BRFRAME
AI&AA framework

AI Prebuild Service &
Automated Machine Learning Leader

한양대 분석가 교육과정

Introduction to Computer Vision with PyTorch

2021년 06월 16일

비알프레임

Machine Learning &
Predictive Analytics

02. Introduction to Computer Vision with PyTorch

- **학습 목표**

  - 컴퓨터 비전 머신러닝 모델을 구축하는 데 사용되는 주요 개념 학습
  - 텐서로서 이미지를 나타내는 방법 학습
  - 밀집 신경망(Dense Neural Network)와 합성곱 신경망(Convolultion Neural Network) 구축 방법 학습

- **전제 조건**

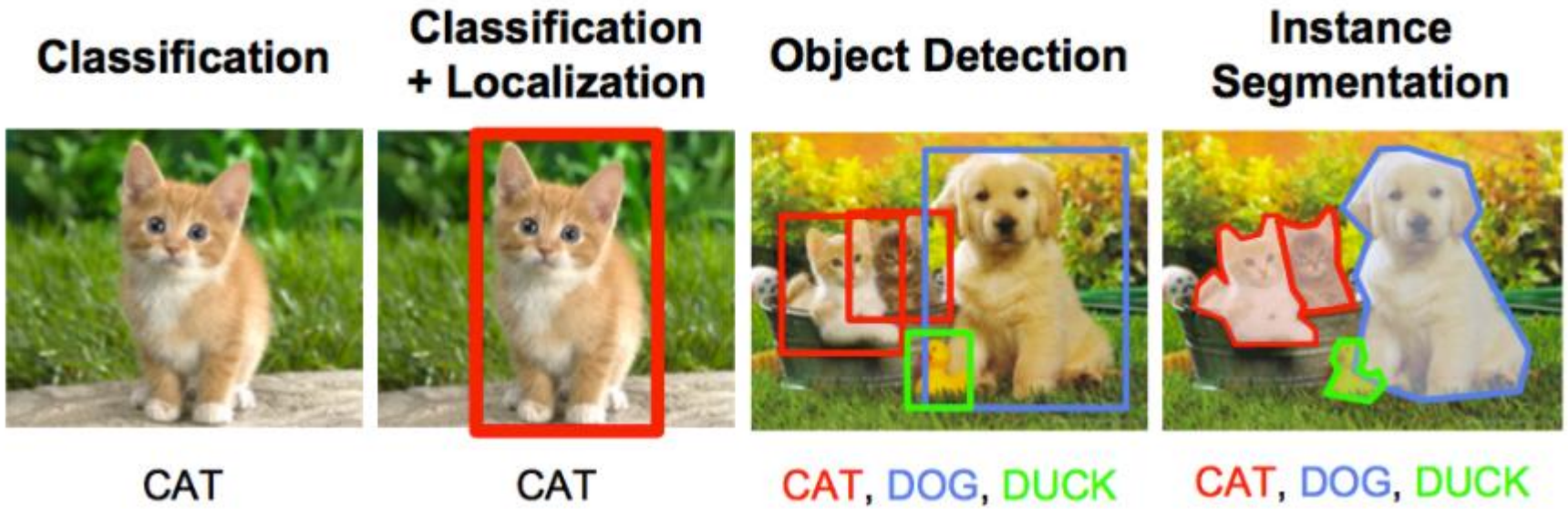
  - 기본 Python 지식
  - Jupyter Notebooks 사용 방법에 대한 기본 지식
  - 기본 머신러닝의 이해

02. Introduction to Computer Vision with PyTorch

- **1. 이미지 데이터 처리 소개**

컴퓨터 비전에서 일반적으로 다음 중 하나의 문제를 다룬다.

- Image Classification : 전체 혹은 이미지 안의 물체(object)의 종류를 구분하는 작업
- Object Detection : 물체가 무엇인지 분류(classification)하는 과정과 물체가 어디에 있는지 위치정보를 알려주는(Localization) 과정이 동시에 수행되는 것을 의미
- Segmentation : bounding box를 찾아주는 Object Detection과는 달리, 인식된 물체의 픽셀 맵 윤곽선을 추출



이미지 출처 : https://cs224d.stanford.edu/index.html

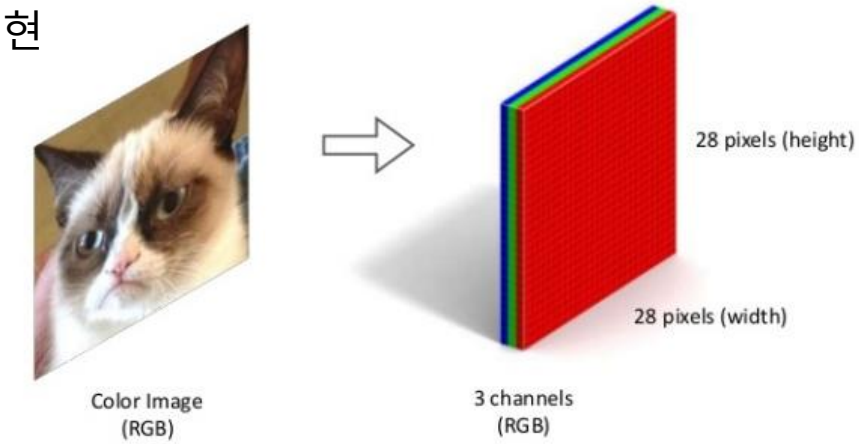
02. Introduction to Computer Vision with PyTorch

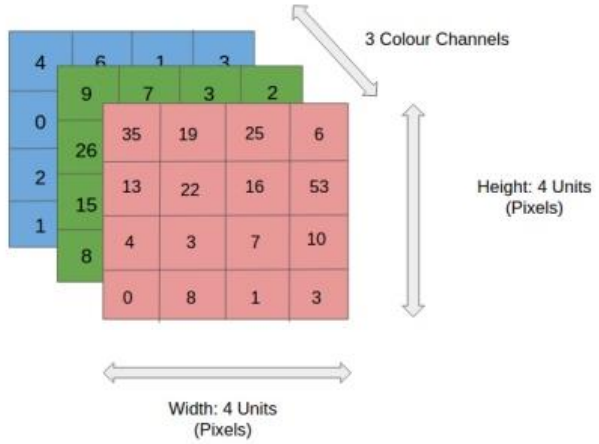
2. 텐서로서의 이미지 데이터

• 컬러 이미지는 3 x W x H 사이즈의 어레이로 표현될 수 있다.
- 3 : 컬러를 표현하기 위해서는 Red, Green, Blue의 채널마다 픽셀 강도 값으로 표현
- W x H : 이미지의 크기인 너비(Width)와 높이(Height) 사이즈의 픽셀 어레이로 표현

• 다차원 배열은 텐서라고 불리며 이미지를 표현하기 위해 텐서를 사용하는 것도 장점이 있다. 왜냐하면 추가 차원을 사용하여 일련의 이미지를 저장할 수 있기 때문입니다. 예를 들어, 800x600 차원을 가진 200 프레임으로 구성된 비디오를 표현하기 위해, 우리는 200x3x600x800 크기의 텐서를 사용할 수 있다.

※ 사용 데이터셋 : MNIST 데이터셋
- 0~9 까지의 숫자를 사람의 손글씨로 작성한 이미지
- 0~1 사이의 실수 값 또는 0~255 사이의 정수 값을 지닌 28 x 28 사이즈의 그레이스케일 이미지
- 학습 데이터셋 : 6만장 / 테스트 데이터셋 : 1만장





이미지 출처 : https://www.slideshare.net/BertonEarnshaw/a-brief-survey-of-tensors

02. Introduction to Computer Vision with PyTorch

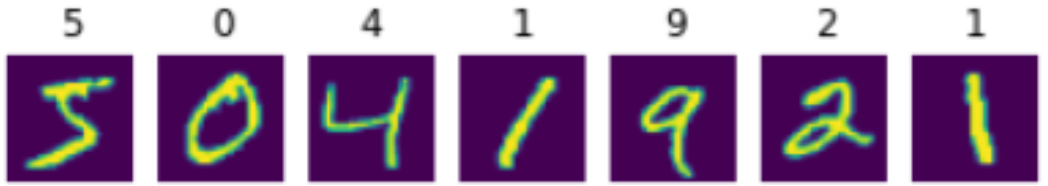
2. 텐서로서의 이미지 데이터

- MNIST 데이터셋 불러오기
  - torchvision.dataset.MNIST 메서드를 사용하여 데이터셋 불러오기
    ※ 파라미터 transform = ToTensor() 를 지정하여 PIL 이미지를 텐서로 변환

```
import torch
import torchvision
import matplotlib.pyplot as plt
import numpy as np

from torchvision.transforms import ToTensor

data_train = torchvision.datasets.MNIST('./data',
        download=True,train=True,transform=ToTensor())
data_test = torchvision.datasets.MNIST('./data',
        download=True,train=False,transform=ToTensor())
```

- 불러온 MNIST 데이터 시각화

```
fig,ax = plt.subplots(1,7)
for i in range(7):
    ax[i].imshow(data_train[i][0].view(28,28))
    ax[i].set_title(data_train[i][1])
    ax[i].axis('off')
```

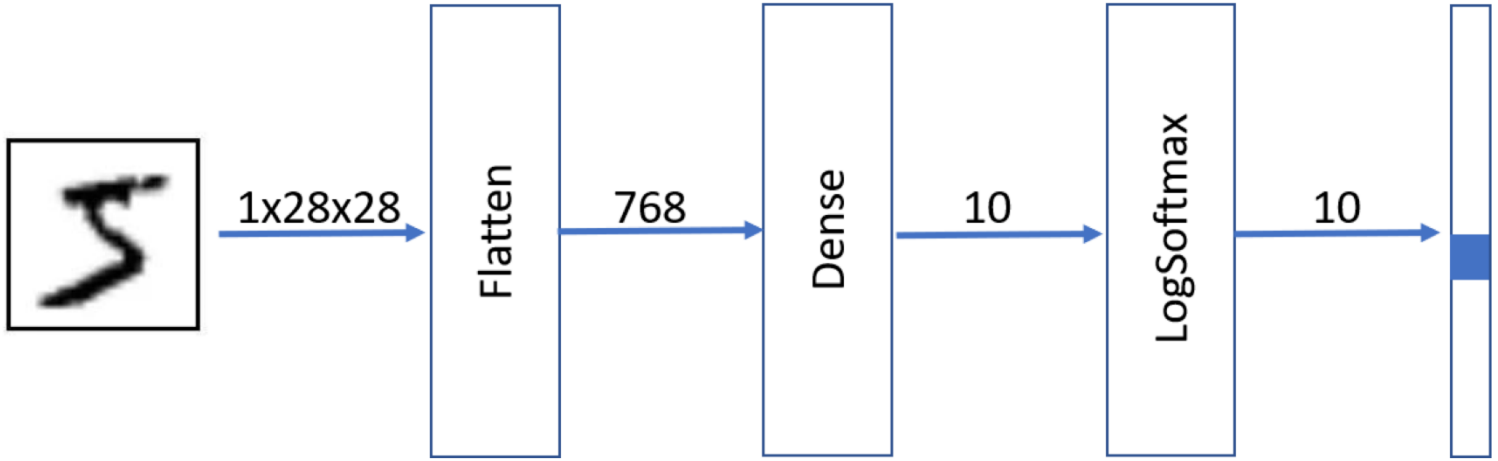


02. Introduction to Computer Vision with PyTorch

- **3. 밀집 신경망 훈련하기(Single-Layer)**

  - 간단한 이미지 분류 문제를 위하여 nn.Sequential()을 사용하여 완전 연결 신경망(Fully-connected Neural Network) 구축
  - 입력 뉴런 수 : 784 (1 x 28 x 28)
  - 출력 뉴런 수 : 10 (0 ~ 9)
  - 활성화 함수 : LogSoftmax

```
net = nn.Sequential(
        nn.Flatten(),
        nn.Linear(784,10), # 784 inputs, 10 outputs
        nn.LogSoftmax())
```



02. Introduction to Computer Vision with PyTorch

3. 밀집 신경망 훈련하기(Single-Layer)

- 학습 프로세스

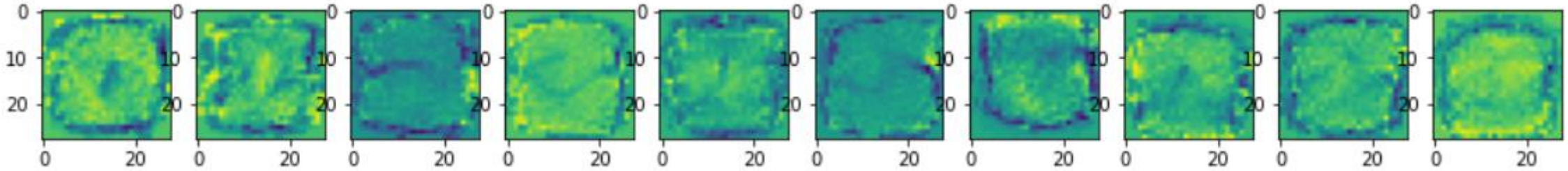
1) 입력 데이터(feature)와 결과(Label)로 구성된 입력 데이터셋에서 미니배치를 얻는다.
2) 미니배치에 대한 예측값 계산
3) 실제값과 예측값의 차이를 손실 함수(Loss function)을 통하여 계산
4) 모델 가중치(매개변수)와 관련하여 이 손실 함수의 기울기를 계산한 다음
   네트워크의 성능을 최적화하기 위해 가중치를 조정
5) 전체 데이터셋이 처리될 때까지 이러한 단계를 반복한다.

```
def train_epoch(net,dataloader,lr=0.01,optimizer=None,loss_fn = nn.NLLLoss()):
    optimizer = optimizer or torch.optim.Adam(net.parameters(),lr=lr)
    net.train()
    total_loss,acc,count = 0,0,0
    for features,labels in dataloader:
        optimizer.zero_grad()
        out = net(features)
        loss = loss_fn(out,labels) #cross_entropy(out,labels)
        loss.backward()
        optimizer.step()
        total_loss+=loss
        _,predicted = torch.max(out,1)
        acc+=(predicted==labels).sum()
        count+=len(labels)
    return total_loss.item()/count, acc.item()/count

train_epoch(net,train_loader)
```

3. 밀집 신경망 훈련하기(Single-Layer)

- 신경망 가중치 시각화
  - net.parameters() 메서드를 사용하여 784 x 10 차원을 갖는 가중치 텐서(weight_tensor) 생성

```
weight_tensor = next(net.parameters())
fig,ax = plt.subplots(1,10,figsize=(15,4))
for i,x in enumerate(weight_tensor):
    ax[i].imshow(x.view(28,28).detach())
```

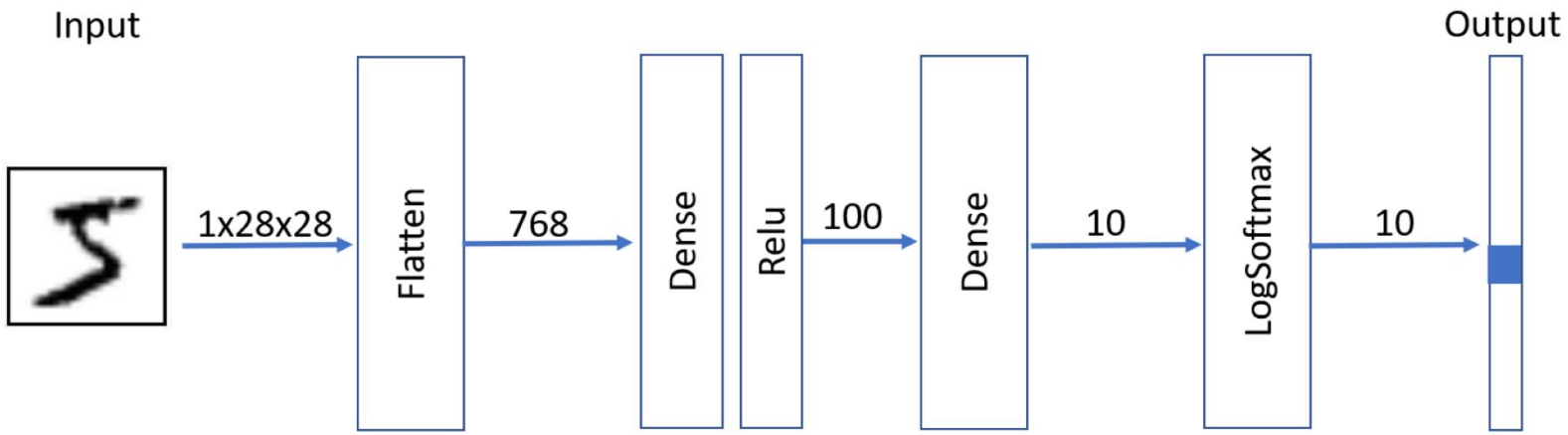


02. Introduction to Computer Vision with PyTorch

- **4. 밀집 신경망 훈련하기(Multi-Layer)**

  - 더욱 정교한 이미지 분류를 다루기 위해 하나의 은닉층을 추가하여 다층 퍼셉트론 구축
  - 입력 뉴런 수 : 784 (1 x 28 x 28)
  - 출력 뉴런 수 : 10 (0 ~ 9)
  - 활성화 함수 : ReLU, LogSoftmax

```
net = nn.Sequential(
        nn.Flatten(),
        nn.Linear(784,100),      # 784 inputs, 100 outputs
        nn.ReLU(),               # Activation Function
        nn.Linear(100,10),       # 100 inputs, 10 outputs
        nn.LogSoftmax())
```



4. 밀집 신경망 훈련하기(Multi-Layer)

• 클래스 기반의 신경망 정의

- Sequential 스타일의 모델을 구축하는 것은 간편하지만 더욱 복잡한 신경망을 정의하기에는 제한이 따른다.
- __init__ : 네트워크에서 사용할 모든 Layer를 정의.
  ※ PyTorch는 훈련할 때 생성된 Layer의 파라미터가 최적화되어야 한다는 것을 자동으로 인식
- forward : 신경망의 포워드 패스 계산 방식을 정의

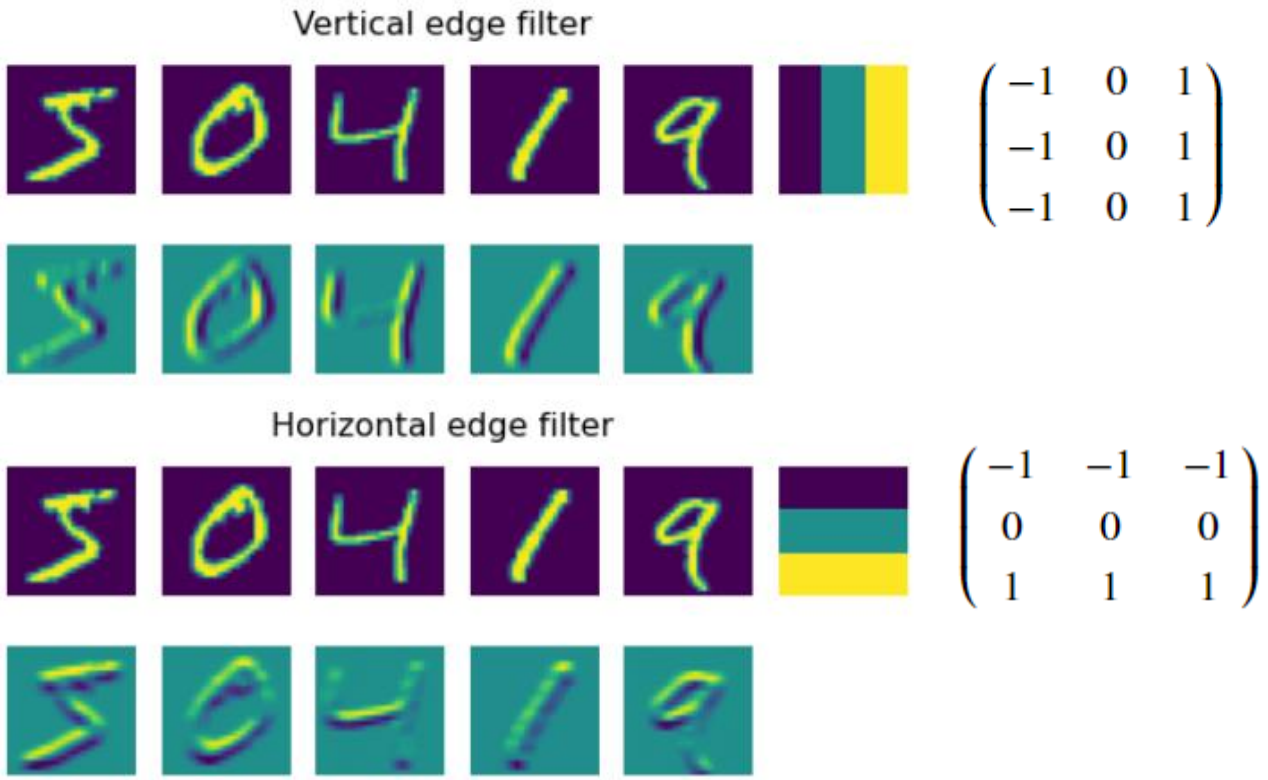
```
from torch.nn.functional import relu,log_softmax

class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.flatten = nn.Flatten()
        self.hidden = nn.Linear(784,100)
        self.out = nn.Linear(100,10)

    def forward(self, x):
        x = self.flatten(x)
        x = self.hidden(x)
        x = relu(x)
        x = self.out(x)
        x = log_softmax(x)
        return x

net = MyNet()
```

02. Introduction to Computer Vision with PyTorch

- **5. 합성곱 신경망 사용하기**

  • 합성곱 필터(Convolutional filters)의 이해

  - 합성곱 필터는 이미지의 각 픽셀에 대해 실행되고 인접 픽셀의 가중 평균을 계산하는 작은 윈도우.
    필터는 가중치 계수의 행렬로 정의되며 MNIST 손글씨 숫자 위에 두 개의 다른 합성곱 필터를 적용하는 예를 살펴보자.
  - 딥러닝에서는 이미지 분류 문제를 해결하기 위하여 최적의 합성곱 필터를 학습하는 네트워크를 구성



02. Introduction to Computer Vision with PyTorch

5. 합성곱 신경망 사용하기

• 합성곱 층(Convolutional layers) : nn.Conv2d 를 사용하여 합성곱 층 정의
- id_channels : 입력 이미지의 채널 수
  ※ 여기서 사용할 MNIST 이미지는 그레이스케일의 이미지이므로 채널 수는 1
- out_channels : 사용할 필터의 개수
- kernel_size : 필터 사이즈 크기. 일반적으로 3x3 또는 5x5를 사용한다.

```
class OneConv(nn.Module):
    def __init__(self):
        super(OneConv, self).__init__()
        self.conv = nn.Conv2d(in_channels=1,out_channels=9,kernel_size=(5,5))
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(5184,10)

    def forward(self, x):
        x = nn.functional.relu(self.conv(x))
        x = self.flatten(x)
        x = nn.functional.log_softmax(self.fc(x),dim=1)
        return x

net = OneConv()
```

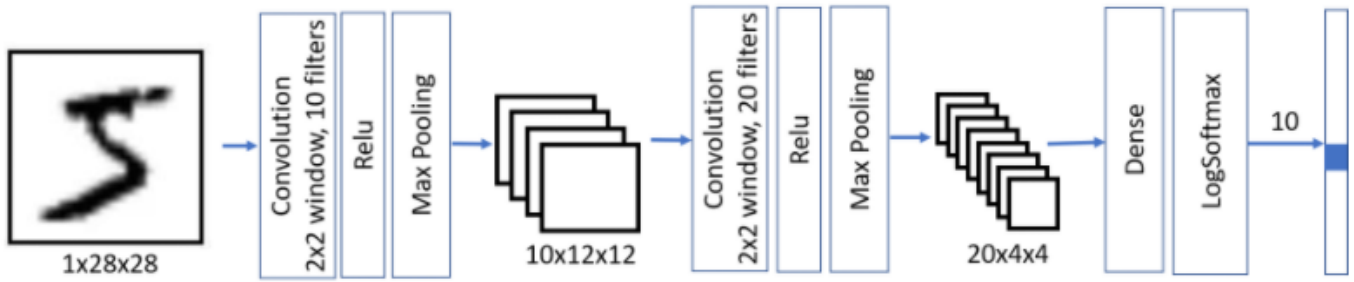
02. Introduction to Computer Vision with PyTorch

6. 다층 합성곱 신경망 사용하기

- 상위 수준의 패턴을 찾기 위하여 합성곱 층 하나를 추가하여 다층 합성곱 신경망(Multi-layered CNNs) 구축
- 풀링 레이어(Pooling Layers)를 통하여 이미지의 사이즈를 줄여 패턴이 나타난 부분만 추출
  - Average Pooling : 슬라이딩 윈도우 내의 값을 평균을 취하여 값 추출
  - Max Pooling : 슬라이딩 윈도우 내의 값 중 최대값을 추출

※ Spatial dimension이 감소하고 feature/filters가 증가하여 이러한 구조를 "피라미드 아키텍처"라 부른다.

```python
class MultiLayerCNN(nn.Module):
    def __init__(self):
        super(MultiLayerCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(10, 20, 5)
        self.fc = nn.Linear(320,10)

    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = x.view(-1, 320)
        x = nn.functional.log_softmax(self.fc(x),dim=1)
        return x

net = MultiLayerCNN()
```



02. Introduction to Computer Vision with PyTorch

- **7. CIFAR-10 데이터셋을 사용하여 실제 이미지 학습하기**

  - CIFAR-10 : 32x32 크기의 6만장 컬러 이미지가 포함되어 있으며, 10개의 클래스로 구성
  - torchvision.datasets.CIFAR10 을 사용하여 데이터셋 불러오기

```
transform = torchvision.transforms.Compose(
    [torchvision.transforms.ToTensor(),
     torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=14, shuffle=True)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=14, shuffle=False)
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

02. Introduction to Computer Vision with PyTorch

- **7. CIFAR-10 데이터셋을 사용하여 실제 이미지 학습하기**

  • LeNet 모델을 사용하여 이미지 분류 모델 구축

  - 이전 MNIST 이미지 분류 모델과의 차이점은 입력 데이터의 채널 수가 1에서 3으로 바뀐 점

    ※ input_size = (1,1,28,28) ➔ (1,3,32,32)

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.conv3 = nn.Conv2d(16,120,5)
        self.flat = nn.Flatten()
        self.fc1 = nn.Linear(120,64)
        self.fc2 = nn.Linear(64,10)

    def forward(self, x):
        x = self.pool(nn.functional.relu(self.conv1(x)))
        x = self.pool(nn.functional.relu(self.conv2(x)))
        x = nn.functional.relu(self.conv3(x))
        x = self.flat(x)
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

net = LeNet()
```

02. Introduction to Computer Vision with PyTorch

- **8. 전이학습을 이용하여 사전학습된 네트워크 사용**

  • 전이학습(Transfer Learning) : 한 신경망 모델에서 다른 신경망 모델로 일부 지식을 전달하는 접근 방식
  - 일반적으로 ImageNet과 같은 일부 대규모 이미지 데이터셋에서 훈련된 사전 훈련된 모델로 시작한다. 이러한 모델은 이미 일반 이미지에서 다른 기능을 추출하는 작업을 잘 수행할 수 있으며, 대부분의 경우 추출된 feature에 분류기를 구축하는 것만으로도 좋은 결과를 얻을 수 있다.

  • 데이터셋 : Kaggle Cats vs Dogs Dataset
  - 25,000장의 고양이와 강아지의 이미지로 구성된 데이터셋

  • 사용할 데이터셋 다운로드 및 압축 해제

```
if not os.path.exists('data/kagglecatsanddogs_3367a.zip'):
    !wget -P data -q https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zi

import zipfile
if not os.path.exists('data/PetImages'):
    with zipfile.ZipFile('data/kagglecatsanddogs_3367a.zip', 'r') as zip_ref:
        zip_ref.extractall('data')
```

02. Introduction to Computer Vision with PyTorch

- **8. 전이학습을 이용하여 사전학습된 네트워크 사용**

  • torchvision.datasets.ImageFolder 메서드를 이용하여 데이터셋 경로와 데이터 변환 방법을 지정하여 PyTorch dataset으로 로드

  • 사전학습된 모델(pre-trained VGG)에 넣기 위하여 resize, ToTensor, standard normalization 등 이미지 전처리 수행

```
std_normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                     std=[0.229, 0.224, 0.225])
trans = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        std_normalize])
dataset = torchvision.datasets.ImageFolder('data/PetImages',transform=trans)
trainset, testset = torch.utils.data.random_split(dataset,[20000,len(dataset)-20000])

display_dataset(dataset)
```

02. Introduction to Computer Vision with PyTorch

- **8. 전이학습을 이용하여 사전학습된 네트워크 사용**

  • 사전학습 모델
  - torchvision 모듈 안에 다양한 사전학습 모델이 있으며, 그 중 VGG-16 모델을 사용

```
vgg = torchvision.models.vgg16(pretrained=True)
```

  • 1,000개의 ImageNet 클래스 테이블 로드

```
import json, requests
class_map = json.loads(requests.get("https://s3.amazonaws.com/deep-learning-models/image-models/imagenet_class_index.json").text)
class_map = { int(k) : v for k,v in class_map.items() }

class_map[res[0].argmax().item()]
```

  • VGG-16 네트워크 구조 확인

```
summary(vgg,input_size=(1,3,224,224))
```

02. Introduction to Computer Vision with PyTorch

- **8. 전이학습을 이용하여 사전학습된 네트워크 사용**

  • VGG 네트워크를 사용하여 전이학습
  - 필요한 개수의 클래스(2 classes)를 생성하는 분류기를 최종 분류기로 배치
  - requires_grad = False 로 설정하여 합성곱 특징벡터 추출기의 가중치를 동결하여 수행
  ➔ 이로 인하여 최종 분류기의 가중치만 학습하게 되므로 더 적은 수의 샘플 데이터로 파라미터 미세 조정 가능

```
vgg.classifier = torch.nn.Linear(25088,2).to(device)

for x in vgg.features.parameters():
    x.requires_grad = False
```

  • 이 프로세스는 시간이 오래 걸리므로 사용자 정의 함수(train_long)을 이용하여 학습 중간 결과 출력

```
trainset, testset = torch.utils.data.random_split(dataset,[20000,len(dataset)-20000])
train_loader = torch.utils.data.DataLoader(trainset,batch_size=16)
test_loader = torch.utils.data.DataLoader(testset,batch_size=16)

train_long(vgg,train_loader,test_loader,loss_fn=torch.nn.CrossEntropyLoss(),epochs=1,print_freq=90)
```

8. 전이학습을 이용하여 사전학습된 네트워크 사용

• 전이학습 파인튜닝(Fine-tuning)

- 이전 섹션에서는 최종 분류기 계층을 훈련하여 자체 데이터셋에서 이미지를 분류했다. 그러나 feature 추출기를 다시 교육하지 않고, 모델이 ImageNet 데이터에서 학습한 featur에 의존했다. 새로운 이미지가 ImageNet 이미지와 시각적으로 다른 경우 이 feature의 조합이 잘 작동하지 않을 수 있다. 따라서 requires_grad = True 로 설정하여 합성곱 레이어 훈련도 시작하는 것이 타당하다.

```
for x in vgg.features.parameters():
    x.requires_grad = True
```

• 이 외 다른 컴퓨터비전 모델

- torchvision 패키지는 가장 간단한 컴퓨터 비전 아키텍처 중 하나인 VGG-16 뿐만 아니라 Microsoft의 ResNet과 Google의 Inception도 제공해준다. 아래 예시는 ResNet-18 모델이다.

```
resnet = torchvision.models.resnet18()
print(resnet)
```

9. MobileNet을 이용하여 비전 문제 해결

- 이전 섹션에서 복잡한 네트워크를 학습하기 위해서는 GPU와 같은 상당한 계산자원이 필요하다. 하지만 고성능 디바이스가 아닌 환경에서 구현되려면 더욱 경량화된 CNN 아키텍처가 필요하다. 따라서 MobileNet은 컴퓨터 성능이 제한되거나 배터리 퍼포먼스가 중요한 곳에서 사용될 목적으로 설계된 CNN 구조이다.

- MobileNet_v2 모델 로드

```
model = torch.hub.load('pytorch/vision:v0.6.0', 'mobilenet_v2', pretrained=True)
model.eval()
print(model)
```

- MobileNet_v2를 이용하여 전이학습 수행

```
for x in model.parameters():
    x.requires_grad = False

device = 'cuda' if torch.cuda.is_available() else 'cpu'
model.classifier = nn.Linear(1280,2)
model = model.to(device)
summary(model,input_size=(1,3,244,244))

train_long(model,train_loader,test_loader,loss_fn=torch.nn.CrossEntropyLoss(),epochs=1,print_freq=90)
```

End of document

Website

http://www.brframe.com/

Contact Point

E-mail admin@brframe.com