

파이썬을 활용한 컴퓨터 비전 입문

Chapter 16. 딥러닝과 OpenCV

동양미래대학교
인공지능소프트웨어학과
권 범

본 강의자료는 길벗 출판사의 『OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝』
교재 내용을 토대로 작성되었습니다.

❖ 16장 딥러닝과 OpenCV

- 16.1 딥러닝과 OpenCV DNN 모듈
- 16.2 딥러닝 학습과 OpenCV 실행
- 16.3 OpenCV와 딥러닝 활용

16.1 딥러닝과 OpenCV DNN 모듈

16.2 딥러닝 학습과 OpenCV 실행

16.3 OpenCV와 딥러닝 활용

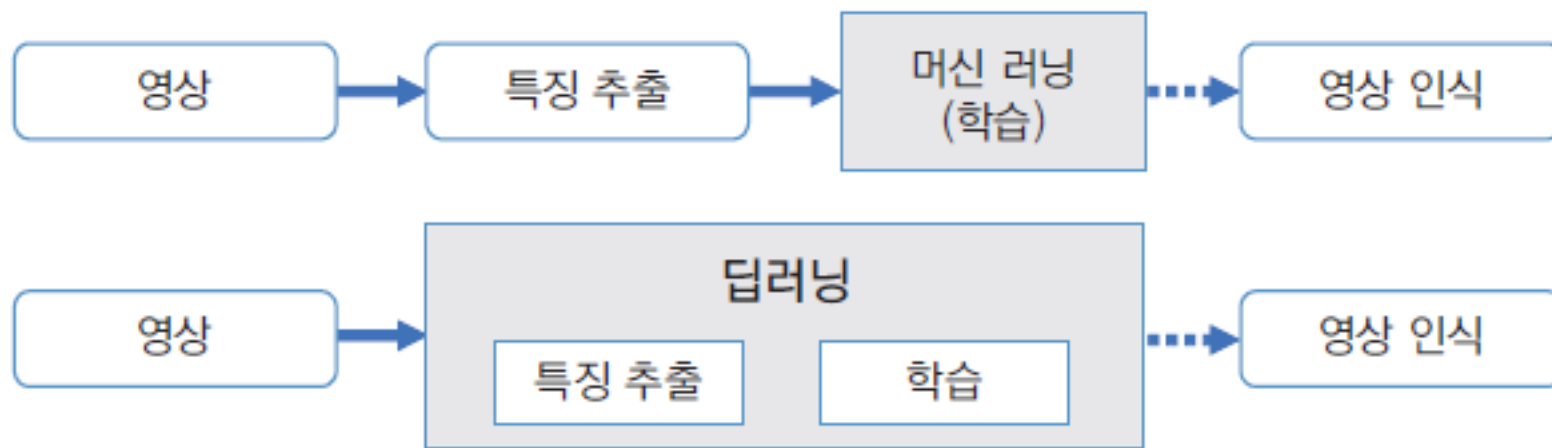
❖ 신경망과 딥러닝 (1/16)

- **딥러닝(deep learning)**은 2000년대부터 사용되고 있는 **심층 신경망(deep neural network)**의 또 다른 이름임
- **신경망(neural network)**은 **인공 신경망(artificial neural network)**이라고도 불림
- 이는 사람의 뇌 신경 세포(neuron)에서 일어나는 반응을 모델링하여 만들어진 고전적인 머신 러닝 알고리즘임
- 딥러닝이란 신경망을 여러 계층(layer)으로 쌓아서 만든 머신 러닝 알고리즘의 한 종류임
- 컴퓨터 비전 분야에서 딥러닝 기술이 크게 주목받고 있는 이유는 객체 인식, 얼굴 인식, 객체 검출, 분할 등의 다양한 영역에서 **딥러닝이 적용된 기술이 기존의 컴퓨터 비전 기반 기술보다 월등한 성능을 보여 주고 있기 때문임**

❖ 신경망과 딥러닝 (2/16)

- 기존의 머신 러닝 학습에서는 영상으로부터 인식에 적합한 특징을 사람이 추출하여 머신 러닝 알고리즘 입력으로 전달함
- 머신 러닝 알고리즘이 특징 벡터 공간에서 여러 클래스 영상을 상호 구분하기에 적합한 규칙을 찾아냄
- 이때 사람이 영상에서 추출한 특징이 영상 인식에 적합하지 않다면 어떤 머신 러닝 알고리즘을 사용한다고 하더라도 좋은 인식 성능을 나타내기 어려움

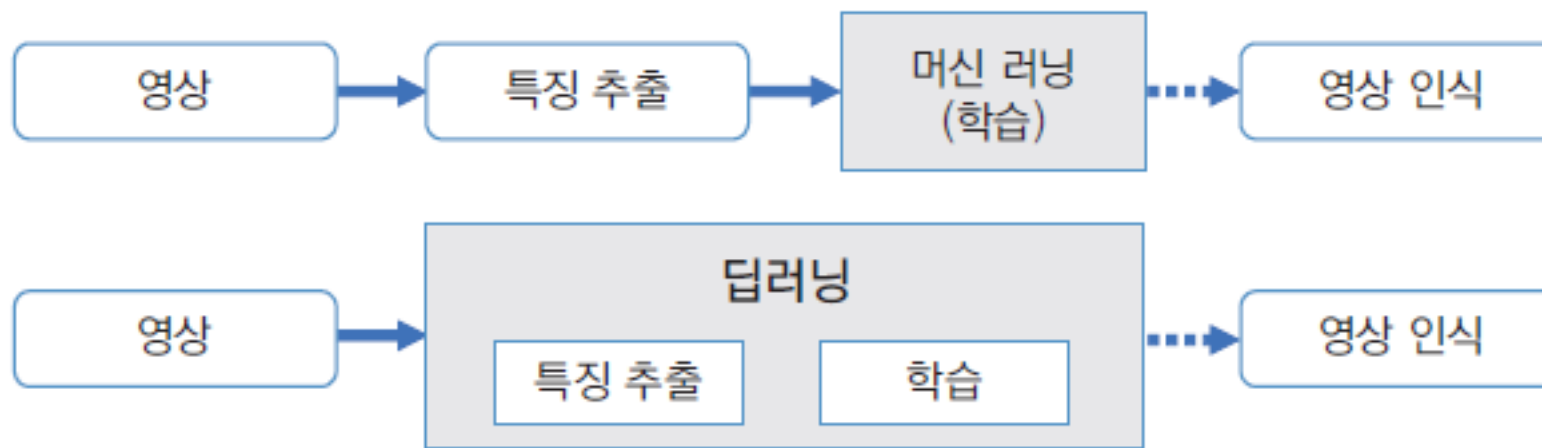
▼ 그림 16-1 전통적인 머신 러닝과 딥러닝 학습 과정



❖ 신경망과 딥러닝 (3/16)

- 최근의 딥러닝은 특징 추출과 학습을 모두 딥러닝이 알아서 수행함
- 여러 영상을 분류하기 위해 적합한 특징을 찾는 것과 이 특징을 잘 구분하는 규칙까지 딥러닝이 한꺼번에 찾아낼 수 있음

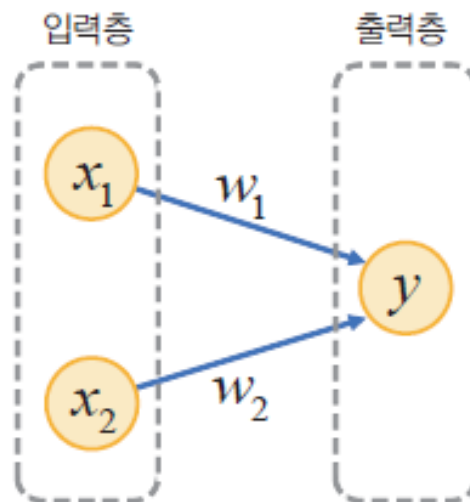
▼ 그림 16-1 전통적인 머신 러닝과 딥러닝 학습 과정



❖ 신경망과 딥러닝 (4/16)

- 신경망의 가장 기초적인 형태는 1950년대에 개발된 **퍼셉트론(perceptron)** 구조임
- 퍼셉트론 구조는 기본적으로 다수의 입력으로부터 가중합을 계산함
- 이를 이용하여 하나의 출력을 만들어 내는 구조임

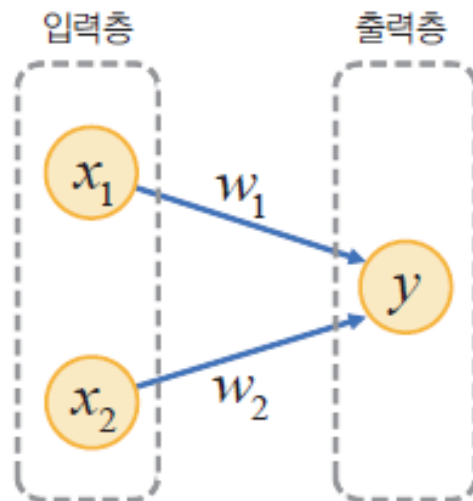
▼ 그림 16-2 기본적인 퍼셉트론



❖ 신경망과 딥러닝 (5/16)

- 그림 16-2에서 원으로 표현된 것을 **노드(node)**라고 함
- 노드 사이에 연결된 선을 **에지(edge)**라고 부름
- 그림 16-2 왼쪽에서 x_1 과 x_2 로 표기된 노드는 입력 노드임
- 오른쪽 y 로 표기된 노드는 출력 노드임

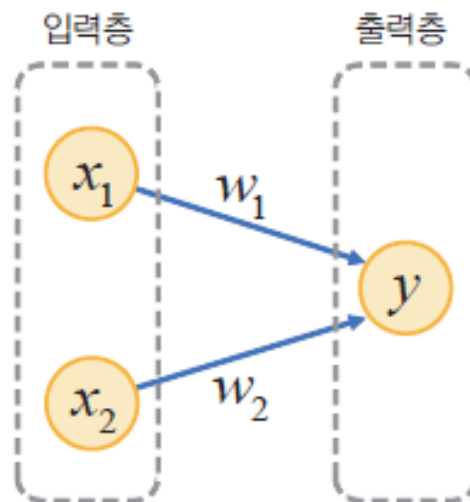
▼ 그림 16-2 기본적인 퍼셉트론



❖ 신경망과 딥러닝 (6/16)

- 입력 노드로 이루어진 계층을 **입력층(input layer)**이라고 함
- 출력 노드로 이루어진 계층을 **출력층(output layer)**이라고 함
- 각각의 에지는 **가중치(weight)**를 가지며, 그림 16-2에서는 두 개의 에지에 각각 w_1 과 w_2 의 가중치가 지정되어 있음

▼ 그림 16-2 기본적인 퍼셉트론



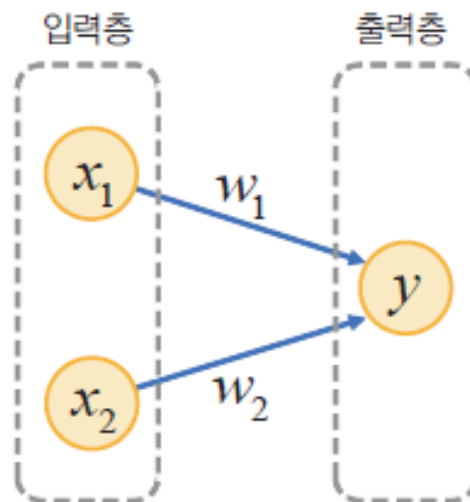
❖ 신경망과 딥러닝 (7/16)

- 이 퍼셉트론의 출력 y 는 다음 수식에 의해 결정됨

$$y = \begin{cases} 1 & w_1x_1 + w_2x_2 + b \geq 0 \text{ 일 때} \\ -1 & w_1x_1 + w_2x_2 + b < 0 \text{ 일 때} \end{cases}$$

- 앞 수식에서 b 는 **편향(bias)**이라고 부르며, y 값 결정에 영향을 줄 수 있는 파라미터임

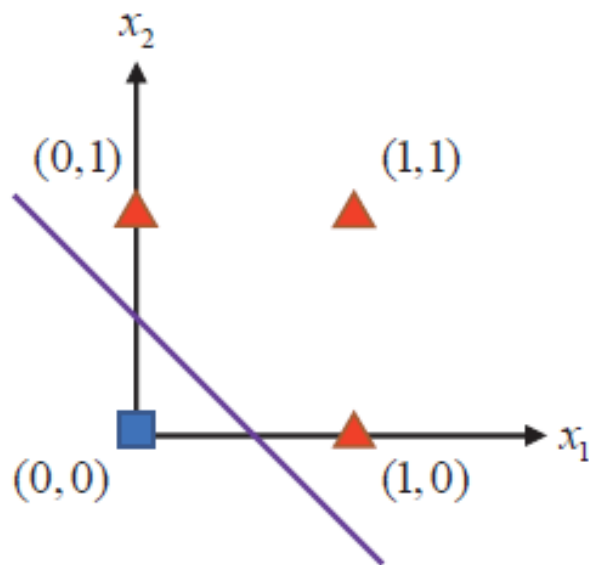
▼ 그림 16-2 기본적인 퍼셉트론



❖ 신경망과 딥러닝 (8/16)

- 그림 16-3은 2차원 평면상에 두 개의 클래스로 나뉜 점들의 분포를 보여 줌
- 빨간색 삼각형 점들과 파란색 사각형 점을 분류하기 위해 퍼셉트론을 사용할 경우, 가중치는 $w_1 = w_2 = 1$ 로 설정하고, 편향은 $b = -0.5$ 로 설정할 수 있음
- 이 경우 출력 y 는 다음과 같이 결정됨

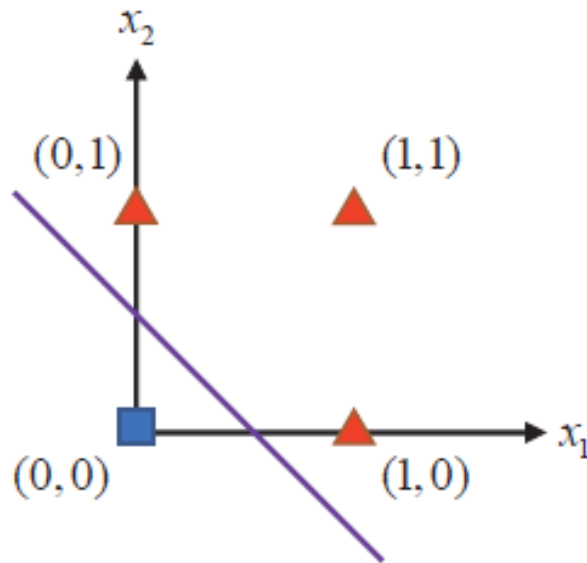
$$y = \begin{cases} 1 & x_1 + x_2 - 0.5 \geq 0 \text{ 일 때} \\ -1 & x_1 + x_2 - 0.5 < 0 \text{ 일 때} \end{cases}$$



◀ 그림 16-3 퍼셉트론에 의한 점들의 분류

❖ 신경망과 딥러닝 (9/16)

- 앞 수식에 빨간색 삼각형 점의 좌표를 입력하면 y 는 1이 됨
- 파란색 사각형 점의 좌표를 입력하면 y 는 -1이 됨
- 그림 16-3에서 두 점들의 분포를 가르는 보라색 직선의 방정식은 $x_1 + x_2 - 0.5 = 0$ 으로 표현됨

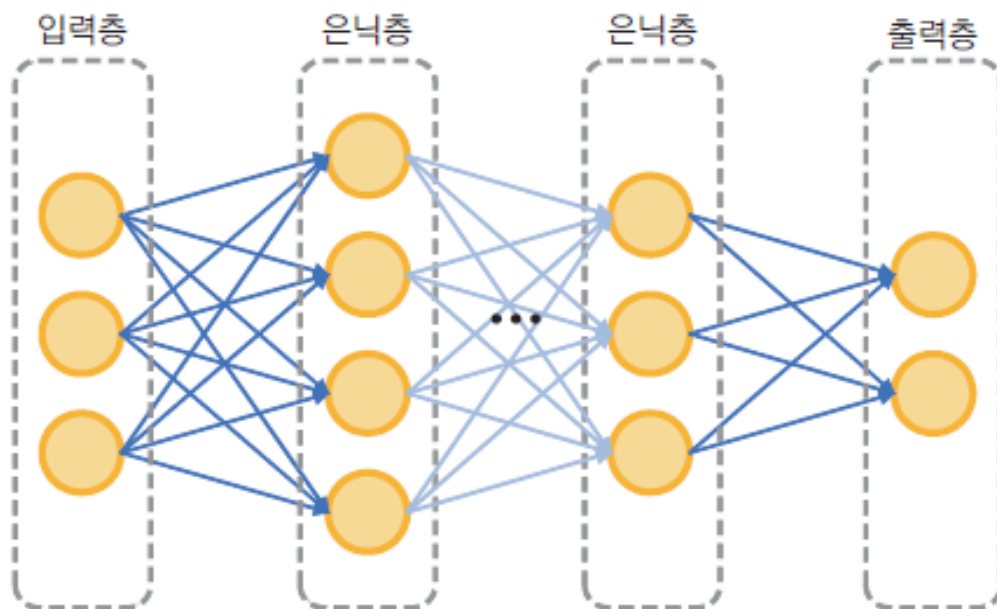


◀ 그림 16-3 퍼셉트론에 의한 점들의 분류

❖ 신경망과 딥러닝 (10/16)

- 기본적인 퍼셉트론은 입력 데이터를 두 개의 클래스로 선형 분류하는 용도로 사용할 수 있음
- 좀 더 복잡한 형태로 분포되어 있는 데이터 집합에 대해서는 노드의 개수를 늘림
- 입력과 출력 사이에 여러 개의 **은닉층(hidden layer)**을 추가하는 형태로 구조를 발전시켜 해결할 수 있음

▼ 그림 16-4 다층 퍼셉트론



❖ 신경망과 딥러닝 (11/16)

- 신경망이 주어진 문제를 제대로 해결하려면 신경망 구조가 문제에 적합해야 함
- 에지에 적절한 가중치가 부여되어야 함
- 에지의 가중치와 편향 값은 **경사 하강법**(gradient descent), **오류 역전파**(error backpropagation) 등의 알고리즘에 의해 자동으로 결정할 수 있음
- **신경망에서 학습이란 결국 학습 데이터셋을 이용하여 적절한 에지 가중치와 편향 값을 구하는 과정이라고 할 수 있음**

❖ 신경망과 딥러닝 (12/16)

- 신경망은 2000년대 초반까지 크게 발전하지 못함
- 그 이유는 은닉층이 많아질수록 학습 시간이 너무 오래 걸림
- 학습도 제대로 되지 않는 문제가 해결되지 않았기 때문임
- 그러다가 2010년 초반부터 신경망은 심층 신경망 또는 딥러닝이라는 새로운 이름으로 크게 발전하기 시작함

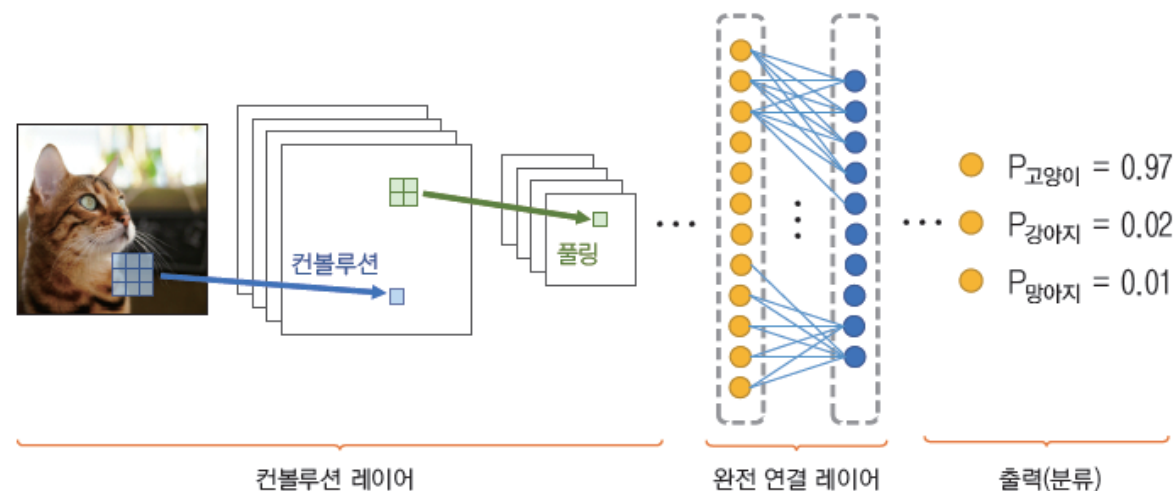
❖ 신경망과 딥러닝 (13/16)

- 2000년대에 딥러닝이 크게 발전한 이유는 크게 세 가지를 들 수 있음
- **첫 번째**는 딥러닝 알고리즘이 개선되면서 은닉층이 많아져도 학습이 제대로 이루어지게 되었다는 점임
- **두 번째**는 하드웨어의 발전, 특히 GPU(Graphics Processing Unit) 성능 향상과 GPU를 활용한 학습 방법으로 인해 딥러닝 학습 시간이 크게 단축되었기 때문임
- **세 번째** 이유는 인터넷의 발전에 따른 빅데이터 활용이 용이해졌다는 점임
- 특히 컴퓨터 비전 분야에서는 Pascal VOC1, ImageNet2과 같이 잘 다듬어진 영상 데이터를 활용할 수 있었다는 점이 큰 장점으로 작용함
- 대용량 영상 데이터셋을 이용한 영상 인식 대회 등을 통해 알고리즘 경쟁과 공유가 활발하게 이루어졌다는 점도 딥러닝 발전에 긍정적인 영향을 끼침

❖ 신경망과 딥러닝 (14/16)

- 다양한 딥러닝 구조 중에서 특히 영상을 입력으로 사용하는 영상 인식, 객체 검출 등의 분야에서는 **합성곱 신경망(CNN, Convolutional Neural Network)** 구조가 널리 사용되고 있음
- CNN 구조는 보통 2차원 영상에서 특징을 추출하는 **컨볼루션 레이어(convolution layer)**와 추출된 특징을 분류하는 **완전 연결 레이어(FC layer, Fully Connected layer)**로 구성됨

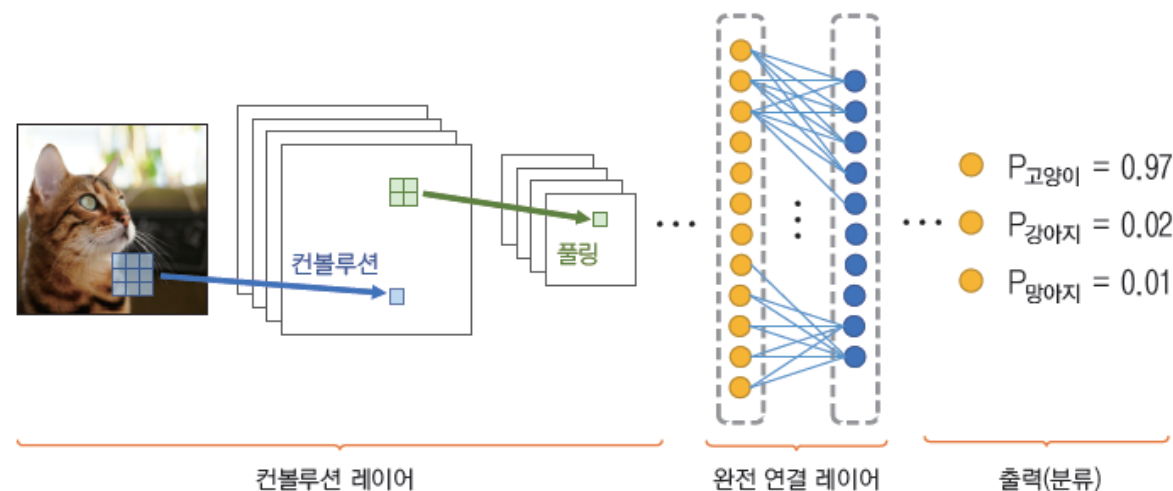
▼ 그림 16-5 일반적인 CNN 구조



❖ 신경망과 딥러닝 (15/16)

- CNN 구조에서 컨볼루션은 영상의 지역적인 특징을 추출하는 역할을 담당함
- **풀링(pooling)**은 비선형 다운샘플링(down sampling)을 수행하여 데이터양을 줄이고 일부 특징을 강조하는 역할을 함
- 완전 연결 레이어는 고전적인 다층 퍼셉트론과 비슷한 구조로서, 앞에서 추출된 특징을 이용하여 출력값을 결정함
- 보통 컨볼루션 레이어를 여러 개 연결하고, 맨 뒤에 완전 연결 레이어를 연결하는 형태로 모델을 구성함

▼ 그림 16-5 일반적인 CNN 구조



❖ 신경망과 딥러닝 (16/16)

- 컴퓨터 비전 분야에서 사용되는 딥러닝 알고리즘은 대부분 CNN 구조를 기본으로 사용하면서 인식의 정확도를 높이거나 연산 속도를 빠르게 하는 등의 목적에 맞게 변형된 형태임
- 컨볼루션 단계에서 사용하는 커널을 3×3 , 5×5 등 다양한 크기로 구성하기도 함
- 레이어 사이의 연결 방식도 새롭게 설계하여 효과적인 성능을 얻기도 함

❖ OpenCV DNN 모듈 (1/15)

- OpenCV dnn 모듈은 이미 만들어진 네트워크에서 순방향 실행을 위한 용도로 설계됨
- **딥러닝 학습은** 기존의 유명한 **텐서플로(TensorFlow)** 등의 **다른 딥러닝 프레임워크에서 진행함**
- **학습된 모델을 불러와서 실행할 때에는 dnn 모듈을 사용하는 방식임**
- 많은 딥러닝 프레임워크가 파이썬 언어를 사용하고 있지만, OpenCV dnn 모듈은 C/C++ 환경에서도 동작할 수 있기 때문에 프로그램 이식성이 높다는 장점이 있음
- dnn 모듈은 OpenCV 3.1에 서는 추가 모듈 형태로 지원되었고, OpenCV 3.3 버전부터 기본 모듈에 포함됨

❖ OpenCV DNN 모듈 (2/15)

- OpenCV dnn 모듈에서 지원하는 딥러닝 프레임워크는 다음과 같음

- 카페(Caffe) <http://caffe.berkeleyvision.org/>
- 텐서플로(TensorFlow) <https://www.tensorflow.org/>
- 토치(Torch) <http://torch.ch/>
- 다크넷(Darknet) <https://pjreddie.com/darknet/>
- DLDT <https://github.com/opencv/dldt>
- ONNX <https://onnx.ai/>

- OpenCV dnn 모듈은 카페, 텐서플로, 토치 등의 프레임워크에서 학습된 모델과 ONNX(Open Neural Network Exchange) 파일 형식으로 저장된 모델을 불러와 실행할 수 있음

❖ OpenCV DNN 모듈 (3/15)

- OpenCV dnn 모듈은 이미 널리 사용되고 있는 딥러닝 네트워크 구성을 지원함
- 최근에 새롭게 개발되고 있는 딥러닝 네트워크도 지속적으로 추가 지원하고 있음
- **영상 인식**과 관련된 AlexNet, GoogLeNet, VGG, ResNet, SqueezeNet, DenseNet, ShuffleNet, MobileNet, Darknet 등의 네트워크가 OpenCV에서 동작됨이 확인됨
- **객체 검출**과 관련해서는 VGG-SSD, MobileNet-SSD, Faster-RCNN, R-FCN, OpenCV face detector, Mask-RCNN, EAST, YOLOv2, tiny YOLO, YOLOv3 등의 모델을 사용할 수 있음
- 이외에도 **사람의 포즈를 인식**하는 OpenPose, **흑백 영상에 자동으로 색상을 입히는** Colorization, **사람 얼굴 인식**을 위한 OpenFace 등의 모델도 OpenCV dnn 모듈에서 사용할 수 있음

❖ OpenCV DNN 모듈 (4/15)

- dnn 모듈에서 딥러닝 네트워크는 cv2.dnn.Net 클래스를 이용하여 표현함
- Net 클래스는 다양한 레이어로 구성된 네트워크 구조를 표현함
- Net 클래스에는 empty(), setInput(), forward() 메서드 등이 존재함
- empty() 메서드는 네트워크가 비어 있으면 True를 반환함
- setInput() 메서드는 네트워크를 순방향으로 실행함
- forward() 메서드는 네트워크 입력을 설정함

❖ OpenCV DNN 모듈 (5/15)

- Net 클래스 객체는 보통 사용자가 직접 생성하지 않으며 readNet() 등의 함수를 이용하여 생성함
- readNet() 함수는 미리 학습된 딥러닝 모델과 네트워크 구성 파일을 이용하여 Net 객체를 생성함

```
net = cv2.dnn.readNet(model, config)
```

model	학습된 가중치를 저장하고 있는 이진 파일 이름
config	네트워크 구성을 저장하고 있는 텍스트 파일 이름
framework	명시적인 딥러닝 프레임워크 이름
net	Net 객체

❖ OpenCV DNN 모듈 (6/15)

- 만약 model 파일에 네트워크 학습 가중치와 네트워크 구조가 함께 저장되어 있다면 config 인자를 생략할 수 있음
- framework 인자에는 모델 파일 생성 시 사용된 딥러닝 프레임워크 이름을 지정함
- 만약 model 또는 config 파일 이름 확장자를 통해 프레임워크 구분이 가능한 경우에는 framework 인자를 생략할 수 있음
- model과 config 인자에 지정할 수 있는 파일 이름 확장자와 framework에 지정 가능한 프레임워크 이름을 표 16-1에 정리함

❖ OpenCV DNN 모듈 (7/15)

▼ 표 16-1 딥러닝 프레임워크에 따른 model 및 config 파일 확장자

딥러닝 프레임워크	model 파일 확장자	model 파일 확장자	framework 문자열
카페	*.caffemodel	*.prototxt	"caffe"
텐서플로	*.pb	*.pbtxt	"tensorflow"
토치	*.t7 또는 *.net		"torch"
다크넷	*.weights	*.cfg	"darknet"
DLDT	*.bin	*.xml	"dldt"
ONNX	*.onnx		"onnx"

❖ OpenCV DNN 모듈 (8/15)

- `readNet()` 함수는 전달된 `framework` 문자열, 또는 `model`과 `config` 파일 이름 확장자를 분석하여 내부에서 해당 프레임워크에 맞는 `readNetFromXXX()` 형태의 함수를 다시 호출함
- 예를 들어 `model` 파일 확장자가 `*.caffemodel`이면 `readNetFromCaffe()` 함수를 호출함
- `model` 파일 확장자가 `*.pb`이면 `readNetFromTensorflow()` 함수를 다시 호출하여 `Net` 객체를 생성함
- `readNetFromTorch()`, `readNetFromDarknet()`, `readNetFromModelOptimizer()`, `readNetFromONNX()` 함수가 OpenCV에서 제공되고 있음
- `readNetFromXXX()` 형태의 함수를 사용자가 직접 호출하여 사용할 수도 있지만, OpenCV 4.0.0 버전부터는 `readNet()` 대표 함수를 사용하는 것이 좋음

❖ OpenCV DNN 모듈 (9/15)

- readNet() 함수를 이용하여 Net 객체를 생성한 후에는 empty() 메서드를 사용하여 Net 객체가 정상적으로 생성되었는지 확인하는 것이 좋음
- 만약 empty() 메서드가 True를 반환하면 예외 처리 코드를 추가함

`net.empty()`

반환값

네트워크가 비어 있으면 True를 반환합니다.

❖ OpenCV DNN 모듈 (10/15)

- 일단 Net 객체가 정상적으로 생성되었다면 이제 생성된 네트워크에 새로운 데이터를 입력하여 그 결과를 확인할 수 있음
- 이때 Net 객체로 표현되는 네트워크 입력으로 2차원 영상을 그대로 입력하는 것이 아니라 **블롭(blob)** 형식으로 변경해야 함

❖ OpenCV DNN 모듈 (11/15)

- 블롭이란 **영상 등의 데이터를 포함할 수 있는 다차원 데이터 표현 방식임**
- OpenCV에서 블롭은 `numpy.ndarray` 타입의 4차원 행렬로 표현됨
- 이때 각 차원은 NCHW 정보를 표현함
- 여기서 N은 영상 개수, C는 채널 개수, H와 W는 각각 영상의 세로와 가로 크기를 의미함

❖ OpenCV DNN 모듈 (12/15)

- OpenCV에서는 `blobFromImage()` 함수를 이용하여 입력 영상으로부터 블롭을 생성함

```
cv2.dnn.blobFromImage(frame, scalefactor, size, mean, swapRB=False)
```

frame	입력 영상. 1 또는 3 또는 4채널
scalefactor	입력 영상 픽셀 값에 곱할 값
size	출력 영상의 크기. 예를 들어 (300, 300) 형식으로 작성합니다.
mean	입력 영상 각 채널에서 빨 평균값. 만약 frame이 BGR 채널 순서이고 swapRB가 True이면 (R 평균, G 평균, B 평균) 순서로 지정합니다.
swapRB	첫 번째 채널과 세 번째 채널을 서로 바꿀 건인지를 결정하는 플래그. 이 값이 True이면 컬러 입력 영상의 채널 순서를 BGR에서 RGB로 변경합니다.
반환값	영상으로부터 구한 블롭. 4차원 numpy.ndarray 행렬입니다.

❖ OpenCV DNN 모듈 (13/15)

- `blobFromImage()` 함수로 생성한 블롭 객체는 Net 클래스 객체의 `setInput()` 메서드를 이용하여 네트워크 입력으로 설정됨

`net.setInput(blob, name, scalefactor, mean)`

<code>blob</code>	블롭 객체
<code>name</code>	입력 레이어 이름
<code>scalefactor</code>	추가적으로 픽셀 값에 곱할 값
<code>mean</code>	추가적으로 픽셀 값에서 뺄 평균값

❖ OpenCV DNN 모듈 (14/15)

- Net 클래스 객체의 setInput() 메서드 인자에도 blobFromImage() 함수에 있는 scalefactor와 mean 인자가 있어서, 추가적인 픽셀 값을 조정할 수 있음
- 결국 네트워크에 입력되는 블롭은 다음과 같은 형태로 설정됨

$$\text{input}(n, c, h, w) = \text{scalefactor} \times (\text{blob}(n, c, h, w) - \text{mean}_c)$$

❖ OpenCV DNN 모듈 (15/15)

- 네트워크 입력을 설정한 후에는 네트워크를 순방향으로 실행하여 결과를 예측할 수 있음
- 네트워크를 실행할 때에는 Net 클래스 객체의 forward() 메서드를 사용함
- forward() 메서드는 순방향으로 네트워크를 실행한다는 의미이며, 이를 **추론(inference)**이라고도 함

`net.forward(outputName)`

outputName	출력 레이어 이름
반환값	지정한 레이어의 출력 블록

16.2 딥러닝 학습과 OpenCV 실행

16.1 딥러닝과 OpenCV DNN 모듈

16.3 OpenCV와 딥러닝 활용

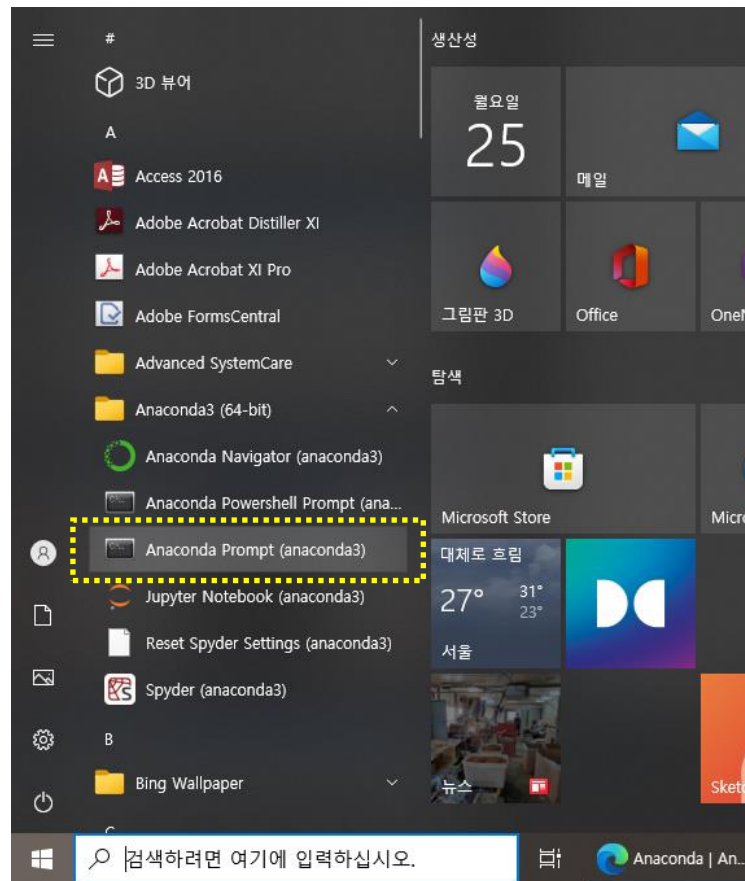
❖ 텐서플로 설치하기 (1/4)

- 많은 딥러닝 프레임워크가 파이썬(Python) 언어를 주력으로 사용함
- 딥러닝 학습을 위한 프레임워크로는 **텐서플로(TensorFlow)**를 사용할 것임
- 파이썬 프로그램에서 제공하는 pip 명령을 이용하여 텐서플로를 설치할 수 있음
- pip는 파이썬에서 사용하는 패키지 설치 및 관리 프로그램 이름임

❖ 텐서플로 설치하기 (2/4)

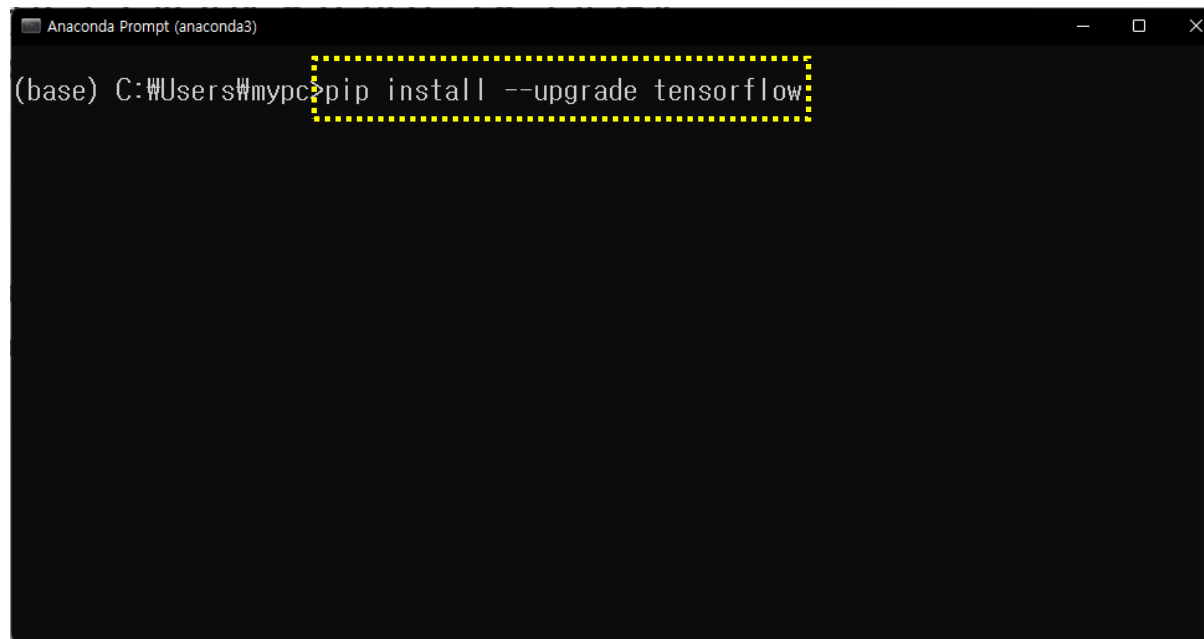
- 작업 표시줄의 윈도우 키를 눌러서 Anaconda3 폴더의 [Anaconda Prompt (anaconda3)] 아이콘을 클릭

윈도우 키



❖ 텐서플로 설치하기 (3/4)

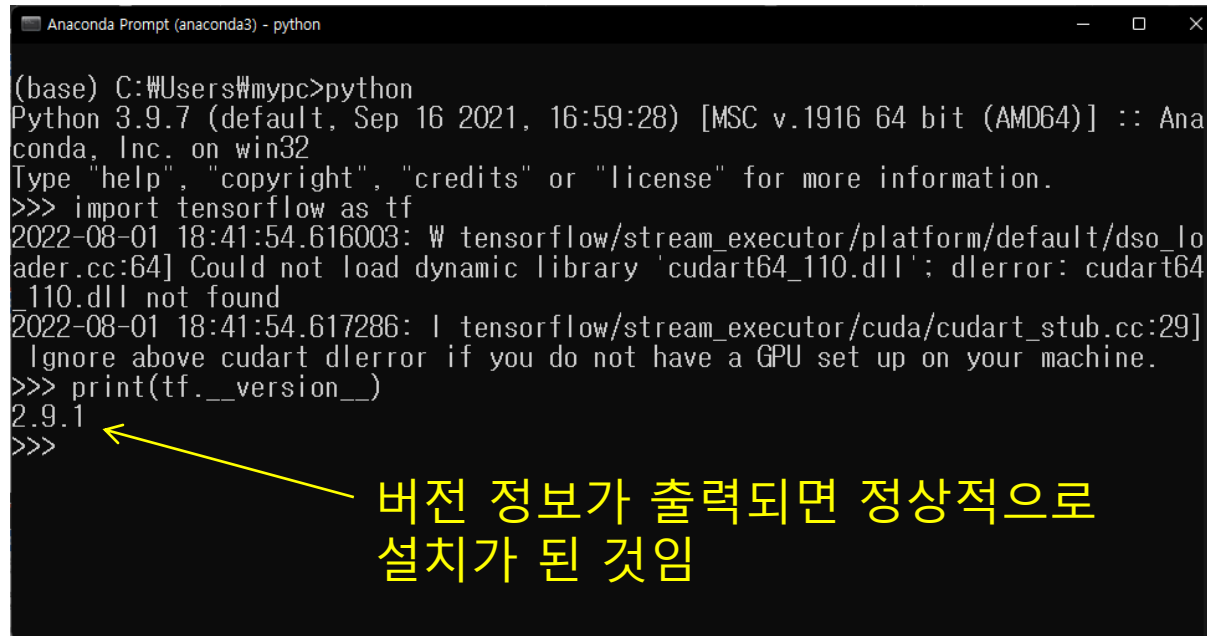
- 아래 명령어를 입력하고 엔터키 누르기
 - ◆ `pip install --upgrade tensorflow`



```
Anaconda Prompt (anaconda3)
(base) C:\Users\mypc>pip install --upgrade tensorflow
```

❖ 텐서플로 설치하기 (4/4)

- 아래 명령어를 순서대로 입력하고 엔터키 누르기
 - ◆ python
 - ◆ import tensorflow as tf
 - ◆ print(tf.__version__)



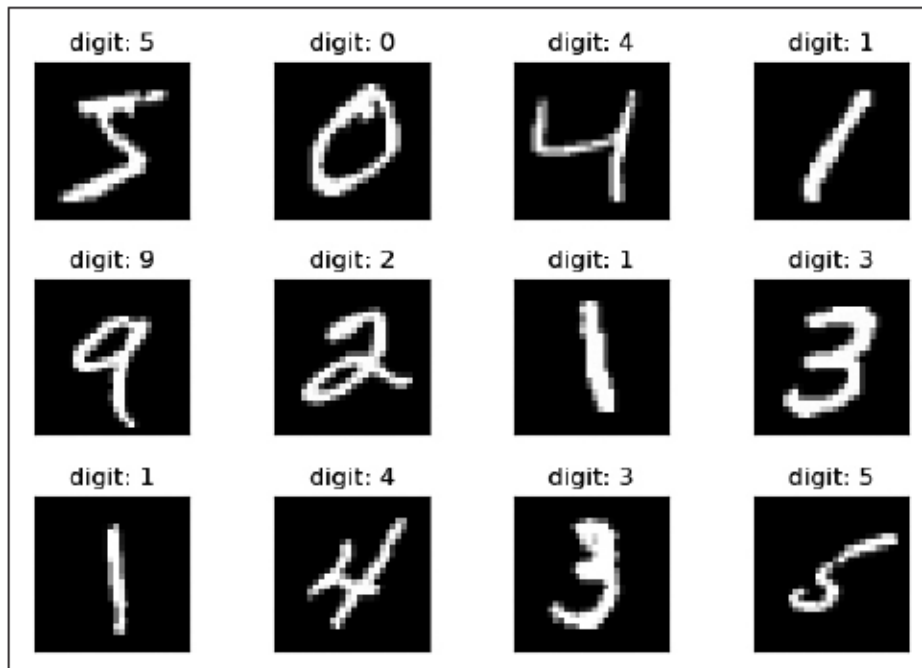
```
Anaconda Prompt (anaconda3) - python

(base) C:\Users\mypc>python
Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
2022-08-01 18:41:54.616003: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlderror: cudart64_110.dll not found
2022-08-01 18:41:54.617286: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
>>> print(tf.__version__)
2.9.1
>>>
```

버전 정보가 출력되면 정상적으로 설치가 된 것임

❖ 텐서플로로 필기체 숫자 인식 학습하기 (1/2)

- 딥러닝 분야에서는 필기체 숫자 인식 학습을 위해 MNIST 데이터셋을 주로 사용함
- MNIST는 뉴욕 대학교 안 르쿤(Yann LeCun) 교수가 우편 번호 등의 필기체 숫자 인식을 위해 사용했던 데이터셋으로, 6만 개의 학습용 영상과 1만 개의 시험용 영상으로 구성되어 있음
- 각 숫자 영상은 28×28 크기로 구성되어 있고, 픽셀 값은 0에서 255 사이의 정수 값으로 되어 있음
- 그림 16-7은 MNIST 데이터셋 일부를 그레이스케일 영상 형식으로 변환하여 나타낸 결과임



◀ 그림 16-7 MNIST 숫자 영상의 예

❖ 텐서플로로 필기체 숫자 인식 학습하기 (2/2)

- 전체 구현 과정은 크게 2단계로 구성되어 있음

- **Step 1)** mnist_cnn.py 소스 코드 구현

- ◆ MNIST 데이터셋 다운로드

- ◆ 필기체 숫자 인식을 위한 CNN 모델을 구축

- ◆ MNIST 데이터셋으로 CNN 모델을 학습시킴

학습 시간이 오래 걸려서, 구글 코랩을 이용

- ◆ 학습 완료된 CNN 모델을 저장 ('best-model.h5'라는 이름으로 모델을 저장)

- **Step 2)** dnnmnist.py 소스 코드 구현

- ◆ best-model.h5를 로드(load)하여 CNN 모델을 불러들임

- ◆ 사용자가 마우스로 그리는 숫자를 CNN 모델에 입력

- ◆ CNN 모델의 필기체 숫자 인식 결과를 실행 결과 창에 출력

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **mnist_cnn.py** 소스 코드 구현 (1/7)

- MNIST 데이터셋은 많은 딥러닝 프레임워크에서도 예제로 널리 사용하고 있음
- 텐서플로도 MNIST 데이터셋을 쉽게 불러올 수 있는 인터페이스를 제공함
- 텐서플로에서 MNIST 데이터셋을 이용하여 필기체 숫자 인식 학습을 수행하는 파이썬 소스 코드를 코드 16-2에 나타냄

코드 16-2 텐서플로를 이용한 필기체 숫자 인식 학습 (mnist_cnn.py)

```
1 from tensorflow import keras
2 import tensorflow as tf
3 from sklearn.model_selection import train_test_split
4 import matplotlib.pyplot as plt
5
6
7 (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
8 x_train = x_train / 255.0
9 x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, random_state=42)
10
```

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **minst_cnn.py** 소스 코드 구현 (2/7)

코드 16-2 텐서플로를 이용한 필기체 숫자 인식 학습 (mnist_cnn.py)

```
11 model = keras.Sequential()
12 model.add(keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='same',
13 input_shape=(28, 28, 1)))
14 model.add(keras.layers.MaxPooling2D(2))
15 model.add(keras.layers.Conv2D(64, kernel_size=3, activation='relu', padding='same'))
16 model.add(keras.layers.MaxPooling2D(2))
17 model.add(keras.layers.Flatten())
18 model.add(keras.layers.Dense(256, activation='relu'))
19 model.add(keras.layers.Dropout(0.3))
20 model.add(keras.layers.Dense(10, activation='softmax'))
21
22 model.summary()
23 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
24
25 checkpoint_cb = keras.callbacks.ModelCheckpoint('best-model.h5', save_best_only=True)
26 early_stopping_cb = keras.callbacks.EarlyStopping(patience=2, restore_best_weights=True)
27
```

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **mnist_cnn.py** 소스 코드 구현 (3/7)

코드 16-2 텐서플로를 이용한 필기체 숫자 인식 학습 (mnist_cnn.py)

```
28 history = model.fit(x_train, y_train, epochs=20, validation_data=(x_val, y_val),
29 callbacks=[checkpoint_cb, early_stopping_cb])
30
31 plt.plot(history.history['loss'], color='blue', label='train')
32 plt.plot(history.history['val_loss'], color='red', label='validation')
33 plt.xlabel('epoch')
34 plt.ylabel('loss')
35 plt.grid(True)
36 plt.legend()
37 plt.show()
38
39 plt.plot(history.history['accuracy'], color='blue', label='train')
40 plt.plot(history.history['val_accuracy'], color='red', label='validation')
41 plt.xlabel('epoch')
42 plt.ylabel('accuracy')
43 plt.grid(True)
44 plt.legend()
45 plt.show()
```

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **mnist_cnn.py** 소스 코드 구현 (4/7)

● mnist_cnn.py 소스 코드 설명

- 1~4행 프로그램 동작에 필요한 파이썬 라이브러리를 포함시킵니다.
- 7행 MNIST 데이터셋을 인터넷에서 내려받아 학습용 데이터와 시험용 데이터를 저장합니다.
- 8행 픽셀 값은 0에서 1 사이의 실수 값으로 정규화시킵니다.
- 9행 `train_test_split()` 함수를 이용해 학습용 데이터의 일부를 검증용 데이터로 분할합니다.
- 11~20행 두 개의 컨볼루션 레이어와 하나의 완전 연결 레이어로 이루어진 CNN 네트워크를 구성합니다.
- 22행 구성된 CNN 모델의 각 레이어를 요약해서 보여줍니다.
- 25행 모델 학습 과정 중 최상의 검증 점수를 낸 모델을 저장하게끔 콜백(callback)을 설정합니다.
- 26행 모델이 과대적합되지 않게, 2번 연속 검증 점수가 향상되지 않으면 학습을 종료시키도록 콜백을 설정합니다.

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **mnist_cnn.py** 소스 코드 구현 (5/7)

- mnist_cnn.py 소스 코드 설명

- 28~29행 CNN 모델을 학습시킵니다.
매 Epoch 마다 모델이 달성한 손실(loss)와 정확도(accuracy) 정보를 history에 저장합니다.
- 31~37행 Epoch에 따른 학습용 데이터와 검증용 데이터에 대한 CNN 모델의 손실 정보를 꺾은선 그래프로 확인합니다.
- 39~45행 Epoch에 따른 학습용 데이터와 검증용 데이터에 대한 CNN 모델의 정확도 정보를 꺾은선 그래프로 확인합니다.

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **minst_cnn.py** 소스 코드 구현 (6/7)

- 여기서는 단순히 이 네트워크에서 입력 데이터로 MNIST 데이터셋을 사용하였기 때문에 각각의 필기체 숫자가 28×28 크기 행렬로 구성되어 있음
- 각 행렬 원소 값은 0에서 255 사이의 정수 값으로 되어 있음 (정규화시켜야 함)
- 출력은 열 개의 노드로 구성되고, 각 노드의 값은 0부터 9까지의 숫자 값에 대한 확률임
- 학습 완료된 CNN 모델의 구성 정보는 best-model.h5 파일에 저장됨

❖ 텐서플로로 필기체 숫자 인식 학습하기: ① **mnist_cnn.py** 소스 코드 구현 (7/7)

- 코드 16-2의 mnist_cnn.py 소스 코드를 실행하게 되면서 필기체 숫자 인식을 위한 학습을 수행함
- 이 작업은 꽤 오랜 시간이 소요되며, 컴퓨터 사양에 따라 수십 분의 시간이 소요될 수 있음
- **그래픽 처리 장비(GPU, graphics processing unit)**를 활용하면 이 시간을 단축시킬 수 있음
- **구글 코랩(google colaboratory)**에서 제공하는 GPU 자원을 활용하면
고가의 GPU가 실습 컴퓨터에 탑재되어 있지 않더라도 모델을 빠르게 학습을 시킬 수 있음

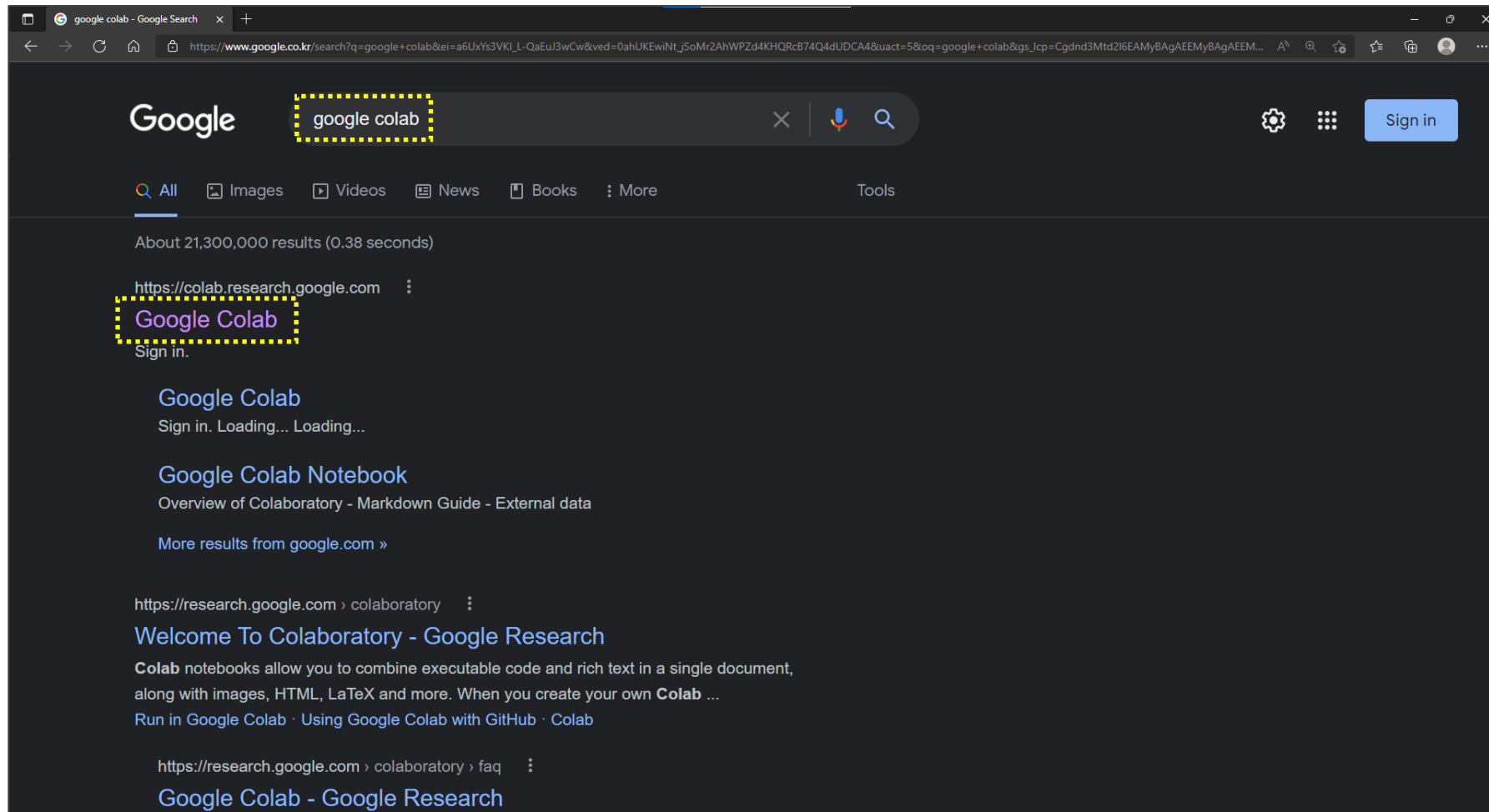
구글 코랩을 이용하기 위해서는 구글 계정 로그인이 필요함

❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (1/13)

- ① Google Colab 사이트 (<https://colab.research.google.com>)에 접속합니다.
- ② Google 계정으로 로그인합니다. (계정이 없다면, 만든 다음에 로그인합니다.)
- ③ 파이썬 프로그래밍을 합니다.
- ④ 프로그래밍을 마치면 프로그램 파일을 구글 드라이브에 저장합니다.

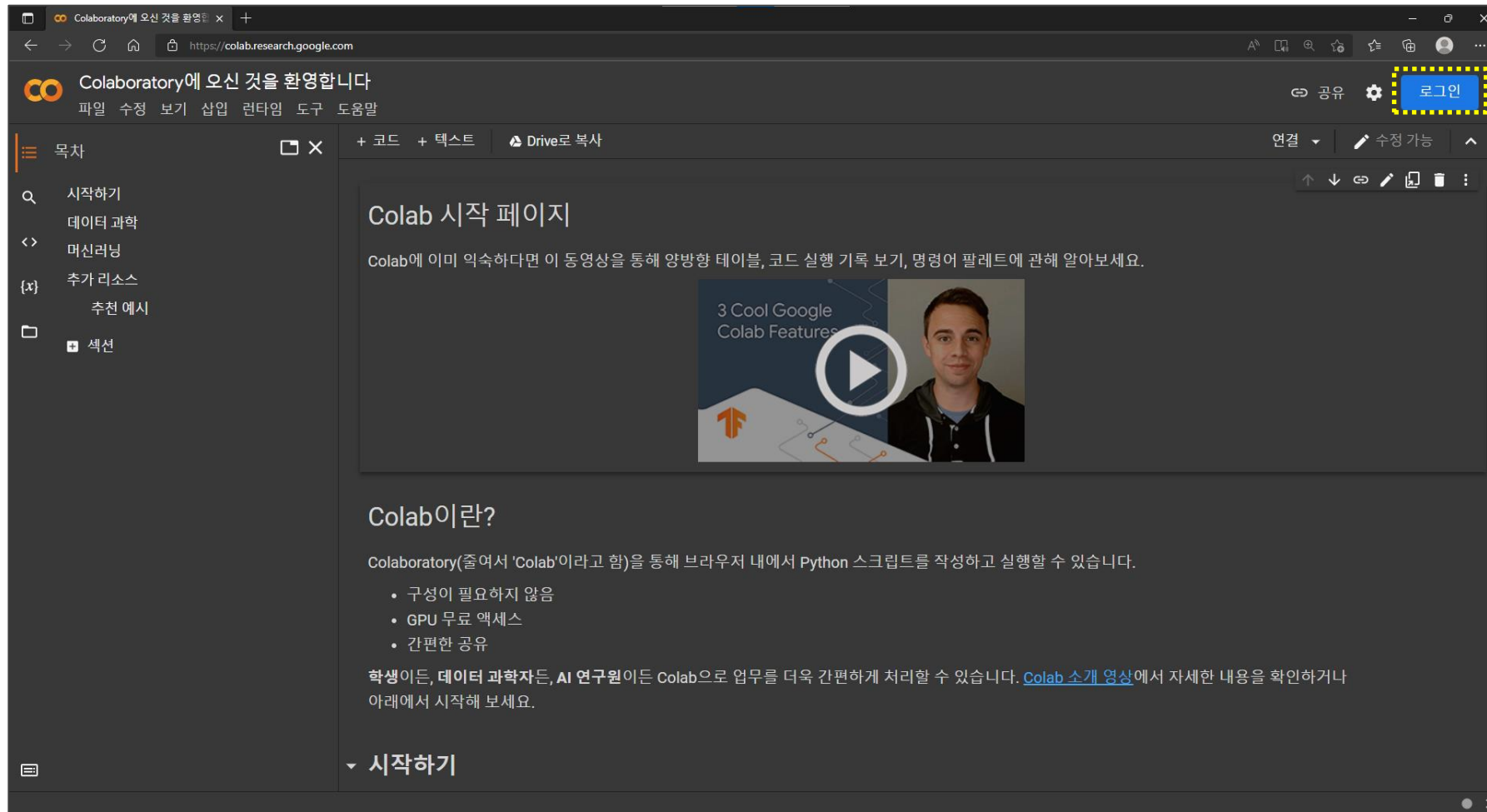
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (2/13)

- 웹 브라우저(Web Browser)에서 'Google Colab' 또는 '구글 코랩'을 검색하고, 클릭합니다.



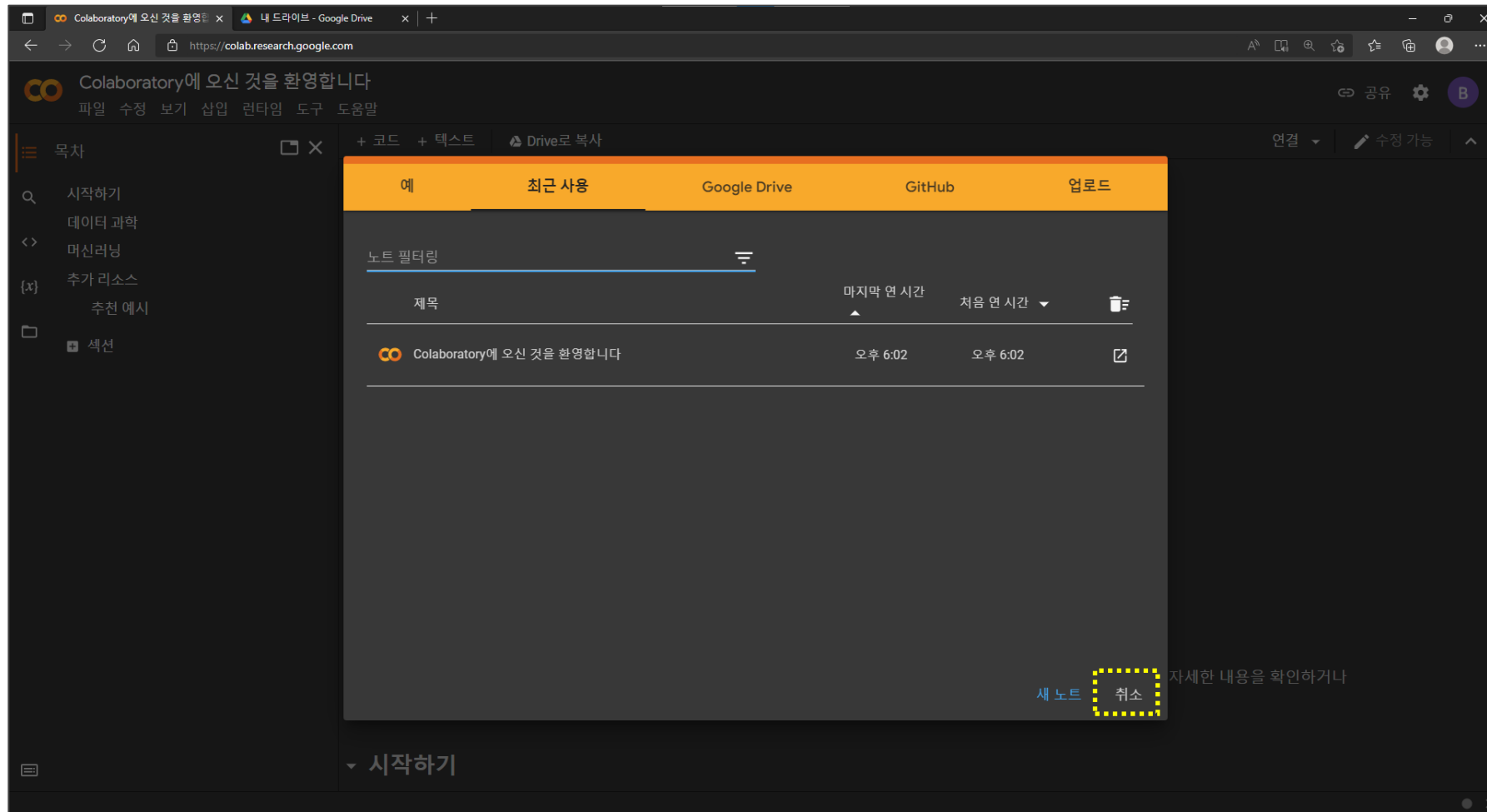
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (3/13)

- 본인의 Google 계정으로 로그인합니다.



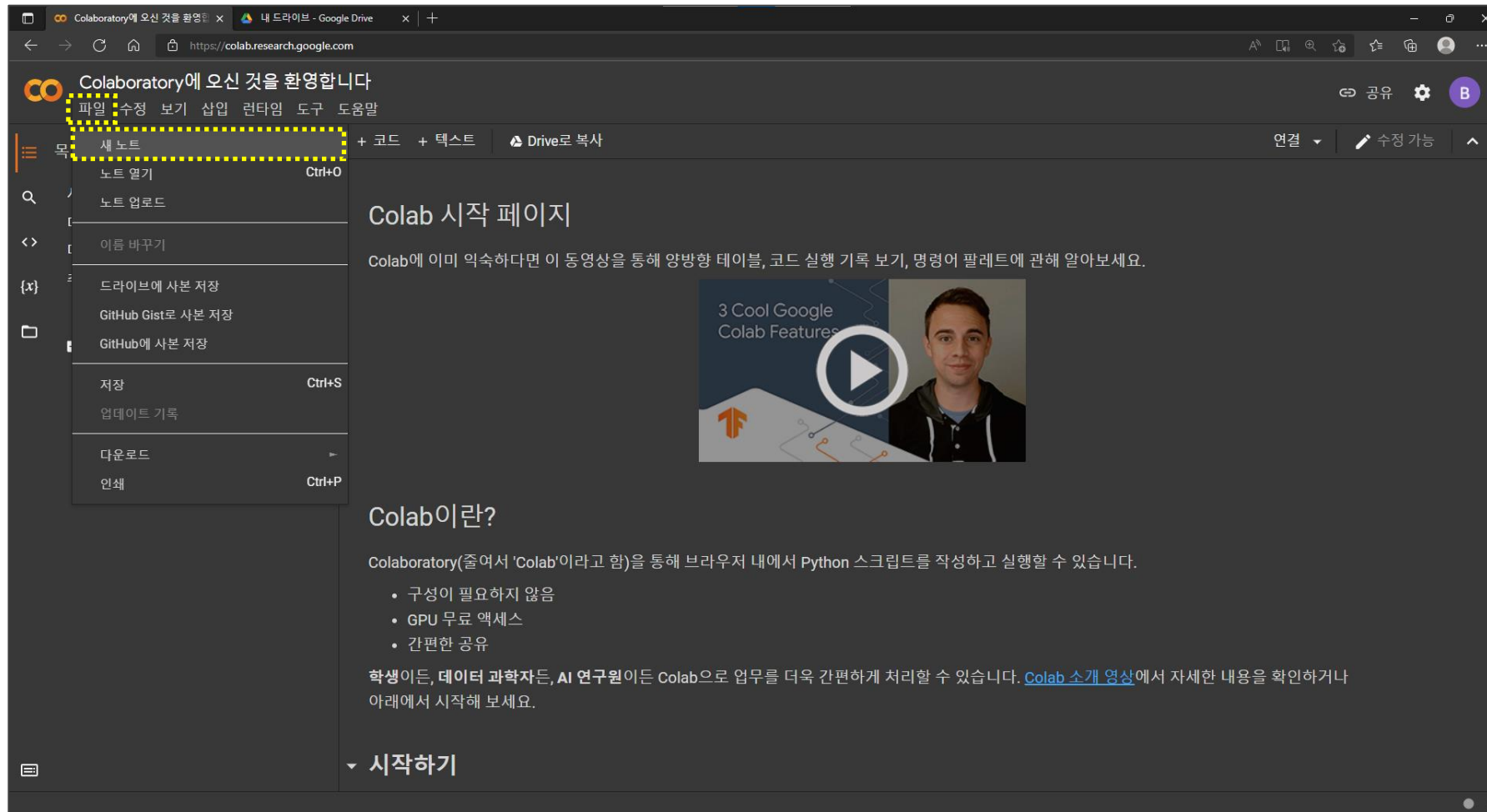
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (4/13)

- 취소 버튼을 눌러 팝업 창을 닫아 줍니다.



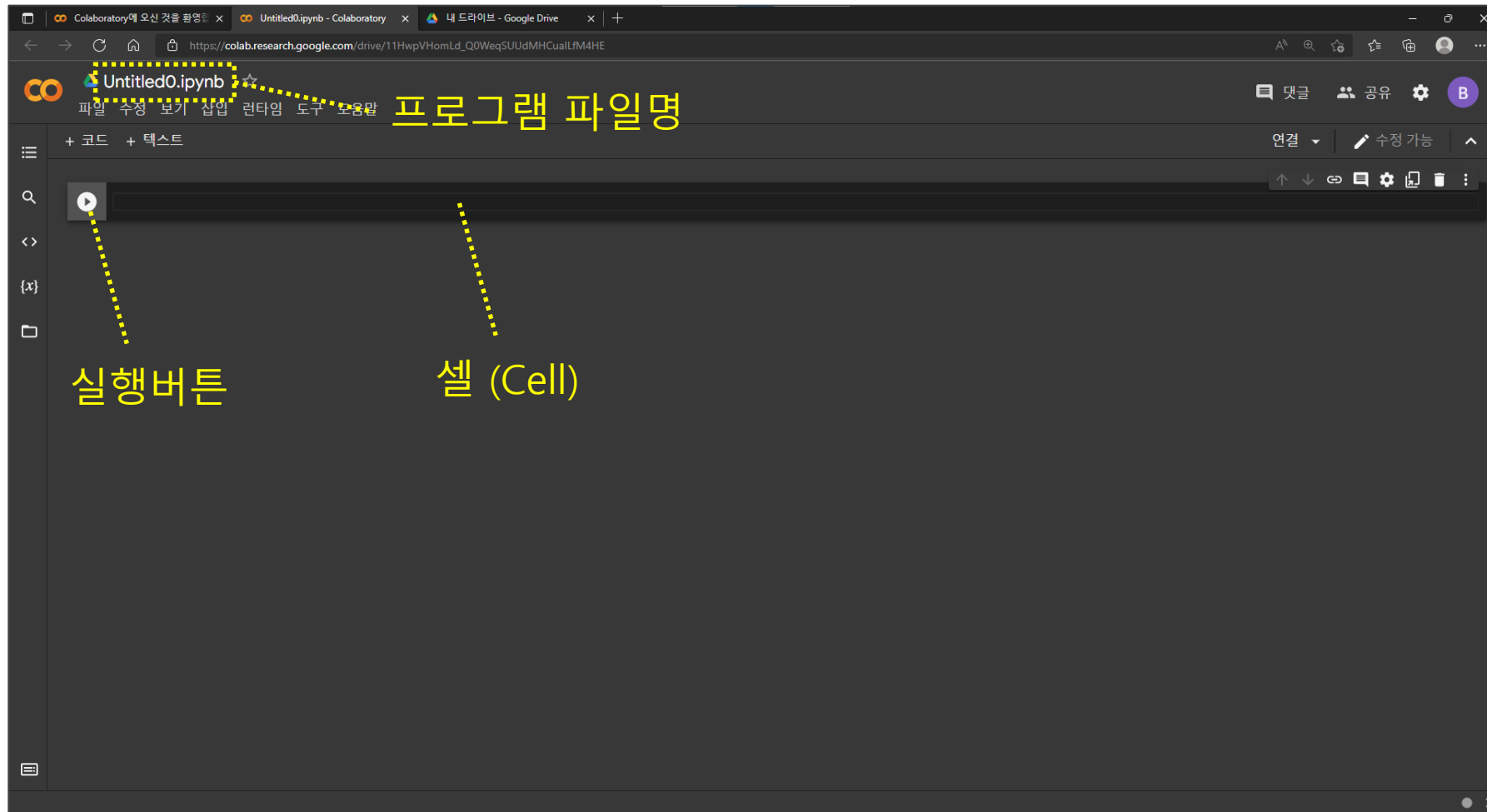
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (5/13)

- [파일] → [새 노트]를 클릭합니다.



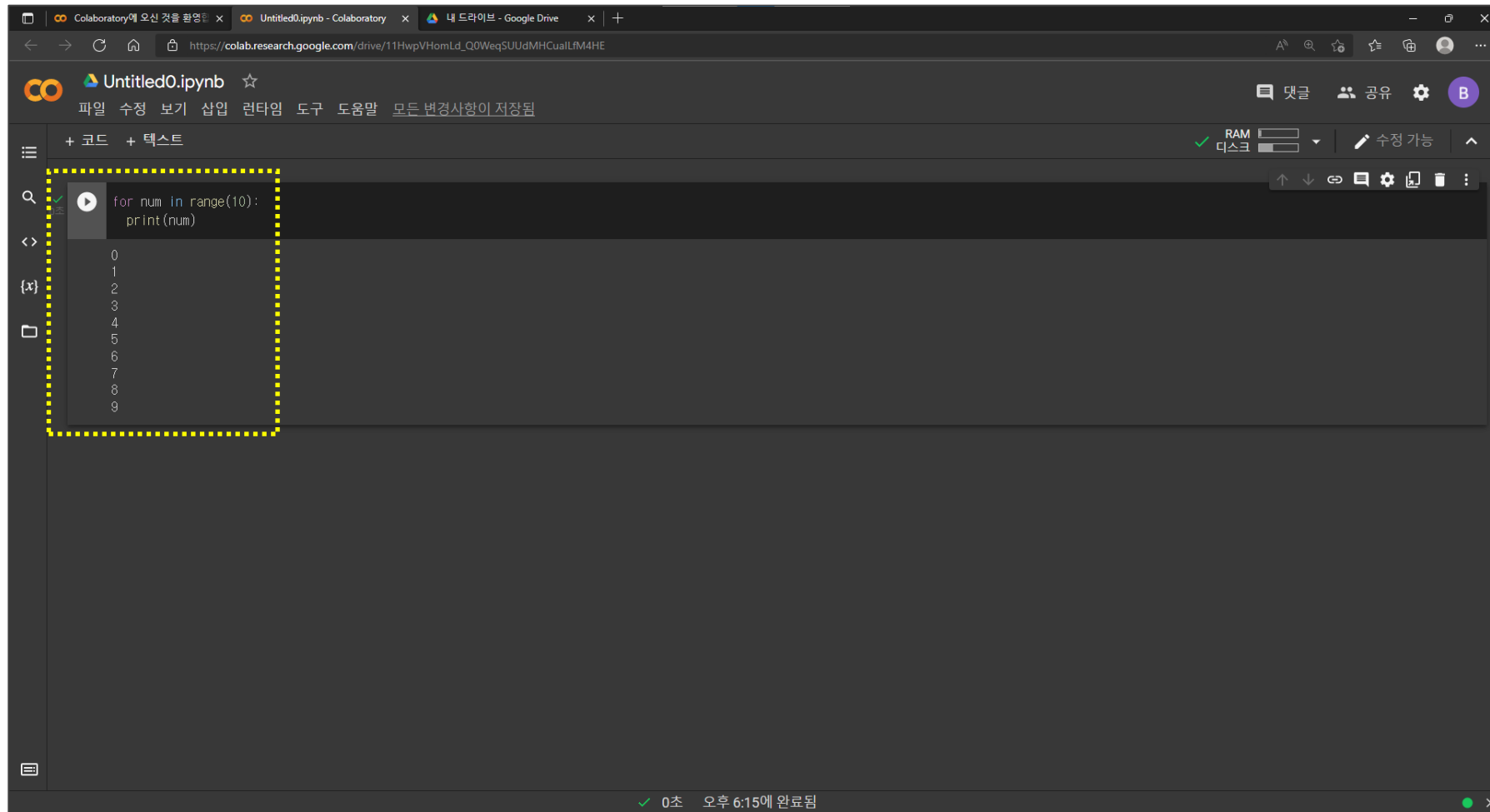
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (6/13)

- 생성된 주피터 노트북 파일 (.ipynb)을 확인할 수 있습니다. (ipynb, IPython Notebook)



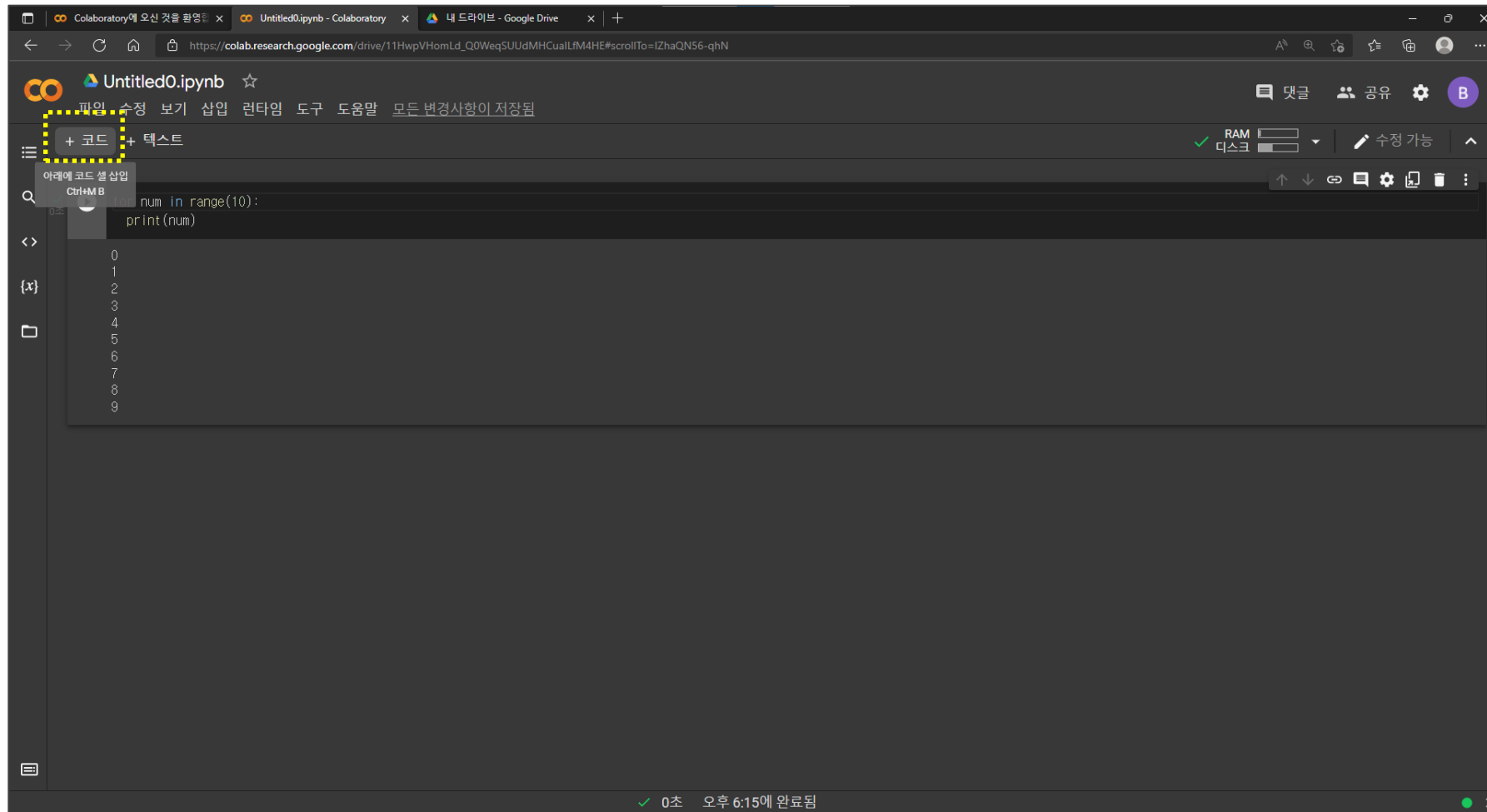
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (7/13)

- 파이썬 코드를 입력하고 실행시켜 결과를 확인해 봅니다.



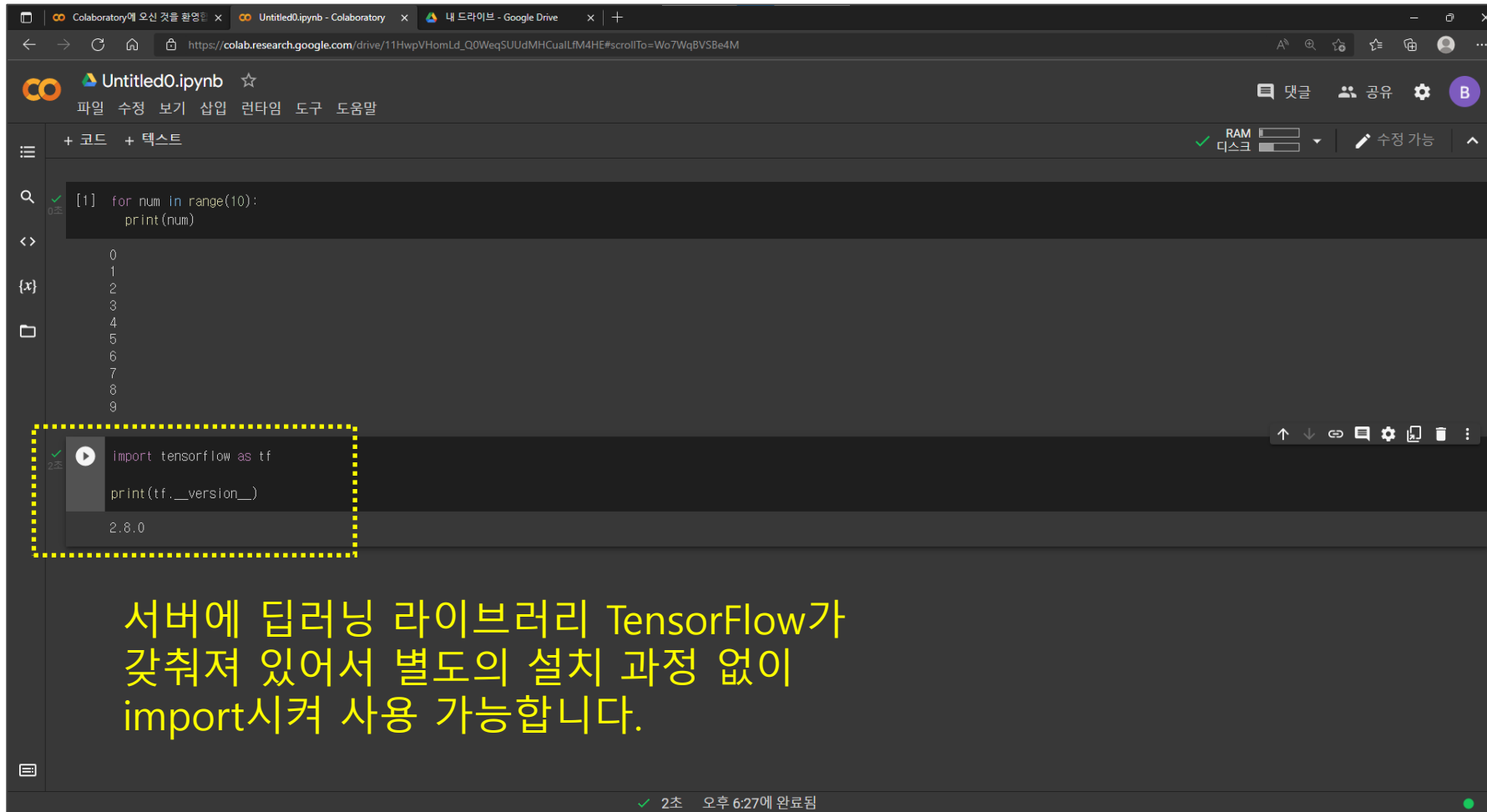
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (8/13)

- 코드 작성을 위한 셀을 추가하기 위해서, [+ 코드] 버튼을 클릭합니다. (단축키: Ctrl + M B)



❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (9/13)

- 추가된 셀에 파이썬 코드를 입력하고 실행시켜 결과를 확인해 봅니다.



The screenshot shows the Google Colaboratory web interface. The top bar includes the Colab logo, the file name 'Untitled0.ipynb', and navigation links. The left sidebar shows a file explorer with a folder icon. The main workspace has a dark theme. A code cell is selected, containing the following Python code:

```
[1] for num in range(10):  
    print(num)
```

The output of this cell is a list of numbers from 0 to 9. Below it, another code cell is highlighted with a yellow dashed border. This cell contains the following Python code:

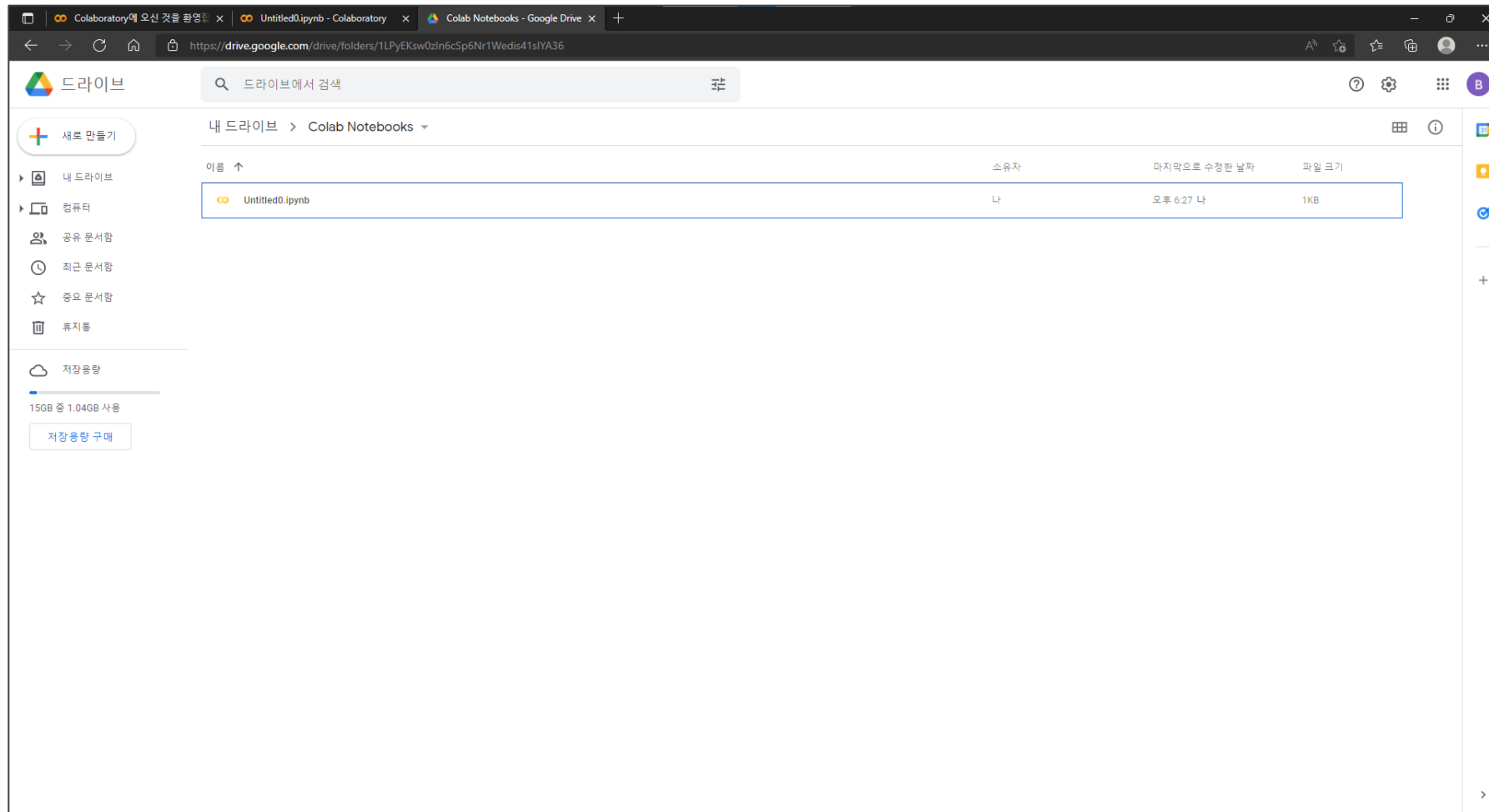
```
import tensorflow as tf  
print(tf.__version__)
```

The output of this cell is '2.8.0'. At the bottom of the interface, a status bar indicates '2초' (2 seconds) and '오후 6:27에 완료됨' (Completed at 6:27 PM).

서버에 딥러닝 라이브러리 TensorFlow가
갖춰져 있어서 별도의 설치 과정 없이
import시켜 사용 가능합니다.

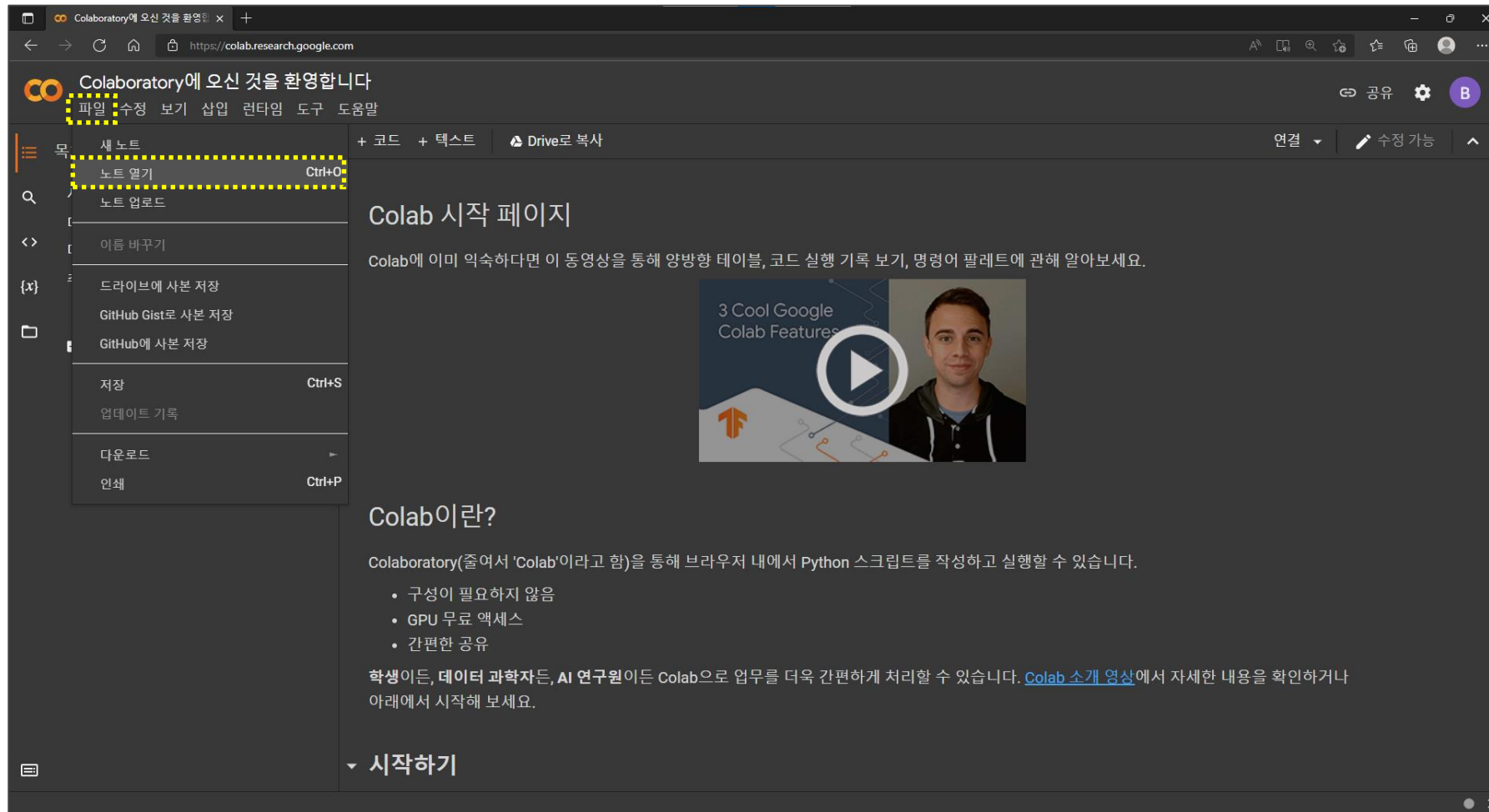
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (10/13)

- 저장 및 작업한 ipynb 파일은 Google 드라이브의 [내 드라이브] – [Colab Notebooks]에 저장됩니다.



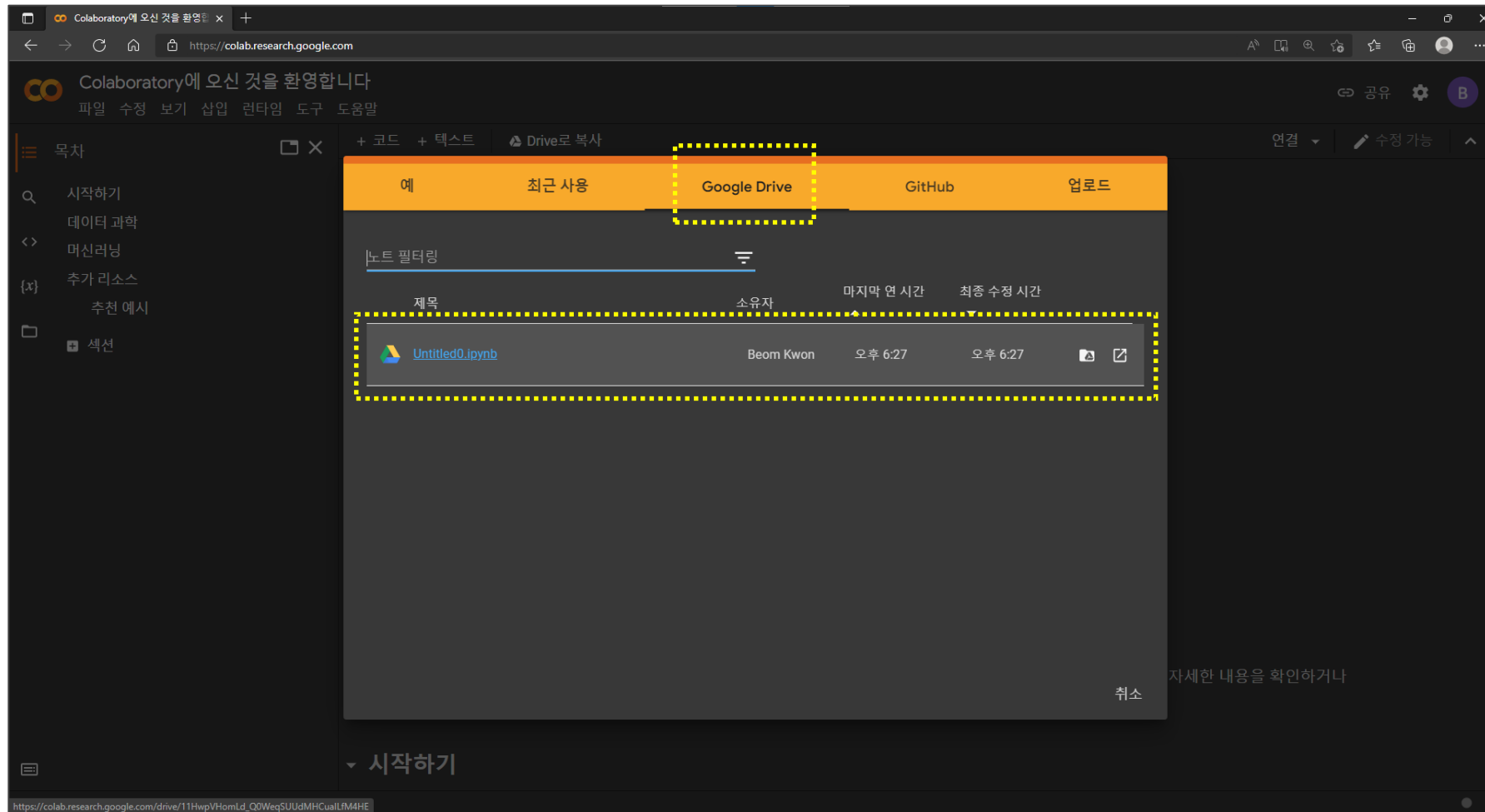
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (11/13)

- 저장되어 있는 파일을 열기 위해서, [파일] - [노트 열기] 버튼을 클릭합니다.



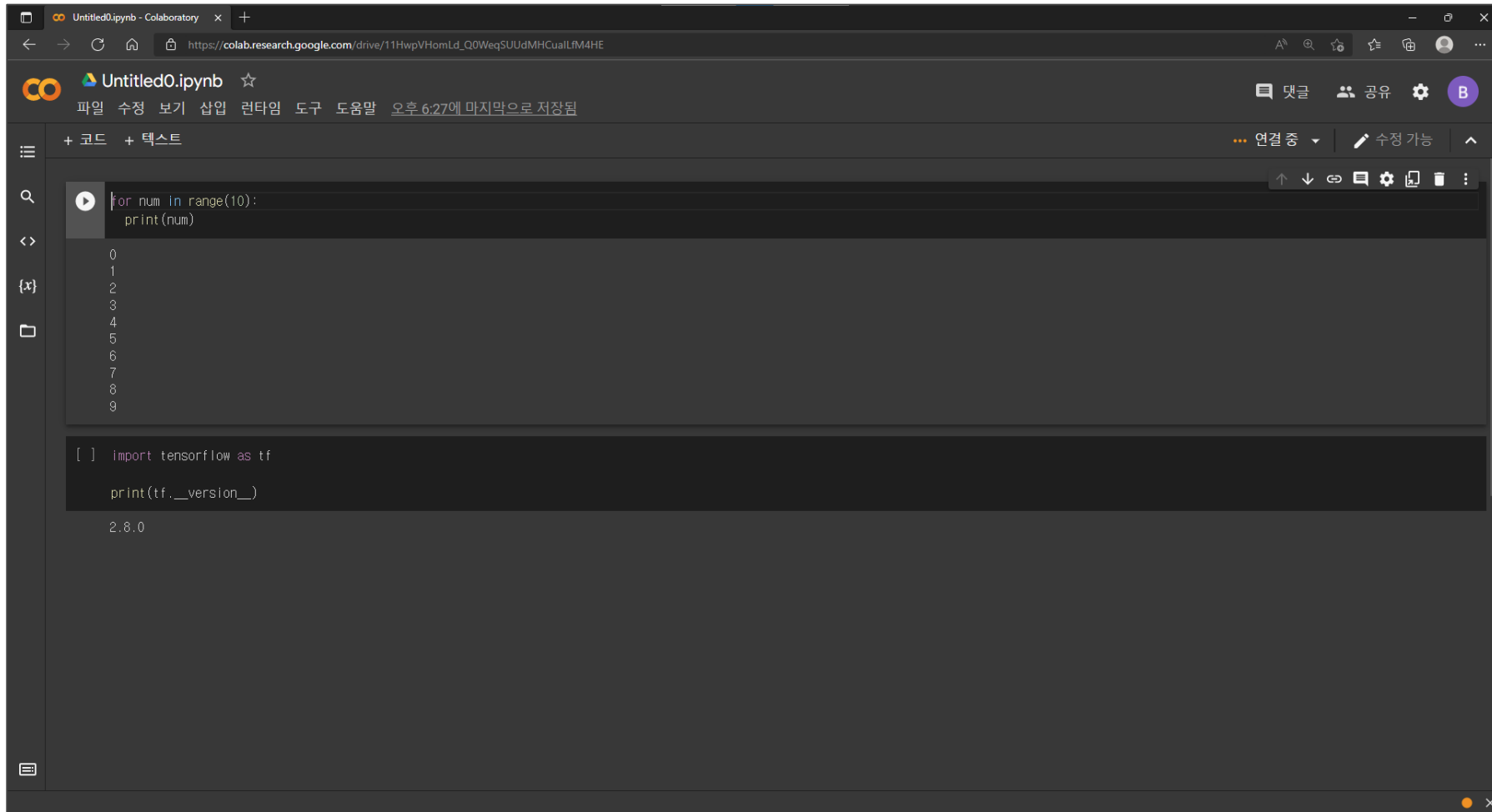
❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (12/13)

- 파업 창에서 [Google Drive] 탭에서 열고자 하는 파일을 클릭합니다.



❖ 텐서플로로 필기체 숫자 인식 학습하기: ② 구글 코랩을 사용하는 절차 (13/13)

- ipynb 파일이 열렸습니다.



The screenshot shows the Google Colaboratory web interface. The browser address bar displays the URL: https://colab.research.google.com/drive/11HwpVHomLd_Q0WeqSUUdMHCUaLLIM4HE. The notebook is titled "Untitled0.ipynb" and shows the following code cells:

```
for num in range(10):  
    print(num)
```

The output of the first cell is a list of numbers from 0 to 9:

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

The second code cell contains:

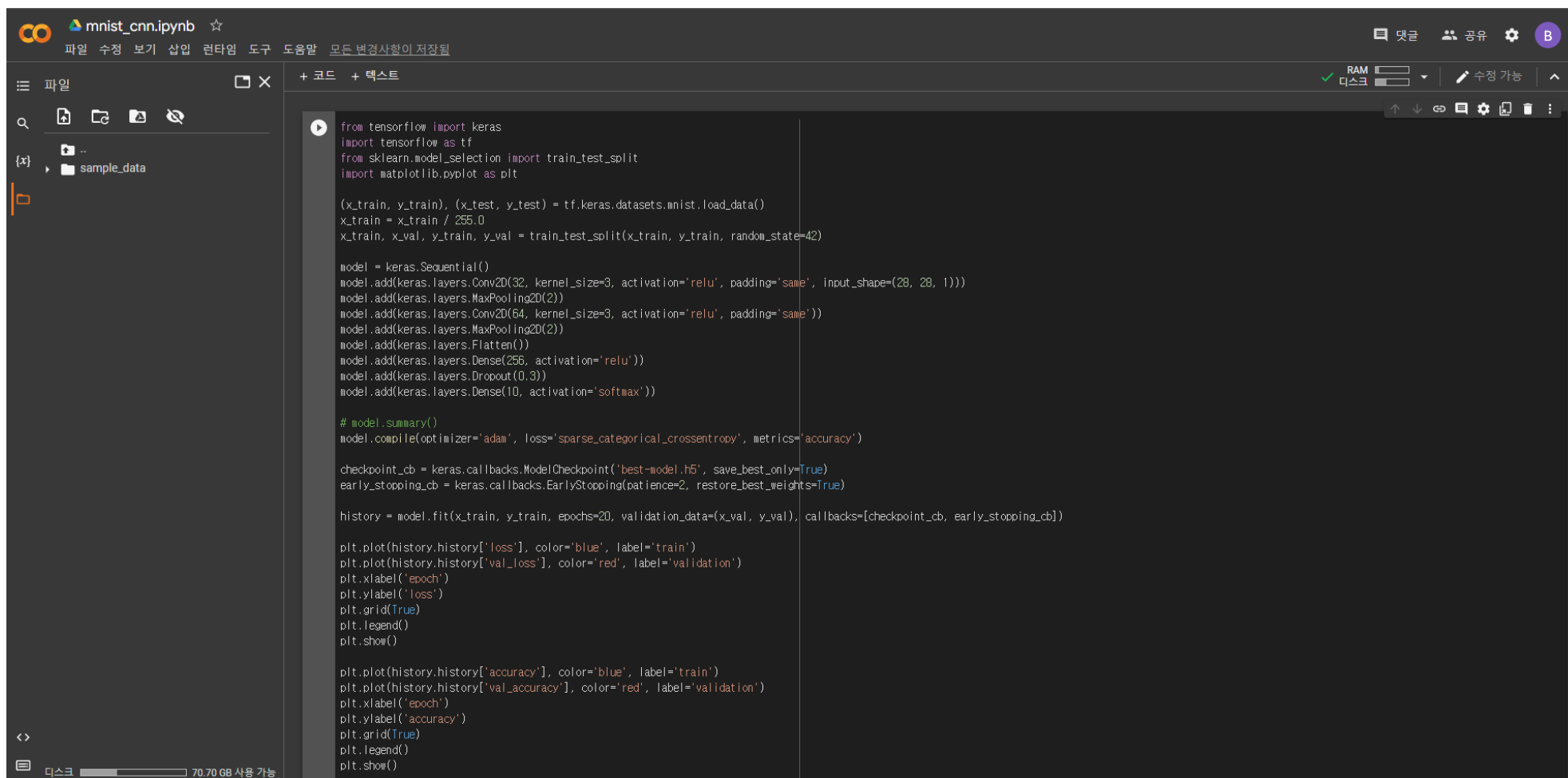
```
[ ] import tensorflow as tf  
  
print(tf.__version__)
```

The output of the second cell is:

```
2.8.0
```

❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 minst_cnn.py 실행 (1/7)

- PyCharm에서 작성한 minst_cnn.py 소스 코드를 복사해서 새 ipynb 파일에 저장합니다.



```
from tensorflow import keras
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train / 255.0
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, random_state=42)

model = keras.Sequential()
model.add(keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='same', input_shape=(28, 28, 1)))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256, activation='relu'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(10, activation='softmax'))

# model.summary()
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')

checkpoint_cb = keras.callbacks.ModelCheckpoint('best-model.h5', save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2, restore_best_weights=True)

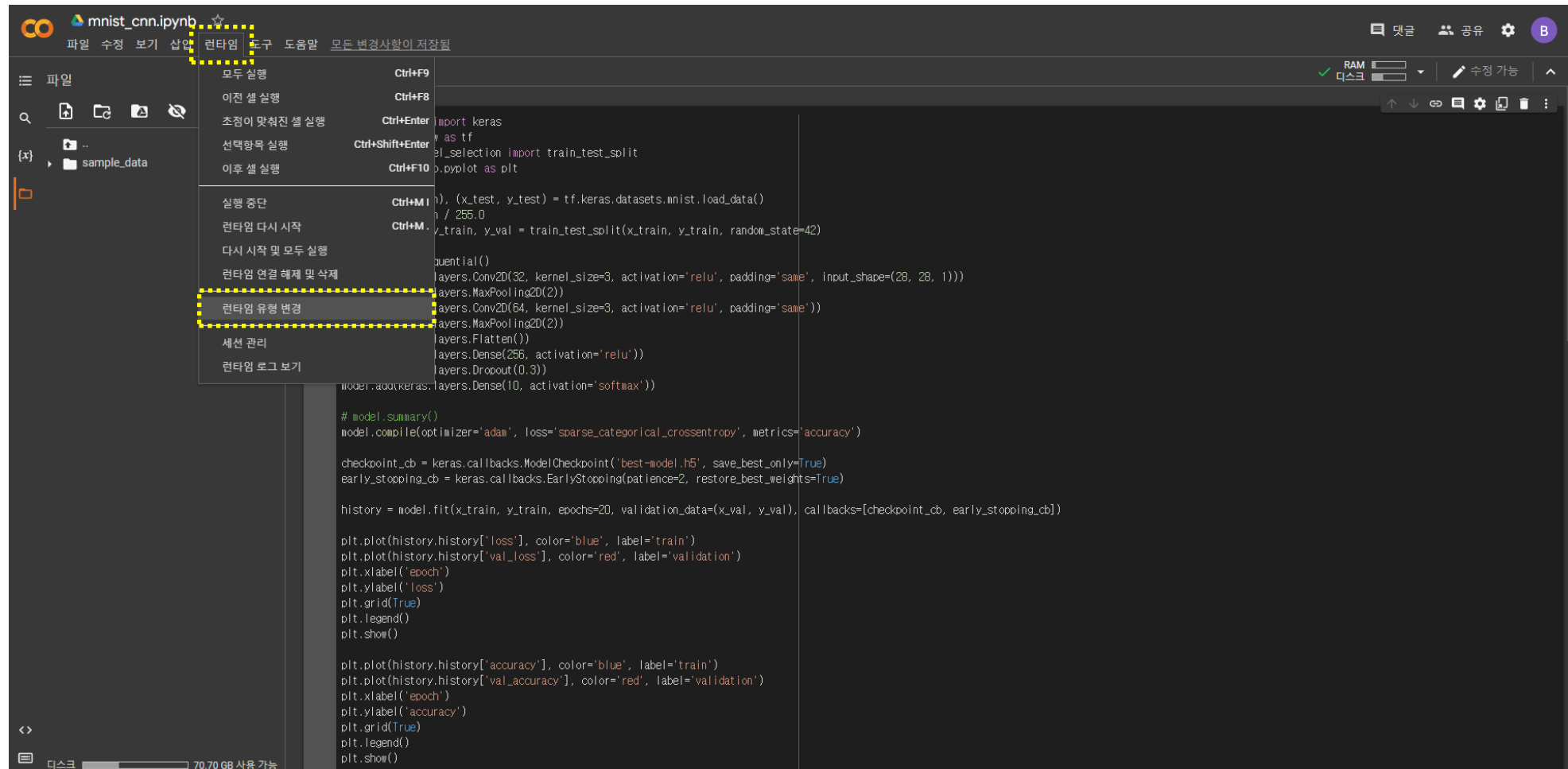
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_val, y_val), callbacks=[checkpoint_cb, early_stopping_cb])

plt.plot(history.history['loss'], color='blue', label='train')
plt.plot(history.history['val_loss'], color='red', label='validation')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='red', label='validation')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.grid(True)
plt.legend()
plt.show()
```

❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 minst_cnn.py 실행 (2/7)

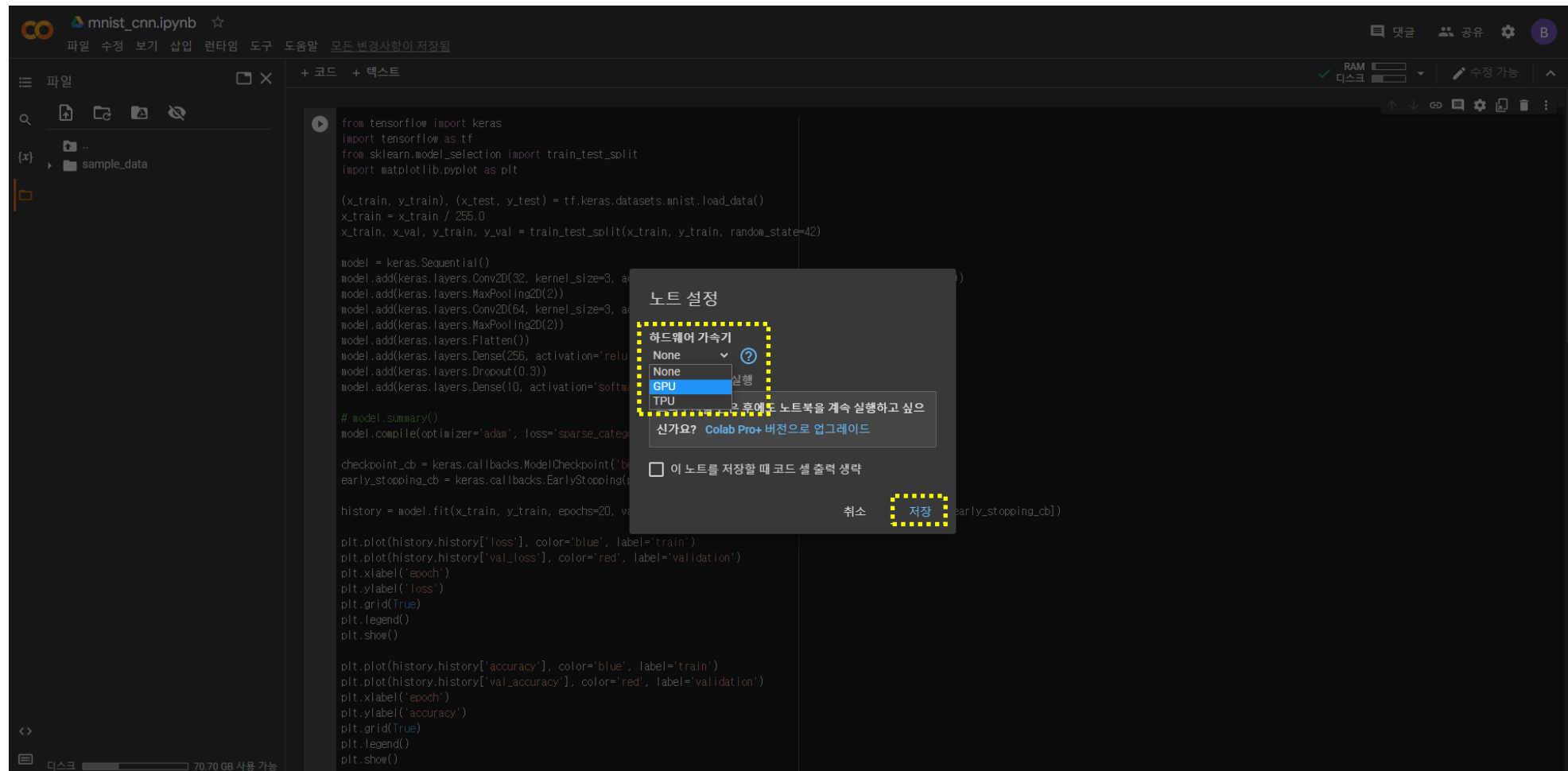
- 메뉴 탭에서 [런타임] → [런타임 유형 변경]을 클릭합니다.



16.2 딥러닝 학습과 OpenCV 실행

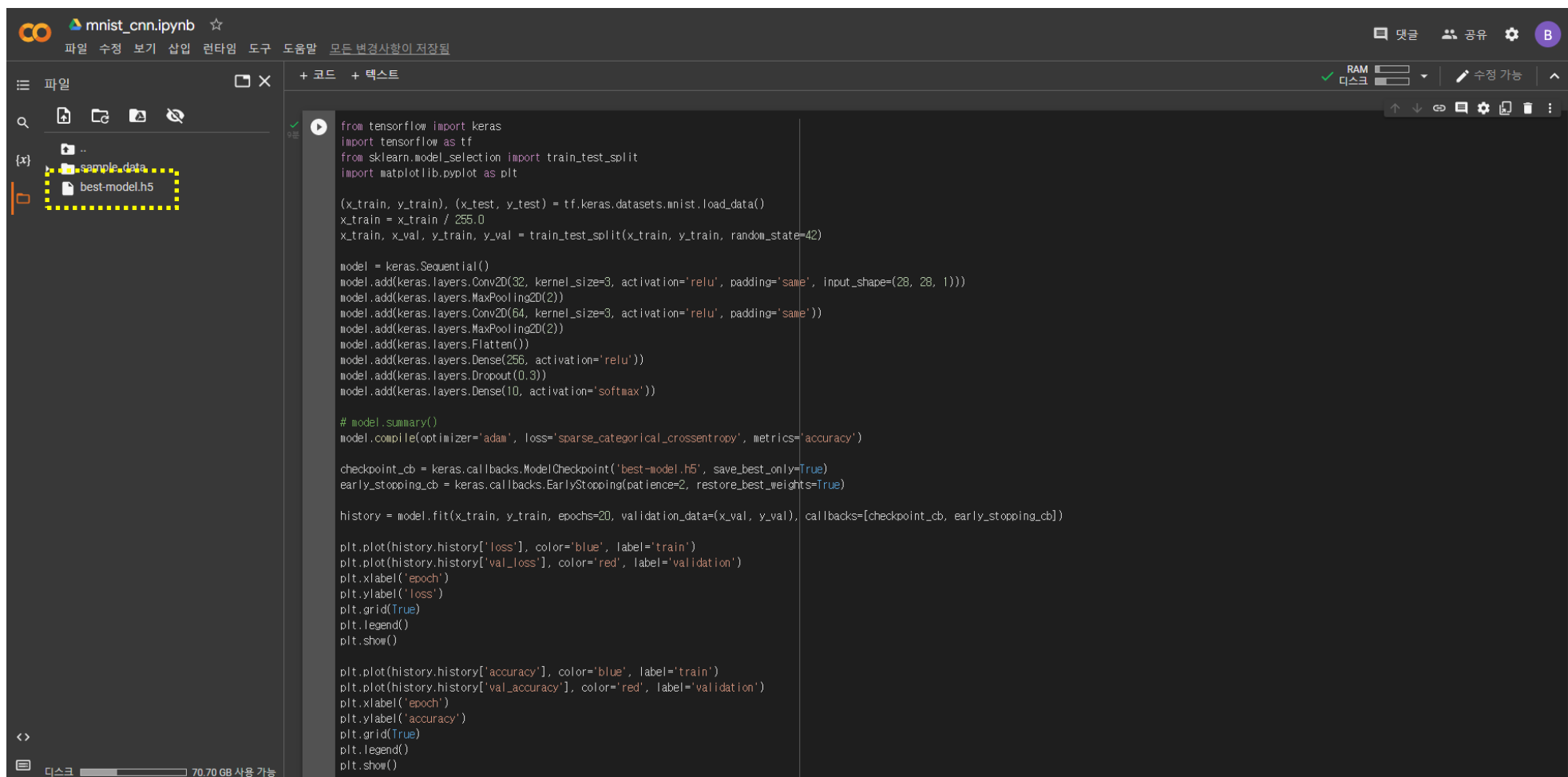
❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 minst_cnn.py 실행 (3/7)

- 하드웨어 가속기를 [None]에서 [GPU]로 변경하고 '저장' 버튼을 클릭합니다.



❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 **mnist_cnn.py** 실행 (4/7)

- 셀(cell) 실행을 통해 학습을 진행시키고, 학습이 완료되면 'best-model.h5' 파일이 생성됩니다.



```
from tensorflow import keras
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train / 255.0
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, random_state=42)

model = keras.Sequential()
model.add(keras.layers.Conv2D(32, kernel_size=3, activation='relu', padding='same', input_shape=(28, 28, 1)))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.Conv2D(64, kernel_size=3, activation='relu', padding='same'))
model.add(keras.layers.MaxPooling2D(2))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256, activation='relu'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(10, activation='softmax'))

# model.summary()
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')

checkpoint_cb = keras.callbacks.ModelCheckpoint('best-model.h5', save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=2, restore_best_weights=True)

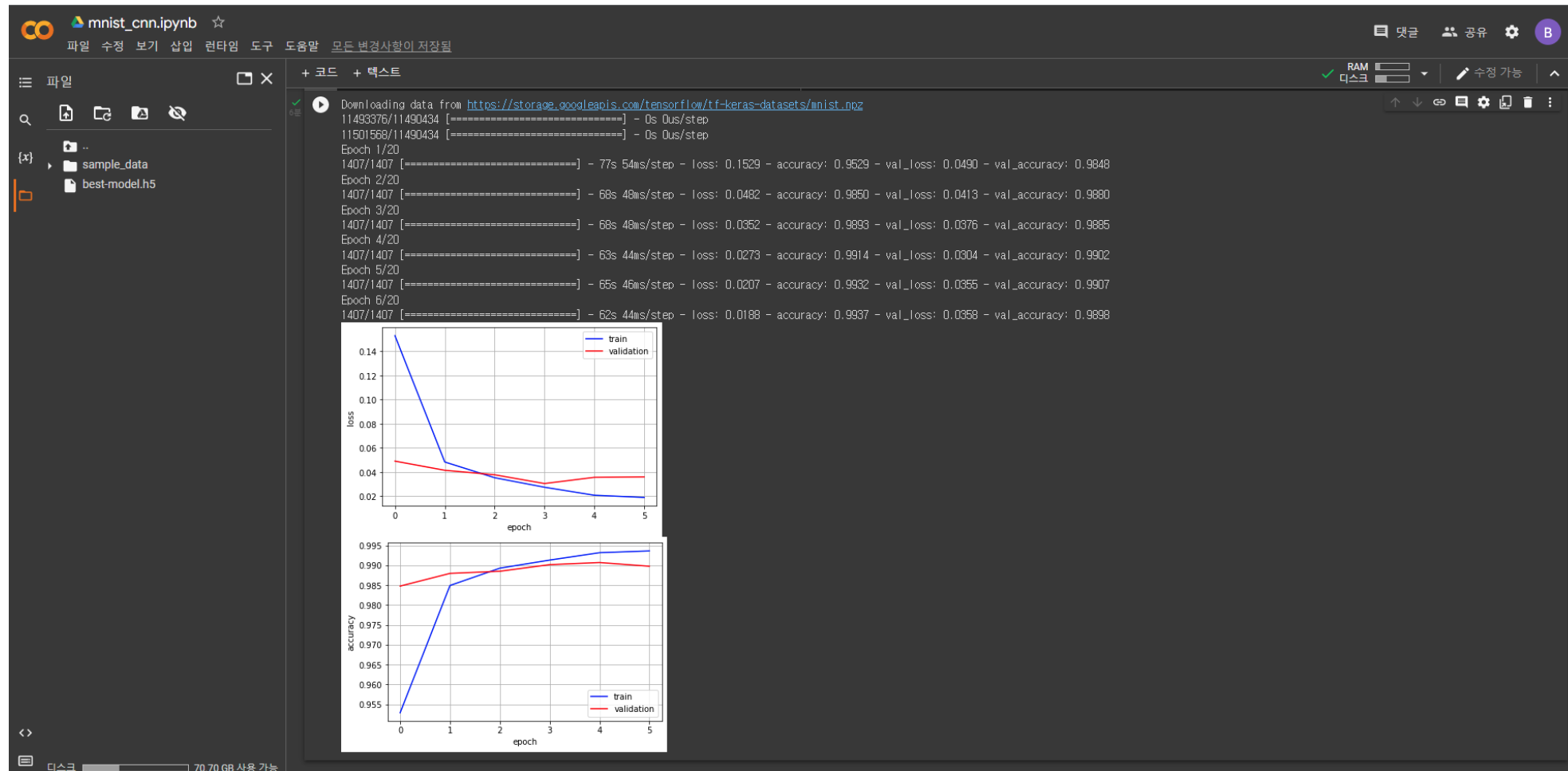
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_val, y_val), callbacks=[checkpoint_cb, early_stopping_cb])

plt.plot(history.history['loss'], color='blue', label='train')
plt.plot(history.history['val_loss'], color='red', label='validation')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
plt.show()

plt.plot(history.history['accuracy'], color='blue', label='train')
plt.plot(history.history['val_accuracy'], color='red', label='validation')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.grid(True)
plt.legend()
plt.show()
```

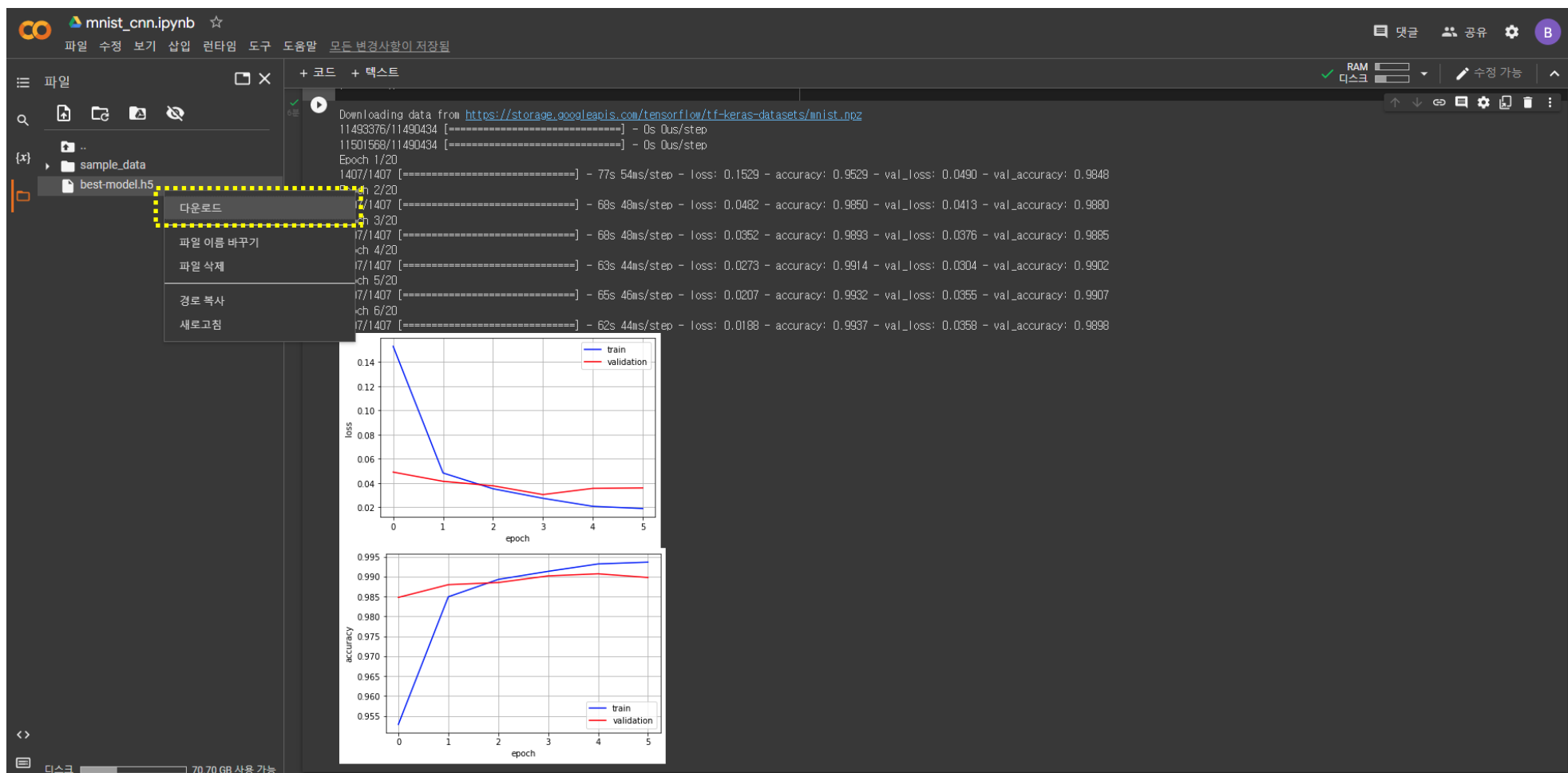
❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 `mnist_cnn.py` 실행 (5/7)

- 화면 하단으로 내려가보면 학습 과정이 기록되어 있습니다.



❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 **mnist_cnn.py** 실행 (6/7)

- 생성된 'best-model.h5'를 마우스 오른쪽 버튼으로 클릭하고, '다운로드' 버튼을 클릭합니다.



mnist_cnn.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

파일

sample_data

best-model.h5

다운로드

파일 이름 바꾸기

파일 삭제

경로 복사

새로고침

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11493376/11490434 [=====] - 0s 0us/step

11501568/11490434 [=====] - 0s 0us/step

Epoch 1/20

1407/1407 [=====] - 77s 54ms/step - loss: 0.1529 - accuracy: 0.9529 - val_loss: 0.0490 - val_accuracy: 0.9848

Epoch 2/20

1407/1407 [=====] - 68s 48ms/step - loss: 0.0482 - accuracy: 0.9850 - val_loss: 0.0413 - val_accuracy: 0.9880

Epoch 3/20

1407/1407 [=====] - 68s 48ms/step - loss: 0.0352 - accuracy: 0.9893 - val_loss: 0.0376 - val_accuracy: 0.9885

Epoch 4/20

1407/1407 [=====] - 63s 44ms/step - loss: 0.0273 - accuracy: 0.9914 - val_loss: 0.0304 - val_accuracy: 0.9902

Epoch 5/20

1407/1407 [=====] - 65s 46ms/step - loss: 0.0207 - accuracy: 0.9932 - val_loss: 0.0355 - val_accuracy: 0.9907

Epoch 6/20

1407/1407 [=====] - 62s 44ms/step - loss: 0.0188 - accuracy: 0.9937 - val_loss: 0.0358 - val_accuracy: 0.9898

loss

epoch

train

validation

accuracy

epoch

train

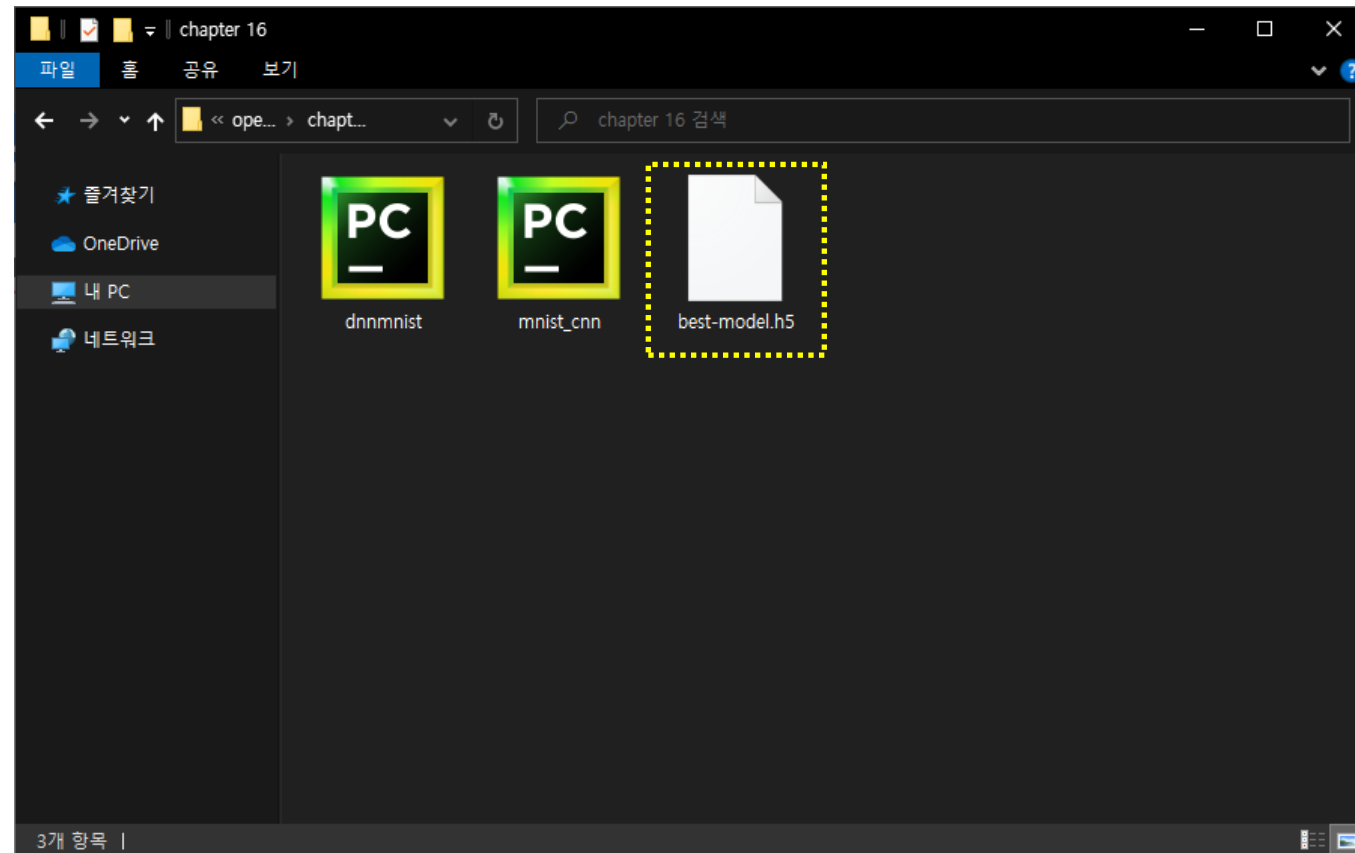
validation

디스크 70.70 GB 사용 가능

❖ 텐서플로로 필기체 숫자 인식 학습하기: ③ 구글 코랩에서 `minst_cnn.py` 실행 (7/7)

- [다운로드] 폴더에 저장되어 있는 'best-model.h5' 파일을 실습 코드가 있는 폴더로 이동시킵니다.

이 파일에 네트워크 구조와 학습된 모든 가중치 값이 저장되어 있습니다.



❖ OpenCV에서 학습된 모델 불러와서 실행하기 (1/4)

- 코드 16-3의 dnnmnist.py 에서는 텐서플로로 학습시킨 best-model.h5 파일을 불러옴
- 사용자가 마우스로 그린 필기체 숫자 영상을 CNN 모델의 입력으로 전달하여 그 결과를 예측함

코드 16-3 학습된 모델 파일을 이용한 필기체 숫자 인식 예제 (dnnmnist.py)

```
1  import numpy as np
2  import cv2
3  from tensorflow import keras
4  import tensorflow as tf
5
6  oldx, oldy = -1, -1
7
8  def on_mouse(event, x, y, flags, _):
9      global oldx, oldy
10
11     if event == cv2.EVENT_LBUTTONDOWN:
12         oldx, oldy = x, y
13
14     elif event == cv2.EVENT_LBUTTONUP:
15         oldx, oldy = -1, -1
16
17     elif event == cv2.EVENT_MOUSEMOVE:
18         if flags & cv2.EVENT_FLAG_LBUTTON:
19             cv2.line(img, (oldx, oldy), (x, y), (255, 255, 255), 40, cv2.LINE_AA)
20             oldx, oldy = x, y
21             cv2.imshow('img', img)
22
```

❖ OpenCV에서 학습된 모델 불러와서 실행하기 (2/4)

코드 16-3 학습된 모델 파일을 이용한 필기체 숫자 인식 예제 (dnnmnist.py)

```
23 model = keras.models.load_model('best-model.h5')
24
25 img = np.zeros((400, 400), np.uint8)
26
27 cv2.imshow('img', img)
28 cv2.setMouseCallback('img', on_mouse)
29
30 while True:
31     c = cv2.waitKey()
32
33     if c == 27:
34         break
35     elif c == ord(' '):
36         img = img / 255.0
37         img_3d = img.reshape(400, 400, 1)
38         resized_img = np.array(tf.image.resize(img_3d, [28, 28])).reshape(1, 28, 28)
39         print(np.squeeze(np.argmax(model.predict(resized_img, verbose=0), axis=-1)))
40
41         img.fill(0)
42         cv2.imshow('img', img)
43
44 cv2.destroyAllWindows()
```

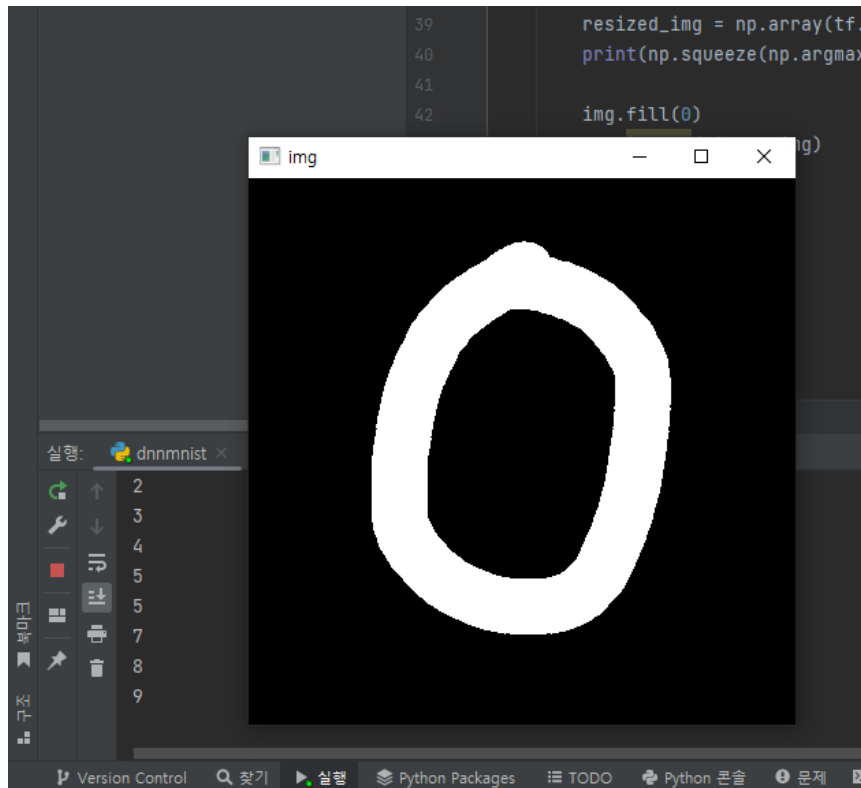
❖ OpenCV에서 학습된 모델 불러와서 실행하기 (3/4)

- dnnmnist.py 소스 코드 설명

- 23행 best-model.h5 파일로부터 CNN 모델을 불러옵니다.
- 25행 400×400 크기의 빈 영상 img를 생성합니다.
이후 img 영상에서 숫자 그림을 그려서 숫자를 인식합니다.
- 35~38행 img 창에서 키보드 [Space] 키를 누르면 img 영상의 픽셀 값을 0부터 1사이로 정규화시키고, (400, 400, 1) 크기를 (28, 28, 1) 크기로 영상을 줄입니다.
predict() 메서드의 입력 형태를 맞추기 위해서 (1, 28, 28)로 변경합니다.
- 39행 CNN 모델의 predict() 메서드로 영상을 입력합니다.
반환되는 확률 행렬에서 최댓값의 위치 인덱스 정보를 찾고, 실행 결과 창에 출력합니다.
- 41~42행 img 영상을 검은색으로 초기화하고, 화면에 출력합니다.
- 33행 [Esc] 키를 누르면 프로그램을 종료합니다.

❖ OpenCV에서 학습된 모델 불러와서 실행하기 (4/4)

- 프로그램이 처음 실행되면 검은색으로 초기화된 img 영상이 화면에 나타남
- 이 위에서 마우스 왼쪽 버튼을 눌러서 숫자를 그린 후 [Space] 키를 누르면 실행 결과 창에 인식 결과가 출력됨



◀ 그림 16-9 dnnmnist 프로그램 실행 결과

16.3 OpenCV와 딥러닝 활용

16.1 딥러닝과 OpenCV DNN 모듈

16.2 딥러닝 학습과 OpenCV 실행

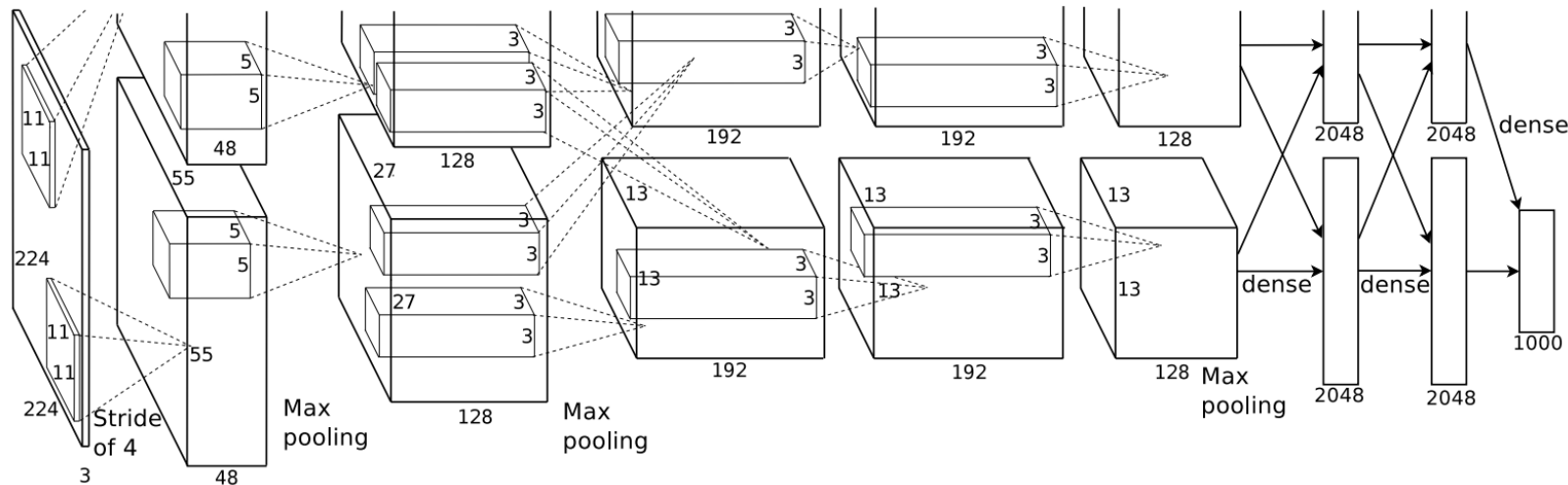
❖ 구글넷 영상 인식 (1/17)

- 딥러닝이 컴퓨터 비전 분야에서 크게 발전할 수 있었던 이유 중에는 **ILSVRC**(ImageNet Large Scale Visual Recognition Competition) 대회의 영향도 있음
- ILSVRC는 영상 인식과 객체 검출 등의 성능을 겨루는 일종의 알고리즘 경진 대회로서 2010년부터 2017년까지 매년 개최되었음



❖ 구글넷 영상 인식 (2/17)

- ILSVRC는 ImageNet이라는 대규모 영상 데이터베이스를 이용함
- 영상 인식 분야에서는 1000개의 카테고리로 분류된 100만 개 이상의 영상을 사용하여 성능을 비교함
- 이 대회에서 2012년에 **알렉스넷(AlexNet)**이라는 딥러닝 알고리즘이
기존 컴퓨터 비전 및 머신 러닝 기반의 알고리즘보다 월등히 높은 성능을 나타냄
- 컴퓨터 비전 분야에 딥러닝 열풍이 시작됨[Krizhevsky12]

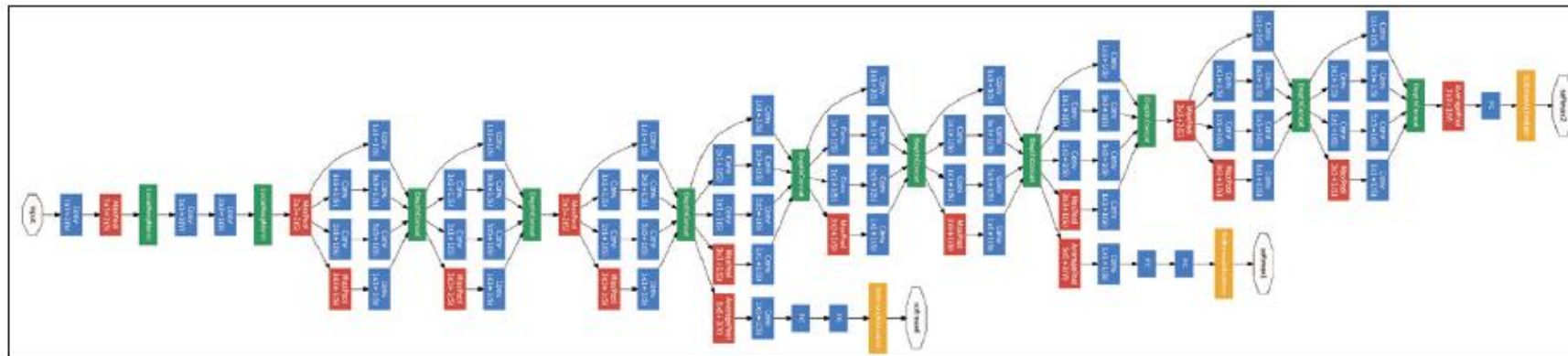


[Krizhevsky12] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Conference on Neural Information Processing Systems (NIPS)*, Dec. 2012, pp. 1-9.

❖ 구글넷 영상 인식 (3/17)

- **구글넷(GoogLeNet)**은 이름에서 알 수 있듯이 구글(Google)에서 발표한 네트워크 구조임
- 2014년 ILSVRC 영상 인식 분야에서 1위를 차지함[Szegedy15]
- 구글넷은 총 22개의 레이어로 구성됨
- 이는 동시대에 발표되었던 딥러닝 네트워크 구조 중에서 가장 많은 레이어를 사용한 형태임

▼ 그림 16-10 구글넷 네트워크 구조

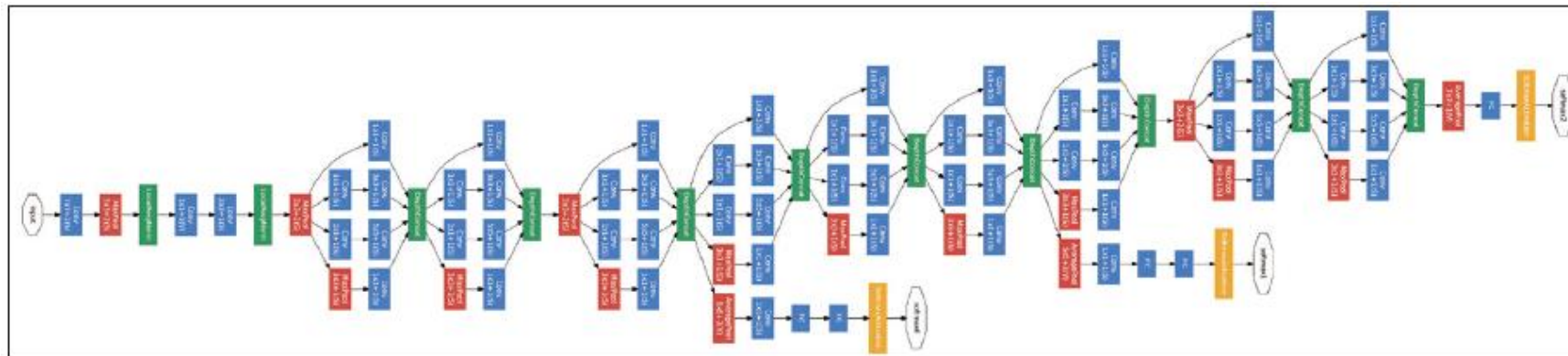


[Szegedy15] C. Szegedy *et al.* "Going deeper with convolutions,"
in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 1-9.

❖ 구글넷 영상 인식 (4/17)

- 레이어를 매우 깊게 설계하였지만 완전 연결 레이어가 없는 구조를 통해 기존의 다른 네트워크보다 파라미터 수가 훨씬 적은 것이 특징임
- 구글넷은 특히 다양한 크기의 커널을 한꺼번에 사용하여 영상에서 큰 특징과 작은 특징을 모두 추출할 수 있도록 설계됨

▼ 그림 16-10 구글넷 네트워크 구조

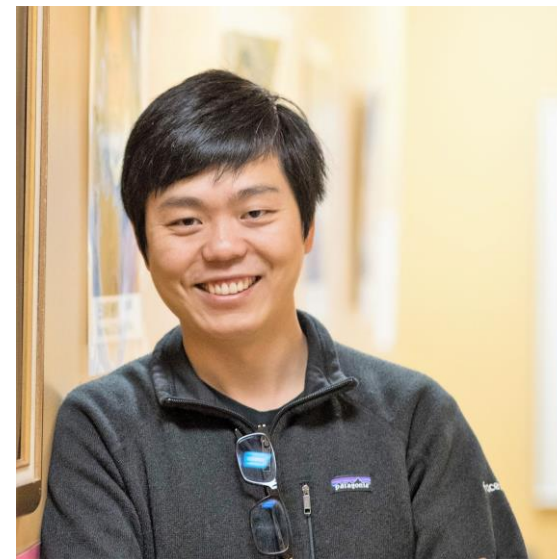


❖ 구글넷 영상 인식 (5/17)

- OpenCV에서 구글넷 인식 기능을 사용하려면 다른 딥러닝 프레임워크를 이용하여 미리 학습된 모델 파일과 구성 파일이 필요함
- 카페는 미리 학습된 딥러닝 모델 파일을 내려받을 수 있는 모델주(model zoo)를 운영하고 있음
- 카페 프레임워크를 이용하여 학습된 구글넷 모델 파일은 다음 링크에서 내려받을 수 있음
 - http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel
- 카페에서 학습된 구글넷 구조를 표현한 구성 파일은 모델 주 깃허브 페이지에서 내려받을 수 있음
 - https://github.com/BVLC/caffe/blob/master/models/bvlc_googlenet/deploy.prototxt

❖ 구글넷 영상 인식 (6/17)

- 카페는 Yangqing Jia가 캘리포니아 대학교(University of California)에서 Ph.D. 과정 중에 제작한 딥러닝 프레임워크임[Jia14]
- 2017년 4월, 페이스북(Facebook)에서 카페2(Caffe2)를 공개함
- 2018년 3월, 카페2가 파이토치(PyTorch)에 합쳐지게 됨

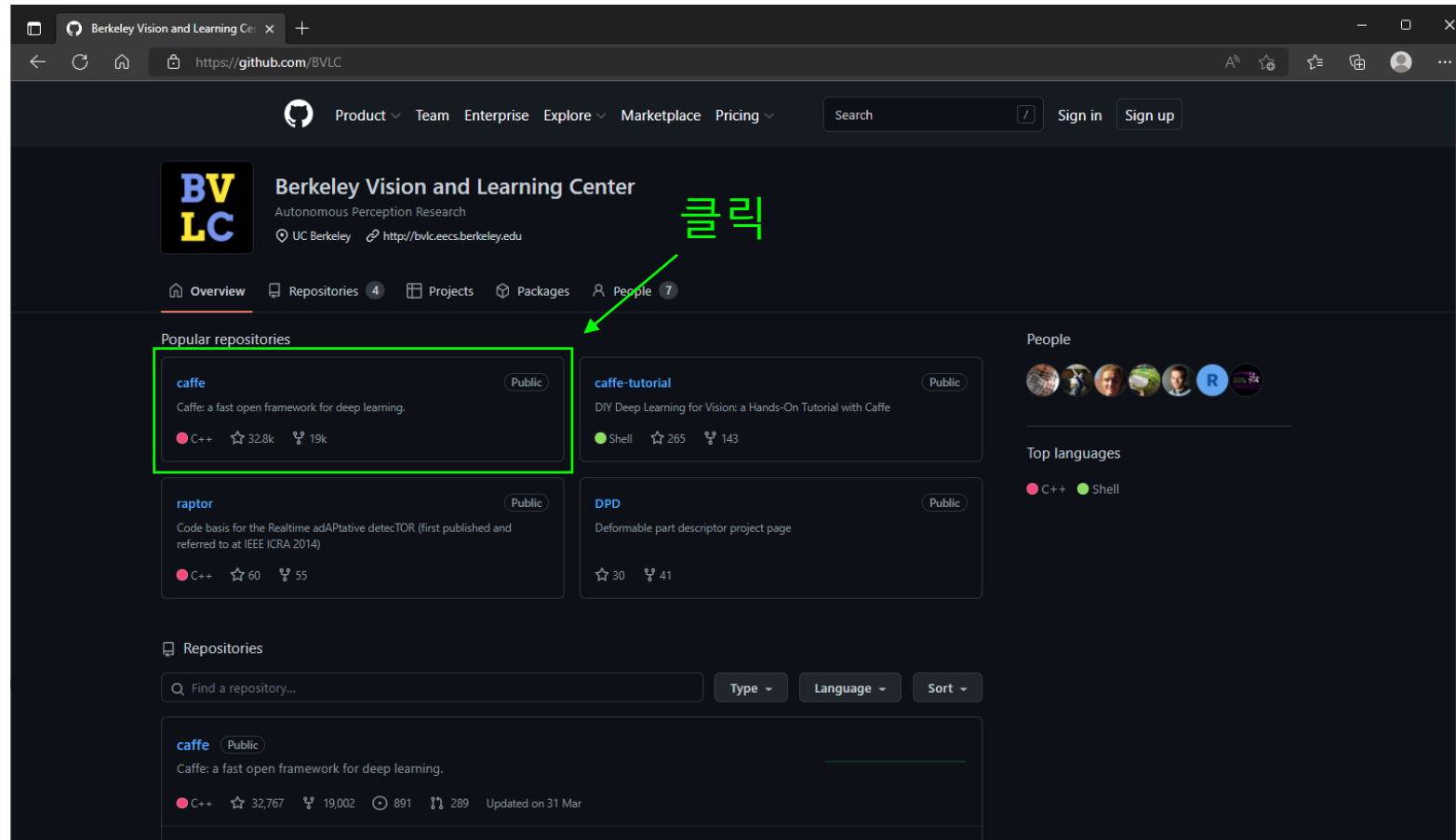


[사진출처] <http://daggerfs.com/>

[Jia14] Y. Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding," arXiv preprint arXiv:1408.5093, 2014.

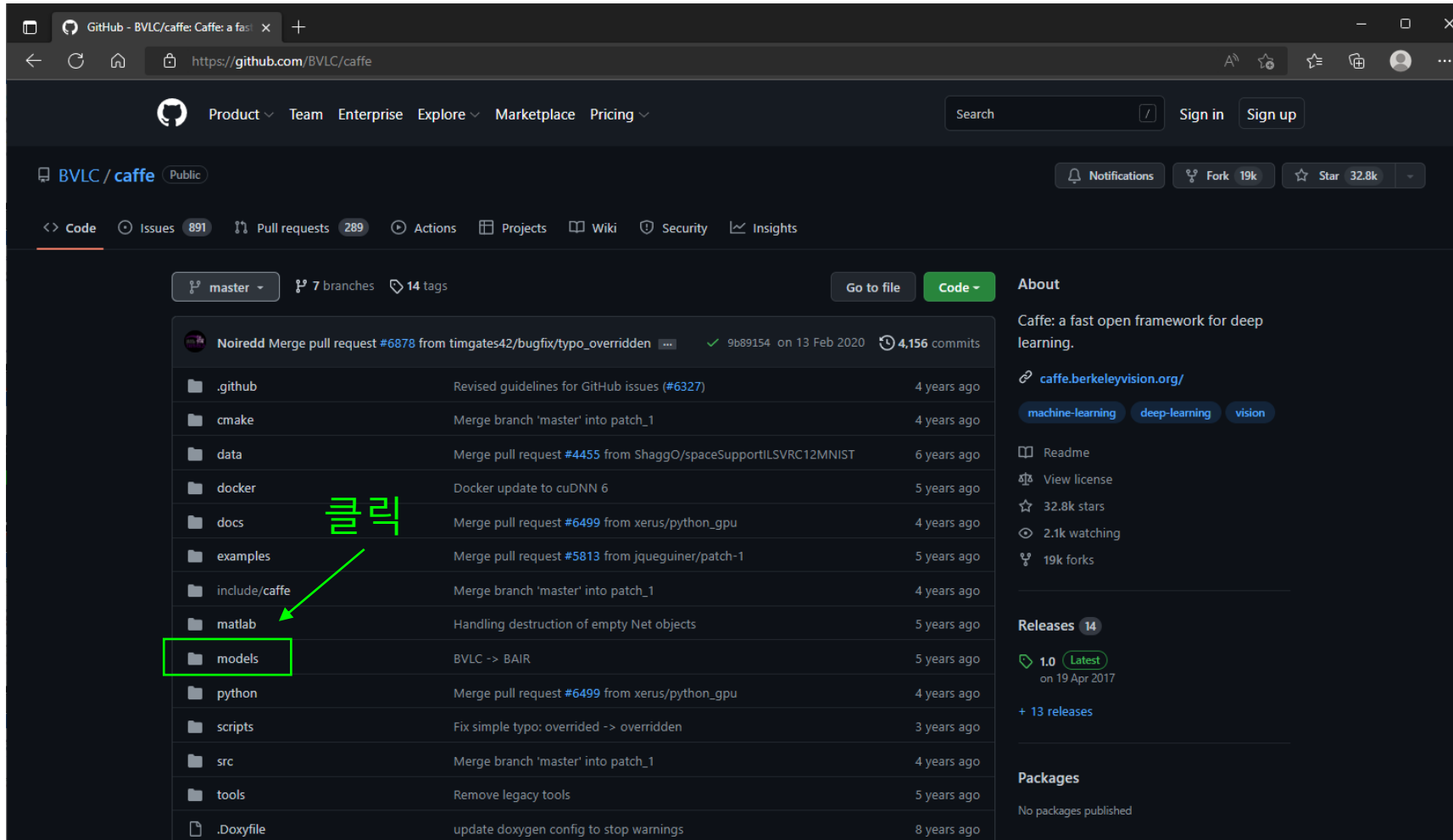
❖ 구글넷 영상 인식 (7/17)

- 카페는 현재 **BVLC**(Berkeley Vision and Learning Center)에서 유지관리되고 있음
- [URL] <https://github.com/BVLC>



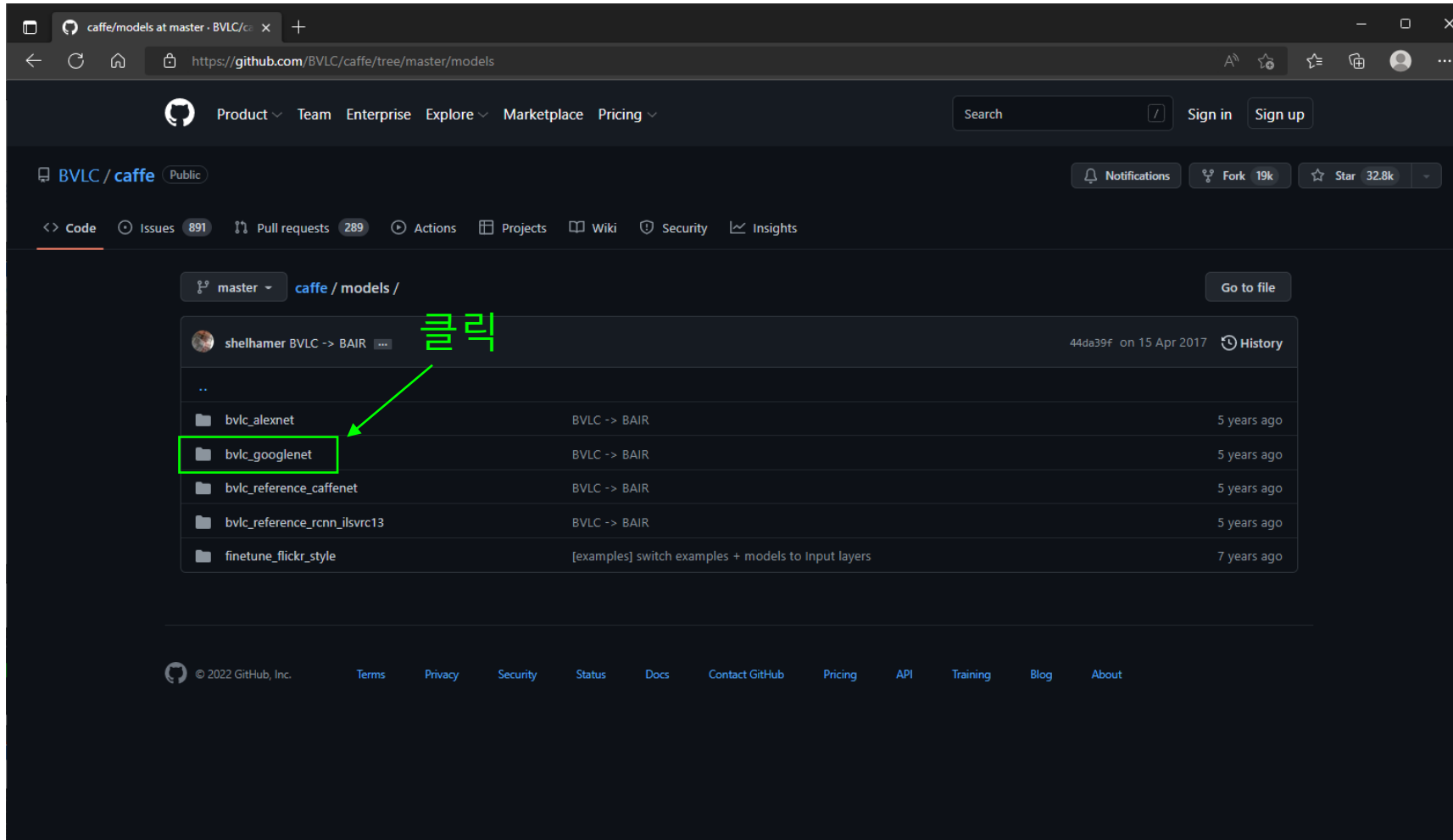
❖ 구글넷 영상 인식 (8/17)

- [models]를 클릭



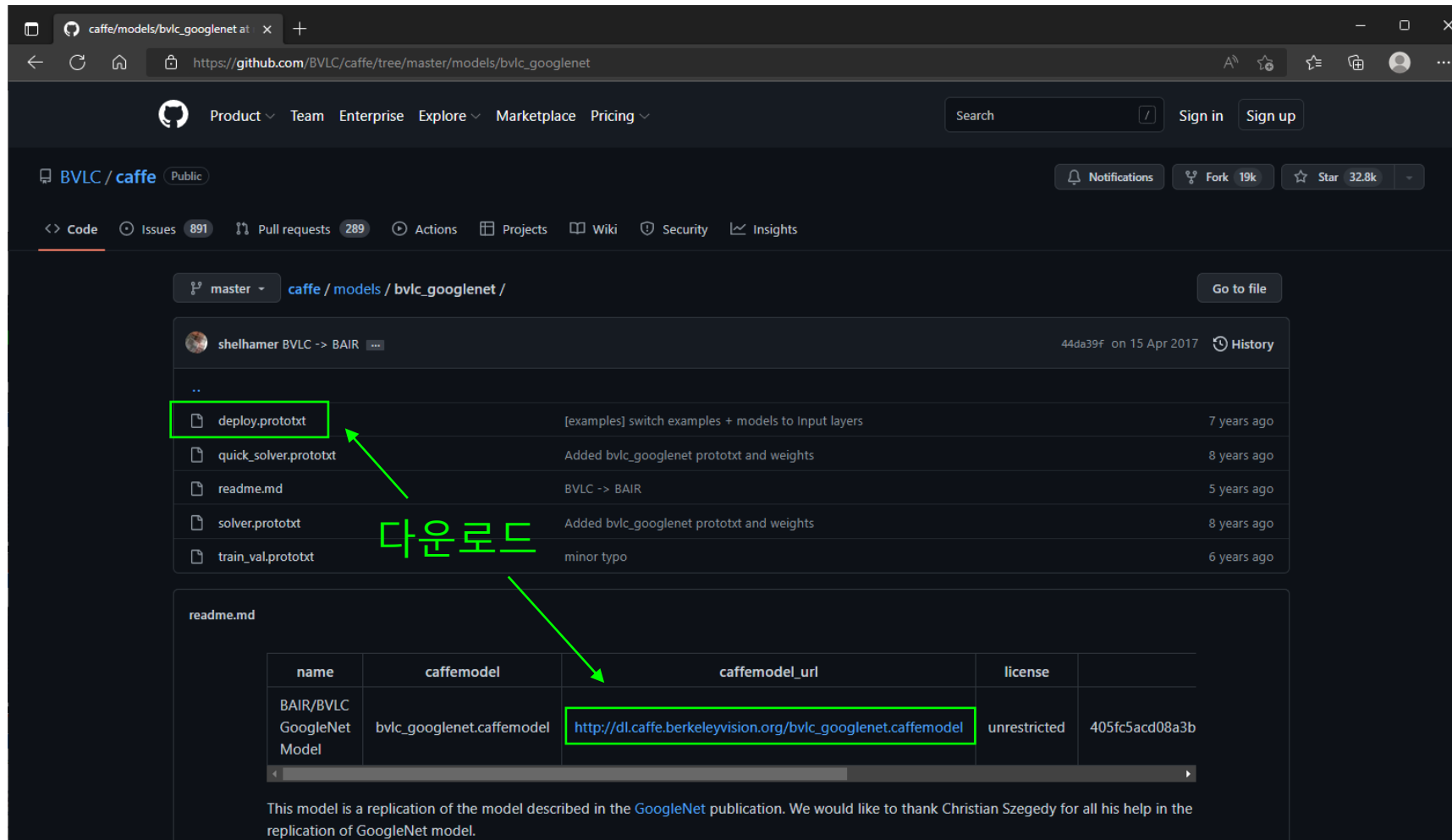
❖ 구글넷 영상 인식 (9/17)

- [bvlc_googlenet]를 클릭



❖ 구글넷 영상 인식 (10/17)

- bvlc_googlenet.caffemodel 파일과 deploy.prototxt 파일을 다운로드 받을 수 있음



master caffe / models / bvlc_googlenet /

shelhamer BVLC -> BAIR 44da39f on 15 Apr 2017 History

- deploy.prototxt [examples] switch examples + models to Input layers 7 years ago
- quick_solver.prototxt Added bvlc_googlenet prototxt and weights 8 years ago
- readme.md BVLC -> BAIR 5 years ago
- solver.prototxt Added bvlc_googlenet prototxt and weights 8 years ago
- train_val.prototxt minor typo 6 years ago

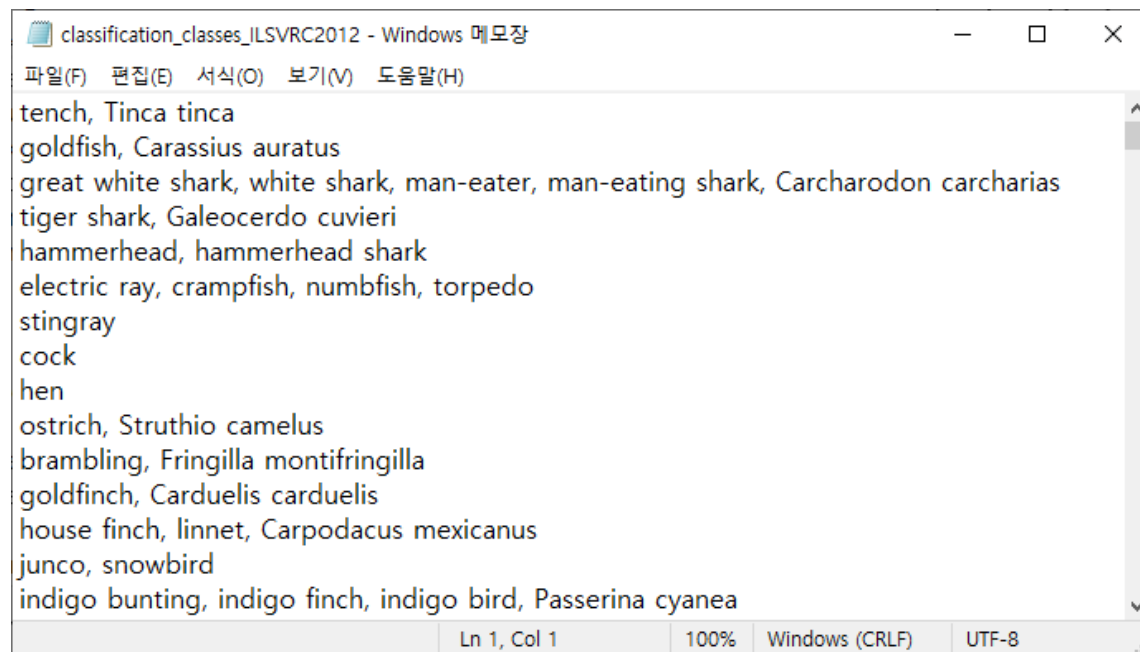
readme.md

name	caffemodel	caffemodel_url	license	
BAIR/BVLC GoogleNet Model	bvlc_googlenet.caffemodel	http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel	unrestricted	405fc5acd08a3b

This model is a replication of the model described in the [GoogleNet](#) publication. We would like to thank Christian Szegedy for all his help in the replication of GoogleNet model.

❖ 구글넷 영상 인식 (11/17)

- 구글넷 인식 기능을 제대로 구현하려면 모델 파일과 구성 파일 외에 인식된 영상 클래스 이름이 적힌 텍스트 파일이 추가로 필요함
- ILSVRC 대회에서 사용된 1000개의 영상 클래스 이름이 적혀 있는 텍스트 파일이 필요하며, 이 파일의 이름은 classification_classes_ILSVRC2012.txt임
- 이 파일에는 1000개의 영상 클래스 이름이 한 줄씩 적혀 있음



❖ 구글넷 영상 인식 (12/17)

- 구글넷 예제 프로그램을 만들기 위해 필요한 세 파일을 정리하면 다음과 같음
 - 학습 모델 파일: `bvlc_googlenet.caffemodel`
 - 구성 파일: `deploy.prototxt`
 - 클래스 이름 파일: `classification_classes_ILSVRC2012.txt`

❖ 구글넷 영상 인식 (13/17)

- 이제 OpenCV에서 이 파일을 이용하여 영상을 인식하는 예제 프로그램 작성 방법에 대해 알아보자
- 먼저 readNet() 함수를 이용하여 Net 클래스 객체를 생성하는 소스 코드를 살펴보자
- readNet() 함수 인자로 모델 파일 이름과 구성 파일 이름을 순서대로 전달하면 Net 클래스 객체가 생성됨

```
model = 'bvlc_googlenet.caffemodel'  
config = 'deploy.prototxt'  
  
net = cv2.dnn.readNet(model, config)
```

- bvlc_googlenet.caffemodel 파일과 deploy.prototxt 파일은 프로그램 실행 폴더에 함께 있어야 함

❖ 구글넷 영상 인식 (14/17)

- 그다음은 각 클래스 이름이 적혀 있는 텍스트 파일로부터 각 클래스 이름을 불러와 리스트(list) 타입의 변수에 저장하는 코드를 살펴보자

```
f = open('classification_classes_ILSVRC2012.txt', 'rt')
classNames = f.read().split('\n')
f.close()
```

- 텍스트 파일에서 1000개의 영상 클래스 이름은 한 줄씩 적혀 있기 때문에 split() 함수에 '\n'을 전달함

❖ 구글넷 영상 인식 (15/17)

- 사용할 네트워크와 클래스 문자열 정보를 제대로 불러왔으면 이제 네트워크에 영상 블록을 전달하여 추론하는 코드를 알아보자
- 시험용 입력 영상은 `imread()` 함수를 이용하여 불러옴
- 불러온 영상을 `blobFromImage()` 함수를 이용하여 블록으로 변환함
- 카페에서 학습된 구글넷은 입력으로 224×224 크기의 BGR 컬러 영상을 사용함
- 각 영상에서 평균값 (104, 117, 123)을 빼서 정규화함
- `blobFromImage()` 함수 인자는 다음과 같이 구성해야 함

```
img = cv2.imread(filename)
inputBlob = cv2.dnn.blobFromImage(img, 1, (224, 224), (104, 117, 123))
```


❖ 구글넷 영상 인식 (16/17)

- 앞 코드에 의해 만들어지는 inputBlob 행렬은 (1, 3, 224, 224) 형태를 갖는 4차원 행렬임
- 이 행렬을 앞서 생성한 네트워크에 입력으로 전달하고, 순방향으로 실행하여 그 결과를 받아 옴

```
net.setInput(inputBlob, 'data')  
prob = net.forward()
```

- 카페 구글넷 네트워크의 경우 최종 레이어는 1000개의 노드를 가지고 있음
- 각 노드 출력은 해당 번호 영상 클래스에 대한 확률을 표현함

❖ 구글넷 영상 인식 (17/17)

- 앞 코드에서 `net.forward()` 코드가 반환하는 `numpy.ndarray` 객체 `prob`는 (1, 1000) 크기의 행렬임
- `prob`는 1행 1000열 행렬이고, 각 열에는 1000개 클래스에 대한 확률이 저장됨
- `prob` 행렬에서 최대 확률을 갖는 열 번호가 입력 영상에 해당하는 클래스 번호라고 판단할 수 있음
- `prob` 행렬에서 최대 확률 위치는 `np.argmax()` 함수를 이용하여 쉽게 알아낼 수 있음
- 만약 최대 확률에 해당하는 클래스 이름을 알고 싶다면 `classNames`에서 인덱싱을 이용하여 해당 문자열을 추출할 수 있음

❖ 구글넷 영상 인식 예제 프로그램 (1/5)

- 코드 16-4에 나타난 classify.py 소스 코드는 영상에 포함된 주된 객체를 판단함
- 해당 객체 이름과 판단 확률을 영상 위에 문자열로 출력함

코드 16-4 구글넷 영상 인식 예제 프로그램 (classify.py)

```
1  import sys
2  import numpy as np
3  import cv2
4
5  filename = 'space_shuttle.jpg'
6
7  img = cv2.imread(filename)
8  if img is None:
9      print('Image load failed!')
10     sys.exit()
11
12  net = cv2.dnn.readNet('bvlc_googlenet.caffemodel', 'deploy.prototxt')
13  if net.empty():
14      print('Network load failed!')
15      sys.exit()
16
```

❖ 구글넷 영상 인식 예제 프로그램 (2/5)

코드 16-4 구글넷 영상 인식 예제 프로그램 (classify.py)

```
17 f = open('classification_classes_ILSVRC2012.txt', 'rt')
18 classNames = f.read().split('\n')
19 f.close()
20
21 inputBlob = cv2.dnn.blobFromImage(img, 1, (224, 224), (104, 117, 123))
22 net.setInput(inputBlob, 'data')
23 prob = net.forward()
24
25 out = prob.flatten()
26 classId = np.argmax(out)
27 confidence = out[classId]
28
29 pred = '%s (%4.2f%%)' % (classNames[classId], confidence * 100)
30 cv2.putText(img, pred, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 1, cv2.LINE_AA)
31
32 cv2.imshow('img', img)
33 cv2.waitKey()
34 cv2.destroyAllWindows()
```

❖ 구글넷 영상 인식 예제 프로그램 (3/5)

● classify.py 소스 코드 설명

- 5행 시험용 입력 영상 파일의 이름을 `filename` 변수에 저장합니다.
- 12~15행 모델 파일과 구성 파일을 이용하여 `net` 객체를 생성합니다.
 `net` 객체 생성에 실패하면 에러 메시지를 출력하고 프로그램을 종료합니다.
- 17~19행 `classification_classes_ILSVRC2012.txt` 파일을 한 줄씩 읽어서 해당 문자열을 리스트 변수 `classNames`에 저장합니다.
- 21~23행 입력 영상 `img`를 이용하여 입력 블롭 `inputBlob`을 생성하고, 이를 네트워크에 입력으로 주고 실행합니다.
 실행 결과는 `prob` 행렬에 저장됩니다.
- 25~27행 `prob` 행렬에서 최댓값과 최댓값 위치를 찾습니다.
 최댓값 위치는 인식된 영상 클래스 인덱스이고, 최댓값은 확률을 의미합니다.
- 29~30행 인식된 클래스 이름과 확률을 문자열 형태로 영상 위에 나타냅니다.

❖ 구글넷 영상 인식 예제 프로그램 (4/5)

- 그림 16-11은 space_shuttle.jpg 우주 왕복선 사진을 인식한 결과임
- img 창 상단에 객체의 이름 space shuttle과 확률 99.99% 문자열이 출력되어 있는 것을 확인할 수 있음

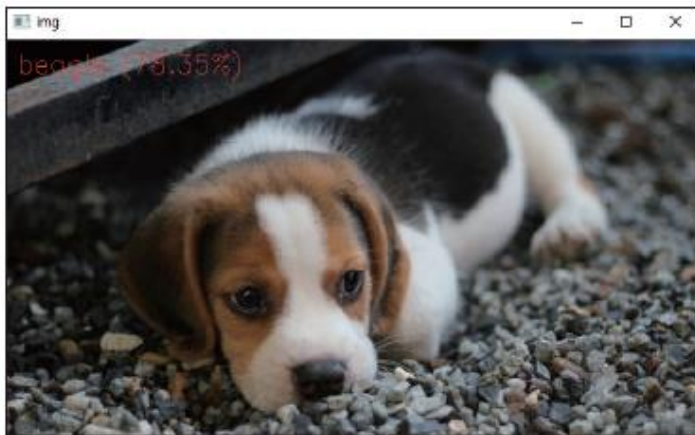
▼ 그림 16-11 구글넷 영상 인식 예제 프로그램 실행 결과



❖ 구글넷 영상 인식 예제 프로그램 (5/5)

- 나머지 시험용 영상 파일에 대해 classify 프로그램으로 인식한 결과를 그림 16-12에 나타냄

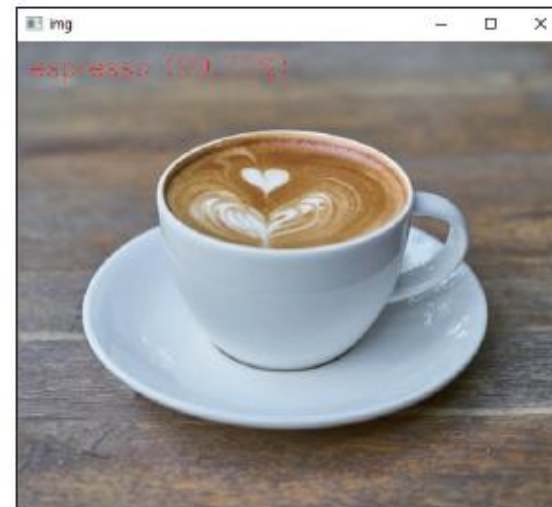
▼ 그림 16-12 다양한 영상에 대한 구글넷 영상 인식 예제 프로그램 실행 결과



(a)



(b)



(c)



(d)

❖ SSD를 이용한 얼굴 검출

- 영상으로부터 실시간으로 얼굴 검출을 하기 위해서, OpenCV와 딥러닝을 활용해 보겠음
- 이번에 사용할 딥러닝 모델은 2016년에 발표된 **SSD(Single Shot multibox Detector)**임[Liu16]
- SSD는 입력 영상에서 특정 객체의 클래스와 위치, 크기 정보를 실시간으로 추출할 수 있는
객체 검출 딥러닝 알고리즘임
- SSD 알고리즘은 원래 다수의 클래스 객체를 검출할 수 있지만,
OpenCV에서 제공하는 파일들은 오직 얼굴만을 검출하도록 학습됨

[Liu16] W. Liu et al. "SSD: Single shot multibox detector,"
in *Proc. European Conference on Computer Vision (ECCV)*, Oct. 2016, pp. 21-37.

❖ SSD를 이용한 얼굴 검출에서 사용할 파일

- OpenCV에서 제공하는 얼굴 검출을 위한 용도로 학습된 딥러닝 모델은 종류가 2가지임
 - ◆ ① 카페(Caffe) 프레임워크에서 학습된 파일
 - ◆ ② 텐서플로(TensorFlow)에서 학습된 파일
- 두 학습 모델은 비슷한 성능으로 동작하고, 어느 것을 사용해도 무방함

❖ SSD를 이용한 얼굴 검출에서 사용할 파일: ① 카페(Caffe)

- 얼굴 검출에서 사용된 네트워크 정보가 담겨 있는 파일

deploy.prototxt

- 얼굴 검출을 위해 미리 Caffe 프레임워크에서 학습된 모델 파일

res10_300x300_ssd_iter_140000_fp16.caffemodel

아래 링크에서 모델 파일을 다운로드 할 수 있음

https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20180205_fp16/res10_300x300_ssd_iter_140000_fp16.caffemodel

❖ SSD를 이용한 얼굴 검출에서 사용할 파일: ② 텐서플로(TensorFlow)

- 얼굴 검출에서 사용된 네트워크 정보가 담겨 있는 파일

opencv_face_detector.pbtxt

- 얼굴 검출을 위해 미리 텐서플로 프레임워크에서 학습된 모델 파일

opencv_face_detector_uint8.pb

아래 링크에서 모델 파일을 다운로드 할 수 있음

https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20180220_uint8/opencv_face_detector_uint8.pb

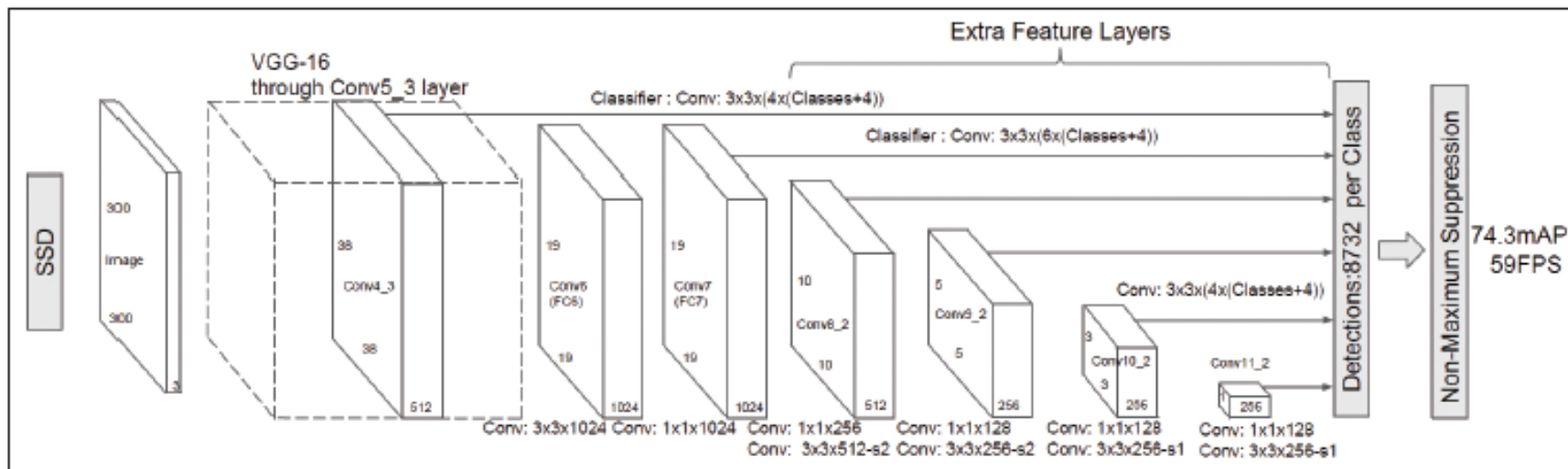
deploy.prototxt와 opencv_face_detector.pbtxt은 아래 링크에서 다운로드 할 수 있음

<https://github.com/sunkyoo/opencv4cvml/tree/master/python/ch16/dnnface>

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (1/14)

- 기본적인 SSD 네트워크 구조를 그림 16-14에 나타냄
- 이 구조에서 입력은 300×300 크기의 2차원 BGR 컬러 영상을 사용함
- 이 영상은 (104, 177, 123) 값을 이용하여 정규화한 후 사용함
- SSD 네트워크의 출력은 추출된 객체의 ID, 신뢰도, 사각형 위치 등의 정보를 담고 있음

▼ 그림 16-14 SSD 네트워크 구조



❖ SSD를 이용한 얼굴 검출 프로그램 구현 (2/14)

- 딥러닝을 이용한 얼굴 검출 예제 프로그램은 컴퓨터에 연결된 카메라로부터 들어오는 매 프레임(frame)마다 얼굴을 검출함
- 검출된 얼굴 위치에 사각형을 그리는 방식으로 만들어 보자
- 카메라로부터 매 프레임을 받아 오기 전에 먼저 cv2.dnn.Net 클래스 객체를 생성해야 함
- 카페에서 학습된 모델 파일과 구성 파일을 이용하여 cv2.dnn.Net 클래스 객체를 생성하려면 다음과 같이 코드를 작성함

```
model = 'res10_300x300_ssd_iter_140000_fp16.caffemodel'  
config = 'deploy.prototxt'  
  
net = cv2.dnn.readNet(model, config)
```

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (3/14)

- 만약 카페 대신 텐서플로에서 학습된 데이터 파일을 사용하고 싶다면 model과 config 변수 내용을 다음과 같이 변경하여 네트워크를 불러옴

```
model = 'opencv_face_detector_uint8.pb'  
config = 'opencv_face_detector.pbtxt'  
  
net = cv2.dnn.readNet(model, config)
```

- 앞 코드에서 사용한 각 모델 파일과 구성 파일은 프로그램 실행 폴더에 함께 있어야 함

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (4/14)

- net 객체가 정상적으로 생성되었다면 이제 카메라 장치를 열어서 매 프레임을 받아 옴
- 매 프레임을 numpy.ndarray 타입의 변수 frame에 저장하였다고 가정하고, frame 영상으로부터 네트워크 입력으로 전달할 블롭 객체를 생성해야 함
- SSD 기본 네트워크 구조에서 입력 영상 크기는 300×300임
- 영상 평균값으로 (104, 177, 123)을 사용하였으므로 blobFromImage() 함수 인자는 다음과 같이 설정함

```
blob = cv2.dnn.blobFromImage(frame, 1, (300, 300), (104, 177, 123))
```

- 이렇게 만들어진 blob 블롭 객체는 (1, 3, 300, 300) 형태의 4차원 행렬이며, Net 클래스 객체 net에 입력으로 사용됨

```
net.setInput(blob)  
detect = net.forward()
```

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (5/14)

- 앞 소스 코드에 의해 생성되는 detect 행렬은 (1, 1, N, 7) 크기의 4차원 행렬로 구성됨
- 처음 두개 차원 크기는 항상 1이고, 네 번째 차원 크기는 항상 7임
- 세 번째 차원 크기 N은 검출된 얼굴 후보 영역 개수를 의미함
- SSD 기반 얼굴 검출 네트워크는 최대 200개까지의 후보 영역을 검출함
- 출력으로 나온 N개의 얼굴 후보 영역 중에서 얼굴일 확률이 높은 영역만 최종 얼굴 영역으로 선택함
- detect 행렬에서의 1, 2차원 크기는 항상 1이므로 res 행렬의 3, 4차원만 이용하여 새로운 2차원 행렬을 구성하여 사용하는 것이 간편함

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (6/14)

- 다음은 4차원의 res 행렬을 2차원 행렬 detect로 변환하는 소스 코드임

```
detect = detect[0, 0, :, :]
```

- 앞 코드에 의해 생성되는 detect 행렬은 N×7 크기의 2차원 행렬이고, 타입은 numpy.ndarray임
- 이 행렬의 0번과 1번 열에는 항상 0과 1이 저장됨
- 2번째 열에는 얼굴 신뢰도(confidence)가 저장됨
- 신뢰도는 0부터 1 사이의 실수로 저장되며, 얼굴일 가능성이 높으면 1에 가까운 값이 저장됨

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (7/14)

- detect 행렬의 3번부터 6번 열에는 얼굴 영역 사각형 좌측 상단 꼭지점 좌표 (x_1, y_1) 과 우측 하단 꼭지점 좌표 (x_2, y_2) 가 차례대로 저장됨
- 이때 사각형 좌표는 영상의 가로와 세로 크기를 1로 정규화하여 저장된 좌표임
- 실제 픽셀 좌표는 영상의 가로 및 세로 크기를 곱해서 계산해야 함
- detect 행렬의 각 행은 얼굴 신뢰도가 높은 순서부터 내림차순 정렬되어 있음
- detect 행렬에서 매 행을 읽고, 이 중 얼굴 신뢰도가 특정 임계값보다 큰 경우에 대해서만 얼굴이라고 간주함

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (8/14)

- 코드 16-5에 나타난 dnnface 예제 프로그램 소스 코드는 카메라 입력 영상에서 얼굴을 검출하고, 그 위치를 녹색 사각형으로 표시함

코드 16-5 SSD 얼굴 검출 예제 프로그램 (dnnface.py)

```
1  import sys
2  import cv2
3
4  model = 'res10_300x300_ssd_iter_140000_fp16.caffemodel'
5  config = 'deploy.prototxt'
6  # model = 'opencv_face_detector_uint8.pb'
7  # config = 'opencv_face_detector.pbtxt'
8
9  net = cv2.dnn.readNet(model, config)
10
11  if net.empty():
12      print('Net open failed!')
13      sys.exit()
14
```

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (9/14)

코드 16-5 SSD 얼굴 검출 예제 프로그램 (dnnface.py)

```
15 cap = cv2.VideoCapture(0)
16
17 if not cap.isOpened():
18     print('Camera open failed!')
19     sys.exit()
20
21 while True:
22     _, frame = cap.read()
23     if frame is None:
24         break
25
26     blob = cv2.dnn.blobFromImage(frame, 1, (300, 300), (104, 177, 123))
27     net.setInput(blob)
28     detect = net.forward()
29
30     (h, w) = frame.shape[:2]
31     detect = detect[0, 0, :, :]
32
```

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (10/14)

코드 16-5 SSD 얼굴 검출 예제 프로그램 (dnnface.py)

```
33     for k in range(detect.shape[0]):
34         confidence = detect[k, 2]
35         if confidence < 0.5:
36             break
37
38         x1 = int(detect[k, 3] * w)
39         y1 = int(detect[k, 4] * h)
40         x2 = int(detect[k, 5] * w)
41         y2 = int(detect[k, 6] * h)
42
43         cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0))
44
45         label = 'Face: %4.3f' % confidence
46         cv2.putText(frame, label, (x1, y1 - 1), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 1, cv2.LINE_AA)
47
48     cv2.imshow('frame', frame)
49
50     if cv2.waitKey(10) == 27:
51         break
52
53 cv2.destroyAllWindows()
```

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (11/14)

● dnnface.py 소스 코드 설명

- 4~5행 사용할 모델 파일과 구성 파일 이름을 각각 `model`과 `config` 변수에 저장합니다.
- 6~7행 만약 텐서플로에서 학습된 모델 파일과 구성 파일을 사용하려면 4~5행을 주석으로 변경하고, 6~7행의 주석을 해제합니다.
- 9~13행 모델 파일과 구성 파일을 이용하여 `net` 객체를 생성합니다.
 `net` 객체 생성에 실패하면 에러 메시지를 출력하고 프로그램을 종료합니다.
- 15~19행 컴퓨터에 연결된 기본 카메라 장치를 열어 `cap`에 저장합니다.
 카메라 열기에 실패하면 에러 메시지를 출력하고 프로그램을 종료합니다.
- 22~24행 카메라의 매 프레임을 `frame` 변수에 저장합니다.
 `frame`을 제대로 받아 오지 못하면 프로그램을 종료합니다.
- 26~28행 `frame` 영상을 이용하여 네트워크 입력 블록을 설정하고, 네트워크 실행 결과를 `detect` 행렬에 저장합니다.

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (12/14)

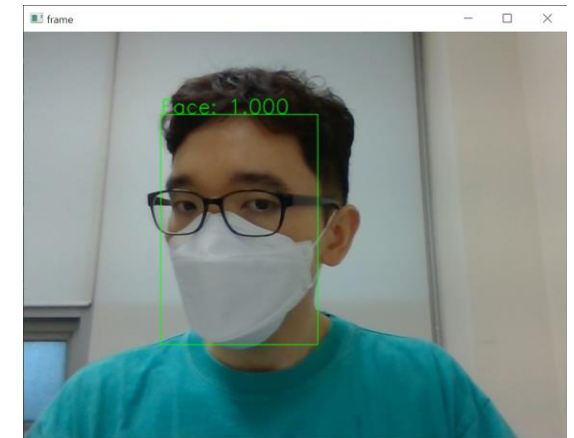
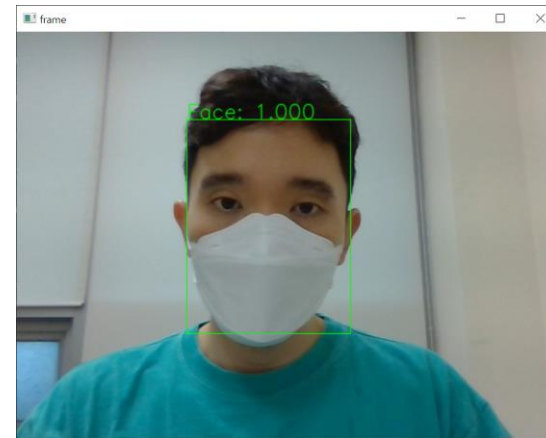
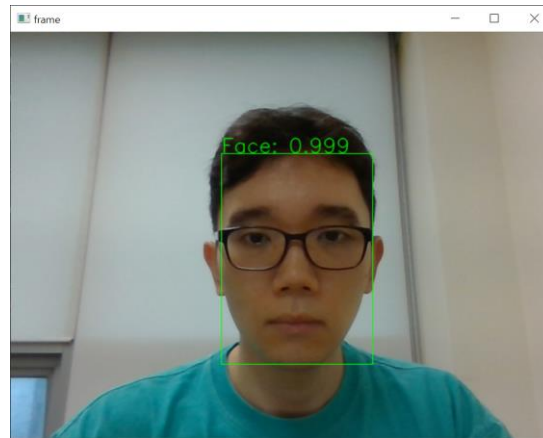
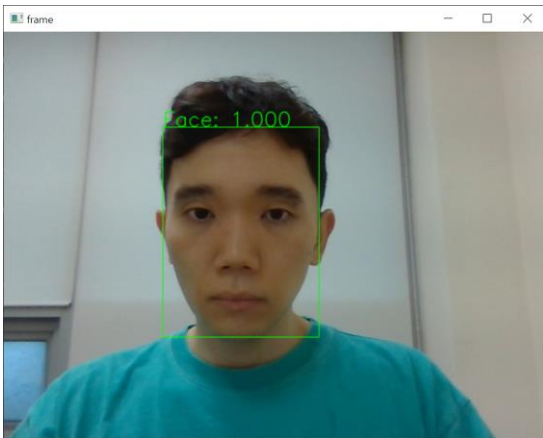
- dnnface.py 소스 코드 설명

- 35~36행 결과 행렬에서 신뢰도 값이 0.5보다 작으면 무시합니다.
- 38~41행 얼굴 검출 사각형 영역의 좌측 상단 좌표 (x1, y1)과 우측 하단 좌표 (x2, y2)를 계산합니다.
- 43~46행 frame 영상에서 얼굴 검출 영역에 녹색 사각형을 그리고, 얼굴 신뢰도를 출력합니다.

❖ SSD를 이용한 얼굴 검출 프로그램 구현 (13/14)

- 코드 16-5의 dnnface 프로그램을 실행하여 얼굴을 검출한 결과 화면의 예를 그림 16-15에 나타냄
- 얼굴 영역을 정확하게 검출하여 녹색 사각형을 그리고, 녹색 사각형 위에 얼굴 검출 신뢰도가 함께 출력된 것을 볼 수 있음

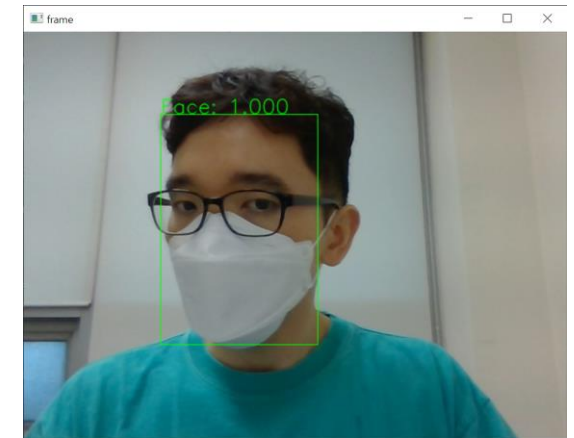
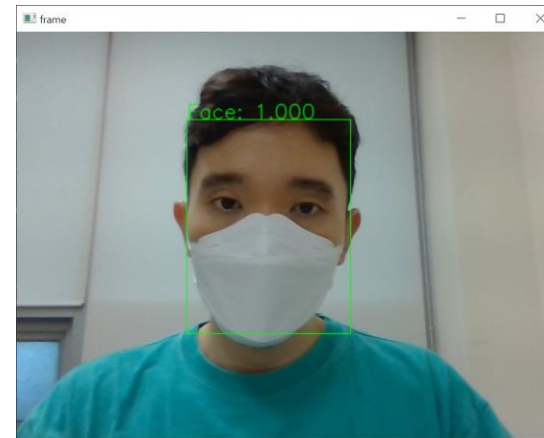
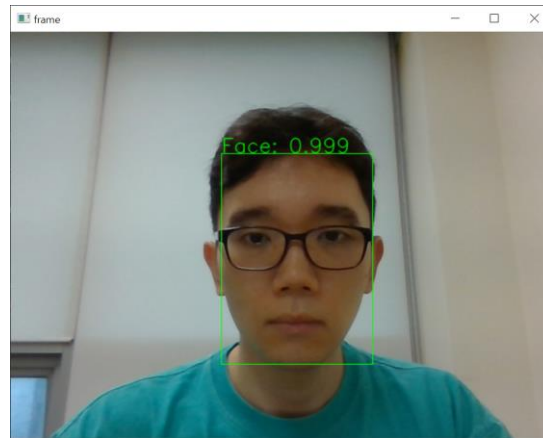
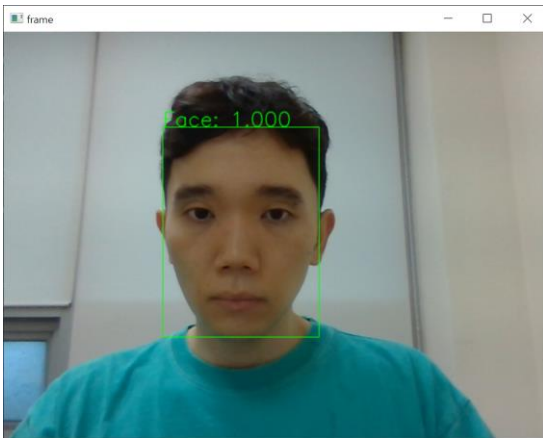
▼ 그림 16-15 SSD 얼굴 검출 예제 프로그램 실행 결과



❖ SSD를 이용한 얼굴 검출 프로그램 구현 (14/14)

- 캐스케이드 분류기 기반의 얼굴 검출 방법은 정면 얼굴이 아니면 얼굴 검출에 실패하는 경우가 많음
- SSD 기반의 얼굴 검출은 얼굴 일부가 가려지거나 얼굴 옆모습이 입력으로 들어가도 안정적으로 얼굴 영역을 검출하는 것을 확인할 수 있음
- 얼굴 검출 속도도 캐스케이드 분류기 기반의 방법보다 SSD 기반의 얼굴 검출이 더 빠르게 동작함

▼ 그림 16-15 SSD 얼굴 검출 예제 프로그램 실행 결과



THANK YOU!

Q & A

- Name: 권범
- Office: 동양미래대학교 2호관 704호 (02-2610-5238)
- E-mail: bkwon@dongyang.ac.kr
- Homepage: <https://sites.google.com/view/beomkwon/home>