<Term Project 2021: Survey on Programming Paradigms>
파이썬 기반의 딥러닝과 주요 프레임워크 비교(TensorFlow vs PyTorch)

과 목 명 : 프로그래밍언어(공통1)

소 속 : 경제통상대학 경제학과

제 출 일 : 2021.05.06

학 번 : 20173027

이 름 : 전보배

목차

주제 선정 이유

그림1

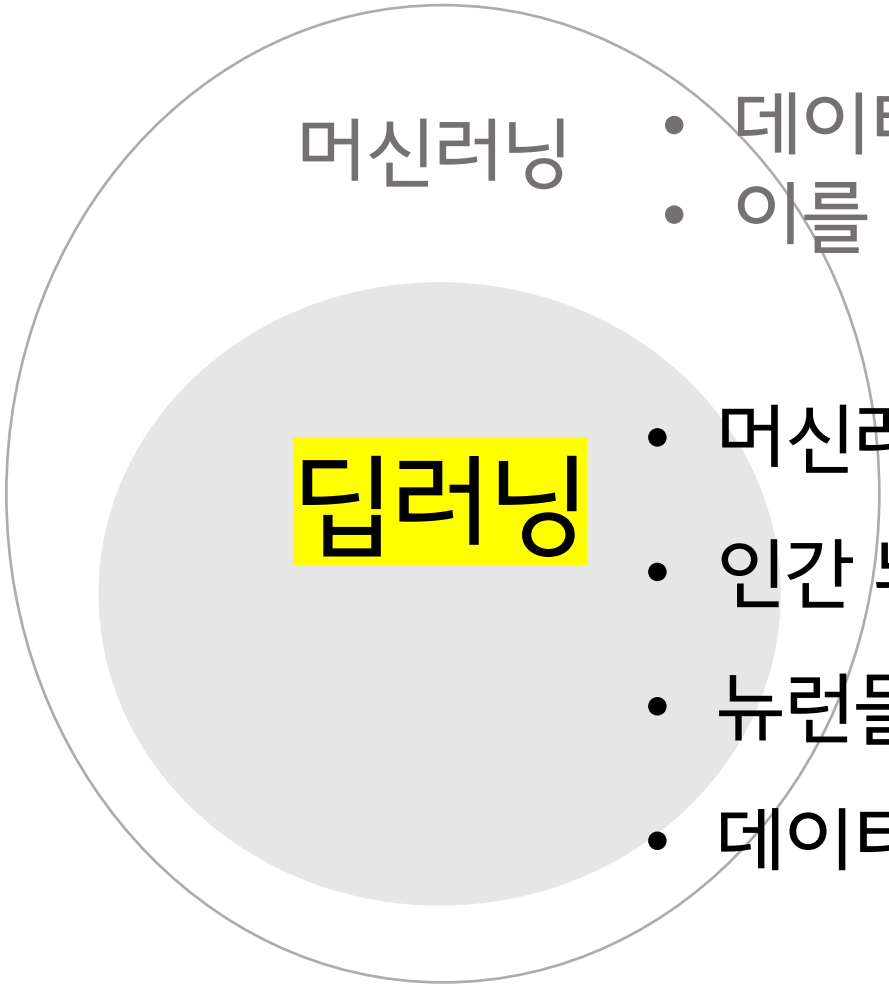
그림2

최근 각광받고 있는 분야인 딥러닝에 대해 알아보고,

딥러닝 알고리즘 개발을 위한 주요 프레임워크를 비교하고자 함

딥러닝 개요

- 딥러닝이란?[1]



- 데이터를 이용하여 데이터 특성과 패턴을 학습
- 이를 바탕으로 미지의 데이터에 대한 미래 결과 예측

- 머신러닝의 세부 분야
- 인간 뇌의 정보처리 과정과 유사하도록 함
- 뉴런들로 구성된 인공신경망을 여러 개 이용한 기계학습
- 데이터의 중요한 특징을 스스로 학습

딥러닝 개요

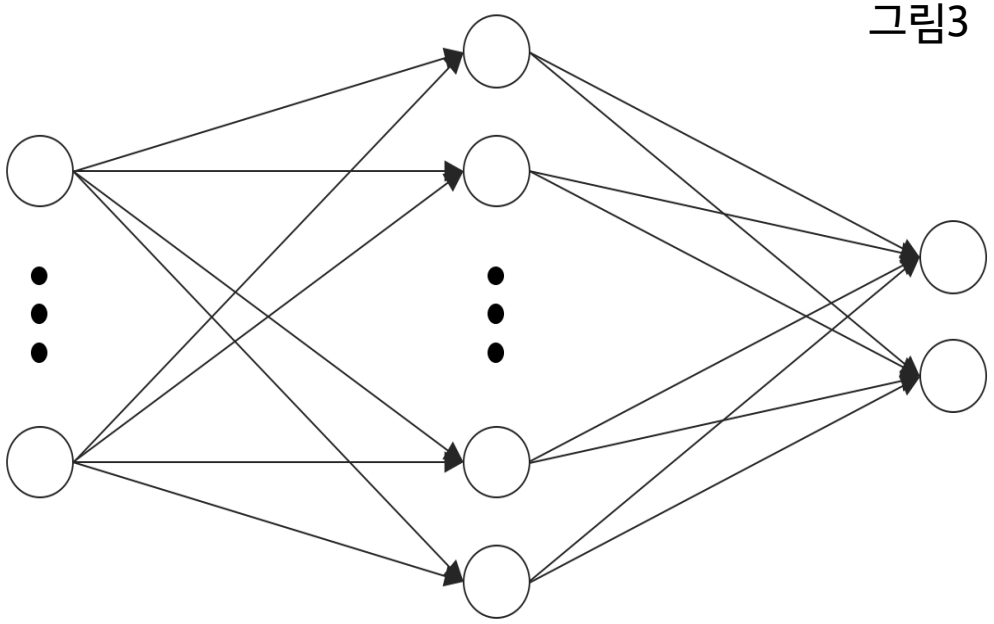
- 딥러닝이란? (Cont.)[2]

- 3개 이상의 Hidden layers로 구성

  → 딥러닝의 deep은 숨겨진 layer가 많음을 의미함

- 가중치 W와 bias로 뉴런간 연결
- 학습시마다 W와 bias를 업데이트해 데이터의 특성을 정확하게 파악하는 최적의 가중치와 바이어스를 찾음

그림3



딥러닝 개요

- 왜 딥러닝을 사용하는가? [3] [4] [5]

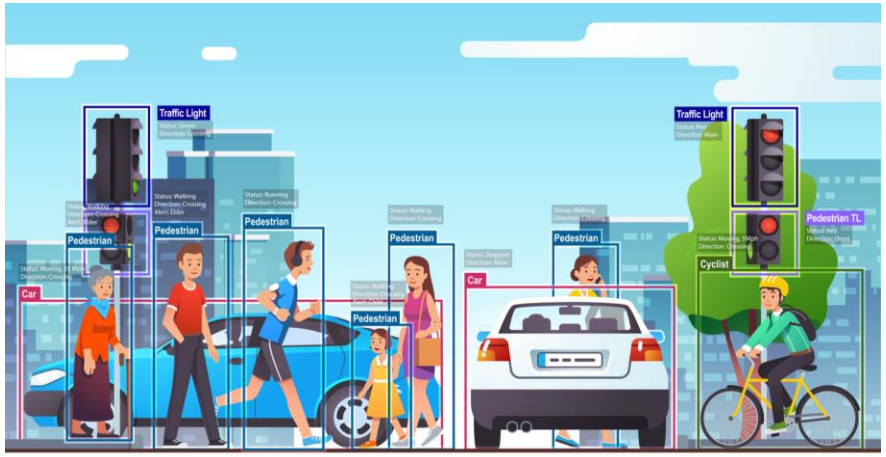
- 높은 정확도
- 사람보다 빠른 학습속도
- 기계학습과 비교하여 데이터 전처리 과정이 간단함

  → 데이터의 특징을 모델 스스로 찾을 수 있기 때문
- 알고리즘의 비약적인 향상
- 하드웨어 발전: GPU를 이용한 연산 소요시간 단축
- 빅데이터: SNS에서 생산되는 다량의 자료와 태그 정보 사용

딥러닝 개요

- 딥러닝의 사용 분야

그림4



컴퓨터비전

그림5

음성인식

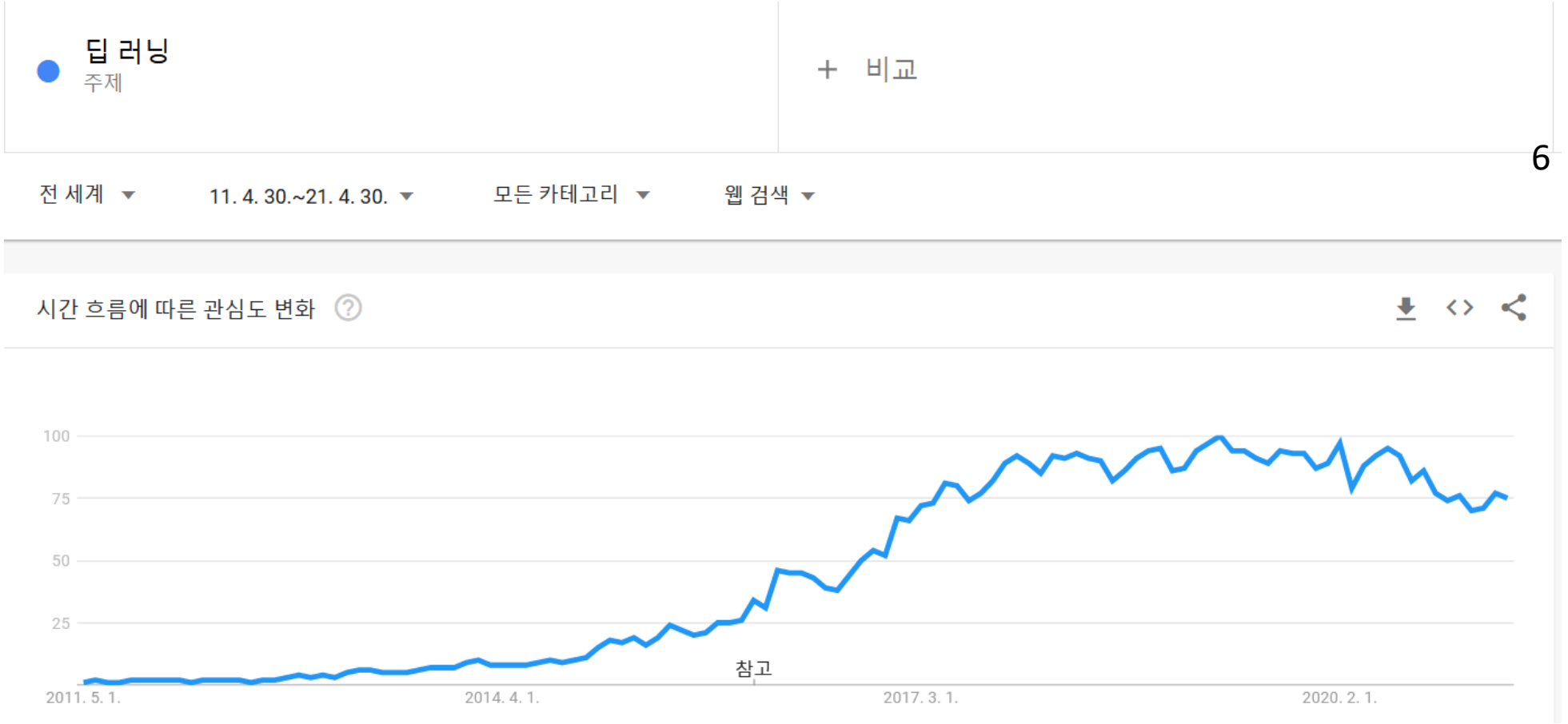
그림6

자연어처리

→ 위 분야에서 뛰어난 성능. 이를 응용한 프로그램들이 개발되어 널리 쓰이고 있음

딥러닝 개요

- 딥러닝 관심도 변화 →10년 전과 비교하여 관심도가 크게 증가한 것을 볼 수 있음



※ 왜 파이썬인가?

그림7

- 머신러닝/딥러닝 분야에서 파이썬을 주로 사용하는 이유[7]

1. 간단한 코드→ 기술적인 측면이 아닌 문제해결에만 집중할 수 있게 함
2. 다양한 라이브러리와 프레임워크 → 효율적인 프로그래밍 가능
3. 플랫폼 독립성(Platform independence) → 다양한 운영체제에서 실행 가능
4. 활발한 커뮤니티 → 양질의 피드백을 받을 수 있음

TensorFlow / PyTorch란?

- 딥러닝 프레임워크란?[8]
  - 프레임워크(Framework): 응용프로그램 개발 위한 클래스와 라이브러리 제공
  - 딥러닝 프레임워크: 많은 라이브러리 및 사전 학습까지 완료된 다양한 딥러닝 알고리즘 제공

- 딥러닝 프레임워크를 이용하면…[9]

[빠른 학습: GPU에서 프로그램을 실행할 수 있음
편리함: 모델 설계를 간편하게 할 수 있고 복잡한 미분 계산을 하지 않아도 됨]

TensorFlow / PyTorch란?

- 딥러닝 프레임워크 예시

그림8

그림9

그림10

그림11

그림12

그림13

Etc.

TensorFlow / PyTorch란?

- TensorFlow 개요[10] [11]

그림8

- 발표일: 2015년 11월
- 개발사: 구글 브레인팀
- 최신버전: 2.4.1
- 언어: 파이썬, C
- 플랫폼: 리눅스, 윈도우, MacOS, 안드로이드, iOS, 자바스크립트
- 특징: 머신러닝을 위한 오픈소스 플랫폼
- 웹사이트: https://www.tensorflow.org/

TensorFlow / PyTorch란?

- PyTorch 개요[12]

그림10

- 발표일: 2016년 10월
- 개발사: 페이스북 AI 리서치 랩
- 최신버전: 1.8.0
- 언어: 파이썬
- 플랫폼: 리눅스, 윈도우, MacOS
- 특징: 파이썬 기반의 과학 연산 패키지- GPU연산, 딥러닝에 사용
- 웹사이트: https://pytorch.org/

TensorFlow / PyTorch란?

- TensorFlow 특징[13 14 15]

- TensorFlow 2.0 공개(2019년): 이전보다 단순, 편리함
  (TensorFlow 1.0에서는 Session 등을 이용해서 직관성 ↓)
- 다양한 플랫폼에서 이용 가능(모바일, js)
- 모델 트레이닝부터 배포까지 지원
- Keras API를 이용해 모델 작성이 쉬워짐(TensorFlow 2.0)
- Static, 혹은 dynamic한 Computational Graph 사용
  (TensorFlow 2.0)
- 모델 학습 중 visualization 가능(TensorBoard 이용)

TensorFlow 2.0 Diagram

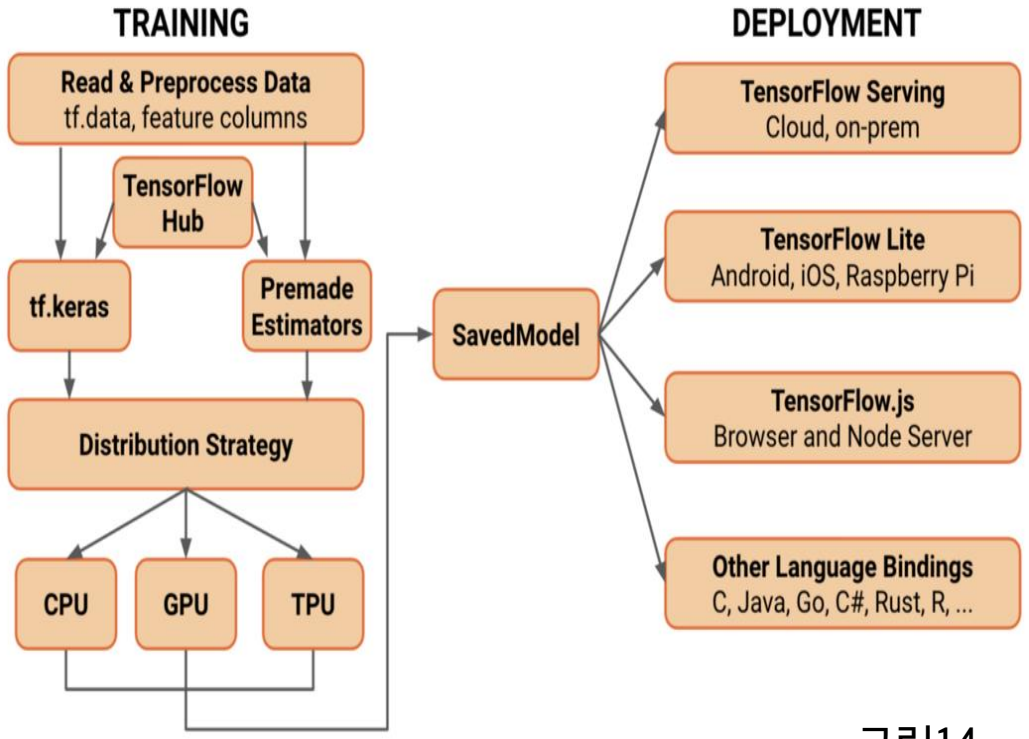


그림14

TensorFlow / PyTorch란?

- PyTorch 특징[14 15]

- 파이썬에 최적화 되어있음 → 쉽게 배울 수 있음
- 파이썬 디버깅 툴로 디버그 가능
- Dynamic한 Computational Graph 사용
- 공식 포럼을 운영하고 documentation이 잘 되어 있음
- 직관적인 모델 작성, 편리성 등은 PyTorch만의 장점이었으나 TensorFlow 2.0의 등장으로 큰 차이 없어짐

15

※ Computational Graph:
노드와 엣지를 이용해 계산 과정을 나타내는 그래프

Static Computational Graph:
모델을 학습하는 동안 파라미터가 고정됨

Dynamic Computational Graph:
모델을 학습하는 동안 파라미터를 바꿀 수 있음

→static graph가 속도상의 이점이 있으나
dynamic graph가 유연성이 높아 더 선호되는 추세

TensorFlow vs PyTorch

- 정확도, 학습 시간, 메모리 사용량 비교[16]

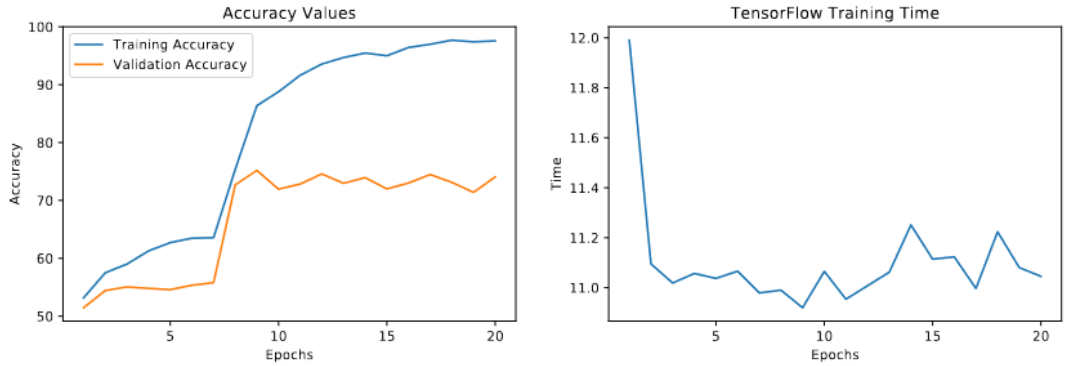


Figure 1: TensorFlow Accuracy and Training Time

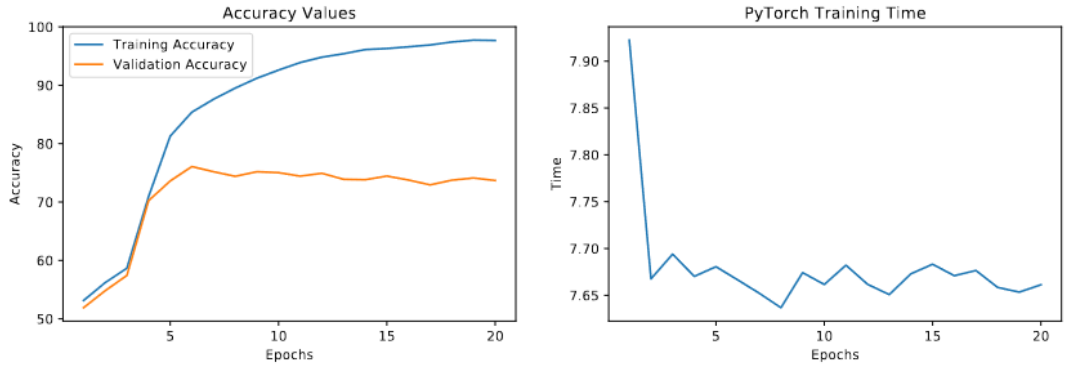


Figure 2: PyTorch Accuracy and Training Time

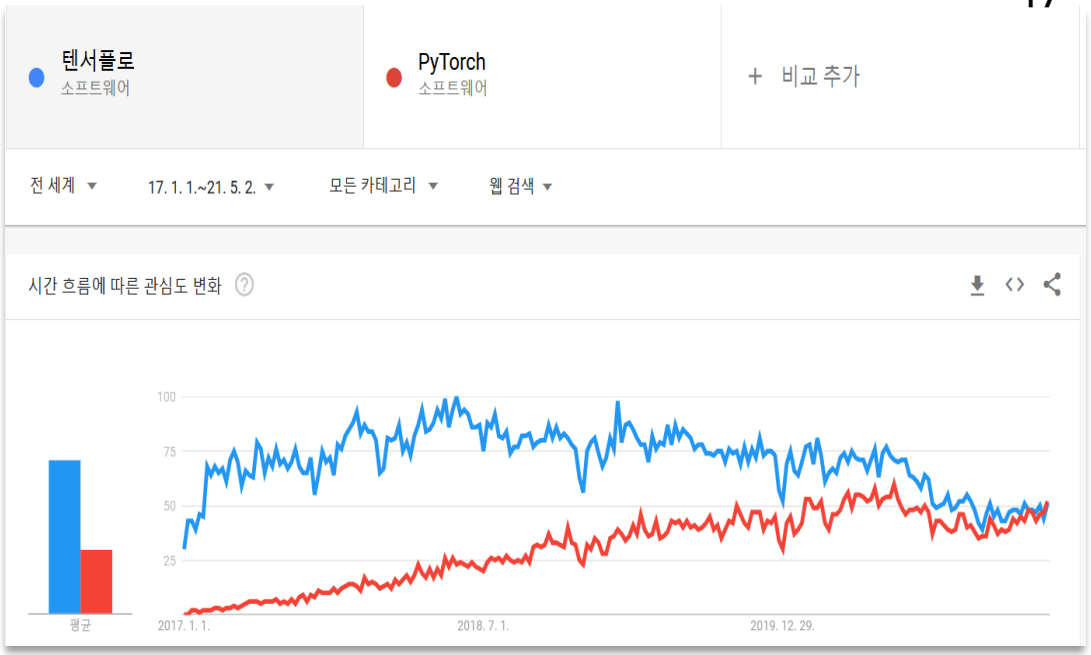
- 정확도: 큰 차이 없음
- 학습 시간: PyTorch가 더 빠름
  (TensorFlow 평균: 11.19초 / PyTorch 평균: 7.67초)
- 학습시 메모리 사용량: TensorFlow가 더 적음
  (TensorFlow: RAM 1.7 GB / PyTorch: RAM 3.5 GB)

← TensorFlow와 PyTorch의 정확도와 학습시간 비교

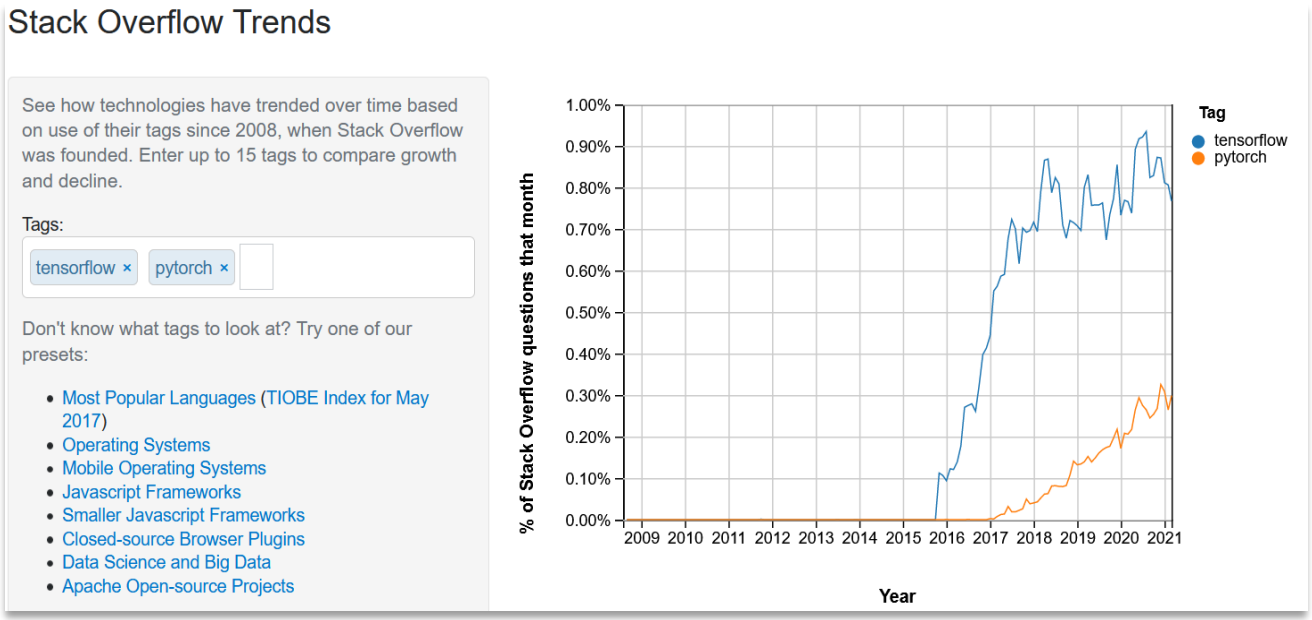
TensorFlow vs PyTorch

- 트렌드(구글, Stack Overflow) 비교

17



18



→ Stack Overflow 트렌드 분석 결과 압도적으로 TensorFlow 사용자가 많은 것으로 보이나

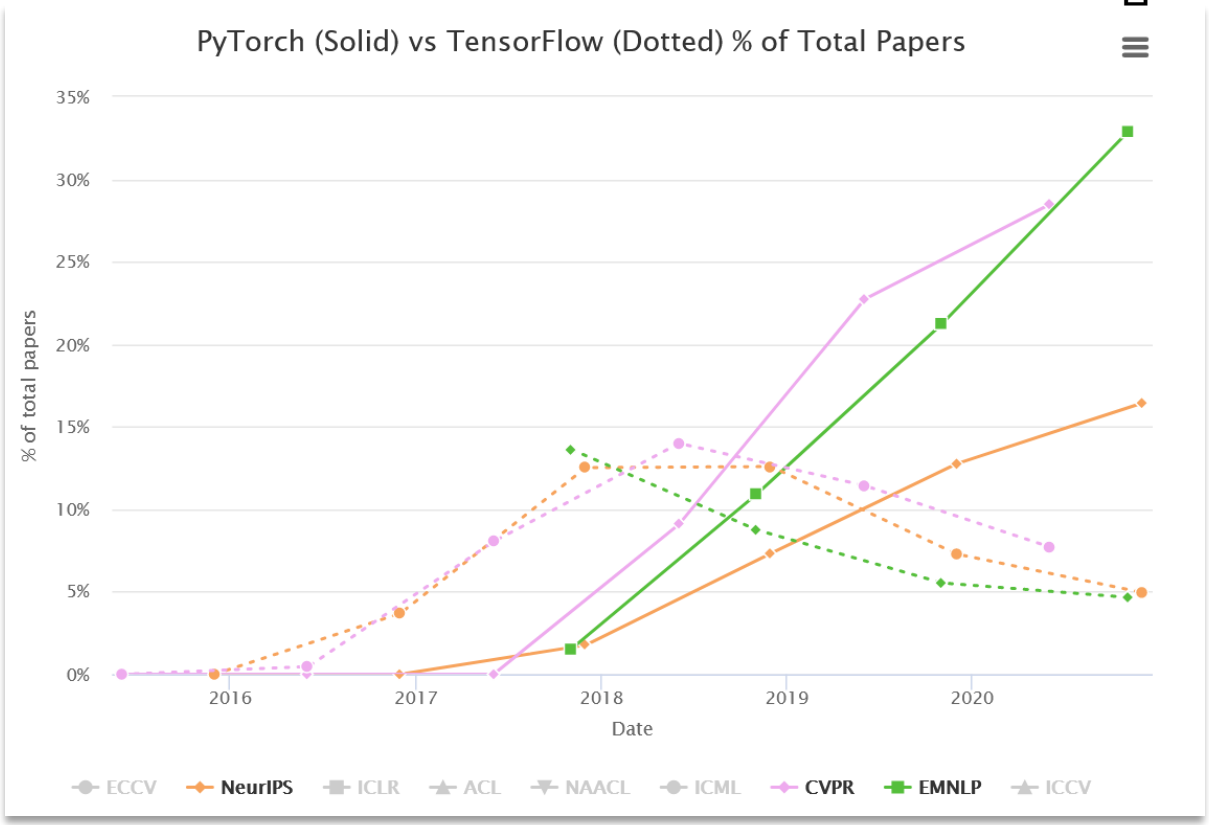
PyTorch 관심도가 빠르게 증가하고 있어 PyTorch 검색량과 TensorFlow 검색량은 비슷해지는 추세

TensorFlow vs PyTorch

그림15

- 트렌드 비교[19]

State of AI Report 2020에 따르면…



- 깃허브에 발표한 논문 분석 결과
  → PyTorch 이용자 47% (TensorFlow 18%)

- 주요 연구논문 중 20-35%만 사용 프레임워크 기술
  → 이 중 75%가 PyTorch를 사용.

- 2018년에 TensorFlow로 논문을 발표한 161명 중 55%가 이후 PyTorch로 바꿈 (반대의 경우는 15%)

TensorFlow vs PyTorch

- 개발환경: 구글 Colab
- 구현 목표: 단순한 CNN 모델을 구축하여 코드 작성 방식 및 정확도 비교
- 모델 학습 및 평가에 사용할 데이터셋: MNIST

(full code: https://github.com/bobaejeon/pl2021)

그림16

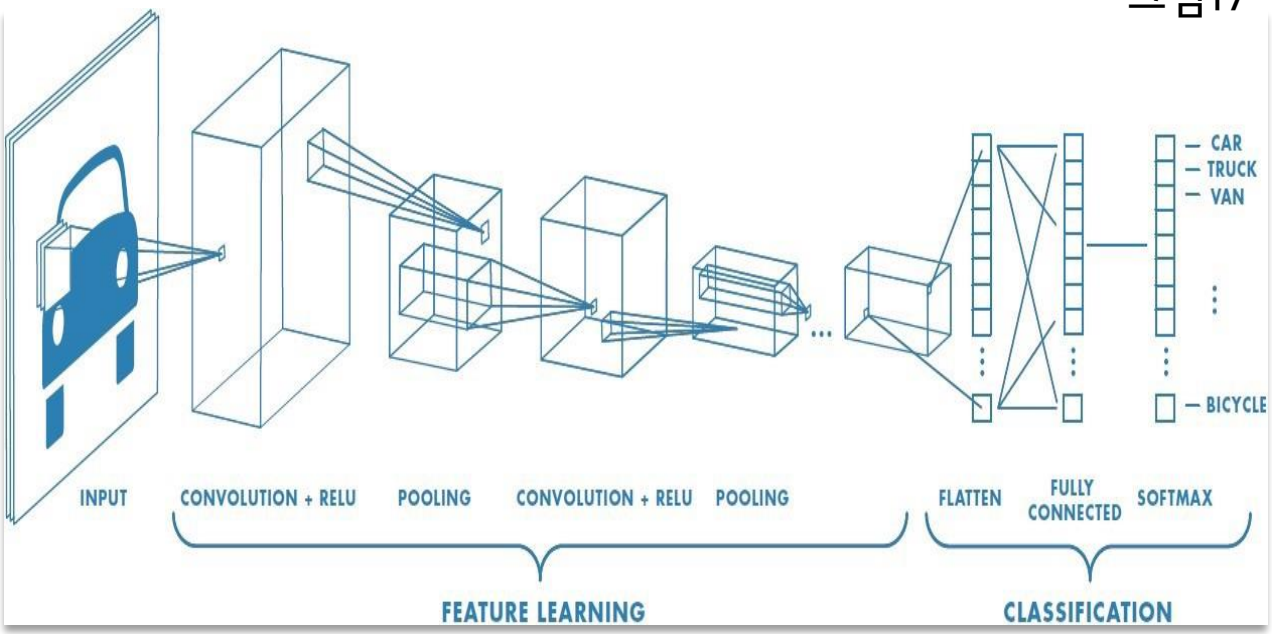
- 0부터 9까지 손글씨 숫자 이미지 데이터베이스[20]
- 학습 이미지 60,000개, 테스트 이미지 10,000개

  → 각 이미지 사이즈는 28x28
- 머신러닝의 “Hello world”라고 할 수 있음

※ CNN 이란?

- Convolutional Neural Network[21]

그림17



- 이미지 데이터 처리에 주로 사용
- Convolution Layer, Pooling Layer 사용

[Convolution Layer: 데이터 특징 추출

Pooling Layer: 샘플 사이즈 줄임

→과적합(overfitting) 방지 / 연산속도 증가

(overfitting: 학습 데이터를 과하게 학습해 실제 데이터를 잘 예측하지 못함)

TensorFlow vs PyTorch: 구현 비교

- import 및 데이터 로드: TensorFlow

- MNIST는 유명한 데이터셋이기 때문에 각 프레임워크에서 함수를 이용하여 쉽게 다운받을 수 있다.

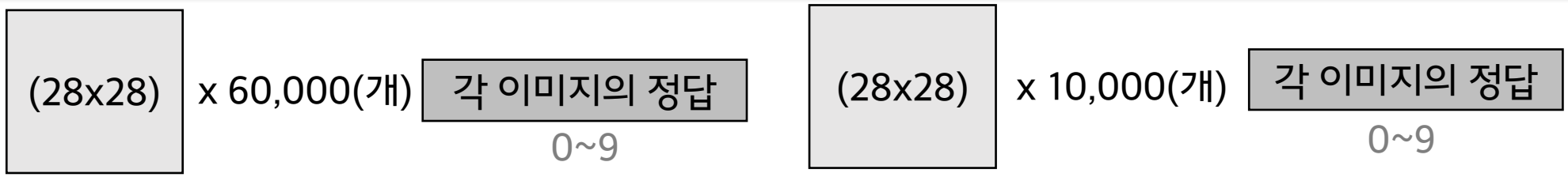
```
import tensorflow as tf
import matplotlib.pyplot as plt ← 샘플 이미지를 출력할 때 필요.
```

Keras 라이브러리를 사용함

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
```

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```



TensorFlow vs PyTorch: 구현 비교

- import 및 데이터 로드: PyTorch

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

학습 데이터인가? True : False

연산을 위해 tensor
데이터 타입으로 바꿔야 함

```
train_dataset = torchvision.datasets.MNIST(root='./data/',
                                           train=True,
                                           transform=torchvision.transforms.ToTensor(),
                                           download=True)
test_dataset = torchvision.datasets.MNIST(root='./data/',
                                          train=False,
                                          transform=torchvision.transforms.ToTensor(),
                                          download=True)

train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=32)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=32)
```

→ iterator처럼 작용함: 데이터 순회 기능

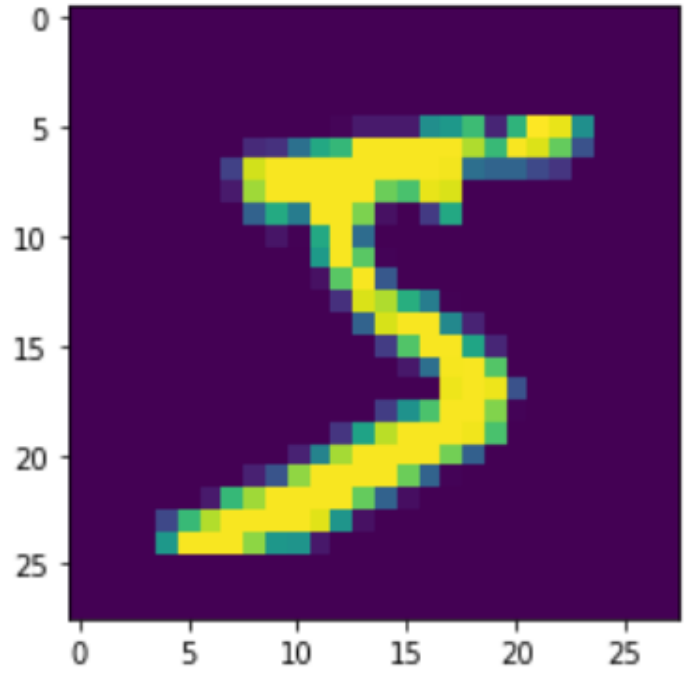
※ 텐서 tensor:
데이터의 배열
(3차원 이상)

TensorFlow vs PyTorch: 구현 비교

- 학습 데이터 표시

```
plt.imshow(train_images[0])
print(train_labels[0])
```

```
5
```



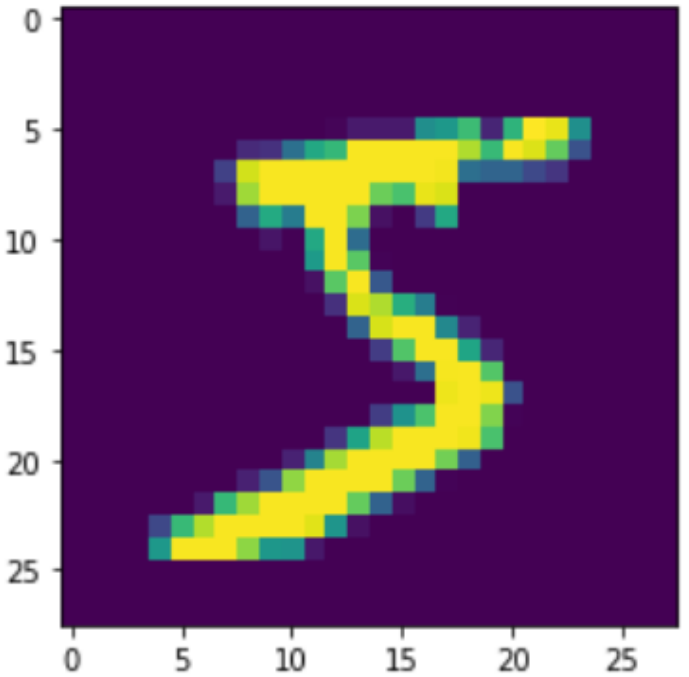
TensorFlow

```
image, label = train_dataset[0]

plt.imshow(image.squeeze())
print(label)
```

→ 사이즈가 1인 차원을 제거

```
5
```



PyTorch

TensorFlow vs PyTorch: 구현 비교

- CNN 모델: TensorFlow

- 데이터 전처리: reshape 및 정규화

```
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# normalize
train_images, test_images = train_images / 255.0, test_images / 255.0
```

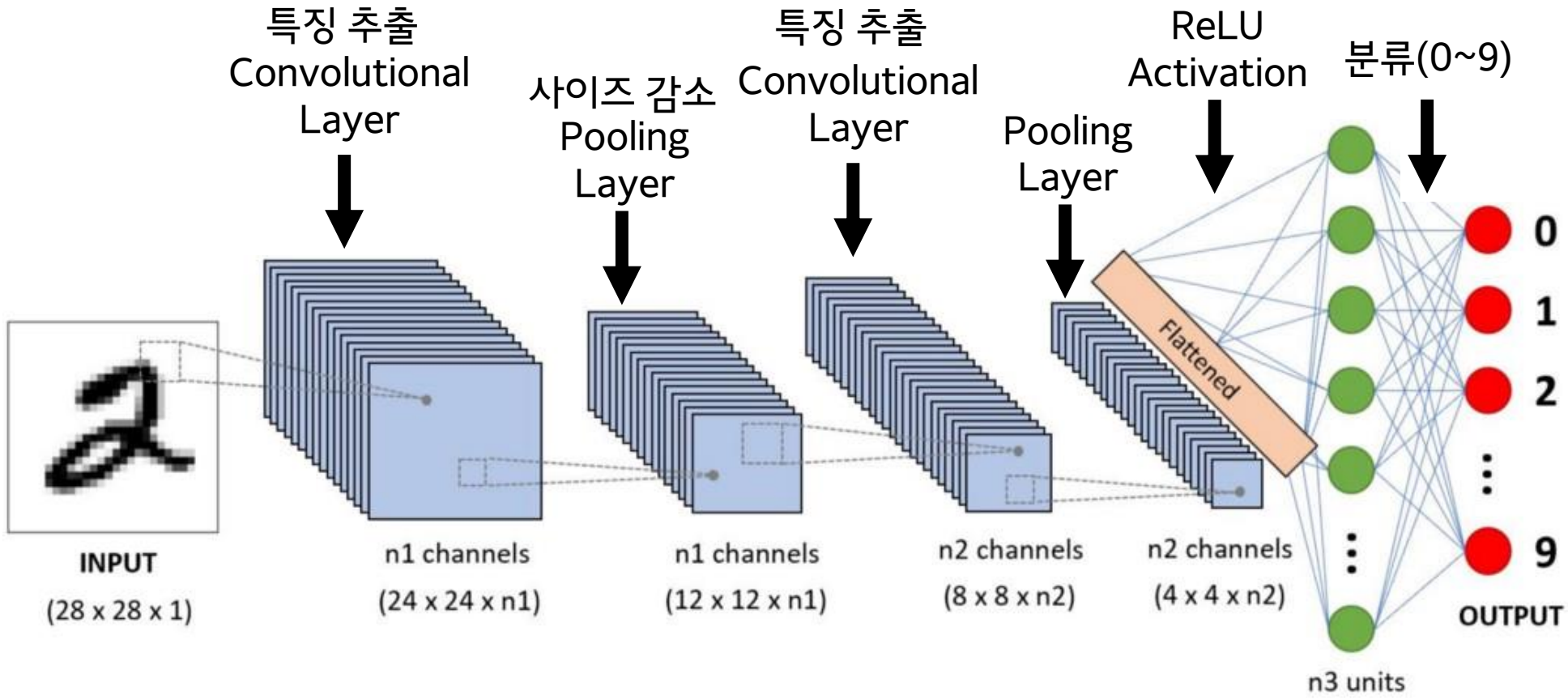
← CNN은 이미지 개수를 제외하고
(이미지 높이, 이미지 너비, 컬러 채널)의
텐서를 입력으로 받음

↑ 픽셀 값을 0~1로 Normalize

TensorFlow vs PyTorch: 구현 비교

- 사용할 모델 구조

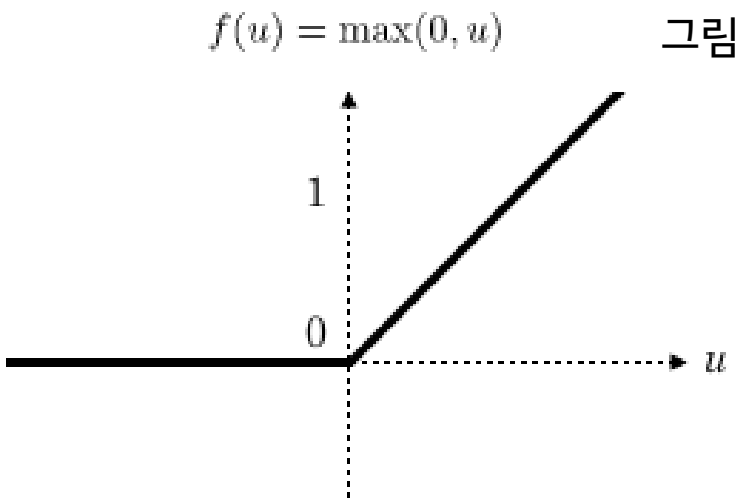
그림18



※ Activation function(활성화 함수)[22]

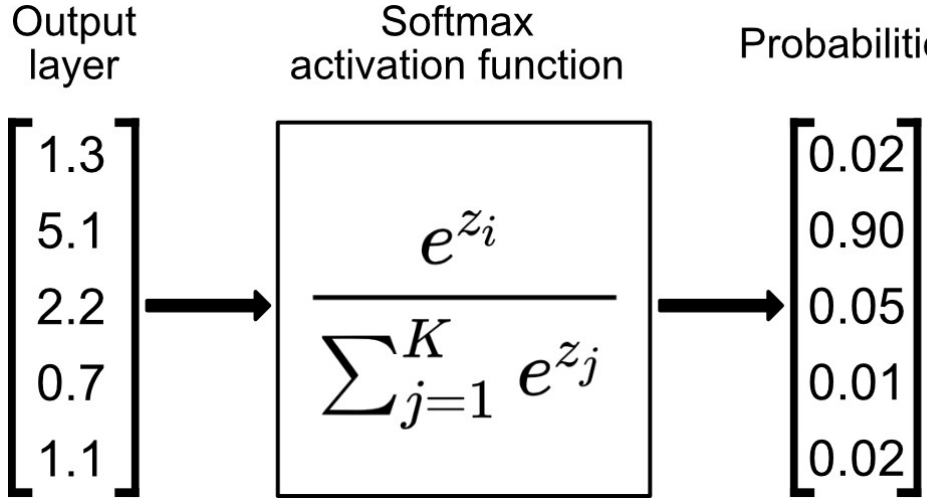
- 레이어에 들어오는 값들에 대한 출력을 결정하는 함수.
- 각 레이어를 비선형 결합으로 만들어 줌.
- 이를 사용하지 않으면 입력값에 대한 출력값이 linear하게 나오게 됨
- 여러 종류가 있으나 본 코드에서는 ReLU와 Softmax 사용

그림19



ReLU(Rectified Linear Unit)
양수만 그대로 보존

그림20



Softmax
Output:각 클래스에 대한 확률

TensorFlow vs PyTorch: 구현 비교

- CNN 모델: TensorFlow

- Keras Sequential API 사용

출력 필터의 수

(5x5)인 필터 사용해 특징 추출

MNIST 이미지 포맷인 (28, 28, 1) 크기의 입력을 처리하는 CNN

```
modeltf = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(5, 5), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=10, activation='softmax')
])
```

→ 3차원 행렬을 1차원으로 변환

→ 10개의 클래스(0~9)로 분류

(2x2) 영역의 max값만 남김

TensorFlow vs PyTorch: 구현 비교

- CNN 모델: PyTorch

- nn.Module을 상속받아 생성자와 forward 메소드 등을 구현해야 함.

```
class PyNet(torch.nn.Module):
  def __init__(self):
        super().__init__()
        self.flatten = torch.nn.Flatten()
        self.cnn_model = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels=1, out_channels=32, kernel_size=(5, 5)),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=(2, 2)),
            torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(5, 5)),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=(2, 2)),
            torch.nn.ReLU()
        )
        self.fc_model = torch.nn.Sequential(
            torch.nn.Linear(in_features=4*4*64, out_features=64),
            torch.nn.ReLU()
        )
        self.classifier = torch.nn.Linear(64, 10)

  def forward(self, x):
        x = self.cnn_model(x)
        x = self.flatten(x)
        x =  self.fc_model(x)
        out = self.classifier(x)
        return out
```

TensorFlow로 작성한 것과 동일한 모델

forward: 호출시마다 수행될 내용 정의

※ Loss function과 Optimizer

- Loss function[23]

- 모델이 얼마나 데이터를 정확하게 예측하는지 평가하는 정량적 지표
- 예측 값과 실제 값에 대한 오차를 계산
- 숫자가 적을 수록 좋음

- Optimizer[24]

- 학습 속도를 빠르게 / 학습을 안정적으로 할 수 있도록 함
- Loss를 줄이는 방향으로 가중치와 bias를 조정함

TensorFlow vs PyTorch: 구현 비교

- Loss function과 Optimizer

- 사용한 loss function: Categorical Cross Entropy Loss
- 사용한 optimizer: Adam Optimizer

```
modelTf.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
```

TensorFlow

```
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(modelPy.parameters())
```

PyTorch

TensorFlow vs PyTorch: 구현 비교

- 모델 학습(Train): TensorFlow
- fit 메소드 사용, 총 5번 학습을 진행함

```
modeltf.fit(train_images, train_labels, epochs=5)
# 60000 examples / 32 batch size = 1875 number of batches
```

```
Epoch 1/5
1875/1875 [==============================] - 55s 29ms/step - loss: 0.2914 - accuracy: 0.9097
Epoch 2/5
1875/1875 [==============================] - 54s 29ms/step - loss: 0.0400 - accuracy: 0.9884
Epoch 3/5
1875/1875 [==============================] - 53s 28ms/step - loss: 0.0262 - accuracy: 0.9916
Epoch 4/5
1875/1875 [==============================] - 53s 28ms/step - loss: 0.0211 - accuracy: 0.9935
Epoch 5/5
1875/1875 [==============================] - 53s 28ms/step - loss: 0.0157 - accuracy: 0.9946
```

loss는 점점 감소하고 정확도는 증가함

TensorFlow vs PyTorch: 구현 비교

- 모델 학습(Train): PyTorch
- TensorFlow와 달리 직접 메소드에 모든 과정을 작성해야 함

```python
def train(dataloader, model, loss_fn, optimizer, epoch):
    model.train() ← train mode로 세팅
    size = len(dataloader.dataset)
    for e in range(epoch):
      loss, test_loss, correct = 0, 0, 0
      for X, y in dataloader:
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        test_loss = loss.item()
        correct += (pred.argmax(1) == y).type(torch.float).sum().item()

      print(f"Epoch: {e+1}, loss: {test_loss:>5f}, accuracy: {(correct / size):>5f}")
```

결과:

```
Epoch: 1, loss: 0.028312, accuracy: 0.954533
Epoch: 2, loss: 0.003180, accuracy: 0.985700
Epoch: 3, loss: 0.000613, accuracy: 0.990600
Epoch: 4, loss: 0.000633, accuracy: 0.993500
Epoch: 5, loss: 0.000072, accuracy: 0.994700
```

```python
train(train_dataloader, modelpy, loss_fn, optimizer, epoch=5)
```

← 마찬가지로 총 5번 학습 진행

TensorFlow vs PyTorch: 구현 비교

- 평가(Test)

```
test_loss, test_acc = modeltf.evaluate(test_images,  test_labels)
# 10000 test imgs / 32 batch size = 313 number of batches
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0301 - accuracy: 0.9914
```

TensorFlow
evaluate 메소드 사용

```
modelpy.eval()
```
← evaluation mode로 세팅

```
size = len(test_dataloader.dataset)
test_loss, correct = 0, 0

with torch.no_grad():
  for X, y in test_dataloader:
    pred = modelpy(X)
    correct += (pred.argmax(1) == y).type(torch.float).sum().item()

print(f"Test Result- accuracy: {(correct / size):>5f}")
```

결과 예측(pred) 및 정확도 계산

```
Test Result- accuracy: 0.989500
```

PyTorch

TensorFlow vs PyTorch: 구현 비교

- 결과(정확도) 비교

```
test_loss, test_acc = modeltf.evaluate(test_images,  test_labels)
# 10000 test imgs / 32 batch size = 313 number of batches
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0301 - accuracy: 0.9914
```

```
modelpy.eval()
size = len(test_dataloader.dataset)
test_loss, correct = 0, 0

with torch.no_grad():
  for X, y in test_dataloader:
    pred = modelpy(X)
    correct += (pred.argmax(1) == y).type(torch.float).sum().item()

print(f"Test Result- accuracy: {(correct / size):>5f}")
```

```
Test Result- accuracy: 0.989500
```

TensorFlow
99.14%

>

PyTorch
98.95%

(0.19% 차이)

결론

- 요약 및 평가

- TensorFlow 2.0 환경에서 코드 작성 방식이 매우 유사하고
- 동일한 언어 파이썬을 사용하므로 readability, writability, reliability, cost 측면에서 거의 차이 없음
- 직접 작성한 코드로는 TensorFlow의 정확도가 더 우세하나 거의 비슷한 양상을 보임

- **TensorFlow**

  Keras API를 사용하면 코드가 매우 심플함. 이 외에도 다양한 수준의 API를 제공하고

  다양한 환경(모바일 등)에서 모델 구축부터 배포까지 할 수 있음. 학습 시 메모리 사용량이 적다.

- **PyTorch**

  TensorFlow로 작성한 코드보다 다소 복잡해 보이나 어렵지 않음. 디버깅이 쉽고 학습 시간이 적게 소요됨.

결론

- 무엇을 사용해야 할까? (개인 소견)

- 각 프레임워크마다 장단점이 있으므로 사용 목적에 따라 선택하면 될 것으로 보임

1. **활발한 커뮤니티를 가진 프레임워크를 찾는다면: TensorFlow and PyTorch**

   →모두 사용자가 많고 활발한 커뮤니티를 가지고 있음.

   단, TensorFlow2.0이 등장한지 얼마 되지 않았고 1.0과 큰 차이가 나므로 인터넷에서 정보를 구할 때 혼란스러울 수도.

2. **딥러닝 초심자가 간단하게 공부하고 싶다면: TensorFlow**

   →코드를 직접 작성하여 비교한 결과 TensorFlow의 Keras API가 비교적 간단했음

3. **딥러닝 프레임워크를 시작으로 깊게 공부하고 싶다면: PyTorch**

   →급격히 발전하는 분야이며 PyTorch를 사용한 논문이 많아지는 추세기 때문에 참고할 게 많음

4. **임베디드 시스템, 모바일에서 사용하거나 애플리케이션을 만들고 배포하고자 한다면: TensorFlow**

   →TensorFlow가 안정적으로 지원함

참고문헌: 글

-딥러닝 개요-

1. 도서: 머신 러닝을 위한 파이썬 한 조각/박성호/비제이퍼블릭/2020
2. 알파고 바둑 실력의 비밀, '딥 러닝(Deep Learning) http://news.unist.ac.kr/kor/column_202/
3. Deep Learning https://www.ibm.com/cloud/learn/deep-learning
4. 딥러닝(위키백과)- 왜 다시 딥러닝인가? https://ko.wikipedia.org/wiki/%EB%94%A5_%EB%9F%AC%EB%8B%9D#%EC%99%9C_%EB%8B%A4%EC%8B%9C_%EB%94%A5_%EB%9F%AC%EB%8B%9D%EC%9D%B8%EA%B0%80?
5. 1.3 왜 딥러닝일까? https://tensorflow.blog/%EC%BC%80%EB%9D%BC%EC%8A%A4-%EB%94%A5%EB%9F%AC%EB%8B%9D/1-3-%EC%99%9C-%EB%94%A5%EB%9F%AC%EB%8B%9D%EC%9D%BC%EA%B9%8C-%EC%99%9C-%EC%A7%80%EA%B8%88%EC%9D%BC%EA%B9%8C/
6. 구글 트렌드(딥러닝) https://trends.google.com/trends/explore?geo=US&q=%2Fm%2F0h1fn8h
7. Why Use Python for AI and Machine Learning? https://steelkiwi.com/blog/python-for-ai-and-machine-learning/

-Tensorflow / PyTorch란-

8. Framework definition https://techterms.com/definition/framework
9. Deep Learning Frameworks https://developer.nvidia.com/deep-learning-frameworks
10. TensorFlow guide https://www.tensorflow.org/guide
11. 텐서플로(위키백과) https://ko.wikipedia.org/wiki/%ED%85%90%EC%84%9C%ED%94%8C%EB%A1%9C
12. PyTorch(README.md) https://github.com/pytorch/pytorch

참고문헌: 글

13. PyTorch vs TensorFlow: Difference you need to know https://hackr.io/blog/pytorch-vs-tensorflow
14. Pytorch vs Tensorflow: A Head-to-Head Comparison https://viso.ai/deep-learning/pytorch-vs-tensorflow/
15. TensorFlow vs PyTorch – A Detailed Comparison https://www.machinelearningplus.com/deep-learning/tensorflow1-vs-tensorflow2-vs-pytorch/

-TensorFlow vs PyTorch-

16. A Comparison of Two Popular Machine Learning Frameworks http://www.ccsc.org/publications/journals/SE2019.pdf#page=20
17. 구글 트렌드(TensorFlow vs PyTorch) https://trends.google.com/trends/explore?geo=US&q=%2Fg%2F11bwp1s2k3,%2Fg%2F11gd3905v1
18. 스택 오버플로우 트렌드(TensorFlow vs PyTorch) https://insights.stackoverflow.com/trends?tags=tensorflow%2Cpytorch
19. State of AI Report 2020 https://docs.google.com/presentation/d/1ZUimafgXCBSLsgbacd6-a-dqO7yLyzll1ZJbiCBUUT4/edit#slide=id.g8b560ae0a6_0_49
20. MNIST database https://en.wikipedia.org/wiki/MNIST_database
21. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
22. Activation function https://en.wikipedia.org/wiki/Activation_function
23. Introduction to loss functions https://algorithmia.com/blog/introduction-to-loss-functions
24. Various Optimization Algorithms For Training Neural Network https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

참고문헌: 그림

-주제 선정 이유-

그림1 https://commons.wikimedia.org/wiki/File:AlphaGo.svg

그림2 https://www.analyticssteps.com/blogs/deep-learning-overview-practical-examples-popular-algorithms

-딥러닝 개요-

그림3 http://news.unist.ac.kr/kor/column_202/

그림4 https://soft-cluster.com/computer-vision/

그림5 https://tweakreviews.com/gadgets/speech-recognition-in-outlook

그림6 https://www.slashgear.com/google-translate-finally-gets-new-languages-in-latest-update-26611357/

그림7 https://www.python.org/community/logos/

-TensorFlow / PyTorch란?-

그림8 https://github.com/tensorflow/tensorflow

그림9 https://keras.io/

그림10 https://icon-icons.com/icon/pytorch-logo/169823

참고문헌: 그림

그림11 https://github.com/apache/incubator-mxnet

그림12 https://en.wikipedia.org/wiki/Microsoft_Cognitive_Toolkit

그림13 https://software.intel.com/content/www/us/en/develop/tools/frameworks.html

그림14 https://medium.com/tensorflow/whats-coming-in-tensorflow-2-0-d3663832e9b8

-TensorFlow vs PyTorch-

그림15 http://horace.io/pytorch-vs-tensorflow/

그림16 https://en.wikipedia.org/wiki/MNIST_database

그림17 https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

그림18 https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (일부 수정)

그림19 https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847

그림20 https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60