ML/ Deep Learning Class PyTorch Tutorials

2020-07-29

SNUVL Lab

Contents

Contents

Installations

Anaconda

PyTorch

Python Packages

Google Colaboratory

Installations - Anaconda

- Python distributions by Continuum Analytics
  - `Including over 450 packages`
  - Managing python packages using virtual environments

- Install Anaconda
  - Official download website
  - Download python 3.6 version!!
  - Test: In the anaconda prompt for Windows users / a terminal window for Linux/Mac users,

```
$ conda --version
# if the following error occurs
$ -bash: conda:  command not found
# Type the following command
$ export PATH=~/anaconda3/bin:$PATH
```

PyTorch

- Open source deep learning platform
  - Facebook에서 Lua용 딥러닝 프레임워크 Torch를 Python library로 개발
  - NumPy 스타일의 직관적인 코딩 스타일: 이해하기 쉬움
  - 짧은 코딩 시간/디버깅 편의성: 연구용/프로토타입용 프로그램 개발에 용이

Installations - PyTorch

- Virtual environment
  - Manage packages by project
  - In the Anaconda prompt,

```
$ conda create --name <your own name> python=3.6
$ conda info --envs # 만들어진 가상 환경 확인
# 가상환경 시작 / 종료
$ conda activate <your own name> / conda deactivate
```

- Install PyTorch
  - PyTorch official website
  - Download none-gpu versions

```
# CPU version
$ conda install pytorch-cpu torchvision -c pytorch
# GPU version
$ conda install pytorch torchvision cudatoolkit=10.2 -c pytorch
```

  - Further Instructions are available in https://pytorch.org/get-started/locally/

Installations - Python packages

- Install required python packages
  - Matplotlib: Python visualization library
  - Jupyter Notebook: Interactive python computing web service

```
$ pip install matplotlib
$ pip install jupyter
```

Google Colaboratory

- 구글 클라우드 기반 Python Jupyter 환경
  - 무료입니다.
  - Pytorch, Matplotlib 등 필요한 파이썬 패키지가 모두 설치되어 있습니다.
  - GPU instance를 제공합니다.
  - https://colab.research.google.com

Tutorials

PyTorch Basic

Linear Regression

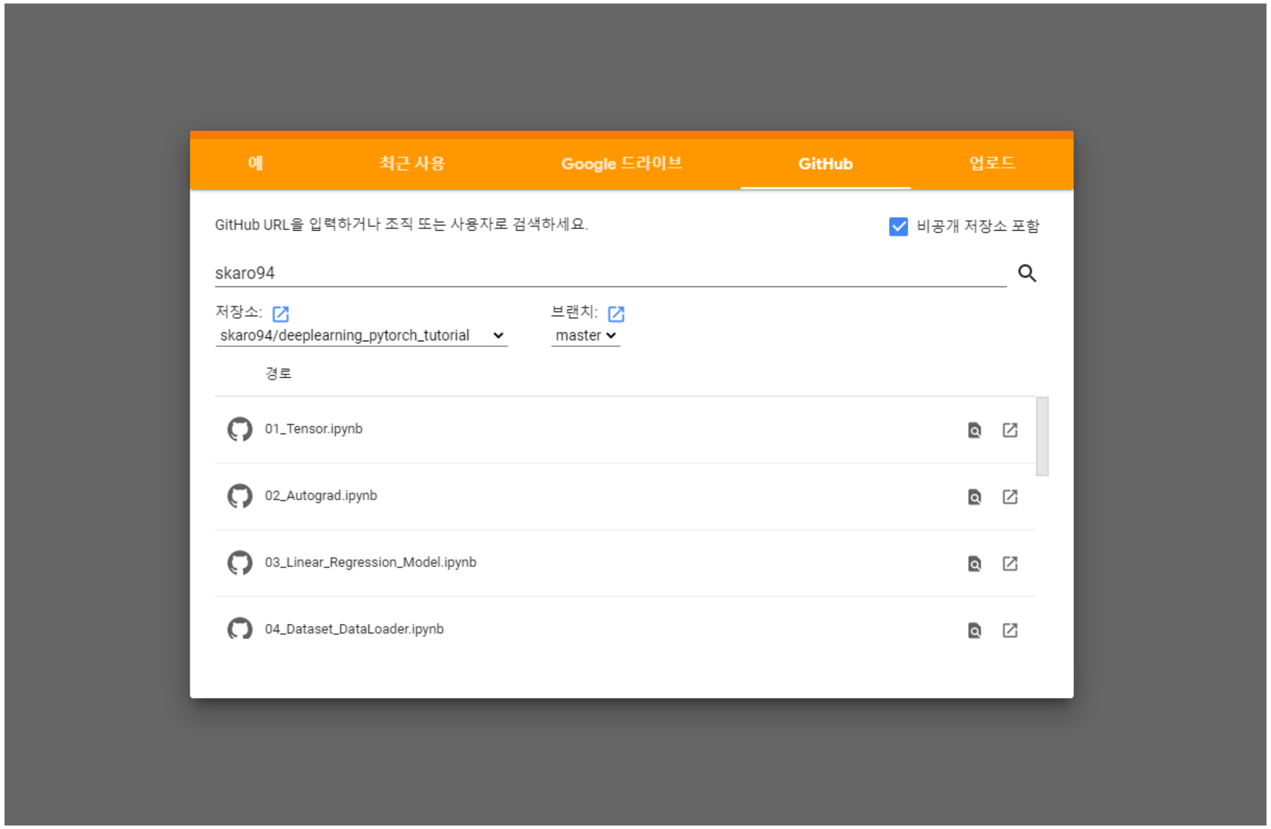
Logistic Regression

Feed-forward Neural Network

Convolutional Neural Network

Tutorials

- Tutorial Jupyter Notebooks
  - Sign in to Google
  - Go to colab.research.google.com/github/skaro94/deeplearning_pytorch_tutorial
  - Click the respective notebooks



GitHub URL을 입력하거나 조직 또는 사용자로 검색하세요.

☑ 비공개 저장소 포함

skaro94

저장소: skaro94/deeplearning_pytorch_tutorial

브랜치: master

경로

01_Tensor.ipynb

02_Autograd.ipynb

03_Linear_Regression_Model.ipynb

04_Dataset_DataLoader.ipynb

PyTorch Basic

- Tensor: 다차원 행렬
  - 모든 Pytorch 연산은 Tensor에서 이루어집니다.
  - 0차원 tensor: 스칼라
  - 1차원 tensor: 벡터
  - 2차원 tensor: 행렬
  - 3차원 이상 tensor: 다차원 행렬

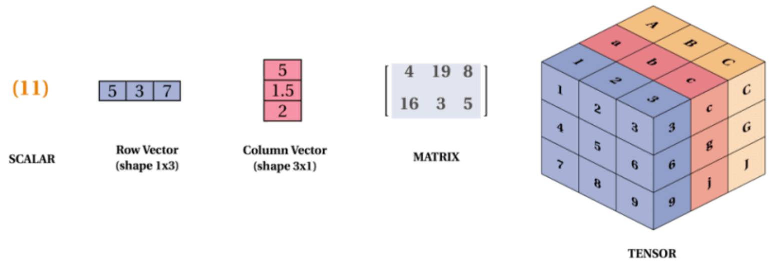


Image from https://dev.to/mmithrakumar/scalars-vectors-matrices-and-tensors-with-tensorflow-2-0-1f66

- Tensor 생성 함수

```
torch.Tensor(data, dtype=None, device=None,
requires_grad=False)
```

- data: tensor의 초기 데이터
- dtype: 생성될 tensor의 데이터 타입 (e.g. float, int, bool)
- device: 생성될 tensor가 배치될 디바이스 (CPU/GPU)
- requires_grad: gradient 계산을 위한 operation 기록 여부

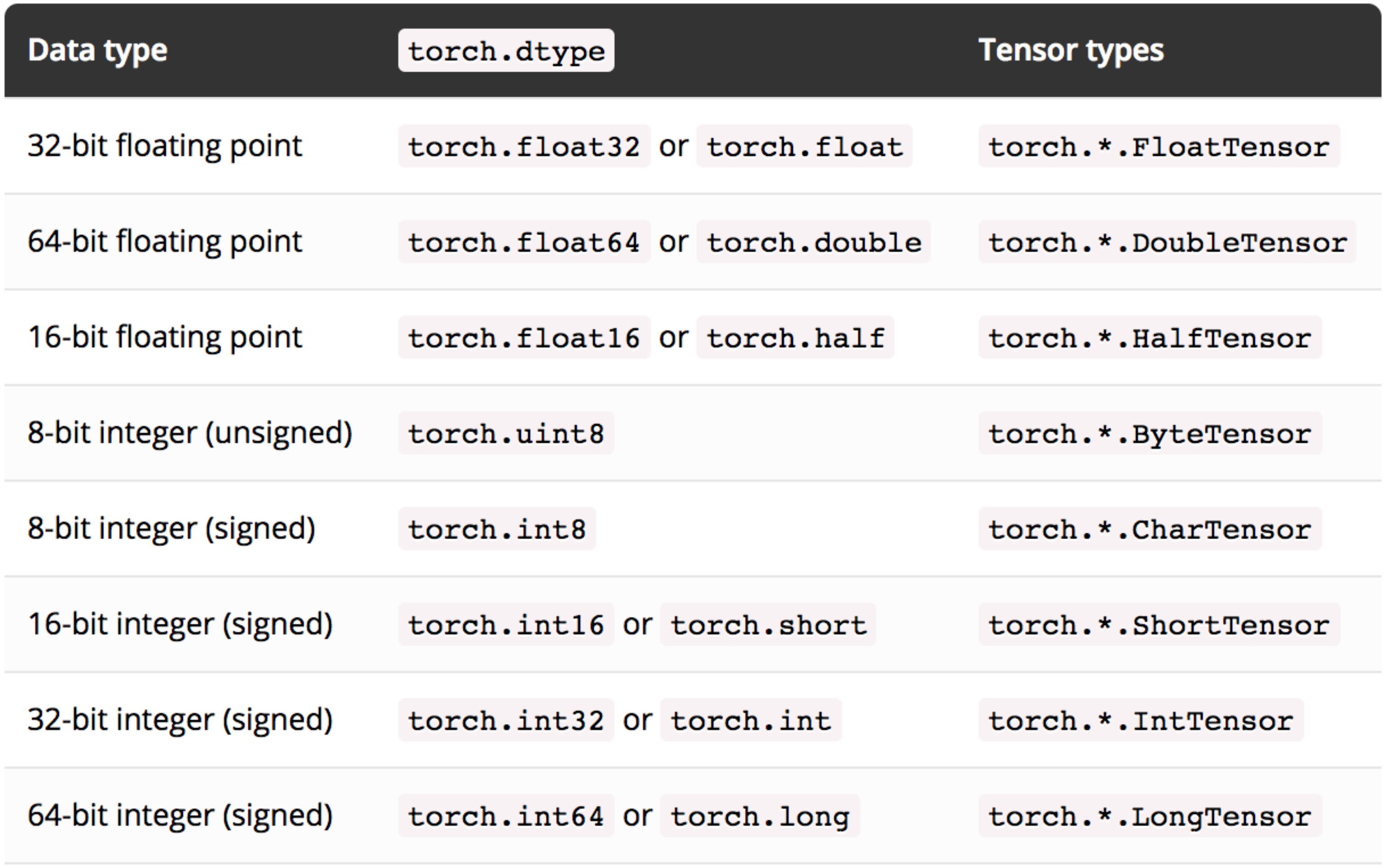
PyTorch Basic

- 여러 tensor 생성 함수
  - torch.ones
  - torch.zeros
  - torch.eye
  - torch.rand
  - torch.randn
  - ...

```
print(torch.ones(3,3), torch.zeros(3,3), torch.eye(3,3),
torch.rand(3,3), torch.randn(3,3), sep='\n')
```

PyTorch Basic

- Tensor 데이터타입
  - NumPy와 유사



Data type	`torch.dtype`	Tensor types
32-bit floating point	`torch.float32` or `torch.float`	`torch.*.FloatTensor`
64-bit floating point	`torch.float64` or `torch.double`	`torch.*.DoubleTensor`
16-bit floating point	`torch.float16` or `torch.half`	`torch.*.HalfTensor`
8-bit integer (unsigned)	`torch.uint8`	`torch.*.ByteTensor`
8-bit integer (signed)	`torch.int8`	`torch.*.CharTensor`
16-bit integer (signed)	`torch.int16` or `torch.short`	`torch.*.ShortTensor`
32-bit integer (signed)	`torch.int32` or `torch.int`	`torch.*.IntTensor`
64-bit integer (signed)	`torch.int64` or `torch.long`	`torch.*.LongTensor`

PyTorch Basic

- Tensor 디바이스
    - 연산 속도 상승을 위해 텐서 (및 딥러닝 모델)를 GPU 상에 배치할 수 있음

```
use_cuda = torch.cuda.is_available() # GPU 사용 가능 여부
device = torch.device("cuda" if use_cuda else "cpu")
```

- Requires_grad
    - 모델 파라미터 최적화를 위해 PyTorch에서는 Loss function에 대한 각 파라미터의 gradient를 저장할 수 있음
    - gradient의 크기는 파라미터의 크기와 동일하므로 모든 텐서의 gradient를 저장하면 메모리 이슈가 발생 가능
    - requires_grad option으로 gradient 저장 여부를 지정

```
x = torch.ones(1) # requires_grad=False (default)
print(x.requires_grad)
```

PyTorch Basic

- Tensor 사이즈
  - Tensor의 size를 tuple 형태로 반환

```
x = torch.randn(3, 2)
x.size() # (3, 2)
x.shape  # (3, 2)
```

- Indexing & Slicing
  - NumPy 스타일의 Indexing & Slicing 지원

```
x = torch.tensor([[1,1,1],[2,2,2],[3,3,3]])
print(x)
print(x[0:2,:])
print(x[0:3,0])
x[0,:] = torch.tensor([0,0,0])
print(x)
```

PyTorch Basic

- Reshaping

```
x  = torch.zeros(2, 1, 2)
print(x)

y = x.view(2,2) # Reahspe Tensor with view funciton
print(y, y.size(), sep='\n')

y = x.view(-1) # -1 is special "Don't Care" symbol
print(y, y.size(), sep='\n’)

y = x.reshape(-1) # reshape does not inputs contiguous

y = x.squeeze()  # remove dimension of 1
print(y, y.size(), sep='\n')

y.unsqueeze_(1)  # insert dimension of 1 / the methods with
the tailing '_' are in-place functions
print(y, y.size(), sep='\n')
```

PyTorch Basic

- Tensor Arithmetic
  - NumPy와 유사한 tensor arithmetic 제공
  - Element-wise operation

```
x1 = torch.tensor([[1,2,3],[4,5,6]], dtype=torch.float)
x2 = torch.tensor([[1,2,3],[4,5,6]], dtype=torch.float)

print(x1 + x2 , x1.add(x2), torch.add(x1, x2), sep='\n')
print(x1 * x2, x1.mul(x2), torch.mul(x1, x2), sep='\n')
print(x1 + 10, x1 * 10, sep='\n') # broadingcasting
print(x1.pow(2), torch.pow(x1, 2), x1 ** 2, sep='\n')
```

PyTorch Basic

- Tensor Arithmetic
  - NumPy와 유사한 tensor arithmetic 제공
  - Other math operations

```
# element-wise square root / logarithm to the base e, 10, 2
print(x1.sqrt(), x1.log(), x1.log10(), x1.log2(), sep='\n')


# summation / maximum / minimum / mean / standard
deviation / absolute value
print(x1.sum(), x1.max(), x.argmax(), x1.min(), x1.mean(),
x1.std(), x1.abs(), sep='\n')
```

PyTorch Basic

- Tensor Arithmetic
    - NumPy와 유사한 tensor arithmetic 제공
    - Matrix (Tensor) operations

```
# torch.mm(mat1, mat2) -> matrix multiplication
x1 = torch.randn(3,4)
x2 = torch.randn(4,5)

# torch.mv(mat1, vector) -> matrix vector multiplication
x = torch.randn(3,4)
v = torch.randn(4)

# torch.bmm(batch1, batch2) -> batch matrix multiplication
x1 = torch.randn(10,3,4)
x2 = torch.randn(10,4,5)

# or simply torch.matmul(batch1, batch2)
print(torch.matmul(x1, x2), torch.matmul(x, v),
torch.matmul(x1,x2).size(), sep='\n')
```

PyTorch Basic

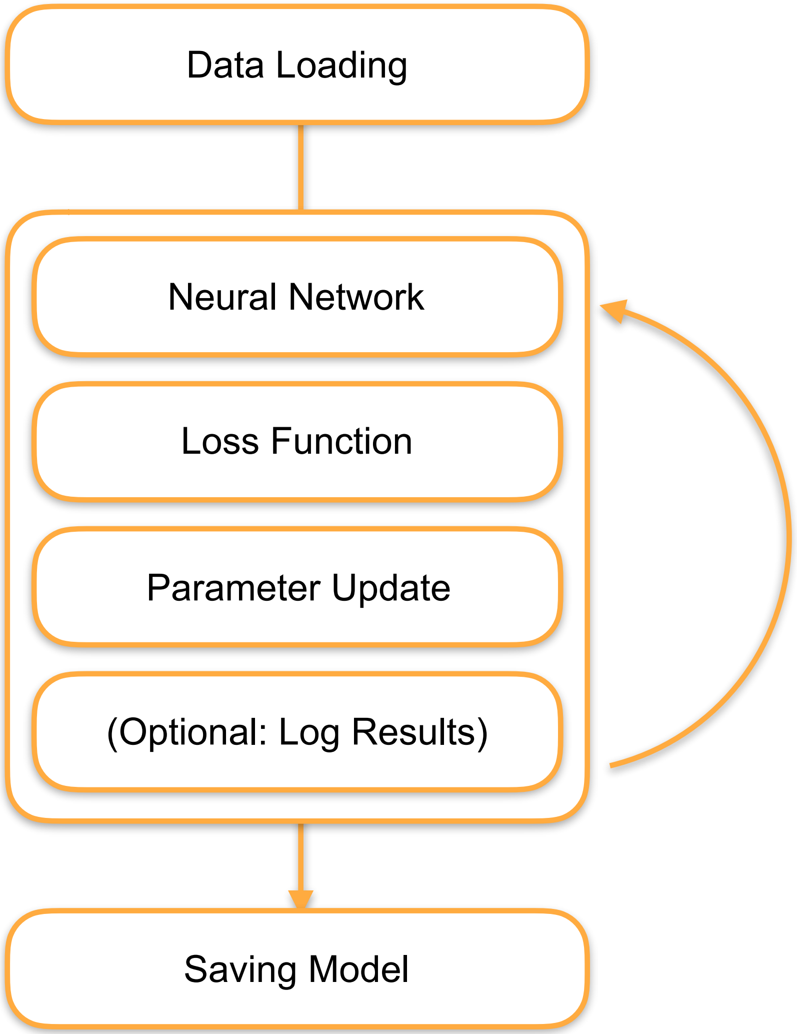
- Tensor Arithmetic
  - NumPy와 유사한 tensor arithmetic 제공
  - Matrix (Tensor) operations

```
# torch.dot(tensor1,tensor2) -> dot product of two tensor
x1 = torch.tensor([1,2,3,4])
x2 = torch.tensor([1,1,1,1])
print(torch.dot(x1, x2))

# torch.t(matrix) -> transposed matrix
x1 = torch.randn(3,4)
print(x1, x1.t(), sep='\n')

# torch.transpose(matrix, axis1, axis2) -> transposed tensor
print(x1, x1.transpose(0, 1), sep='\n')
```

Let’s Check the Codes!!

01_Tensor

Training a Neural Network



Linear Regression

- nn.Linear
- nn.MSELoss
- torch.optim.SGD
- tensorboardX.SummaryWriter
- torch.save

torch.nn

- Machine Learning/Deep Learning 모델 생성을 위한 패키지
    - 모델 생성을 위한 모든 빌딩 블록을 제공

- nn.XX
    - Linear transformation, Convolution, Softmax 등 다양한 텐서 연산 제공
    - Cross Entropy, Mean Squared Loss 등 다양한 로스 함수 제공
    - 학습 가능한 매개변수를 저장하기 위한 객체 지향형 모듈
    - 따로 지정하지 않은 경우 모든 매개변수가 랜덤 생성
    - **기본적으로 nn.XX 모듈들은 배치 연산을 지원: 모든 input/output tensor의 첫 번째 차원은 배치 차원**

torch.nn

- nn.Linear

```
torch.nn.Linear(in_features, out_features, bias=True)
```

  - 선형 변환 y = xA^T + b
  - in_features: size of each input sample
  - out_features: size of each output sample
  - bias: bias (b) 벡터 사용 유무

```
m = nn.Linear(20, 30)
input = torch.randn(128, 20)
output = m(input)
print(output.size()) # torch.Size([128, 30])
```

torch.nn

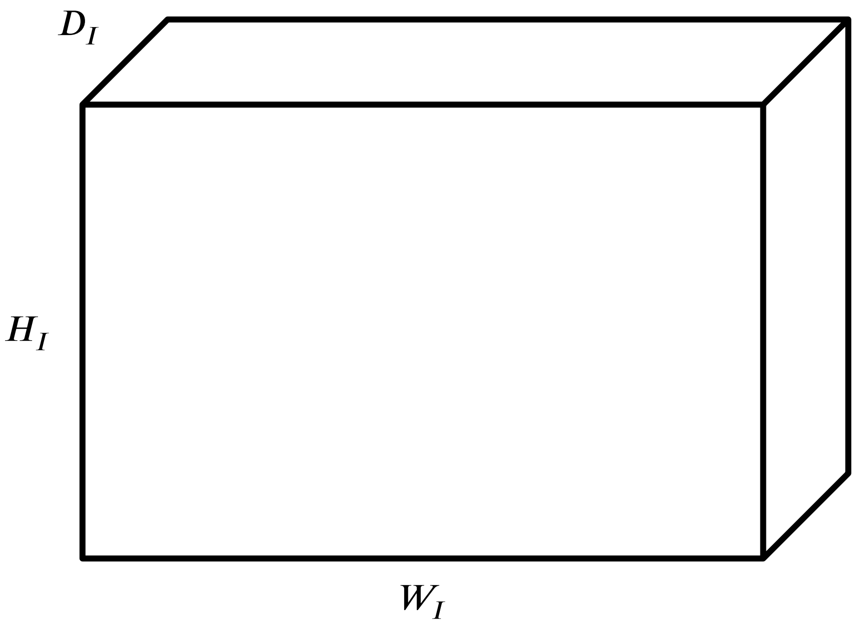
- nn.Conv2d

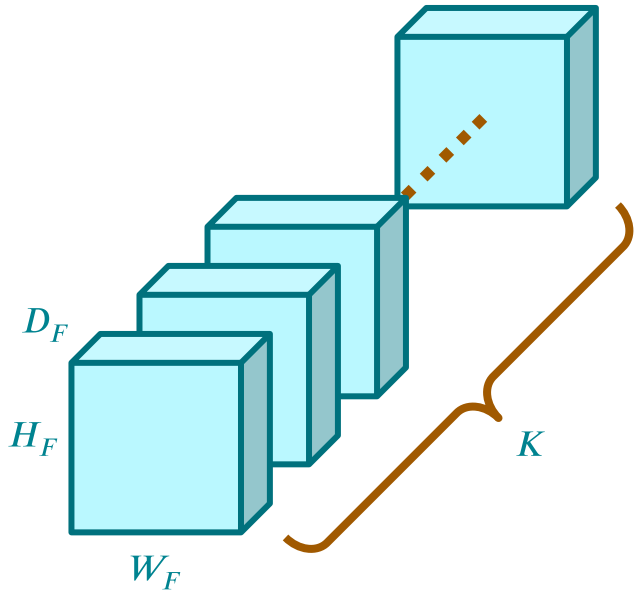
```
torch.nn.Conv2d(in_channels, out_channels, kernel_size,
stride=1, padding=0, bias=True)
```

  - 3차원 데이터 (ex) image) 상에서 합성곱(convolution) 연산을 수행
  - in_channels: Number of channels in the input image
  - out_channels: Size of the convolving kernel
  - kernel_size: Size of the convolving kernel
  - stride: Stride of the convolution. Default: 1
  - Bias: bias 벡터 사용 유무

```
m = nn.Conv2d(16, 33, 3, stride=2)
# N: 배치, C: 채널, H: 세로 크기, W: 가로 크기
input = torch.randn(20, 16, 50, 100) # (N, C, H, W)
output = m(input) # torch.Size([20, 33, 24, 49])
```

Digression: Convolution Arithmetic





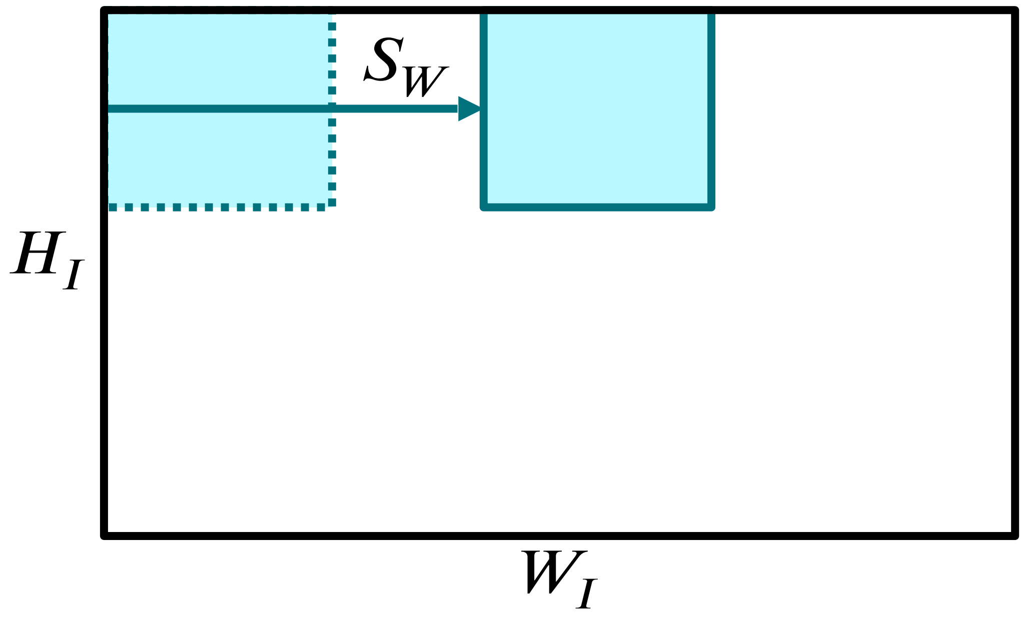
Input $I\colon \left[W_I, H_I, D_I\right] \in \mathbb{R}^3$

Filter

$F\colon \left[W_F, H_F, D_F, K\right] \in \mathbb{R}^4$

Digression: Convolution Arithmetic

- Assumption: use zero padding of size P



Output

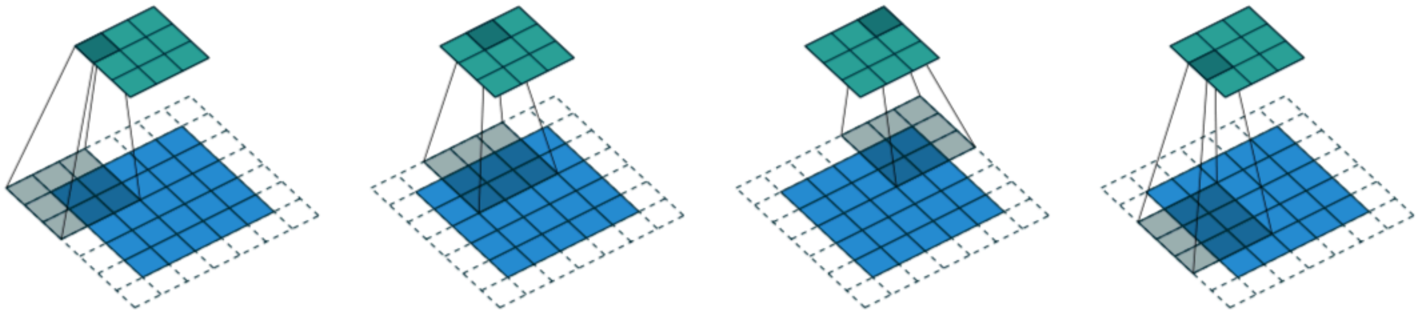
$$O: \left[W_O, H_O, D_O\right] \in \mathbb{R}^3$$

$$W_O = \left\lfloor \frac{W_I - W_F + 2P}{S_W} \right\rfloor + 1$$

Digression: Convolution Arithmetic

- Valid convolution: $W_F = 3,\ P = 1, S_W = 2$

$W_I = 5$



$W_I = 6$

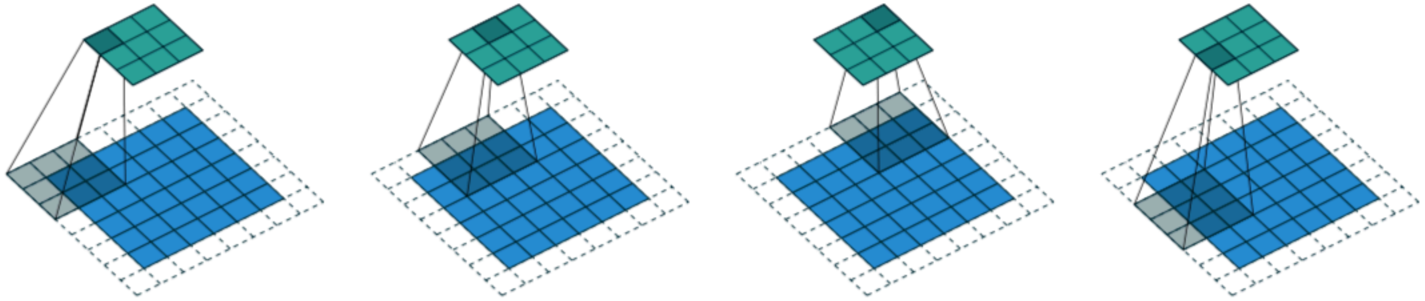


Image Credit: V. Dumoulin et al.. A Guide to Convolution Arithmetic for Deep Learning. arXiv preprint arXiv:1603.07285, 2016

torch.nn

- nn.MaxPool2D

```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0)
```

  - 3차원 데이터 (e.g. image) 상에서 MaxPooling 연산 수행

- nn.ReLU

```
m = nn.ReLU()
input = torch.randn(2)
output = m(input)
```

  - Element-wise rectified linear unit. y = max(x, 0)

torch.nn

- nn.MSELoss

```
loss = nn.MSELoss()
input = torch.randn(3, 5, requires_grad=True)
target = torch.randn(3, 5)
output = loss(input, target)
output.backward() # Compute the gradient of output
```

  - input: (N, *)
  - Target: (N,*), input과 동일한 size
  - Linear regression 모델을 위한 loss 함수

torch.nn

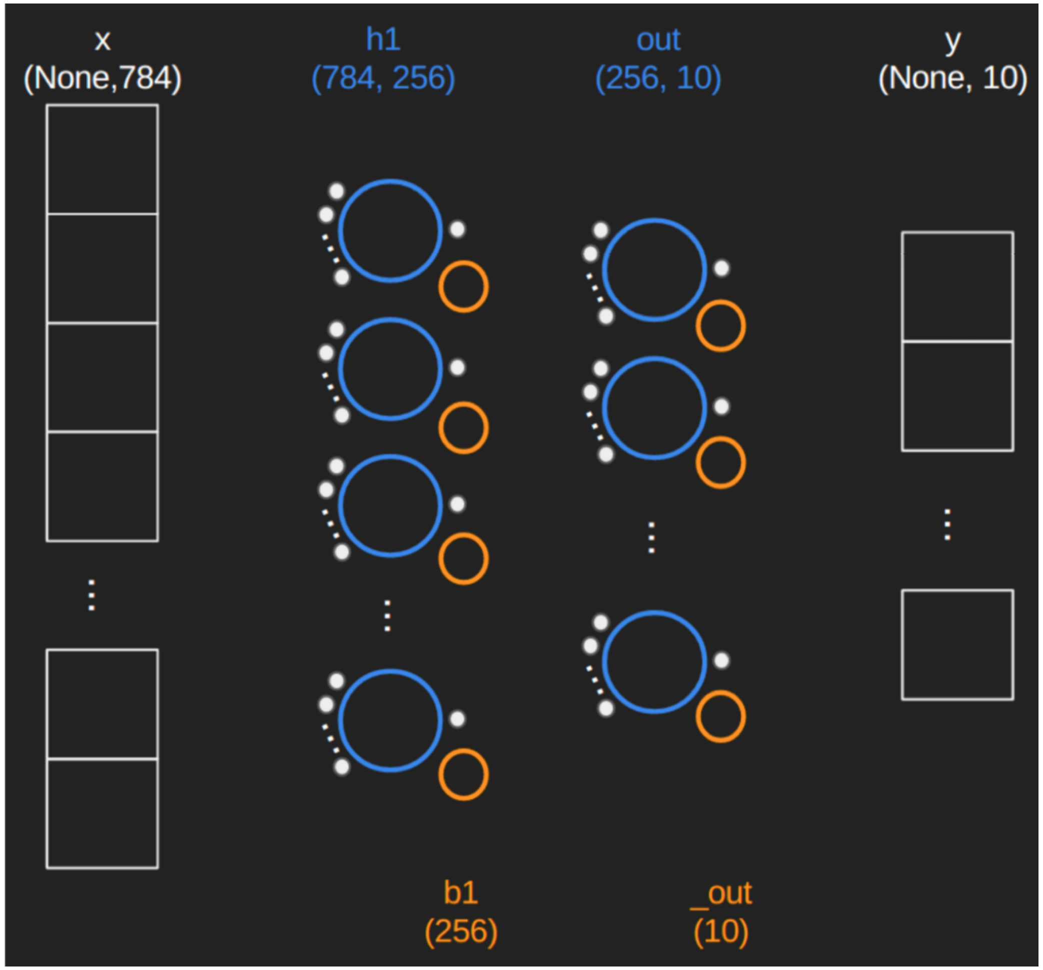
- nn.CrossEntropyLoss

```
loss = nn.CrossEntropyLoss()
input = torch.randn(3, 5, requires_grad=True)
target = torch.empty(3, dtype=torch.long).random_(5)
output = loss(input, target)
output.backward() # Compute the gradient of output
```

  - input: (N, C), logits 값, CrossEntropyLoss 함수 내에서 softmax 진행
  - Target: (N,)
  - Logistic regression 모델을 위한 loss 함수

torch.nn

- 새로운 딥러닝 모델 만드는 방법
  - 다음의 연산을 수행하는 feed-forward network는 어떻게 만들 수 있을까요?
  - input size: 784, hidden size: 256, output size: 10, activation function: ReLU



torch.nn

- 새로운 딥러닝 모델 만드는 방법
  - nn.Module을 상속
  - __init__ 함수에서 딥러닝 모델이 사용할 모듈 정의
  - forward 함수에서 input, output 간의 관계 정의

```
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size,
num_classes):
        super().__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        # Define the input processing through network
        z1 = self.fc1(x)
        h1 = self.relu(z1)
        out = self.fc2(h1)
        return out
```

torch.nn

- 새로운 딥러닝 모델 만드는 방법
  - nn.Module을 상속
  - __init__ 함수에서 딥러닝 모델이 사용할 모듈 정의
  - forward 함수에서 input, output 간의 관계 정의

```
model = NeuralNet(input_size, hidden_size,
num_classes).to(device)
print(model)
------------------------------------------------------------
--
NeuralNet(
  (fc1): Linear(in_features=784, out_features=256,
bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=256, out_features=10, bias=True)
)
```

torch.optim

- optim.SGD

```
torch.optim.SGD(params, lr)
```

  - 기본적인 Stochastic Gradient Descent optimizer
  - params: 학습 대상인 모델 파라미터
  - lr: learning rate

```
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
optimizer.zero_grad() # zeroing the gradients of model
params
loss_fn(model(input), target).backward() # Compute gradients
optimizer.step() # Perform the Gradient Descent
```

torch.optim

- optim.Adam

```
torch.optim.Adam(params, lr)
```

  - ADAM: SGD의 변형 optimizer
  - 학습이 진행됨에 따라 초기 learning rate에서 수정된 값을 사용
  - 여러 이슈가 있으나, 일반적으로 많은 문제에서 가장 높은 성능을 보이는 optimizer

Tensorboard

- Visualization tool
    - 학습의 진행 과정 및 결과를 시각화하기 위해 사용하는 툴
    - 학습의 문제 사항 등을 파악하기 위한 디버깅 툴로서도 역할
    - 본래 Tensorflow의 visualization tool로 개발되었으나 PyTorch에서도 사용 가능 (tensorboardX)

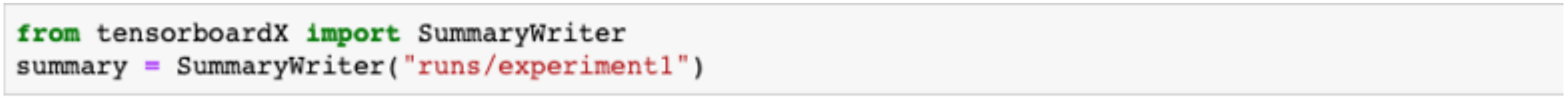
- tensorboardX.SummaryWriter

```
tensorboardX.SummaryWriter(logdir=None)
```

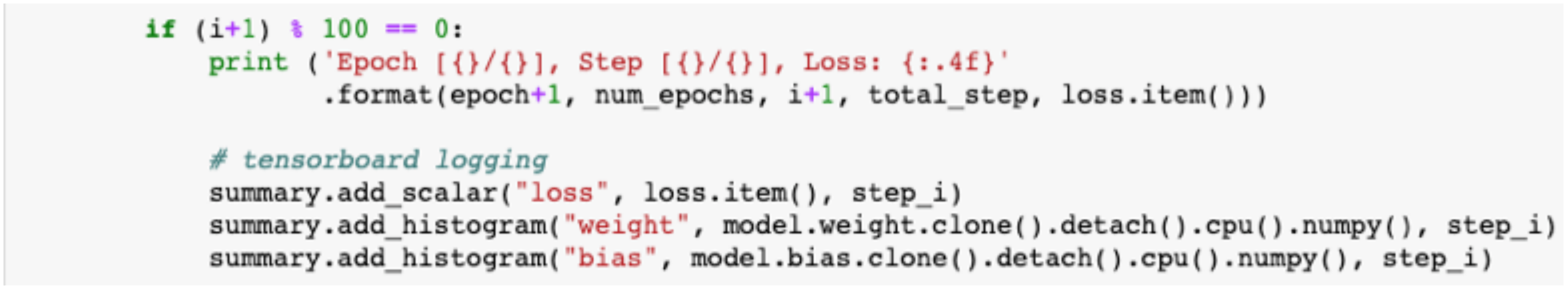
- 학습 과정에서 발생하는 텐서의 값을 기록할 수 있는 writer
- logdir: tensorboard log 파일을 저장할 directory

Tensorboard

- Scalar summary와 Histogram summary
  - Loss, accuracy와 같은 스칼라 텐서 (0차원 텐서)는 scalar summary
  - 모델의 파라미터와 같은 다차원 텐서는 histogram summary로 기록



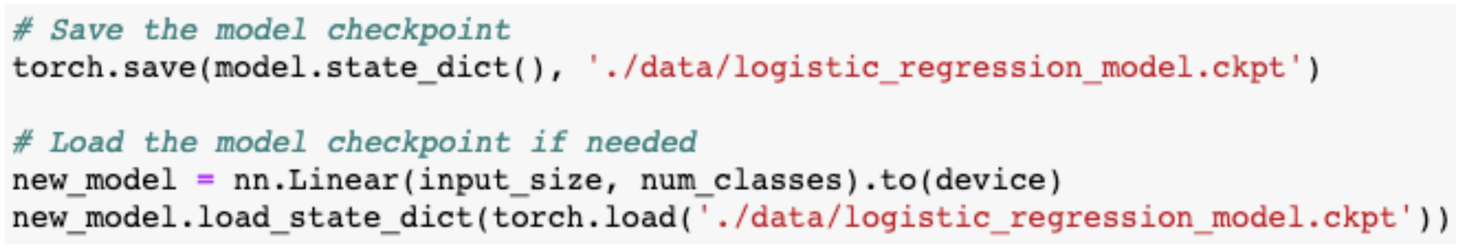
```python
from tensorboardX import SummaryWriter
summary = SummaryWriter("runs/experiment1")
```



```python
        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))

            # tensorboard logging
            summary.add_scalar("loss", loss.item(), step_i)
            summary.add_histogram("weight", model.weight.clone().detach().cpu().numpy(), step_i)
            summary.add_histogram("bias", model.bias.clone().detach().cpu().numpy(), step_i)
```

모델 파라미터 저장

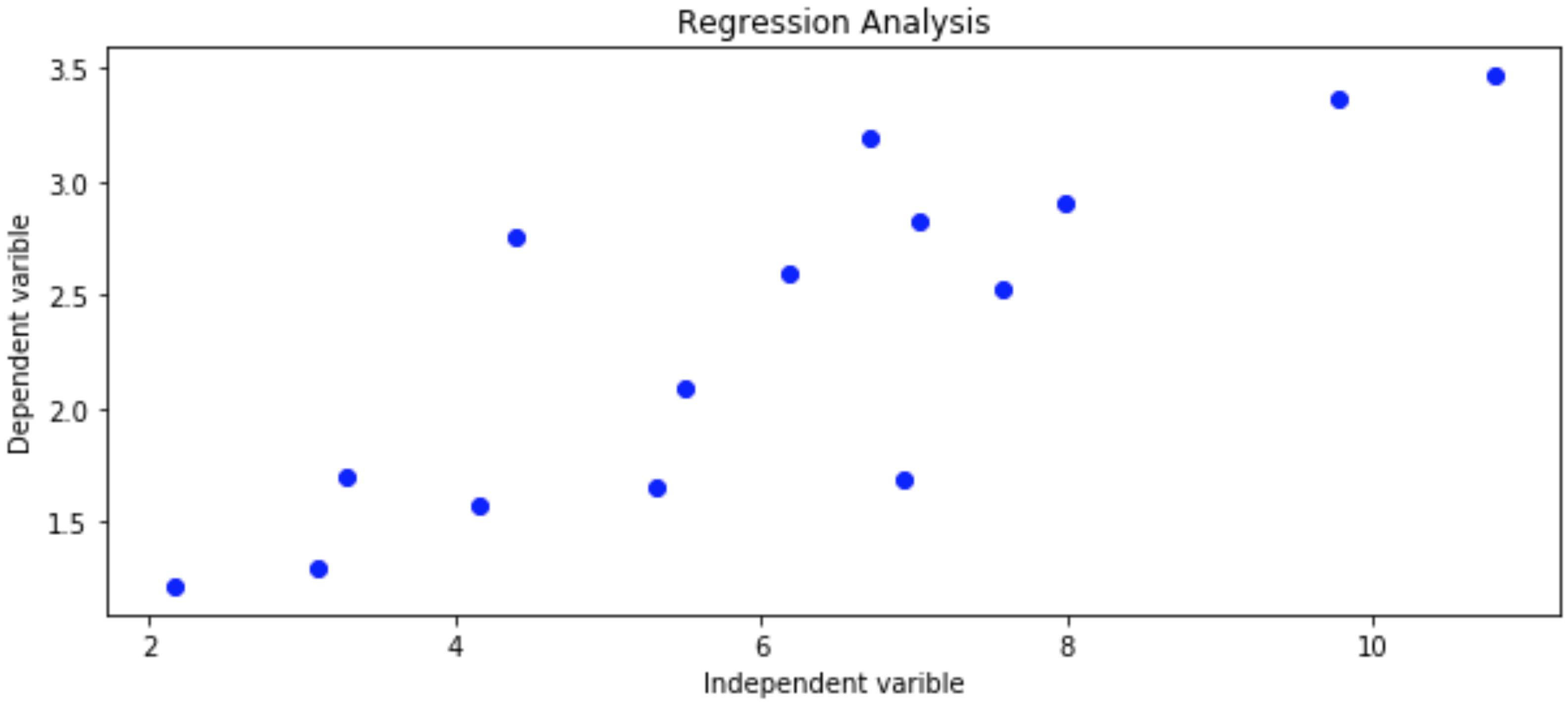
- torch.save/torch.load
  - 학습된 모델의 파라미터를 파일에 저장, 후에 다시 저장하기 위해 로드 가능
  - nn.Module.state_dict(): 모델의 파라미터를 Dictionary 형태로 반환



```python
# Save the model checkpoint
torch.save(model.state_dict(), './data/logistic_regression_model.ckpt')

# Load the model checkpoint if needed
new_model = nn.Linear(input_size, num_classes).to(device)
new_model.load_state_dict(torch.load('./data/logistic_regression_model.ckpt'))
```

Example: Linear Regression

- Line fitting
  - 다음의 15 points에 linear regression model (line)을 fitting



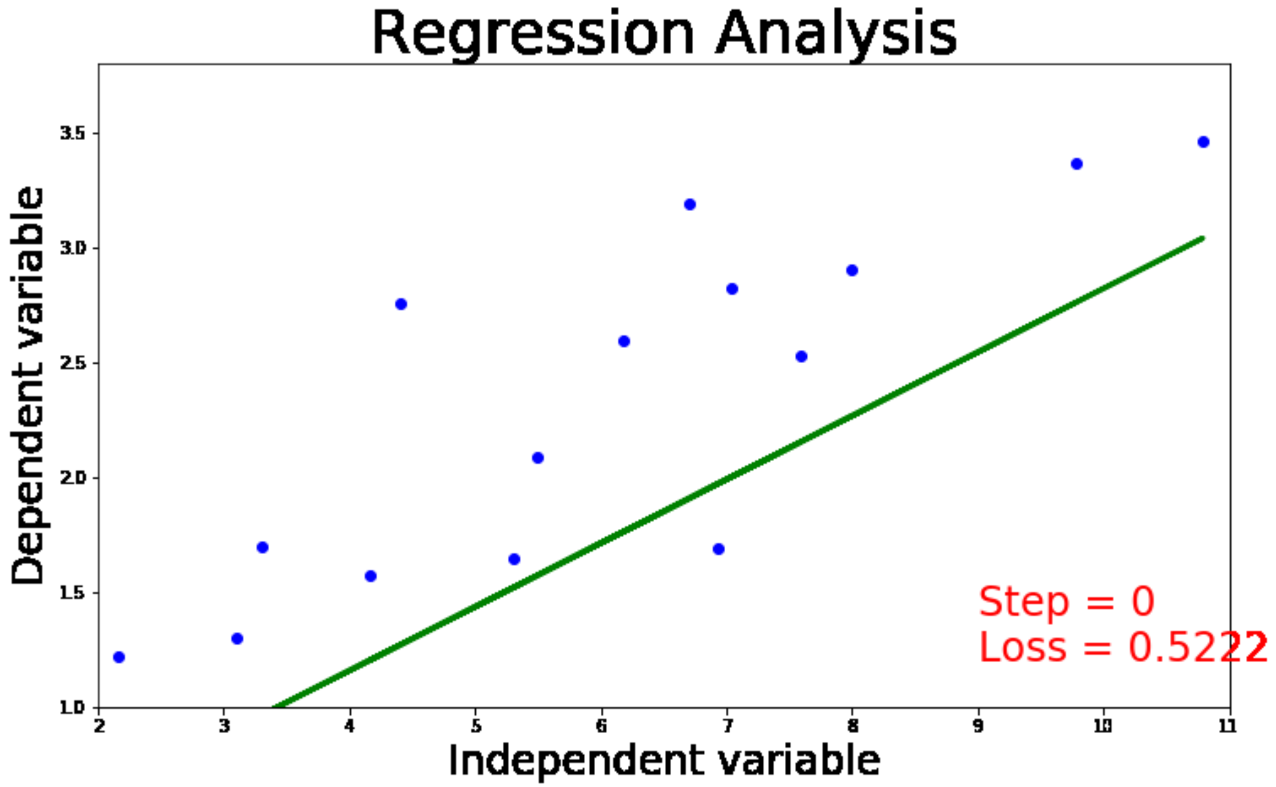
Example: Linear Regression

- Line fitting
  - 학습 모델: linear regression model (nn.Linear)
  - Loss: nn.MSELoss
  - Optimizer: optim.SGD, learning rate: 0.001
  - 총 학습 step: 60

```
# Linear regression model
model = nn.Linear(1, 1)
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
```

Example: Linear Regression

- Line fitting
    - 학습 모델: linear regression model (nn.Linear)
    - Loss: nn.MSELoss
    - Optimizer: optim.SGD, learning rate: 0.001
    - 총 학습 step: 60

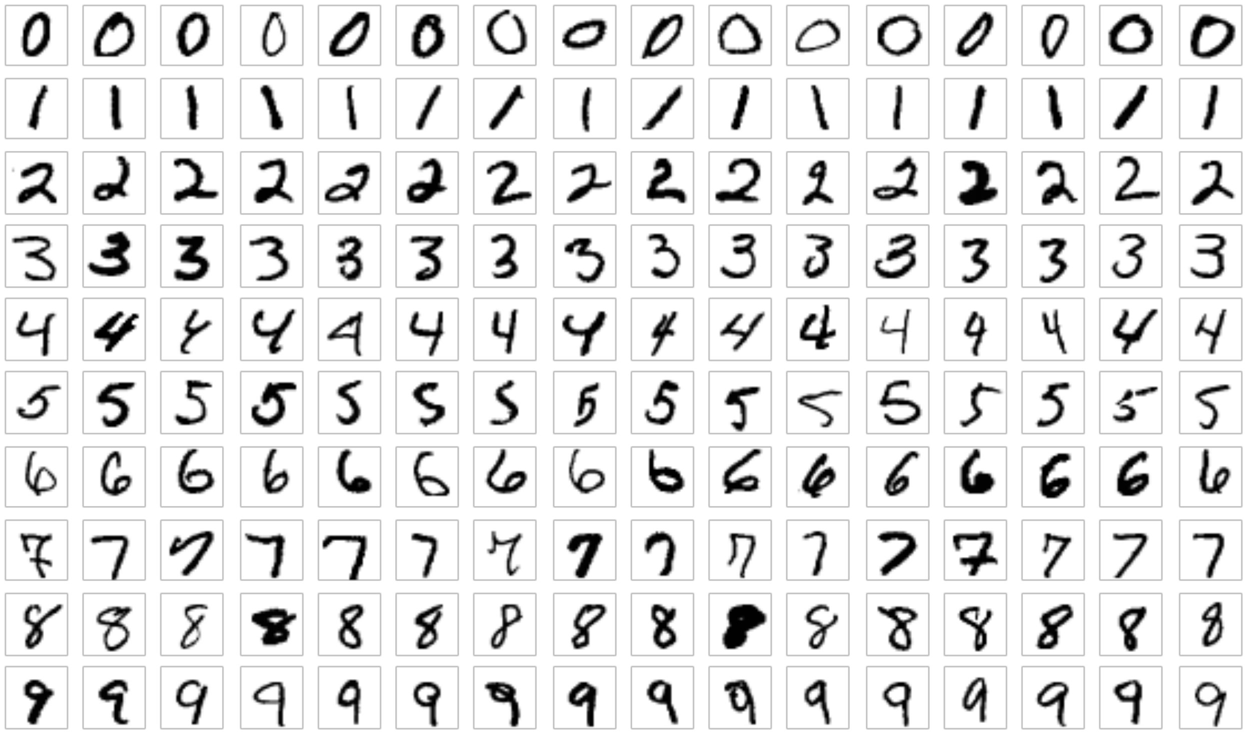


Let’s Check the Codes!!

03_Linear_Regression_Model

Example: Logistic Regression

- MNIST
  - Database of handwritten digits
  - Training set: 60,000 images, Test set: 10,000 images
  - Image: 1 x 28 x 28 ([C, H, W]) gray image



Example: Logistic Regression

- MNIST
  - 학습 모델: Logistic regression (nn.Linear)
  - Loss: nn.CrossEntropyLoss
  - Optimizer: optim.Adam (SGD의 변형 optimizer), learning rate: 0.001
  - 배치 사이즈: 100, 총 5 epoch 학습

```
# Linear regression model
model = nn.Linear(784, 10) # 784 = 28x28, 10 = # of classes
(0~9)
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer =
torch.optim.Adam(model.parameters(),lr=learning_rate)
```

Example: Logistic Regression

- MNIST

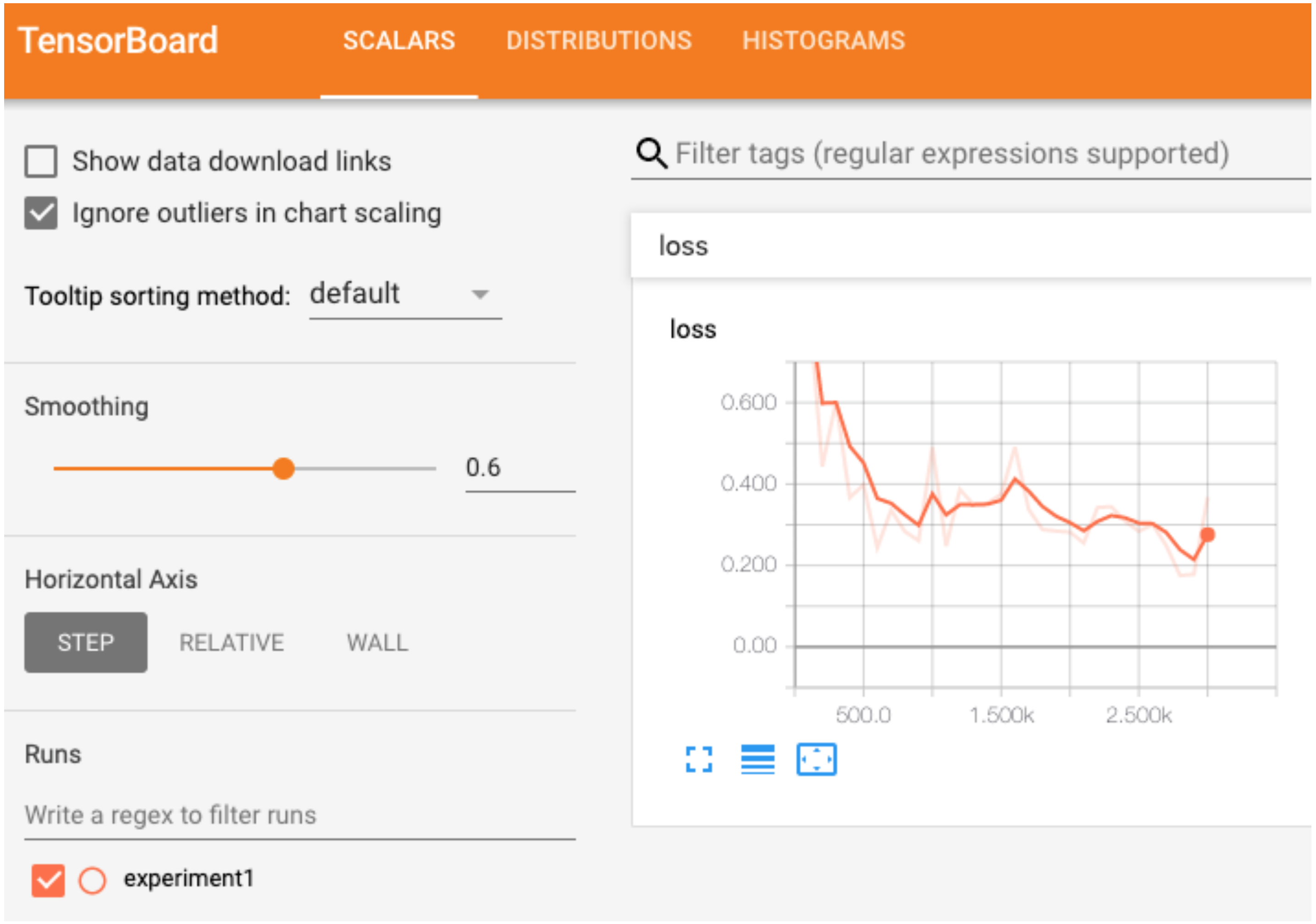
Train the network

```python
num_epochs = 5
# Train the model
total_step = len(train_loader)
step_i = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Reshape images to (batch_size, input_size)
        images = images.reshape(-1, 28*28).to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        step_i += 1

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))

            # tensorboard logging
            summary.add_scalar("loss", loss.item(), step_i)
            summary.add_histogram("weight", model.weight.clone().detach().cpu().numpy(), step_i)
            summary.add_histogram("bias", model.bias.clone().detach().cpu().numpy(), step_i)
```

```
Epoch [1/5], Step [100/600], Loss: 0.8579
Epoch [1/5], Step [200/600], Loss: 0.4429
Epoch [1/5], Step [300/600], Loss: 0.6020
Epoch [1/5], Step [400/600], Loss: 0.3668
Epoch [1/5], Step [500/600], Loss: 0.3985
Epoch [1/5], Step [600/600], Loss: 0.2436
Epoch [2/5], Step [100/600], Loss: 0.3362
Epoch [2/5], Step [200/600], Loss: 0.2835
Epoch [2/5], Step [300/600], Loss: 0.2611
```

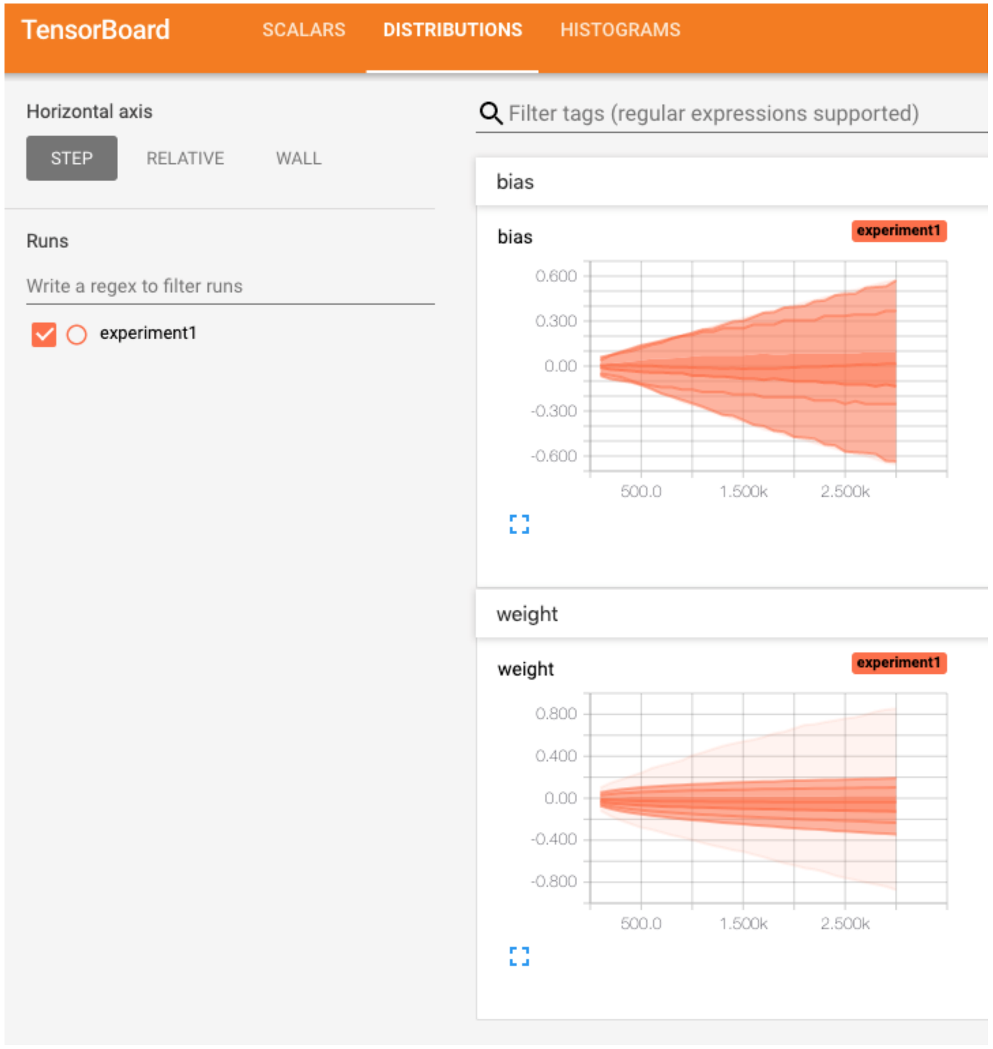
Example: Logistic Regression

- MNIST
    - Tensorboard logging: Loss



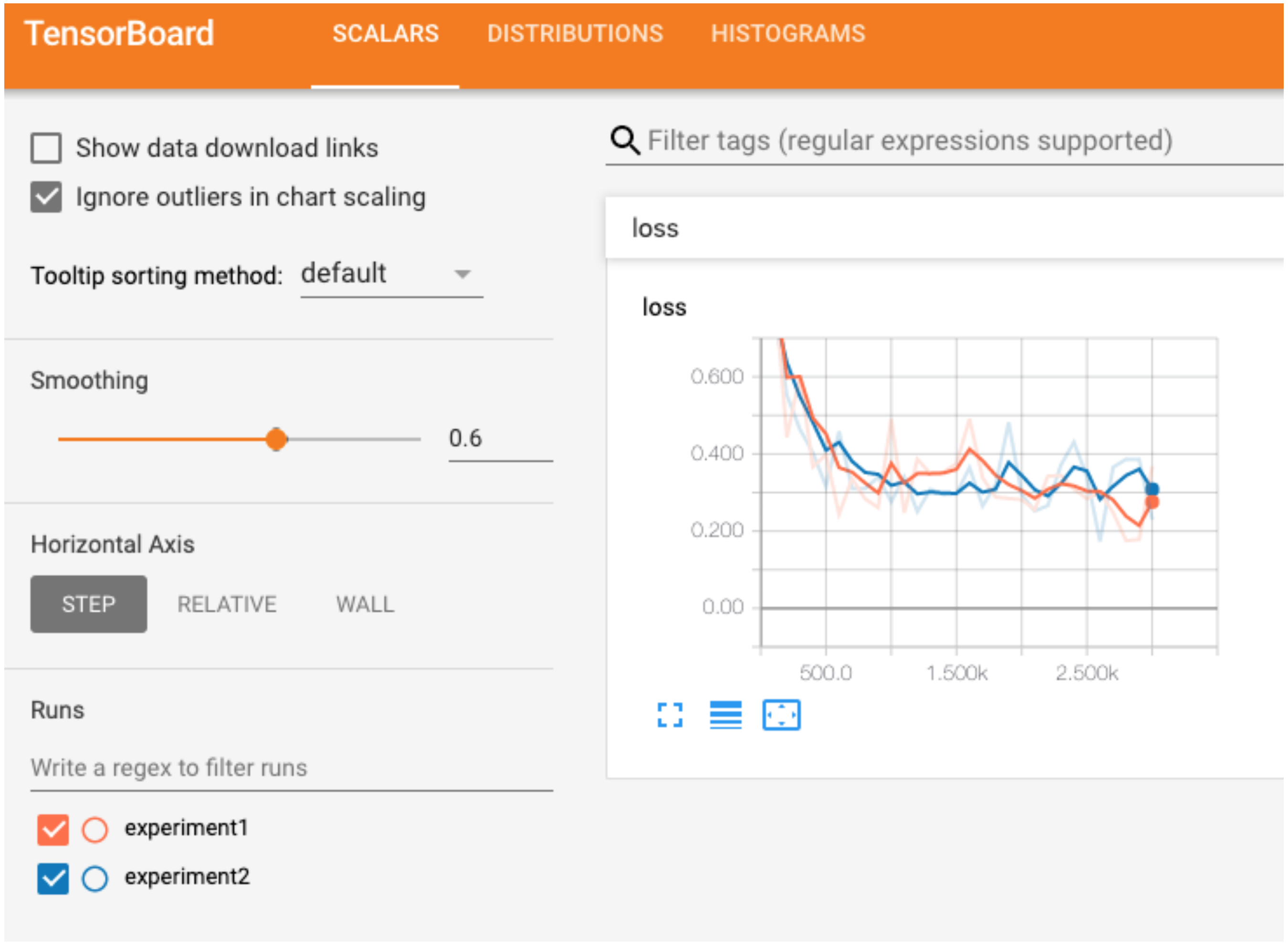
Example: Logistic Regression

- MNIST
  - Tensorboard logging: Weight/Bias distribution



Example: Logistic Regression

- MNIST
  - Tensorboard logging: 서로 다른 실험 결과를 한 그래프에서 비교 가능 (weight/bias의 초기 값을 다르게 하고 실험한 결과)



☐ Show data download links
☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

☑ experiment1
☑ experiment2

Let’s Check the Codes!!

05_Logistic_Regression_Model

Let’s Check the Codes!!

06 - 07

Applications

Black and White Image Coloring

Object Detection

Task

- Image Colorization
  - Computer vision task of converting 1-channel (greyscale) images to 3-channel (RGB) images
  - Neural-network-based methods show SOTA performance

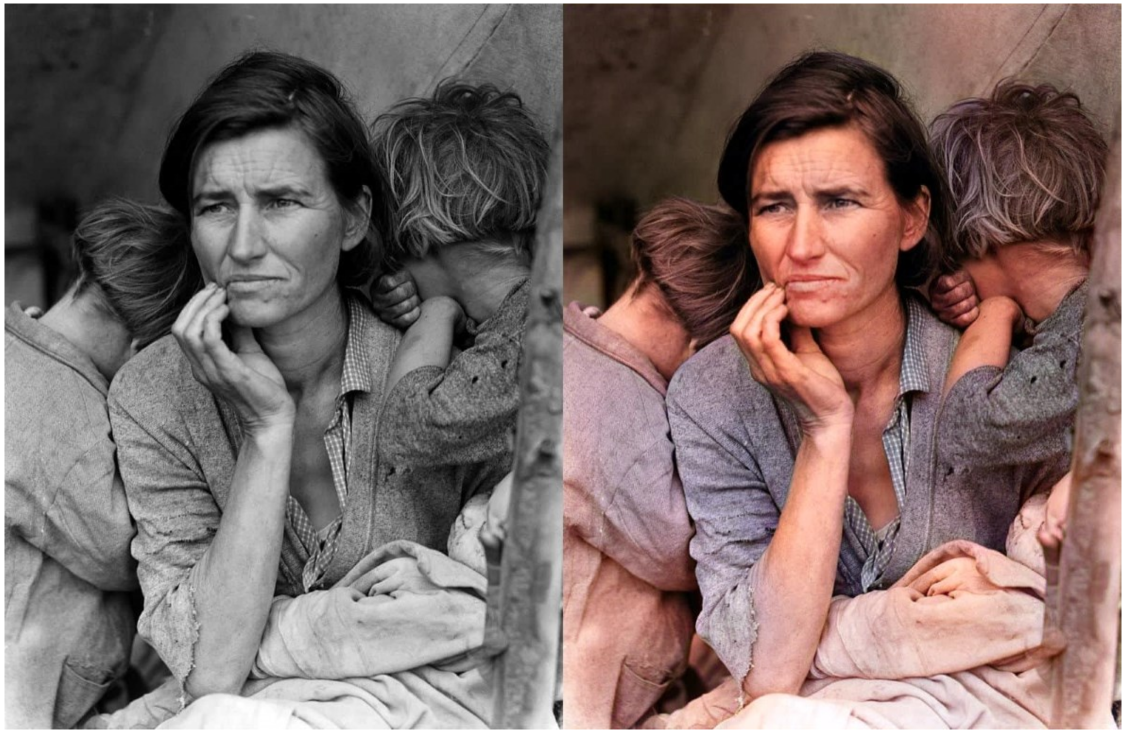


Image from https://github.com/jantic/DeOldify

Method

- GAN
    - A generative model (Capable of generating artificial data)
    - GAN uses two competing networks: the generator and discriminator
    - The generator tries to fool the discriminator by creating realistic samples
    - The discriminator aims to discern whether the sample is real of fake (created by the generator)

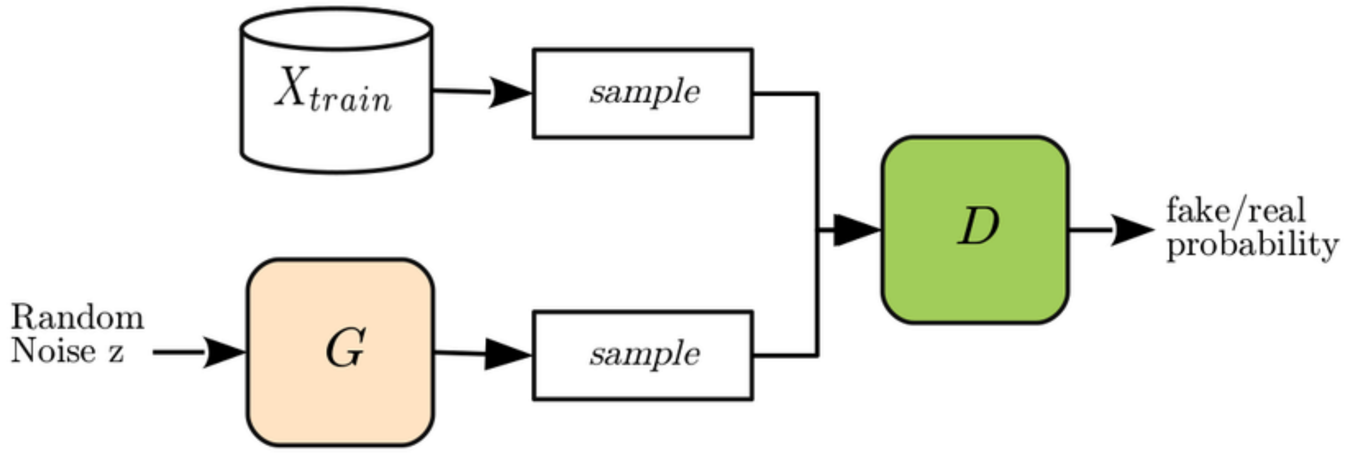


Image from *LOGAN: Evaluating Privacy Leakage of Generative Models Using Generative Adversarial Networks*

Task

- Object Detection
  - Computer vision task of detecting objects in images or videos.
  - Object detection is related to various tasks such as segmentation and pose detection

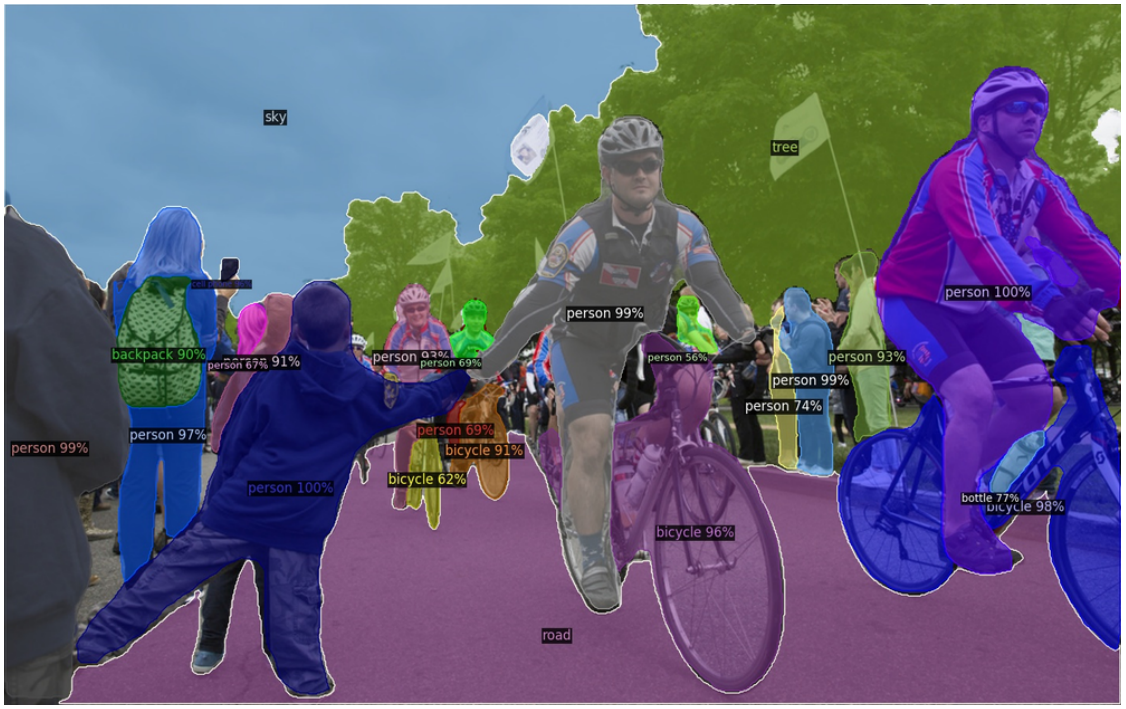


Image from *https://github.com/facebookresearch/detectron2*

Let’s Check the Codes!!

08_Image_Colorization

Method

- R-CNN: Regions with CNN features
  - R-CNN consists of two modules: the region proposal generator and the CNN
  - The region proposal generator suggests region candidates
  - The CNN calibrates the region boxes and classify the objects
  - Modern variants of R-CNN use neural networks for both modules

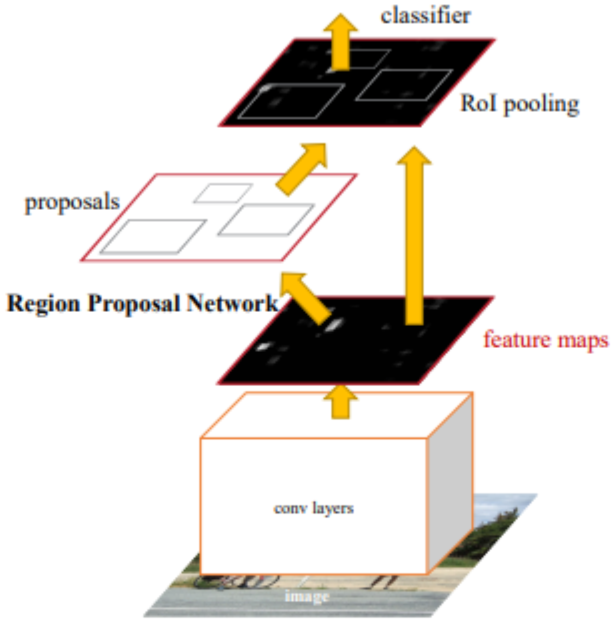


Image from *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*

Library

- Detectron2
  - Facebook AI Research's next generation software system that implements state-of-the-art object detection algorithms
  - https://github.com/facebookresearch/detectron2

Detectron2

Image from https://github.com/facebookresearch/detectron2

Let’s Check the Codes!!

09_Object_Detection