

Convolutional Neural Networks

김현우
시각 및 학습 연구실

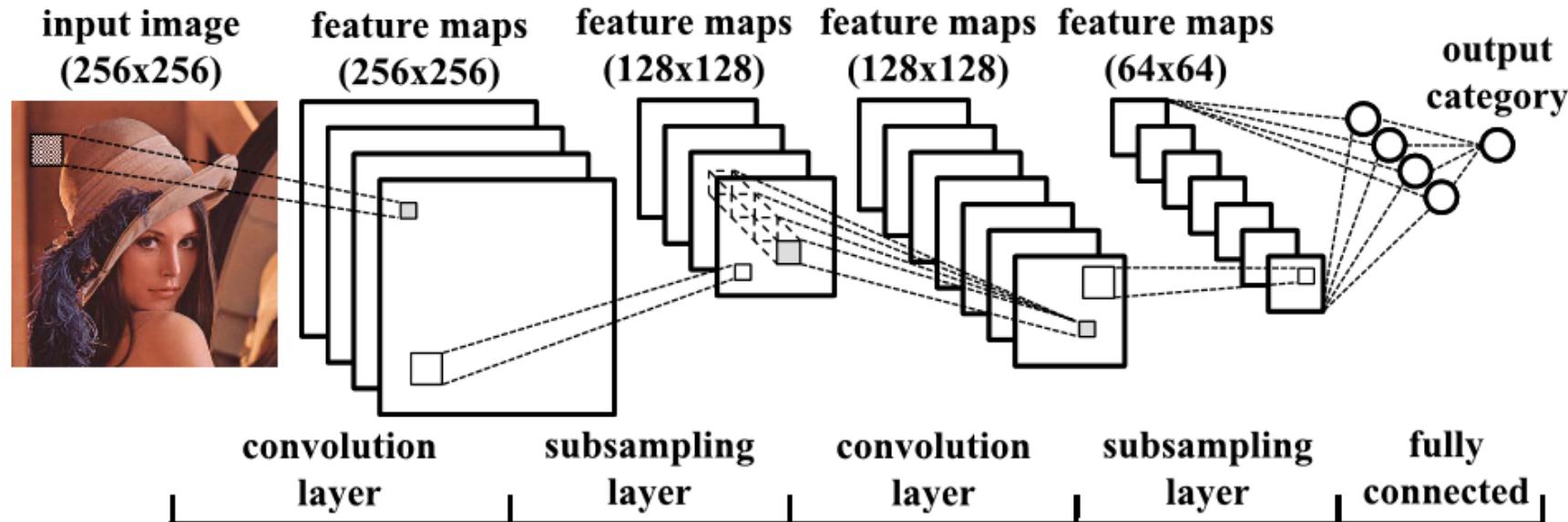


공지사항

- 질문을 비밀글로 올리시면 답변해드리지 않아요.
- 오늘 실습은 다음 파일에 들어 있는 연습 문제를
수업시간에 완성하는 것으로 평가할 예정이에요.
 - Convolutional_Neural_Networks_PyTorch.ipynb
- 과제는 다음 파일에 있는 연습 문제를 완성해서 ETL에 제출하시면 돼요.
 - 03_[BOTH]_CIFAR10_CNN_PyTorch.ipynb

목차

- 실습 환경 세팅
- Data loader
 - notMNIST 데이터셋
- CNN에 기반한 이미지 분류(image classification)

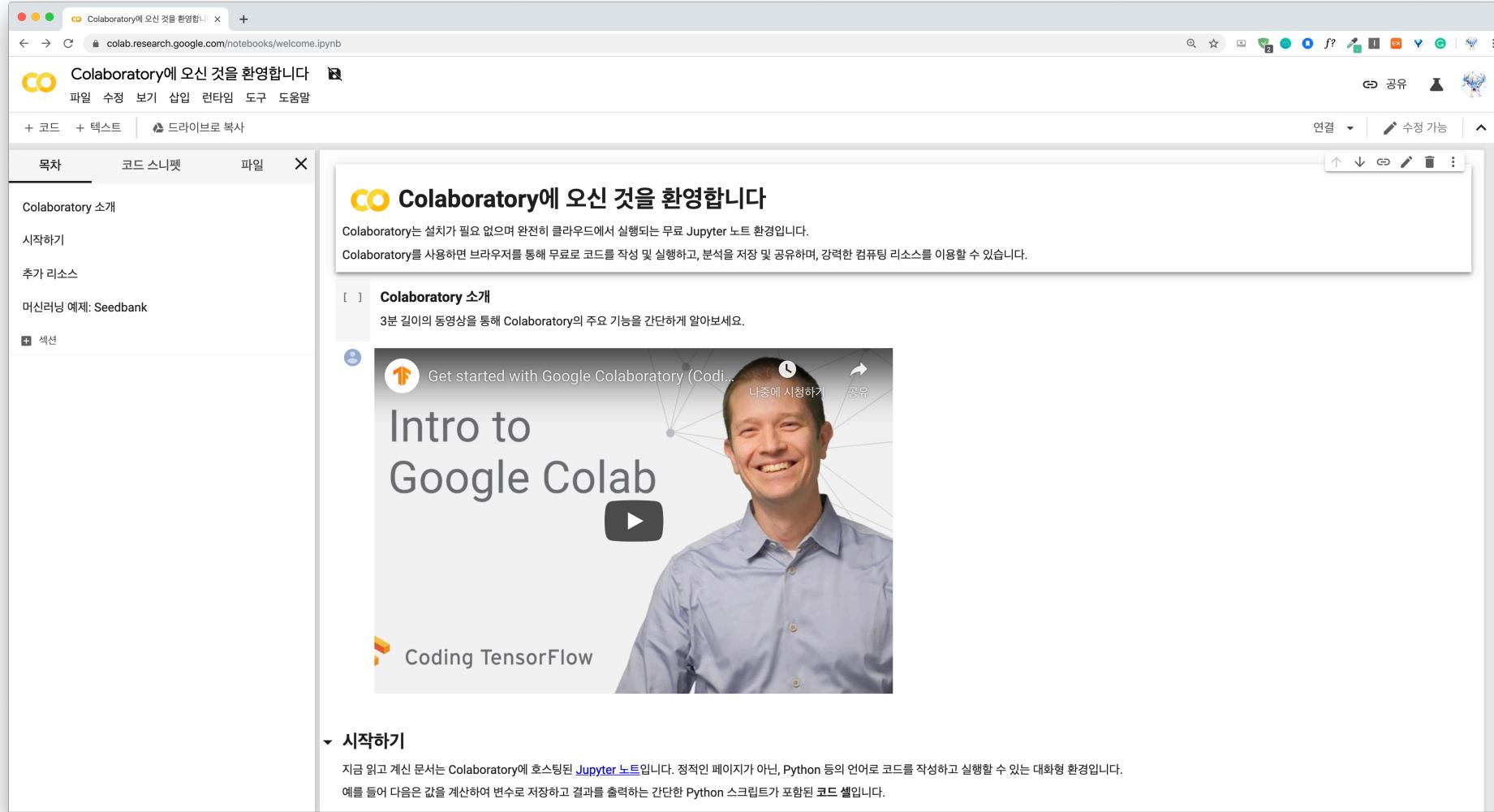


Colab 사용자를 위한 실습 환경 세팅



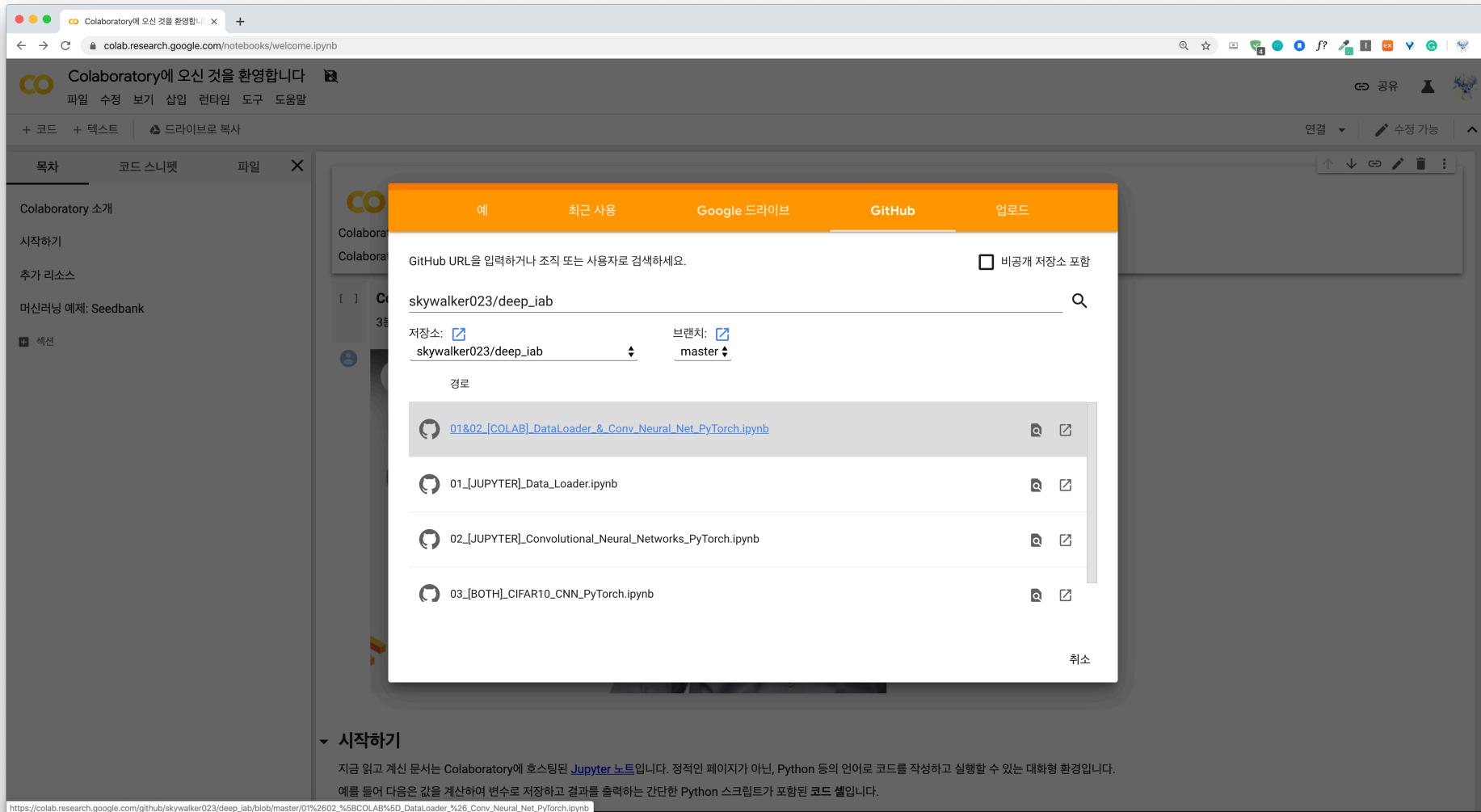
Colab 실습 환경 세팅

<https://colab.research.google.com>



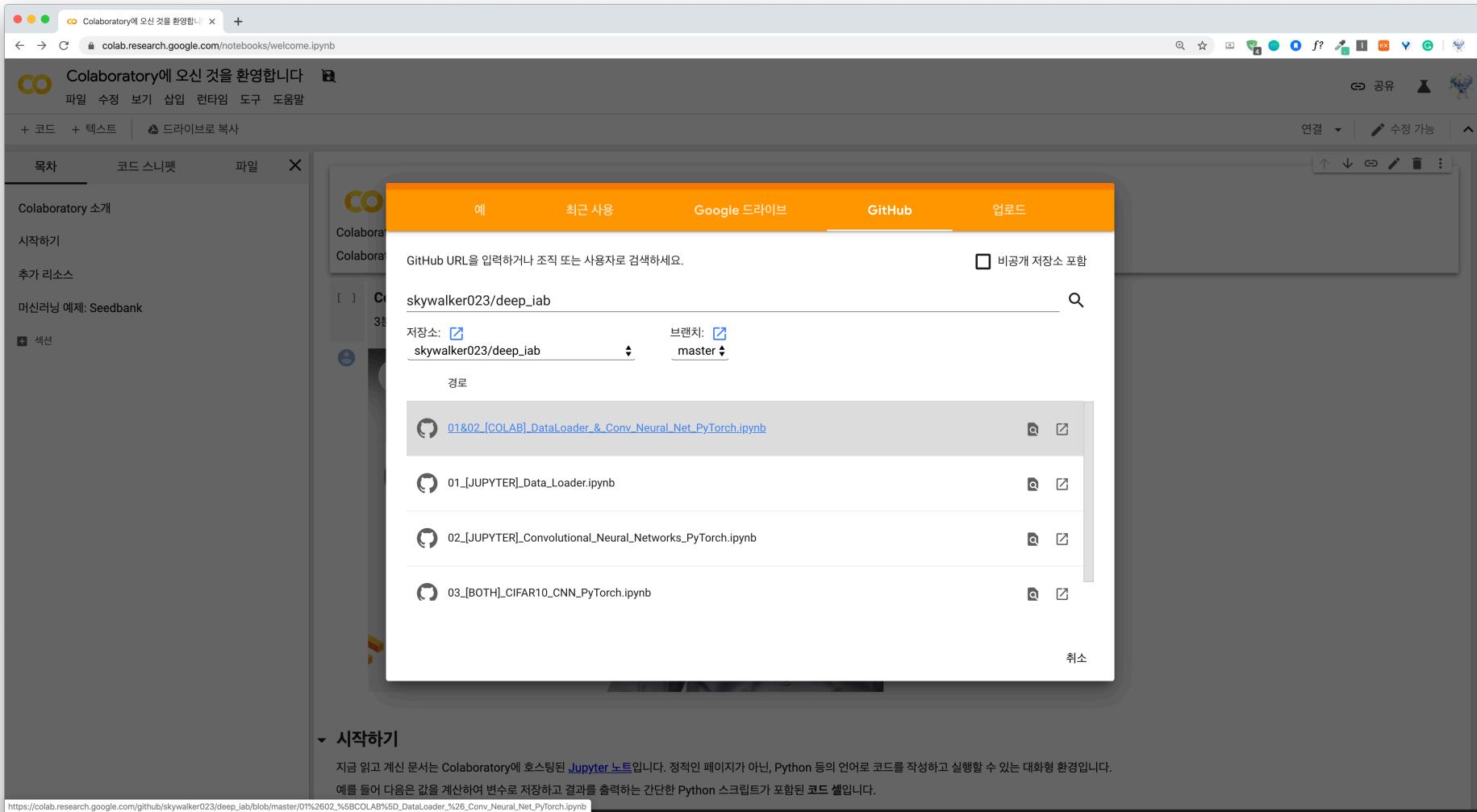
Colab 실습 환경 세팅

파일 → 노트 열기 → GitHub탭 → skywalker023/deep_iab 입력



Colab 실습 환경 세팅

01&02_[COLAB]_DataLoader_&_Conv… 노트북 파일 열기



Colab 실습 환경 세팅

“드라이브로 복사” 클릭

The screenshot shows a Google Colab notebook titled "01&02_[COLAB]_DataLoader_&_Conv_Neural_Net_PyTorch.ipynb". The interface includes a toolbar at the top with various icons for file operations, search, and help. Below the toolbar, there are tabs for "코드" (Code) and "드라이브로 복사" (Copy to Drive). The main content area displays the following text:

딥러닝 모델을 학습시키기 위해 준비되어야 할 4가지 요소

1. 데이터
2. 모델
3. 손실 함수(objective function, loss function 등으로 불려요): 정답과 모델의 예측값을 어떤 식으로 비교할지 결정해주는 함수
4. optimizer: gradient descent를 해줄 애. 즉, 모델의 파라미터를 어느 방향으로 조금 수정할지 결정하고 수정해주는 함수

Part 1 : Data Loader

```
[ ] !pip install scipy==0.19.1

[ ] import numpy as np
import os, sys
import tarfile
import matplotlib.pyplot as plt

from IPython.display import display, Image
from scipy import ndimage
from six.moves import urllib.request as urlretrieve
from six.moves import cPickle as pickle
from tqdm import tqdm

# Config the matplotlib backend as plotting inline in Ipython
%matplotlib inline
```

데이터 다운로드 하기

```
[ ] url = 'https://commondatastorage.googleapis.com/books1000/'

data_root = './data'
if not os.path.exists(data_root):
    os.mkdir(data_root)
```

anaconda & jupyter notebook
사용자를 위한 실습 환경 세팅



실습 환경 세팅 (anaconda & jupyter notebook 사용시)

```
$ conda create -n deep_iab python=3.6  
$ conda activate deep_iab  
$ git clone git@github.com:skywalker023/deep\_iab.git  
$ cd deep_iab/  
$ pip install -r requirements.txt  
$ conda deactivate && conda activate deep_iab  
$ jupyter notebook
```

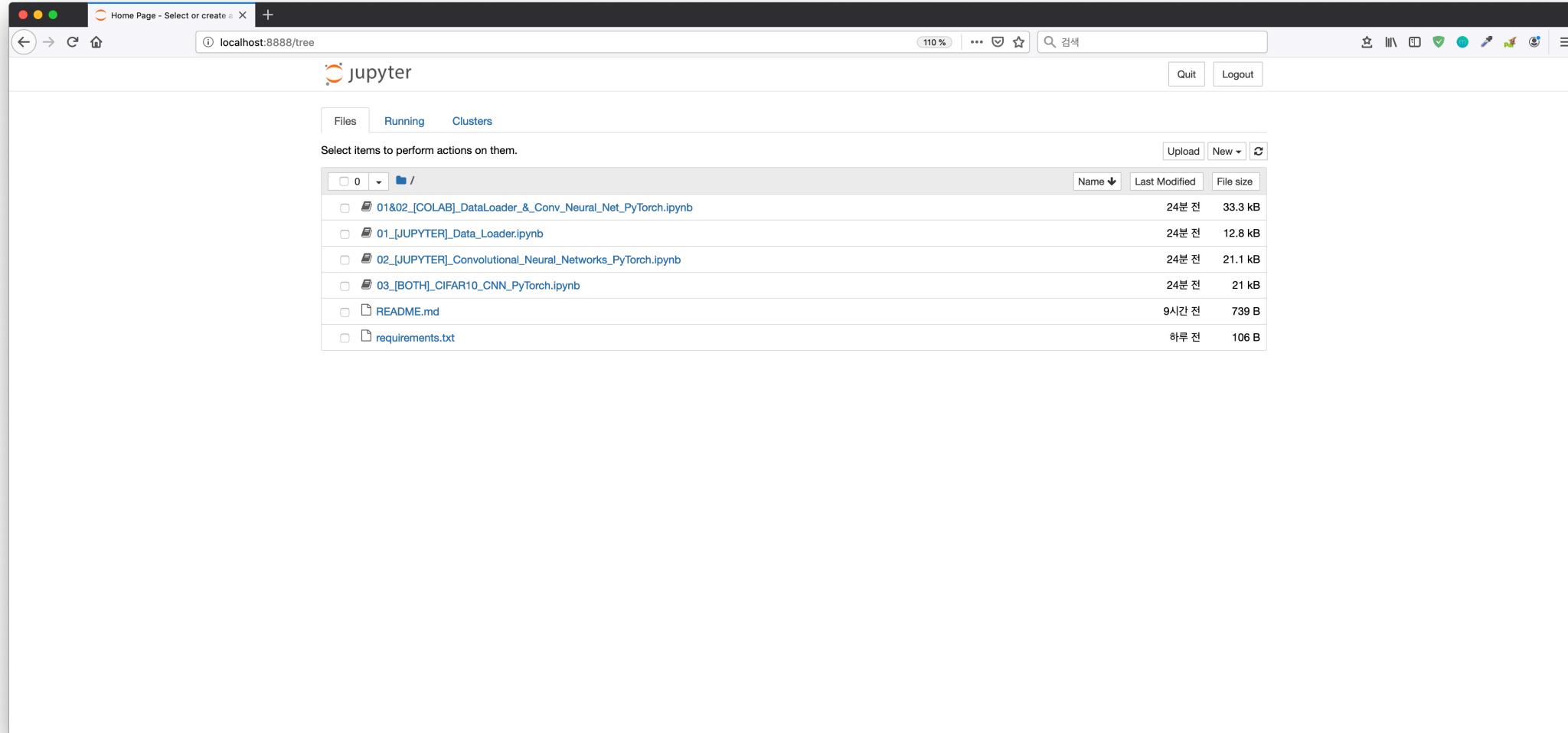


실습 환경 세팅 (anaconda & jupyter notebook 사용시)

```
$ conda create -n deep_iab python=3.6   ← python 3.6 버전의 가상환경 만들고 거기에 deep_iab라고 이름 붙이기  
$ conda activate deep_iab   ← deep_iab 가상환경 활성화  
$ git clone git@github.com:skywalker023/deep\_iab.git   ← 이 저장소에서 파일 복사해서 가져오기  
$ cd deep_iab/   ← deep_iab 디렉토리로 이동하기  
$ pip install -r requirements.txt   ← pip이라는 툴로 requirements.txt 에 들어있는 패키지들 설치하기  
$ conda deactivate && conda activate deep_iab   ← 가상환경 한 번 껐다 다시 켜주기 (설정 적용 위해)  
$ jupyter notebook
```



실습 환경 세팅 (anaconda & jupyter notebook 사용시)



딥러닝



딥러닝 모델 학습을 위해 필요한 4가지 요소

1. 모델에 들어가도록 알맞게 (전)처리된 데이터
2. 모델
3. 손실함수 (loss function, objective function)
4. Optimizer :
 - gradient descent를 해주는 애.
 - = 모델의 파라미터를 어느 방향으로 수정할지 결정하고 수정해주는 애
 - 예: Adam, SGD(Stochastic Gradient Descent), RMSProp 등

딥러닝 모델 학습을 위해 필요한 4가지 요소 (현실)

1. 모델에 들어가도록 알맞게 (전)처리된 데이터

2. 모델

3. 손실함수 (loss function, objective function)

4. Optimizer :

- gradient descent를 해주는 애. 모델의 파라미터를 어느 방향으로 수정할지 결정하고 수정 해주는 애
- 예: Adam, SGD(Stochastic Gradient Descent), RMSProp 등

딥러닝 모델을 위해 데이터를 전처리하는 절차

1. 데이터 파일 존재

2. 불러와서 numpy array 형식으로 변환

3. 필요한 전처리를 시행

4. Numpy array를 torch.Tensor() 형태로 변환
5. 그걸 data loader에 넣어주기.
 - Data loader란?
트레이닝 데이터를 모델에 몇 개씩 나눠서(=mini-batch) 전달해주는 애.

Dataset: notMNIST

이미지 분류 과제

- 10개의 클래스: 각기 다른 폰트로 이루어진 A ~ J
- 28x28 사이즈 이미지들로 구성



Data loader

데이터셋을 다운로드 합니다

```
In [*]: url = 'https://commondatastorage.googleapis.com/books1000/'
```

```
data_root = './data'  
if not os.path.exists(data_root):  
    os.mkdir(data_root)  
  
def dataset_downloader(filename):  
    """데이터셋 파일이 없으면 다운로드 합니다."""  
    dest_dir = os.path.join(data_root, filename)  
    if not os.path.exists(dest_dir):  
        print('다운로드 시도 중 : ', filename)  
        filename, _ = urlretrieve(url + filename, dest_dir)  
        print(filename, ' 다운로드 완료!')  
    else:  
        print(dest_dir, ' 이미 있습니다.')  
  
    return dest_dir
```

```
train_filename = dataset_downloader('notMNIST_large.tar.gz')  
test_filename = dataset_downloader('notMNIST_small.tar.gz')
```

‘url + filename’ 링크로부터 다운로드를 하고
‘dest_dir’에 저장을 합니다.

Data loader

다운로드 한 파일의 압축을 풁니다

```
In [3]: num_classes = 10
np.random.seed(1000)

def data_extract(filename):
    root = os.path.splitext(os.path.splitext(filename)[0])[0] # remove .tar.gz

    if os.path.isdir(root):
        print('{} 이미 있습니다 - {} 는 추출을 건너뜁니다.'.format(root, filename))
    else:
        print('{}에서 데이터를 추출합니다.'.format(root))
        tar = tarfile.open(filename)
        sys.stdout.flush()
        tar.extractall(data_root)
        tar.close()

    data_folders = [
        os.path.join(root, d) for d in sorted(os.listdir(root))
        if os.path.isdir(os.path.join(root, d))]

    if len(data_folders) != num_classes:
        raise Exception('{} folders 기대했는데, {} 개가 있네요.'.format(num_classes, len(data_folders)))

    print(data_folders)
```

Untar *.tar.gz file.

Data loader

각 글자 데이터들을 각각 pickle 파일로 저장합니다

```
In [4]: image_size = 28 # Pixel width and height.  
pixel_depth = 255.0 # Number of levels per pixel.
```

```
def load_letter(folder):  
    """한 글자 클래스 데이터를 로드합니다."""  
    image_files = os.listdir(folder)  
    dataset = np.ndarray(shape=(len(image_files), image_size, image_size), dtype=np.float32)  
  
    num_images = 0  
    for image in image_files:  
        image_file = os.path.join(folder, image)  
        try:  
            image_data = (ndimage.imread(image_file).astype(float) - pixel_depth / 2) / pixel_depth  
            if image_data.shape != (image_size, image_size):  
                raise Exception('이미지가 이상한 크기인데요?: {}'.format(str(image_data.shape)))  
            dataset[num_images, :, :] = image_data  
            num_images = num_images + 1  
        except IOError as e:  
            print('{} - skip'.format(e))  
  
    dataset = dataset[0:num_images, :, :]  
    print('전체 데이터셋 모양은 다음과 같습니다:', dataset.shape)  
  
    return dataset
```

픽셀값을 0~1로 변경해주기
normalization의 일종

Data loader

각 글자 데이터들을 각각 pickle 파일로 저장합니다

```
def make_pickle(data_folders):
    dataset_names = []
    for folder in data_folders:
        set_filename = folder + '.pickle'
        dataset_names.append(set_filename)
        if os.path.exists(set_filename):
            print('{} 이미 있습니다 - pickling을 건너뜁니다.'.format(set_filename))
            continue
        print('Pickling {}'.format(set_filename))
        dataset = load_letter(folder)
        with open(set_filename, 'wb') as f:
            pickle.dump(dataset, f)

    return dataset_names
```

각 글자 데이터를
Pickle로 저장하기

Pickle 모듈?

list, dictionary, class와 같은 파이썬 객체들을 그대로 저장해줍니다
그대로 다시 불러와도 파이썬 자료형이 살아있습니다. 매우 편리!



Data loader

각 글자 클래스마다 개수가 비슷한지 점검합니다 (불균형 싫어요)

```
In [6]: print("==== 트레이닝셋을 위한 데이터들 ====")
for i in range(len(train_datasets)):
    set_filename = train_datasets[i]
    with open(set_filename, 'rb') as f:
        dataset = pickle.load(f)
    print("글자 {} 에 대한 트레이닝 데이터 개수는 {} 개입니다.".format(os.path.basename(set_filename)[0], len(dataset)))

print("\n==== 테스트셋을 위한 데이터들 ====")
for i in range(len(test_datasets)):
    set_filename = test_datasets[i]
    with open(set_filename, 'rb') as f:
        dataset = pickle.load(f)
    print("글자 {} 에 대한 테스트 데이터 개수는 {} 개입니다.".format(os.path.basename(set_filename)[0], len(dataset)))
```

Data loader

각 글자 별로 나누어져있던 거를 2개의 트레이닝셋, 테스트셋으로 통칩니다

```
In [7]: def merge_datasets(pickle_files, dataset_size):
    num_classes = len(pickle_files)
    dataset = np.ndarray((dataset_size, image_size, image_size), dtype=np.float32)
    labels = np.ndarray(dataset_size, dtype=np.int32)
    tsize_per_class = dataset_size // num_classes

    start_t = 0
    end_t = tsize_per_class
    for label, pickle_file in enumerate(pickle_files):
        with open(pickle_file, 'rb') as f:
            letter_set = pickle.load(f)
            np.random.shuffle(letter_set)

            letter = letter_set[0:tsize_per_class, :, :]
            dataset[start_t:end_t, :, :] = letter
            labels[start_t:end_t] = label
            start_t += tsize_per_class
            end_t += tsize_per_class

    return dataset, labels

train_size = 200000
test_size = 10000
```

Data loader

데이터셋을 섞어줍니다.

라벨들이 잘 섞여 있어야 트레이닝셋, 테스트셋에 골고루 들어갑니다.

```
In [9]: def shuffle(dataset, labels):
    permutation = np.random.permutation(labels.shape[0])
    shuffled_dataset = dataset[permutation,:,:]
    shuffled_labels = labels[permutation]
    return shuffled_dataset, shuffled_labels

train_dataset, train_labels = shuffle(train_dataset, train_labels)
test_dataset, test_labels = shuffle(test_dataset, test_labels)
```

Data loader

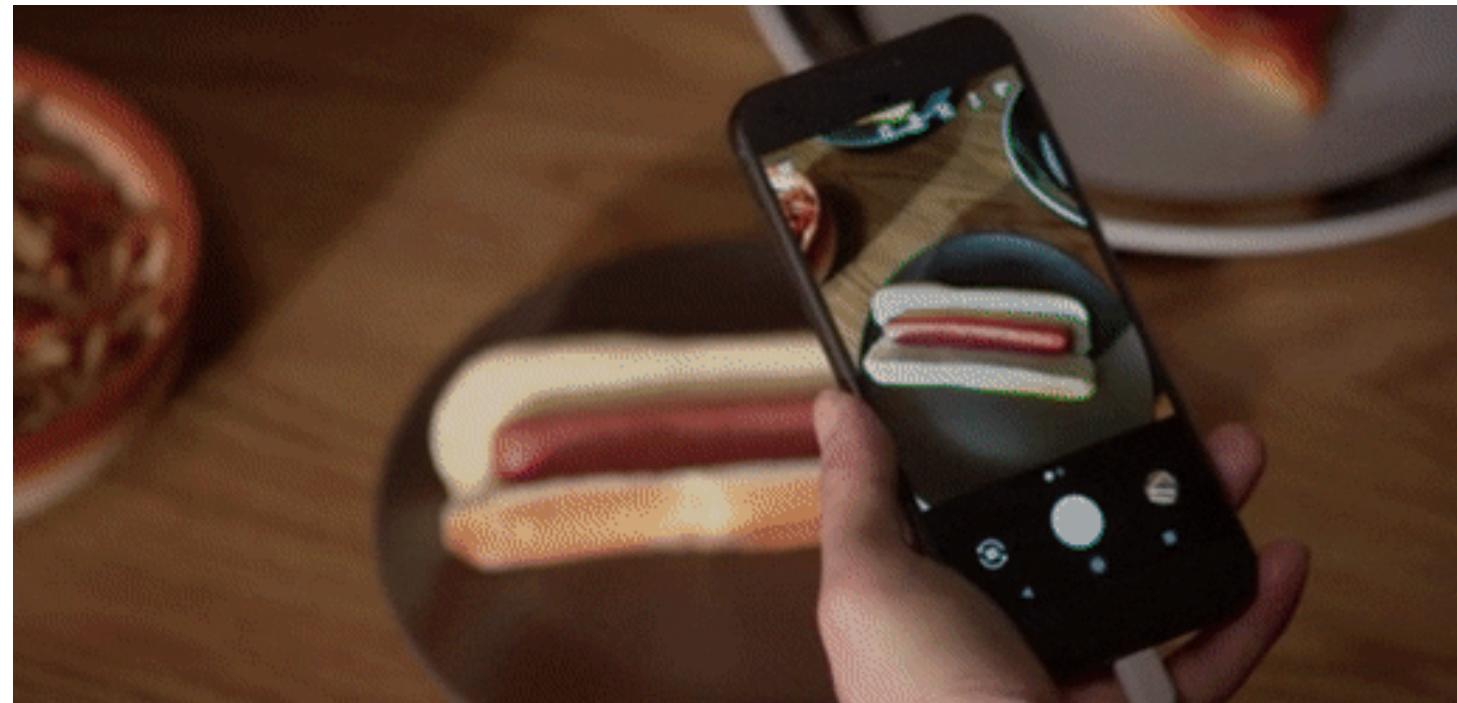
데이터를 이제 파일로 저장합니다

```
In [10]: pickle_file = os.path.join(data_root, 'notMNIST.pickle')

f = open(pickle_file, 'wb')
save = {
    'train_dataset': train_dataset[:50000],
    'train_labels': train_labels[:50000],
    'test_dataset': test_dataset[:5000],
    'test_labels': test_labels[:5000],
}
pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
f.close()
```

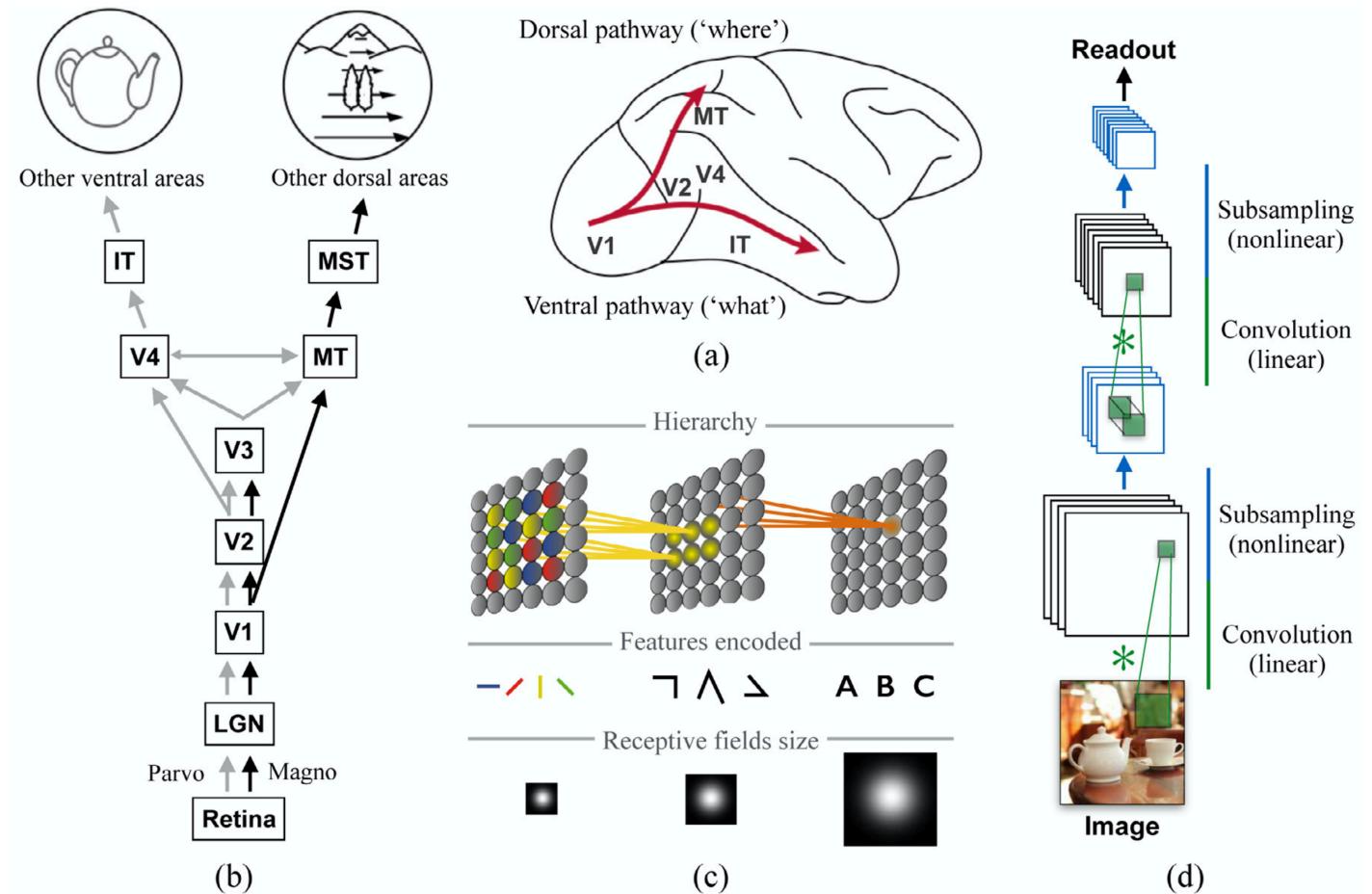
CNN based Image Classification

드라마 “실리콘 밸리” 중
Hotdog, Not hotdog



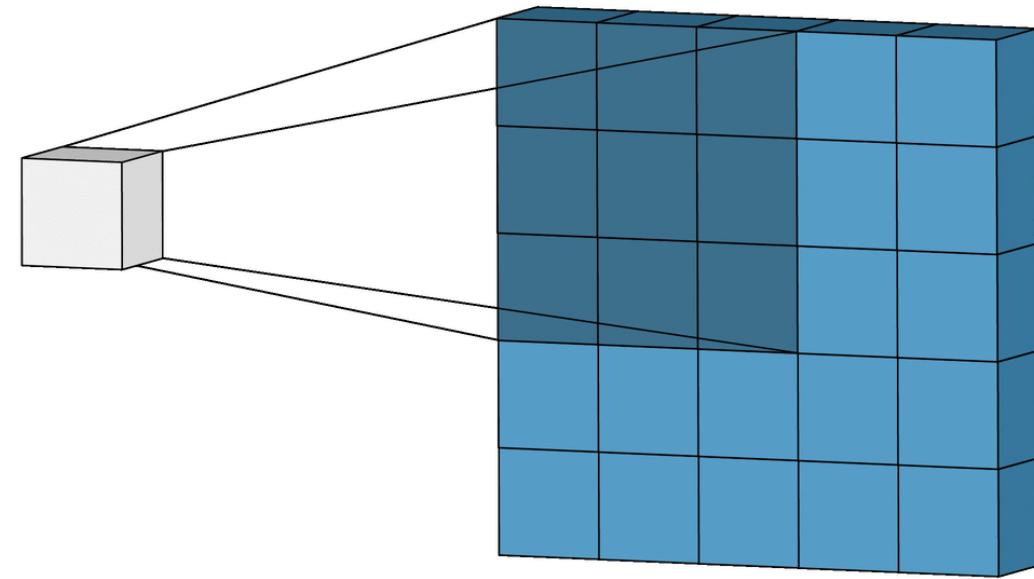
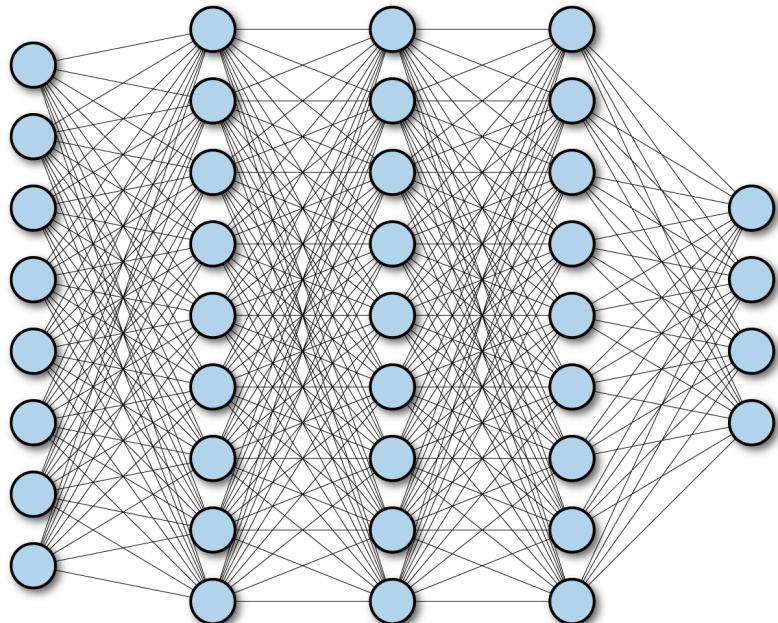
Why Convolution?

우리 사람도 이렇게 시각 정보를 convolution으로 처리합니다



Why Convolution?

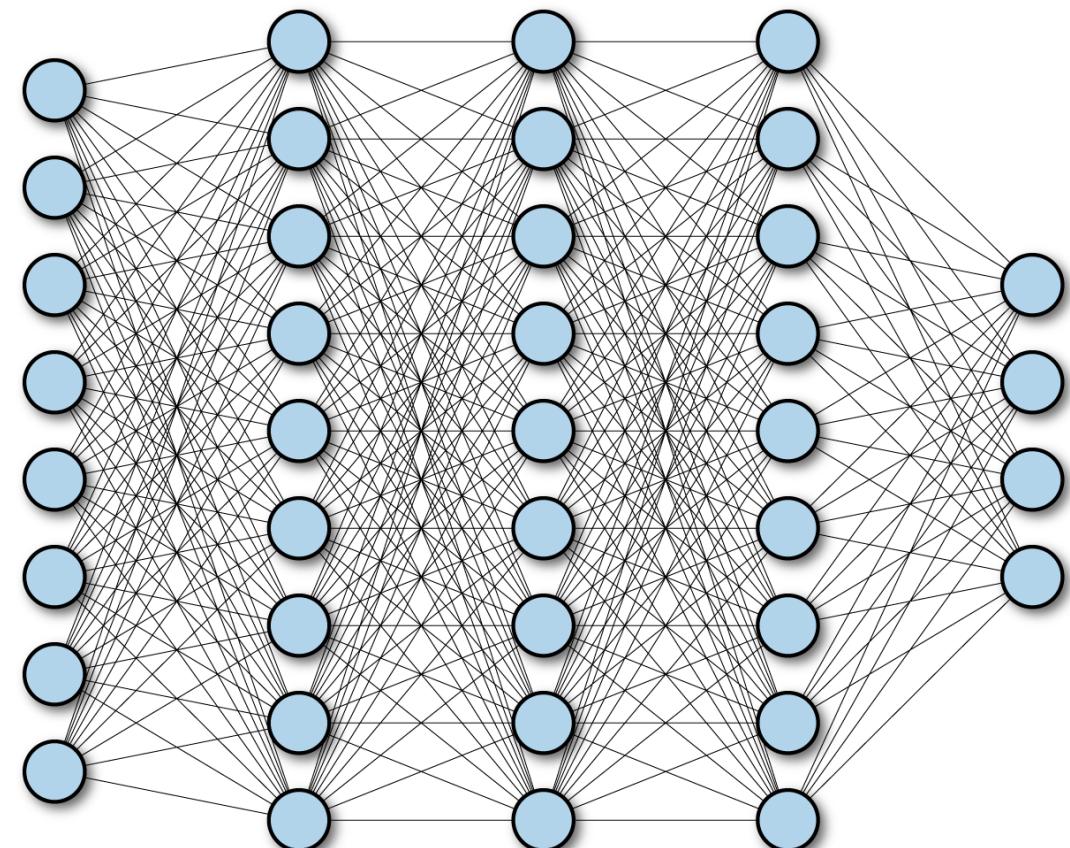
공간 정보를 같이 반영할 수 있습니다.



Why Convolution?

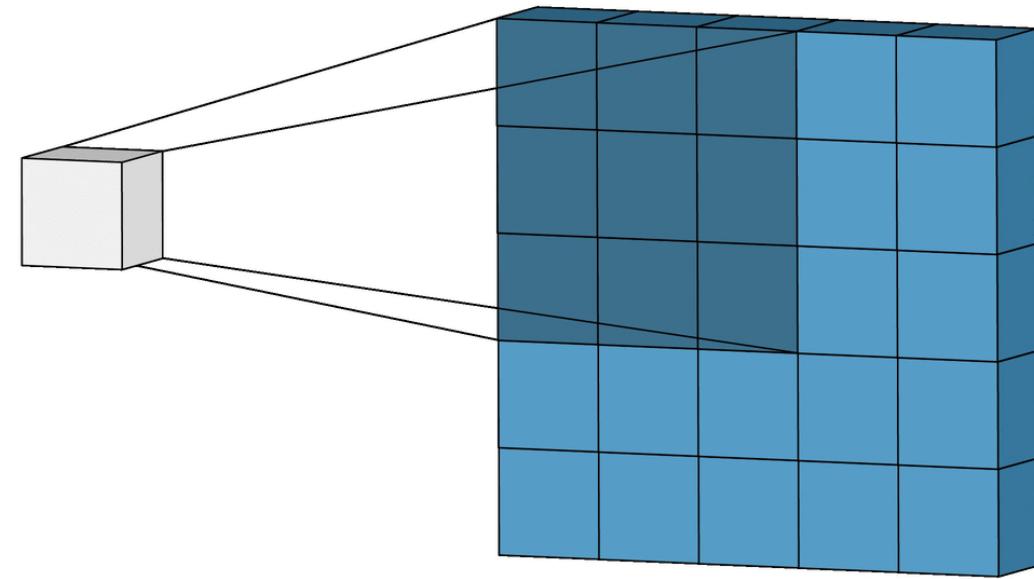
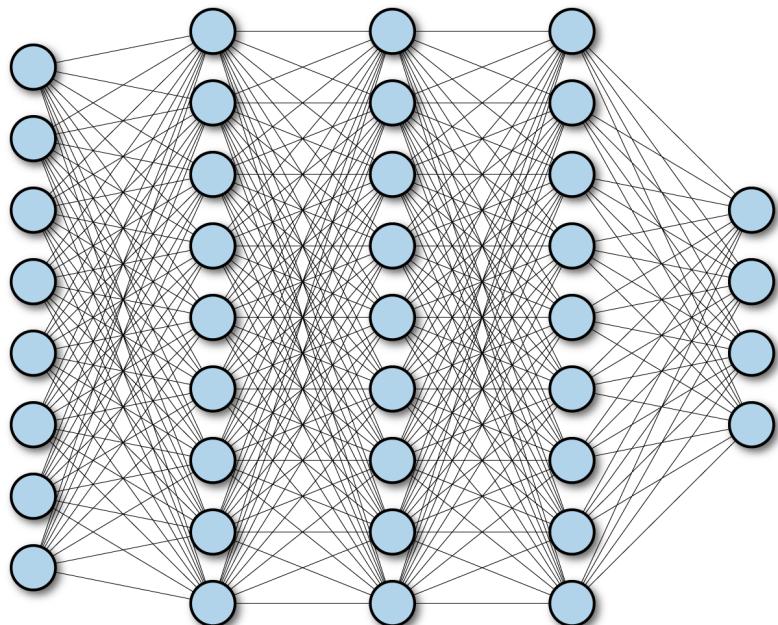
일반 fully connected neural net은 이미지 학습을 잘 못합니다.
이유: fully connected라서.

예를 들어,
28x28짜리 이미지가 주어진다고 하면
Input layer 1층의 weight만 해도
 $28 \times 28 \times 3 = 2,352$ 개!

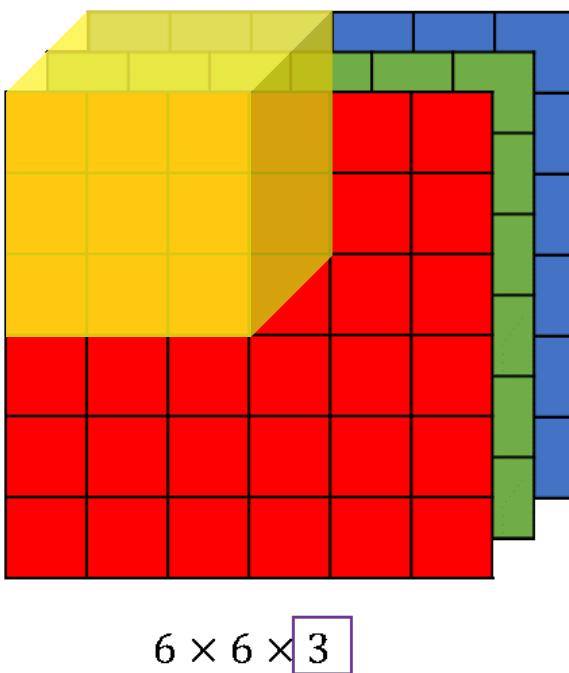


Why Convolution?

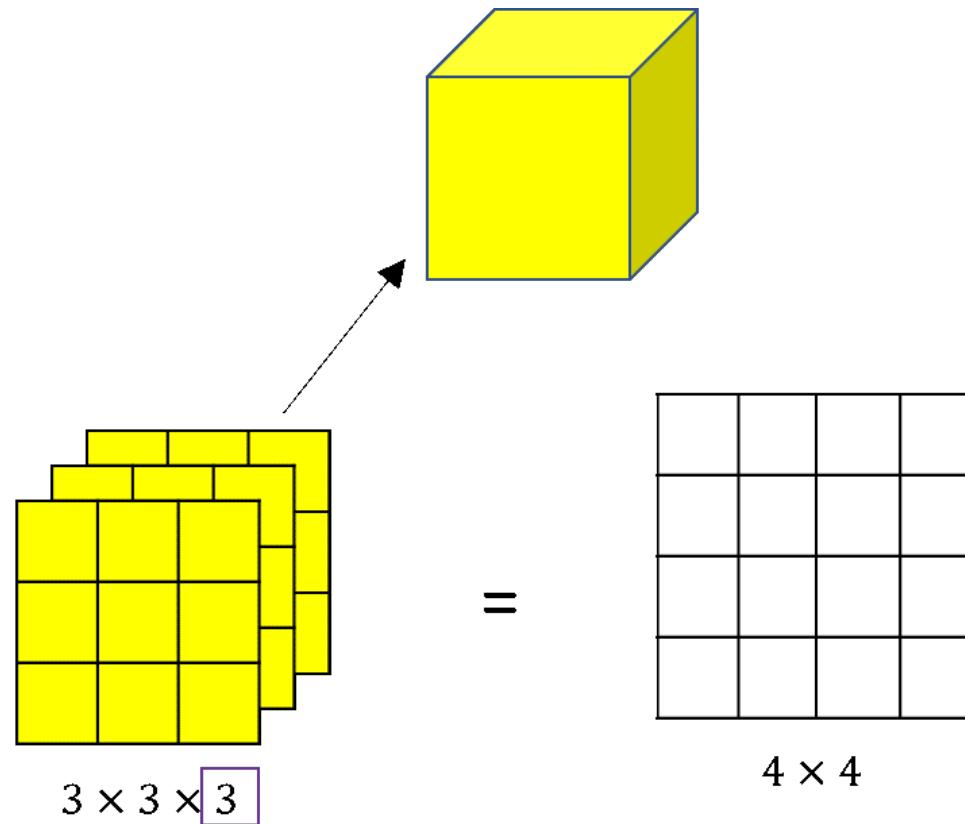
파라미터 개수를 확 줄일 수 있습니다



Convolution over RGB images

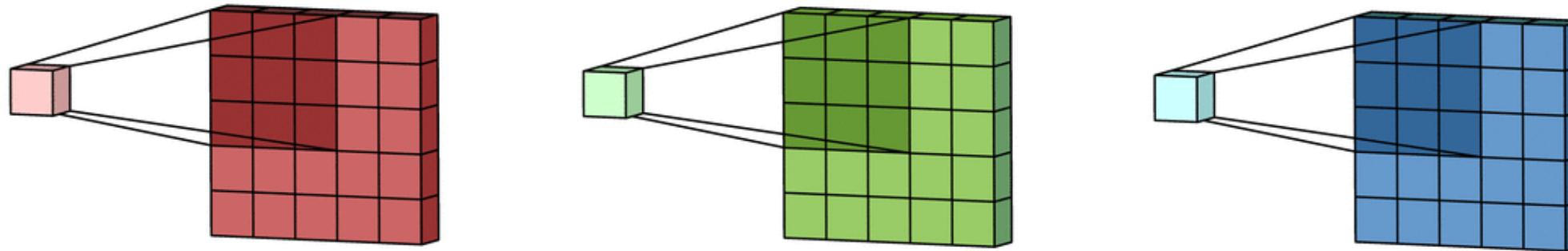


*



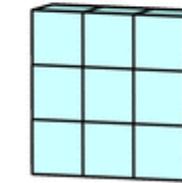
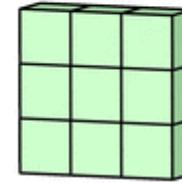
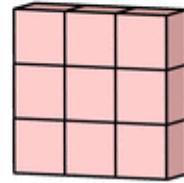
Convolution over RGB images

각 채널에 convolution을 합니다



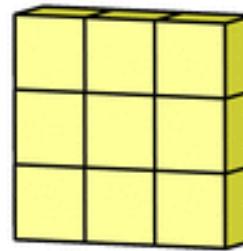
Convolution over RGB images

하나로 더해줍니다

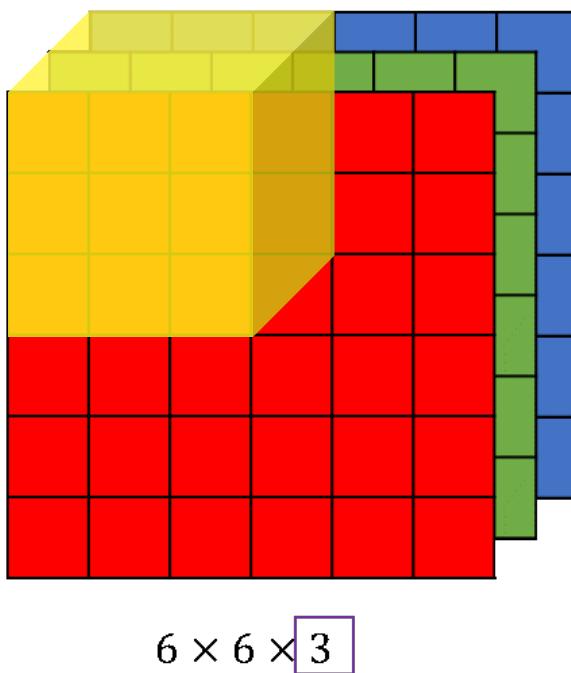


Convolution over RGB images

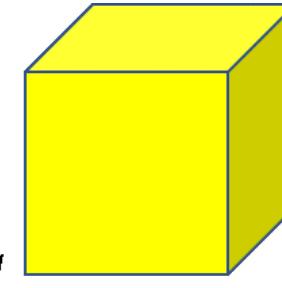
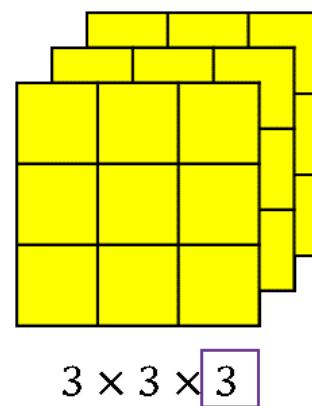
Bias도 있습니다



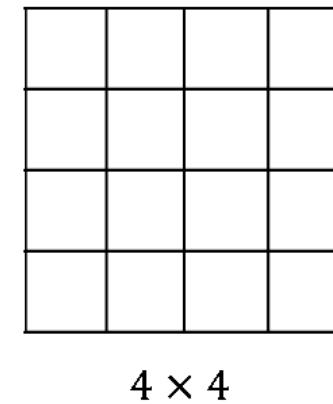
Convolution over RGB images



*



=



Convolution over RGB images

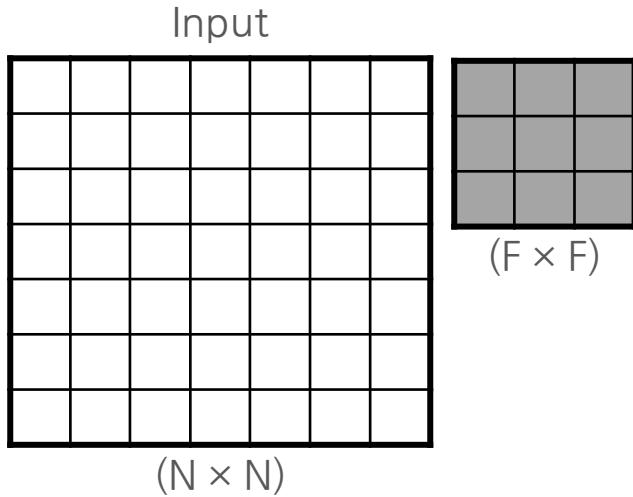
1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Convolution 결과 크기 계산하기



Input size : $N \times N \times \text{channel}$
Filter size : $F \times F$
Padding : p per side
Stride : s

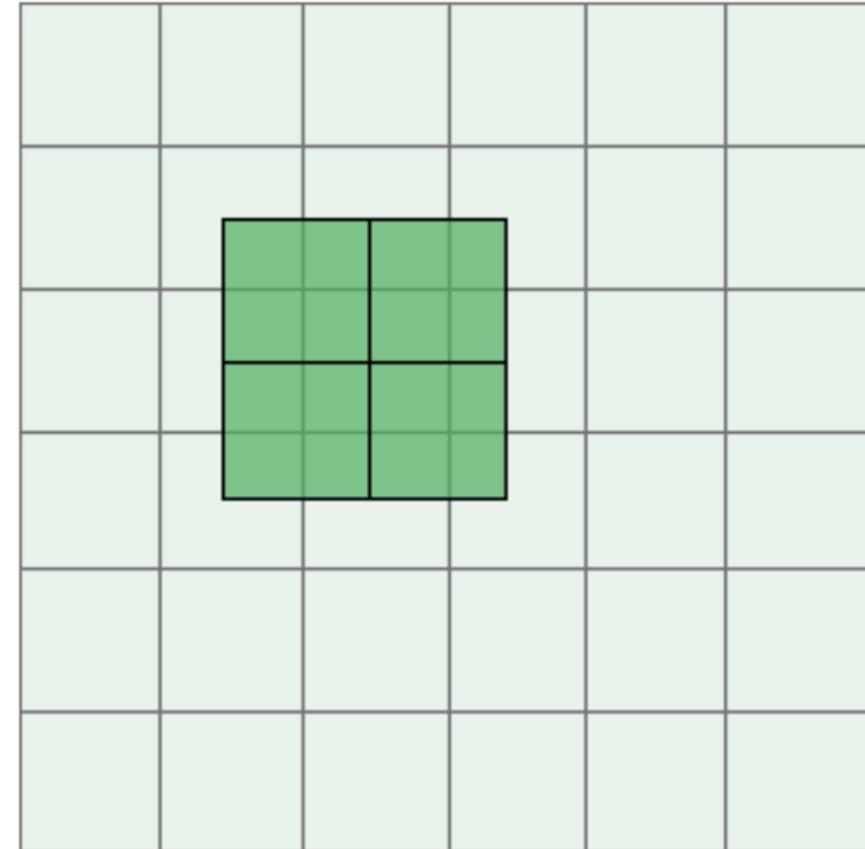
Output size : $\underbrace{\left\lfloor \frac{N+2p-F}{s} + 1 \right\rfloor}_{\text{height}} \times \underbrace{\left\lfloor \frac{N+2p-F}{s} + 1 \right\rfloor}_{\text{width}} \times \underbrace{\# \text{ of filters}}_{\text{depth}}$

짝수 높이의 filter는 안 쓰나요?

2x2, 4x4 같은 거 써도 되죠?

대개 잘 안 써요

기준점으로 작용할
정중앙 요소가 없어져요!

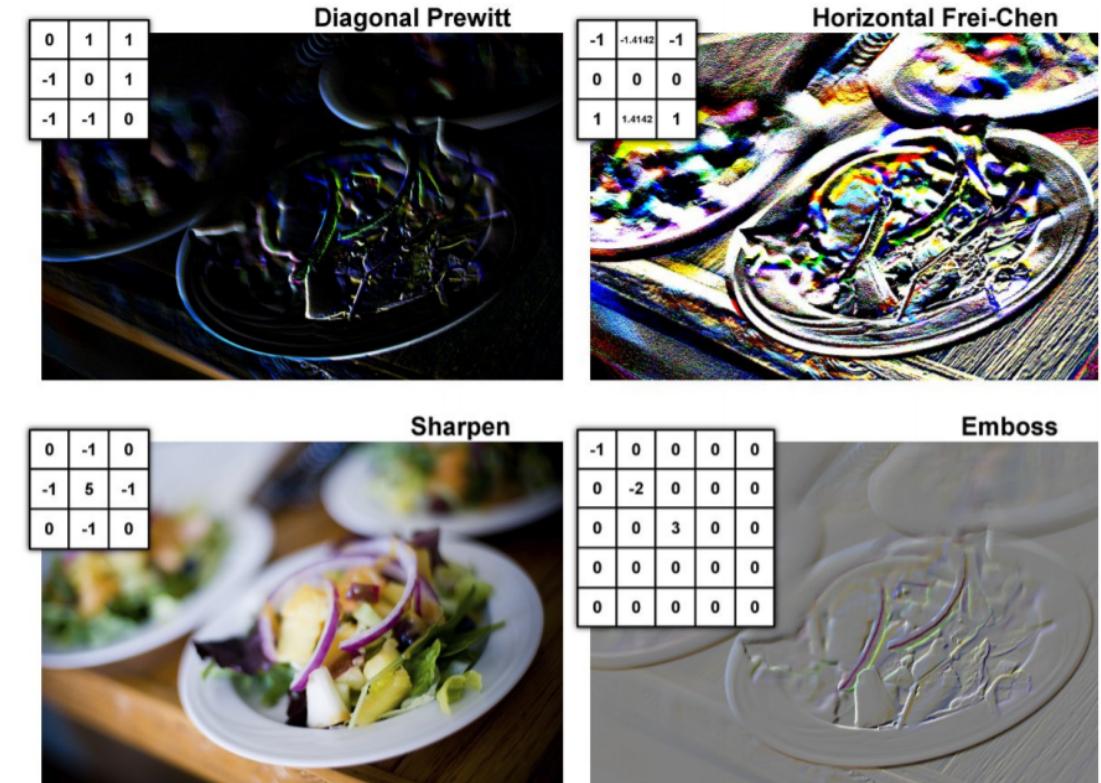


Convolution의 filter

필터는 feature detector입니다.

딥러닝 이전에는 학습이 아니라 사람이 직접 값을 지정해줬...

= Hand-designed (conventional ML)

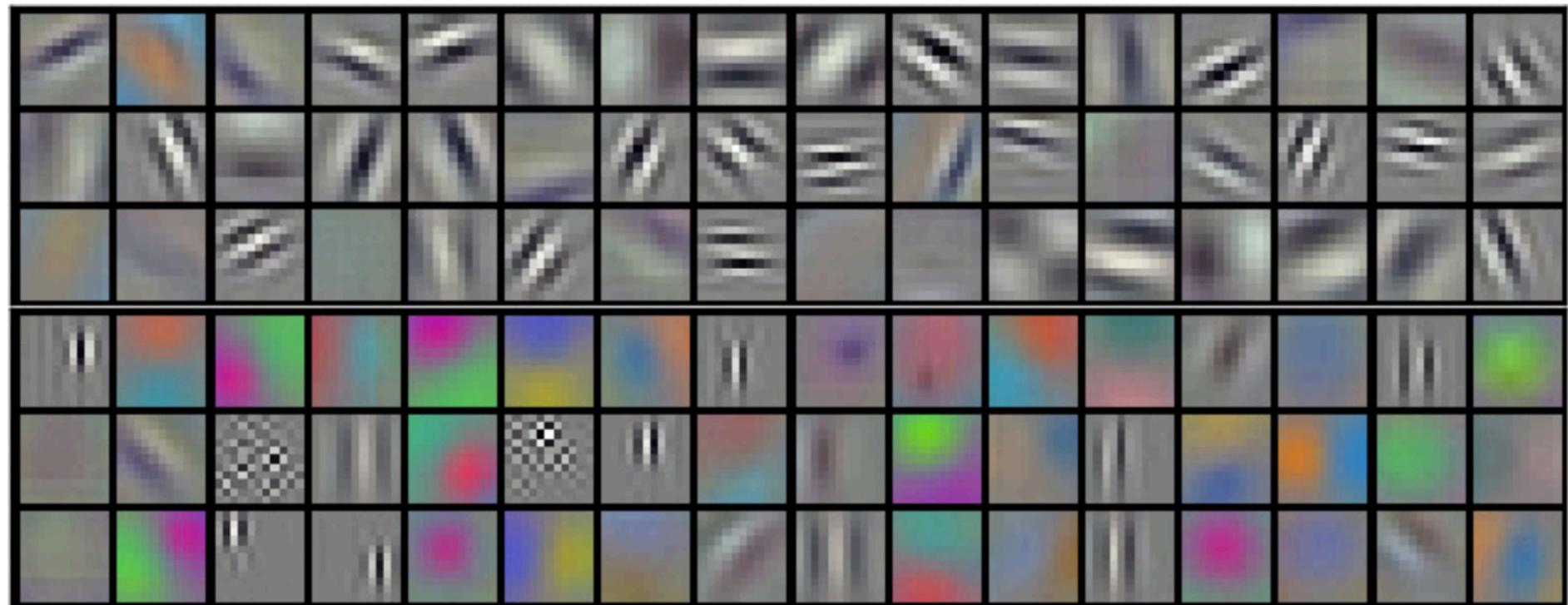


Convolution의 filter

필터는 feature detector입니다.

딥러닝에서는 필터의 weight값을 학습합니다!

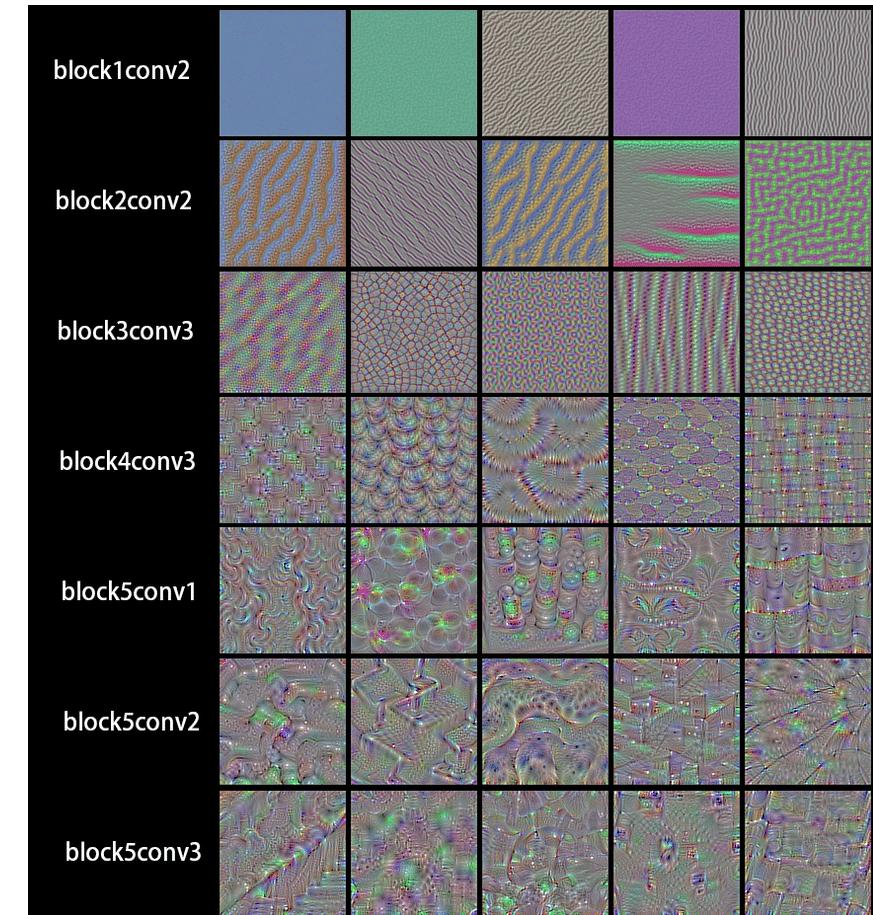
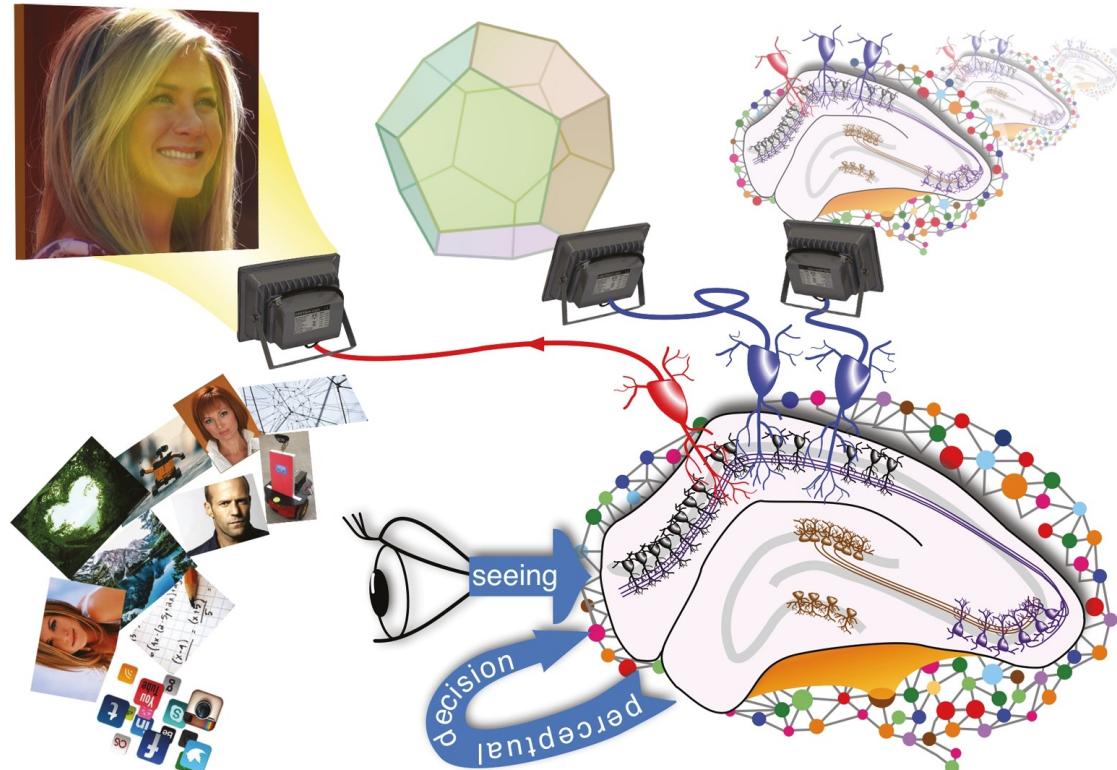
아래는 AlexNet의 96개 필터(사이즈 11x11)가 무엇을 배웠는지 시각화한 예시입니다



Convolution의 계층 구조

Hierarchical feature learning

낮은 층에서는 색깔이라든지, edge라든지 그런 low level feature를 학습
높은 층에서는 좀더 복잡한 패턴과 같은 high level feature를 학습



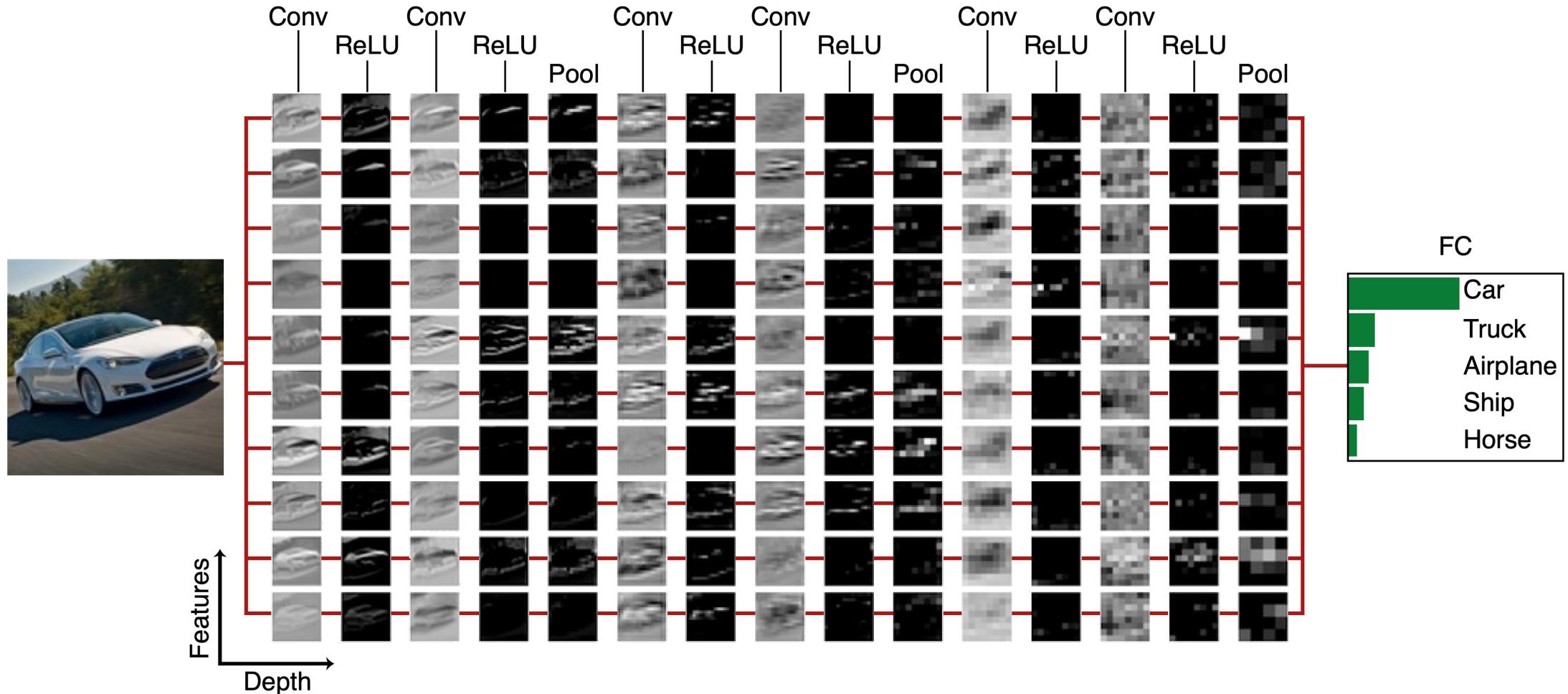
Convolutional Neural Net의 요소

ConvNet Representation

- 1. INPUT layer: Raw pixel values of the image
- 2. COV (Convolutional) layer: Output of units connected to local regions in the input
 - Compute a dot product between their weights and the region connected to in the input volume
- 3. Activation layer (RELU): Element-wise activation function
- 4. POOL layer: Downsampling (summation)
- 5. FC (Fully-connected) Layer: Computing a class score

Convolutional Neural Net

이들을 쌓아서 ConvNet 모델을 만듭니다

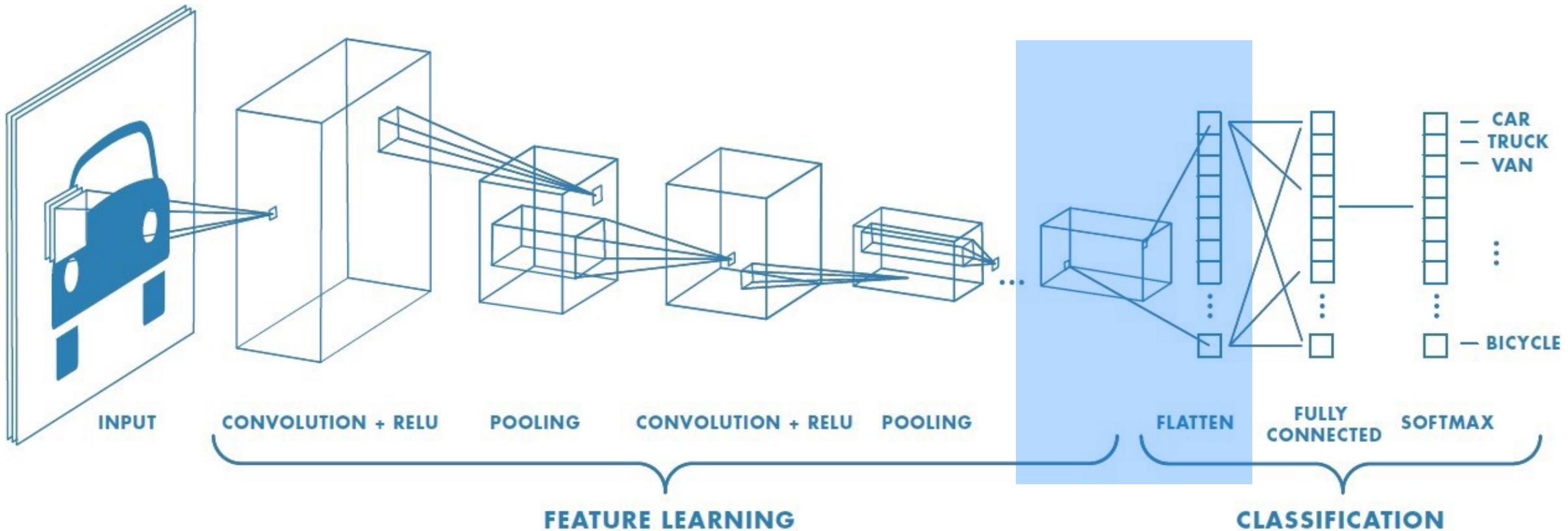


이제 모델 코드를 확인해봅시다
Convolutional_Neural_Networks_Pytorch

Flattening?

Height * width 의 행렬을 1 * (height * width) 형태로 1줄로 짹 펴주는 것

이걸 x.view(-1, #filter * width * height) 로 펴주는 거예요



딥러닝 처음 시작할 때 공부하기 좋은 자료

모두의 딥러닝 시즌2

우리말이라서 좋아요

<https://deeplearningzerotoall.github.io/season2/>