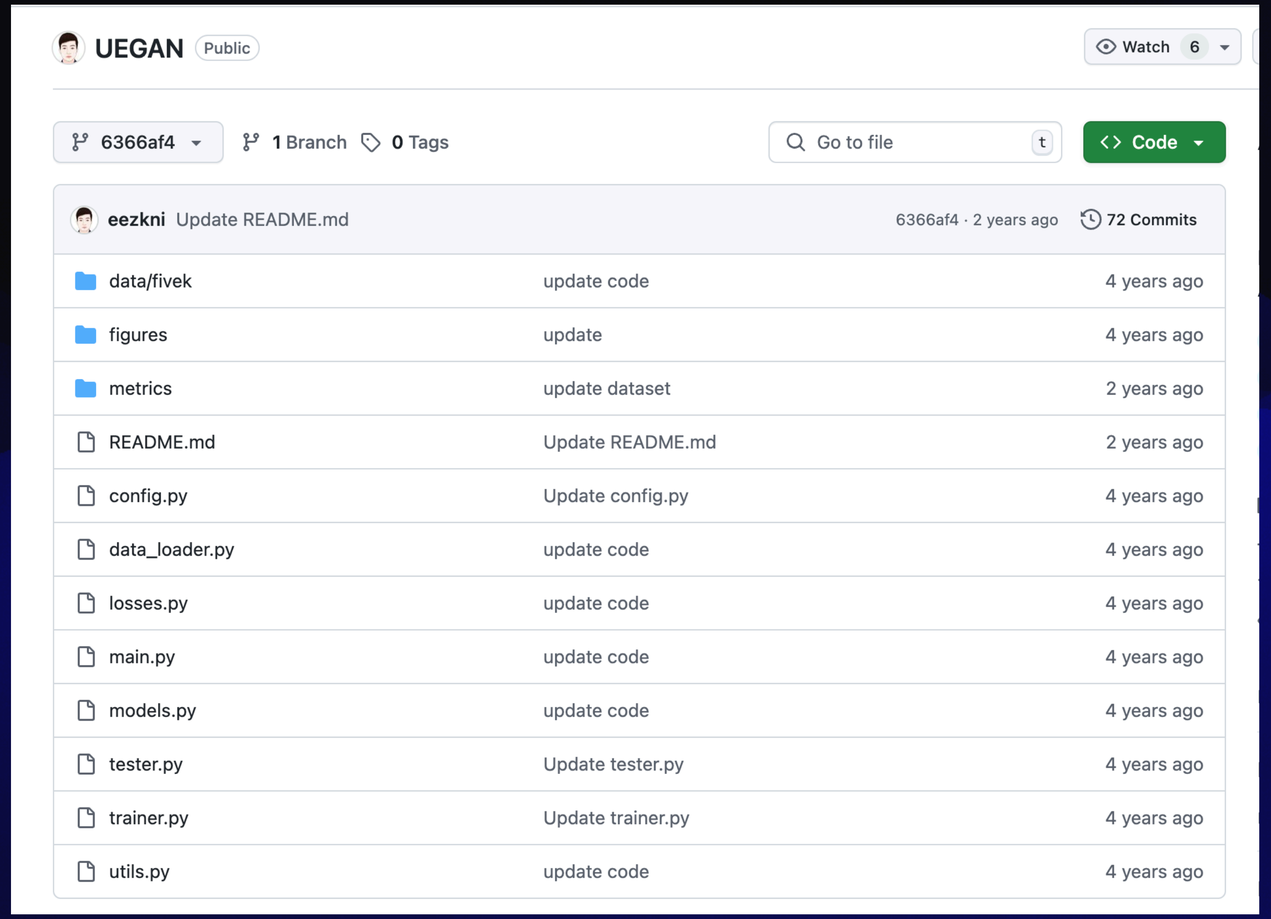
So Large Dataset Is NECESSARY

Code Structure

(Recap) Abstract View

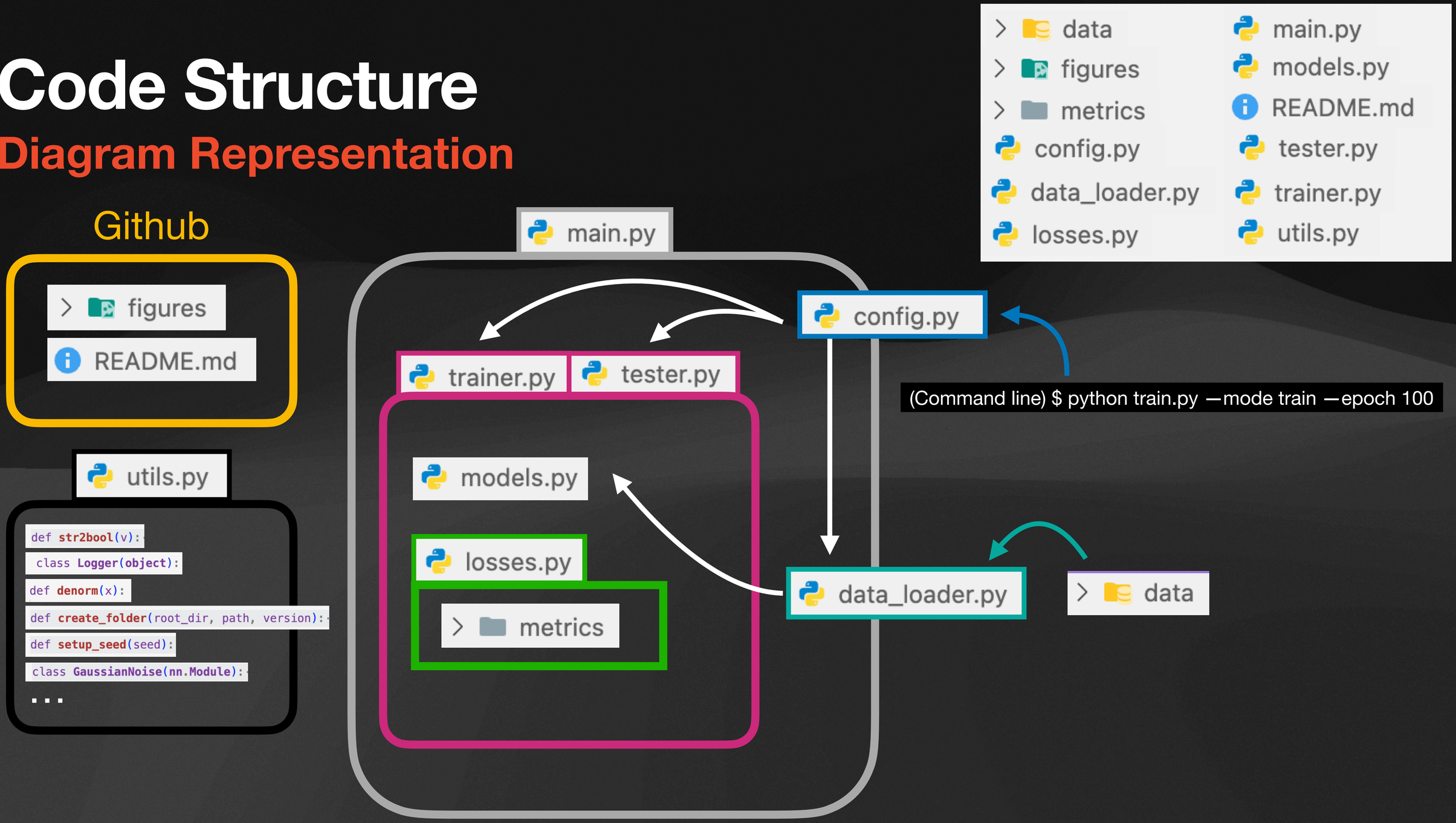


Code Structure



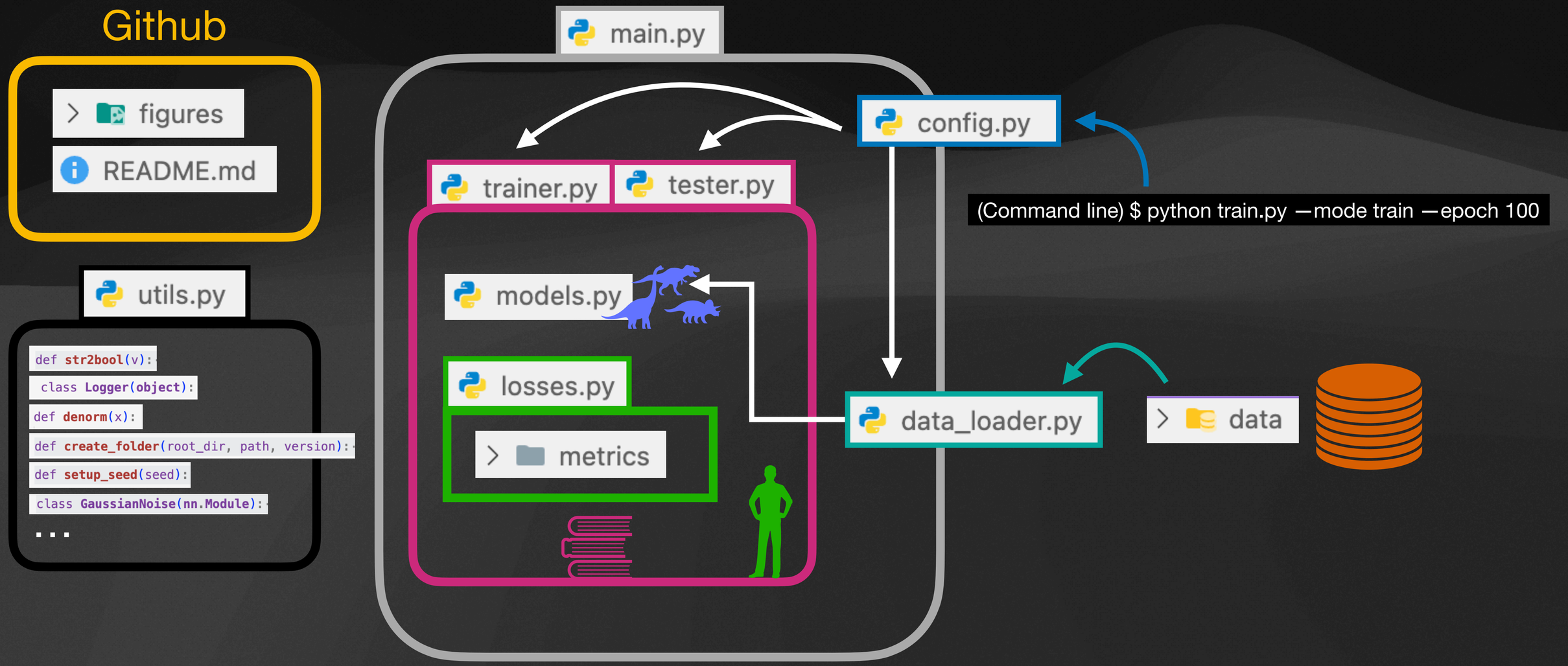
Code Structure

Diagram Representation



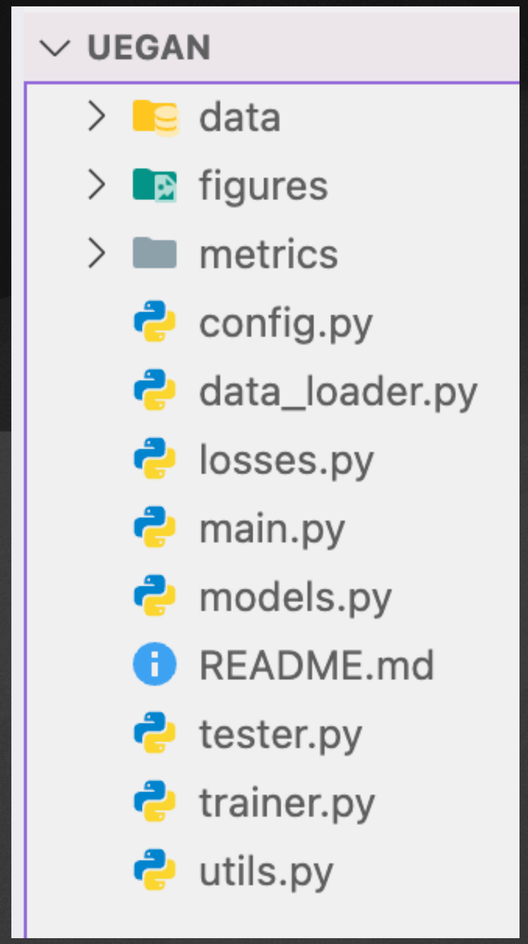
Code Structure

Diagram Representation



Code Structure

main.py



Where real data (e.g., .jpg, .png) is stored.

Stores figures (e.g., graphs, images) generated by the project.

Contains metric functions (e.g., VGG perceptual loss) for evaluation or loss calculation.

Defines and parses configuration settings from the command line using argparse.

Loads and preprocesses data for model input.

Defines loss functions for training.

Entry point that manages the main workflow and initializes the environment.

Defines model architectures as subclasses of nn.Module.

Provides an overview and usage instructions for the project on GitHub.

Manages and executes model evaluation.

Manages and executes model training.

Contains common utility functions used across the project.

Let’s see each code block

dataloader, model, loss, optimizer

Simple Python Grammar

Class

```python
class Car:
    def __init__(self, color):
        self.color = color

    def drive(self):
        print(f"{self.color} 자동차가 달립니다!")

car1 = Car("빨간색")
car2 = Car("보라")
```

Class : 설계도
Instance : 완제품 (class를 기반으로 한)

__init__ : instance가 생성되었을 때 실행, 초기화 역할

method : class가 가지는 함수들

self :
클래스 내부에서 자기 자신(인스턴스)을 가리키는 변수

Simple Python Grammar

Class

```python
class Car:
    def __init__(self, color):
        self.color = color

    def __call__(self, time):
        print(f"{self.color} 자동차가 {time}초 동안 달립니다!")

    def __len__(self):
        return len(self.color)

    def drive(self):
        print(f"{self.color} 자동차가 달립니다!")

car1 = Car("빨간색")
car2 = Car("보라")

car1.drive()

car2(10)

len(car1)
```

✓ 0.0s

```
빨간색 자동차가 달립니다!
보라 자동차가 10초 동안 달립니다!

3
```

Special Methods:

- __init__ :
  auto-called when an instance is created
- __getitem__ :
  called when accessing with brackets [ ]
- __call__ :
  called when the instance is called like a function ()
- __len__ :
  called when using len() on the object

Dataloader

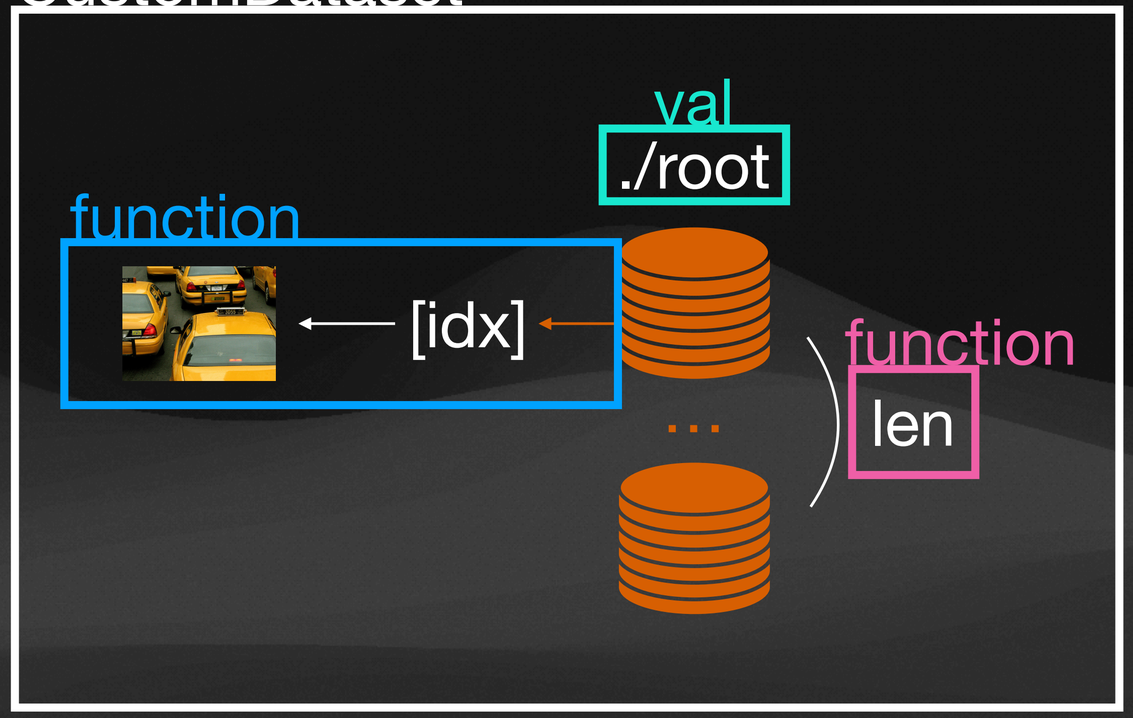
give me the briefest pytorch dataloader structure with specified root and custom dataset class

Here's the **briefest PyTorch** `DataLoader` **structure** using a **custom dataset class** and a specified `root` directory:

python 복사 편집

```python
from torch.utils.data import Dataset, DataLoader
import os

class CustomDataset(Dataset):
    def __init__(self, root):
        self.files = os.listdir(root)
        self.root = root

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        path = os.path.join(self.root, self.files[idx])
        x = open(path, 'rb').read()  # replace with actual processing
        return x

root = '/path/to/data'
dataset = CustomDataset(root)
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

What is class?
Ans : containing data and behavior



Dataloader

give me the briefest pytorch dataloader structure with specified root and custom dataset class

Here's the **briefest PyTorch** `DataLoader` **structure** using a **custom dataset class** and a specified `root` directory:

python 복사 편집

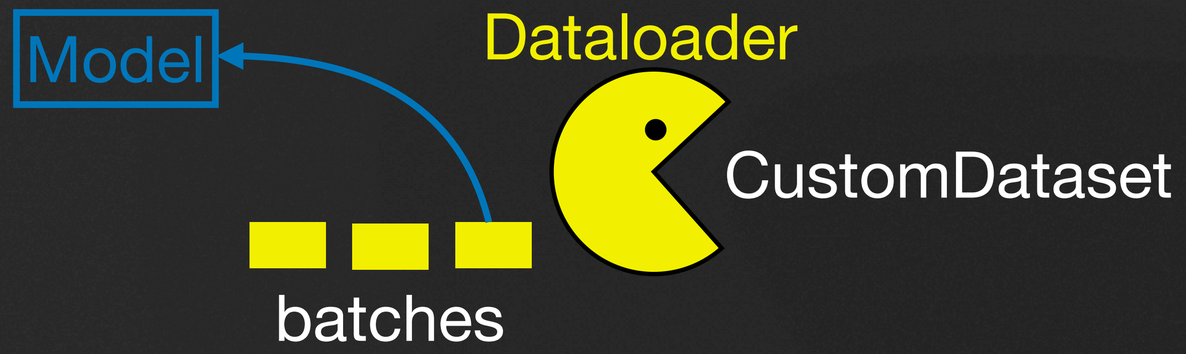
```python
from torch.utils.data import Dataset, DataLoader
import os

class CustomDataset(Dataset):
    def __init__(self, root):
        self.files = os.listdir(root)
        self.root = root

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        path = os.path.join(self.root, self.files[idx])
        x = open(path, 'rb').read()  # replace with actual processing
        return x

root = '/path/to/data'
dataset = CustomDataset(root)
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

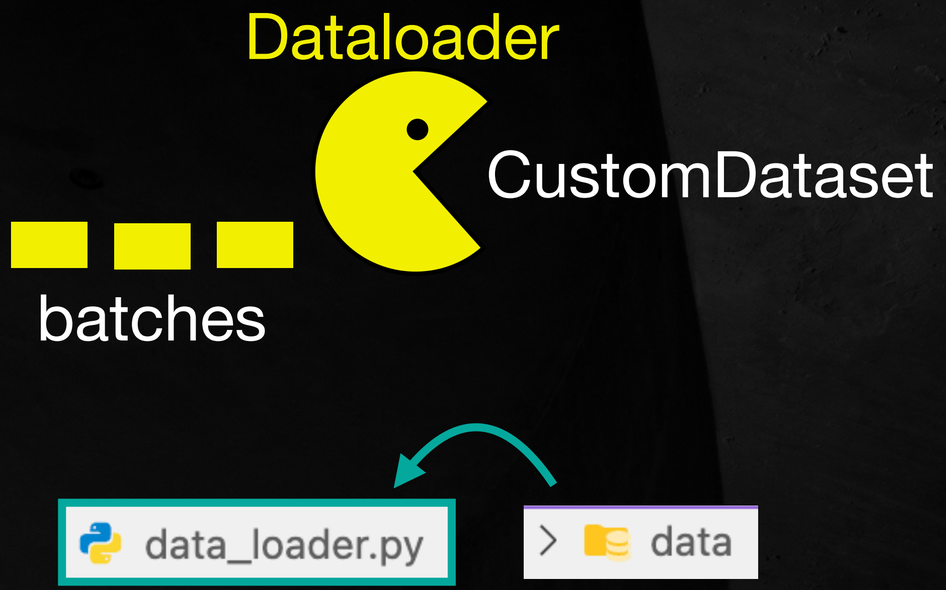
Why sperate Dataset and Datalaoder?

Key reason
= Lazy loading
= Memory Efficient

If we load whole dataset at once, memory efficiency is highly degraded.

So Dataloader manages the loading timing

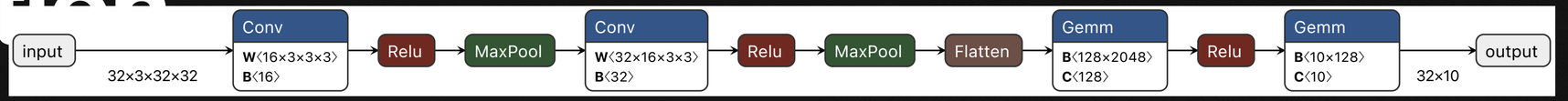




Code View (data_loader.py)

UEGAN Code

Model Definition



Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Linear

CLASS torch.nn.Linear(*in_features*, *out_features*, *bias=True*, *device=None*, *dtype=None*) [SOURCE]

Applies an affine linear transformation to the incoming data: $y = xA^T + b$.

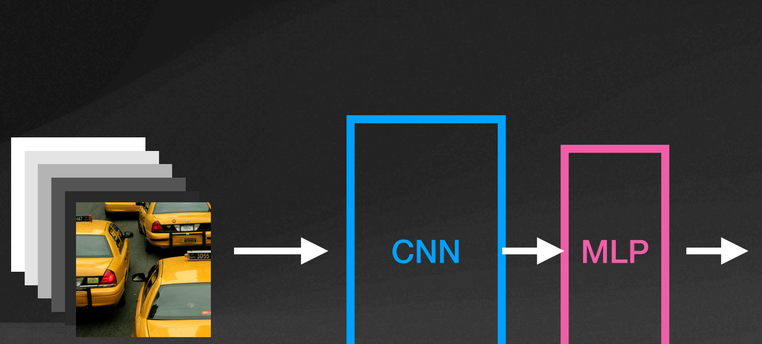
🔧 코드 예제

python 복사 편집

```python
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.mlp = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.cnn(x)
        x = self.mlp(x)
        return x
```

https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Model Definition



CNN

MLP

1
2
3
4
5
6
7
8
9
0

Model

Dataloader

CustomDataset

batches

https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

give me the briefest pytorch model def structure (with several mlp and cnn to classify 10 labels)

가장 **짧고 핵심만 담은** PyTorch 모델 구조예요 — CNN + MLP 조합으로 10-class 분류기입니다.

🔧 코드 예제

python 복사 편집

```python
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.mlp = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.cnn(x)
        x = self.mlp(x)
        return x
```

Linear

CLASS torch.nn.Linear(*in_features*, *out_features*, *bias=True*, *device=None*, *dtype=None*) [SOURCE]

Applies an affine linear transformation to the incoming data: $y = xA^T + b$.

Model Definition

What is class __init__ ?

__init__ is ALWAYS called when class instance is made.
So __init__ initialize the attributes of instance.

What is self.cnn / self.len ?

self = reference to the object itself
with . method we can define or access the attributes (variables)

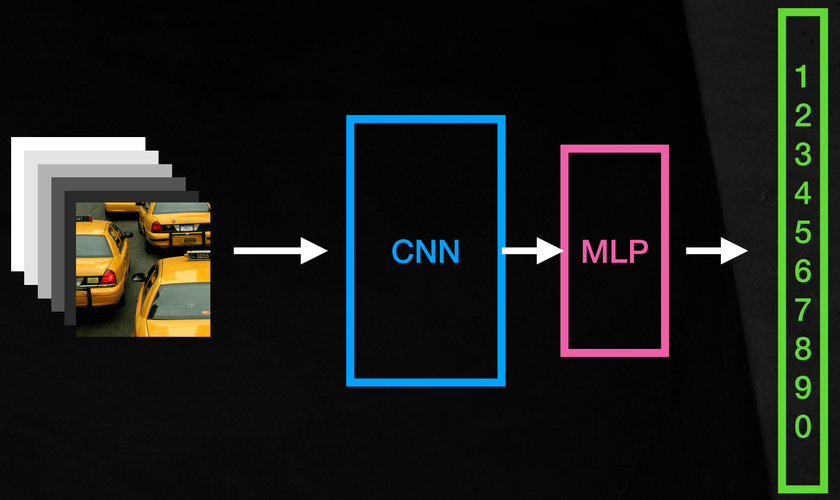
give me the briefest pytorch model def structure (with several mlp and cnn to classify 10 labels)

가장 **짧고 핵심만 담은** PyTorch 모델 구조예요 — CNN + MLP 조합으로 10-class 분류기입니다.

🔧 **코드 예제**

python 복사 편집

```python
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(3, 16, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.mlp = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.cnn(x)
        x = self.mlp(x)
        return x
```



models.py

Code View (models.py)

UEGAN Code

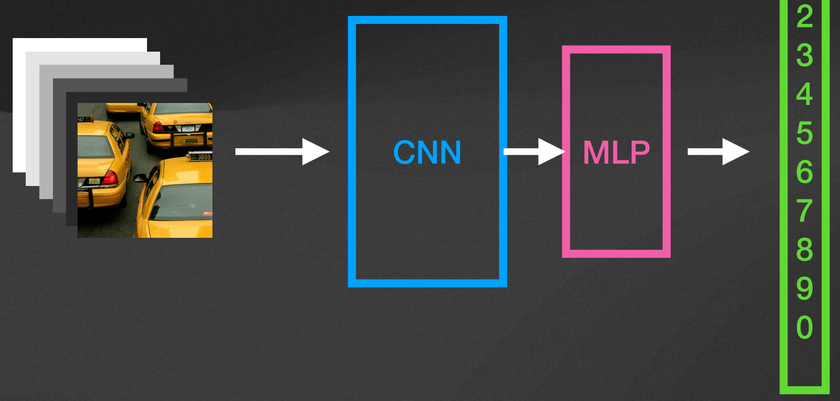
Loss function

Remember: Loss function is output dependent.

Classification - CrossEntropyLoss
Image Restoration - MSE, L1 loss
Image Generation - Adversarial loss, Perceptual loss
Text Generation - Negative Log Likelihood loss
Text Summury - Cross Entropy Loss, Coverage loss

...

Loss function



Predicted

```
{
    'cab': 0.08239,
    'dog': 0.04366,
    'cat': 0.09582,
    'car': 0.22994,
    'bird': 0.03967,
    'plane': 0.03967,
    'boat': 0.24323,
    'tree': 0.10801,
    'horse': 0.03135,
    'bike': 0.08626
}
```

Answer

```
{
    'cab': 1,
    'dog': 0,
    'cat': 0,
    'car': 0,
    'bird': 0,
    'plane': 0,
    'boat': 0,
    'tree': 0,
    'horse': 0,
    'bike': 0
}
```

Cross Entropy Loss

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

$$y_j = 1, \quad y_{i \neq j} = 0$$

$$L = -\log(\hat{y}_j)$$

Cross Entropy Loss 계산 결과: loss=2.4963

Loss function

```python
# 구성
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = MyModel().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

dataset = CustomDataset('/path/to/data')
loader = DataLoader(dataset, batch_size=32, shuffle=True)

# 1 training step
for x, y in loader:
    x, y = x.to(device), y.to(device)
    out = model(x)
    loss = criterion(out, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    break  # 1 step만 예시
```

__init__ 만 실행된 상태 / CrossEntropyLoss 객체(인스턴스) 생성

__call()__ 호출 / self.forward(out, y) 호출 -> return Loss tensor

Loss tensor의 backward() method 호출 -> backpropagation에 필요한 gradient 계산

Predicted

```
{
    'cab': 0.08239,
    'dog': 0.04366,
    'cat': 0.09582,
    'car': 0.22994,
    'bird': 0.03967,
    'plane': 0.03967,
    'boat': 0.24323,
    'tree': 0.10801,
    'horse': 0.03135,
    'bike': 0.08626
}
```

Answer

```
{
    'cab': 1,
    'dog': 0,
    'cat': 0,
    'car': 0,
    'bird': 0,
    'plane': 0,
    'boat': 0,
    'tree': 0,
    'horse': 0,
    'bike': 0
}
```

Cross Entropy Loss

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

$$y_j = 1, \quad y_{i \neq j} = 0$$

$$L = -\log(\hat{y}_j)$$

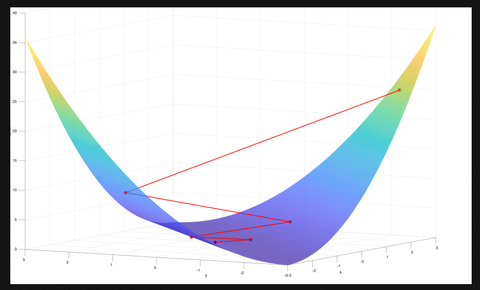
Cross Entropy Loss 계산 결과: loss=2.4963

PyTorch Seminar

Code View (losses.py)

UEGAN Code

Optimizer



```python
# 구성
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = MyModel().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1e-3)

dataset = CustomDataset('/path/to/data')
loader = DataLoader(dataset, batch_size=32, shuffle=True)

# 1 training step
for x, y in loader:
    x, y = x.to(device), y.to(device)
    out = model(x)
    loss = criterion(out, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    break  # 1 step만 예시
```

Parameter update 할 수식 지정

grad 계산 전 grad=0으로 초기화 (예방 코드)

계산한 grad 기반으로 param 값 업데이트

Cross Entropy Loss

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

$$y_j = 1, \quad y_{i \neq j} = 0$$

$$L = -\log(\hat{y}_j)$$

SGD optimizer

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_\theta \cdot lr$$

Optimizer

```python
# 구성
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = MyModel().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1e-3)

dataset = CustomDataset('/path/to/data')
loader = DataLoader(dataset, batch_size=32, shuffle=True)

# 1 training step
for x, y in loader:
    x, y = x.to(device), y.to(device)
    out = model(x)
    loss = criterion(out, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    break  # 1 step만 예시
```

Let's See only for `cnn.0.weight`

1) loss calculation

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

2) optimizer zero_grad

```
[After optimizer.zero_grad()] Parameter name: cnn.0.weight
  shape: torch.Size([16, 3, 3, 3])
  data (first 5 elements): tensor([-0.1312, -0.0523,  0.0782,  0.1126, -0.1212])
  grad (first 5 elements): None
```

3) back propagation

```
[After loss.backward()] Parameter name: cnn.0.weight
  shape: torch.Size([16, 3, 3, 3])
  data (first 5 elements): tensor([-0.1312, -0.0523,  0.0782,  0.1126, -0.1212])
  grad (first 5 elements): tensor([-0.0325,  0.0690, -0.0045, -0.1044,  0.0348])
```

4) optimizer step

```
[After optimizer.step()] Parameter name: cnn.0.weight
  shape: torch.Size([16, 3, 3, 3])
  data (first 5 elements): tensor([-0.1308, -0.0530,  0.0783,  0.1137, -0.1215])
  grad (first 5 elements): tensor([-0.0325,  0.0690, -0.0045, -0.1044,  0.0348])
```

updated

PyTorch Seminar

Code View (optimizer)

UEGAN Code

Summary

Overall Code Structure

CNN

MLP

1 2 3 4 5 6 7 8 9 0

Predicted

```
{
    'cab': 0.08239,
    'dog': 0.04366,
    'cat': 0.09582,
    'car': 0.22994,
    'bird': 0.03967,
    'plane': 0.03967,
    'boat': 0.24323,
    'tree': 0.10801,
    'horse': 0.03135,
    'bike': 0.08626
}
```

Answer

```
{
    'cab': 1,
    'dog': 0,
    'cat': 0,
    'car': 0,
    'bird': 0,
    'plane': 0,
    'boat': 0,
    'tree': 0,
    'horse': 0,
    'bike': 0
}
```

Dataloader

CustomDataset

batches

Cross Entropy Loss

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

$$y_j = 1, \quad y_{i \neq j} = 0$$

$$L = -\log(\hat{y}_j)$$

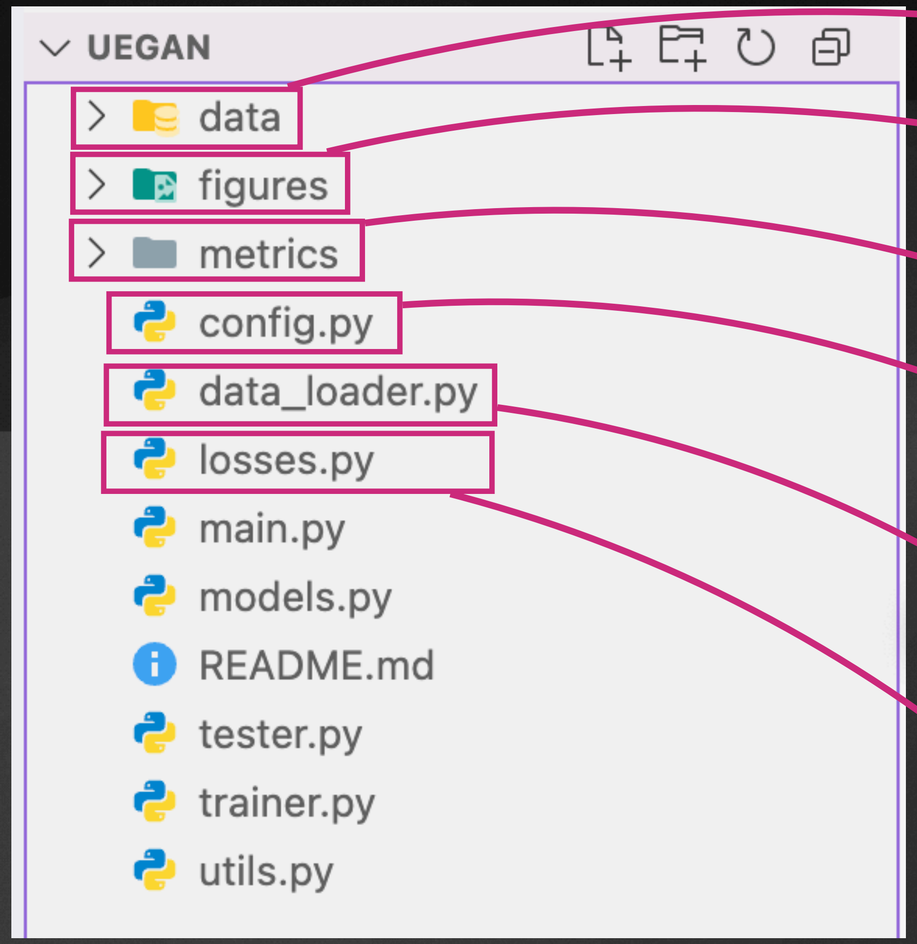
SGD optimizer

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta \cdot lr$$

Thank you

Code Structure

Directory View



Data(image) is stored

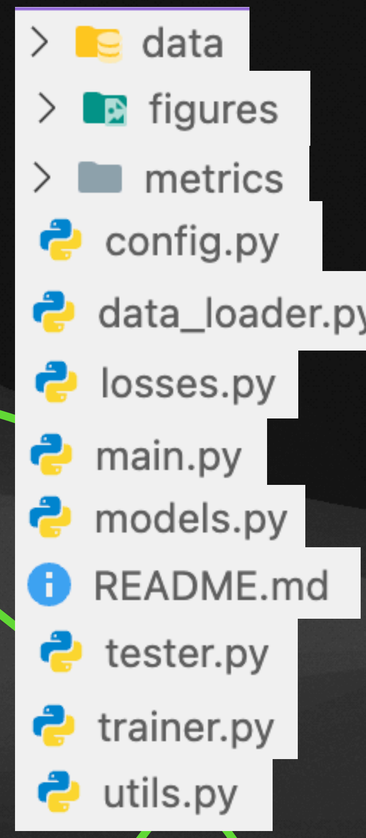
Paper Figures(e.g.) are stored

Loss function Metrics are stored

Command Line inputs
/ main function args are definced

[>data] is loaded and transformed
to fit input shape

Loss function are Defined



https://github.com/eezkni/UEGAN