Deep Learning Overview

김 대 열

2023.03.10

강사 소개

- 김대열
- 주요 연구 분야 : 생체 신호 분석, 영상처리

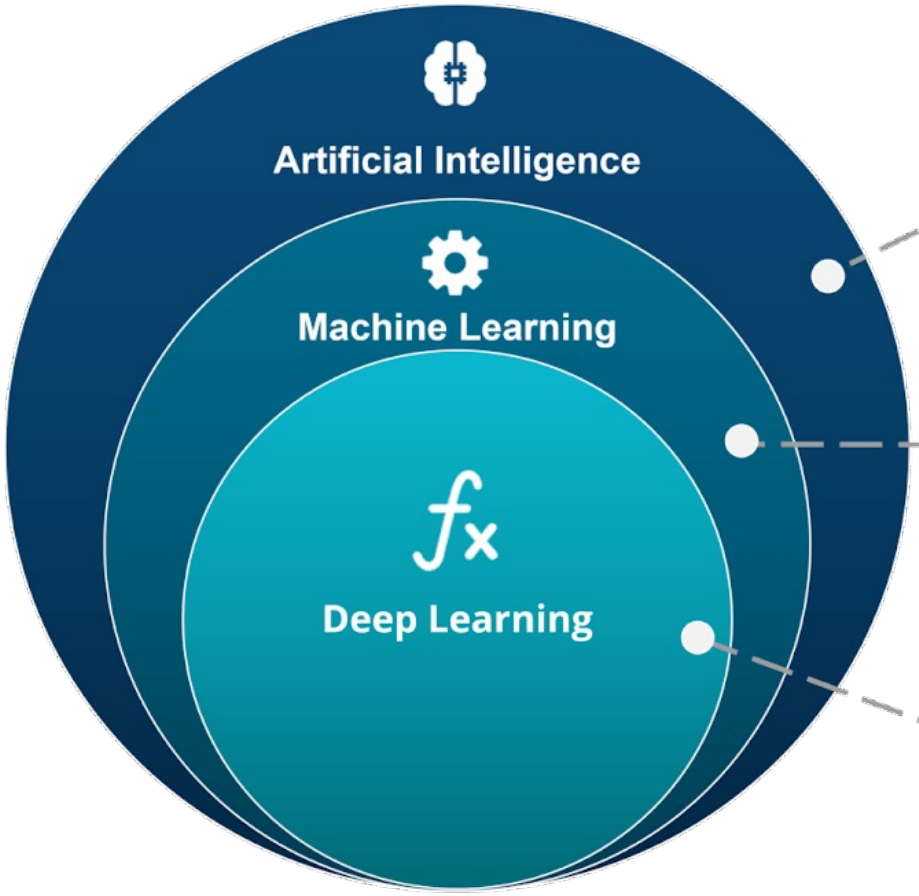
- Email : kwelcomm@gmail.com

- GPU 이용할 계획이 있다면 드라이버 및 쿠다 설치
  - https://www.nvidia.co.kr/Download/index.aspx <- conda
  - https://developer.nvidia.com/cuda-11-7-0-download-archive <- cuda
  - https://developer.nvidia.com/rdp/cudnn-archive <-cudnn
- Conda 설치
  - https://www.anaconda.com/products/individual
  - Sh 파일로 받았다면, $./Anaconda3-2021.11-Linux-x86_64.sh
- Conda 환경 구축
  - conda create -n pytorch python=3.9
  - conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia
  - conda install captum –c pyotorch
  - pip install wandb
- Pycharm install
  - https://www.jetbrains.com/ko-kr/pycharm/download/
    - 학교 계정이 있고, 연구실 서버를 외부에서 이용할 계획이 있다면 Professional 버전 추천
  - https://cosmosproject.tistory.com/480
    <- interpreter 설정 / professional 버전이라면 ssh 외부 interpreter 설정가능
- Tqdm install
  - conda install -c conda-forge tqdm
- Numpy install
  - Pip install numpy
- Matplotlib install
  - Pip install matplotlib

AL vs ML vs DL

- AI(Artificial Intelligence)
  - 사람이 할 수 있는 일을 대신하는 것을 통칭
- ML(Machine Learning)
  - AI의 종류 중 하나로 수 많은 데이터셋을 이용하여 동작
- DL(Deep Learning)
  - ML 방식 중 하나로 MLP(Multi Layer Perceptron)을 이용한 방식

AL vs ML vs DL

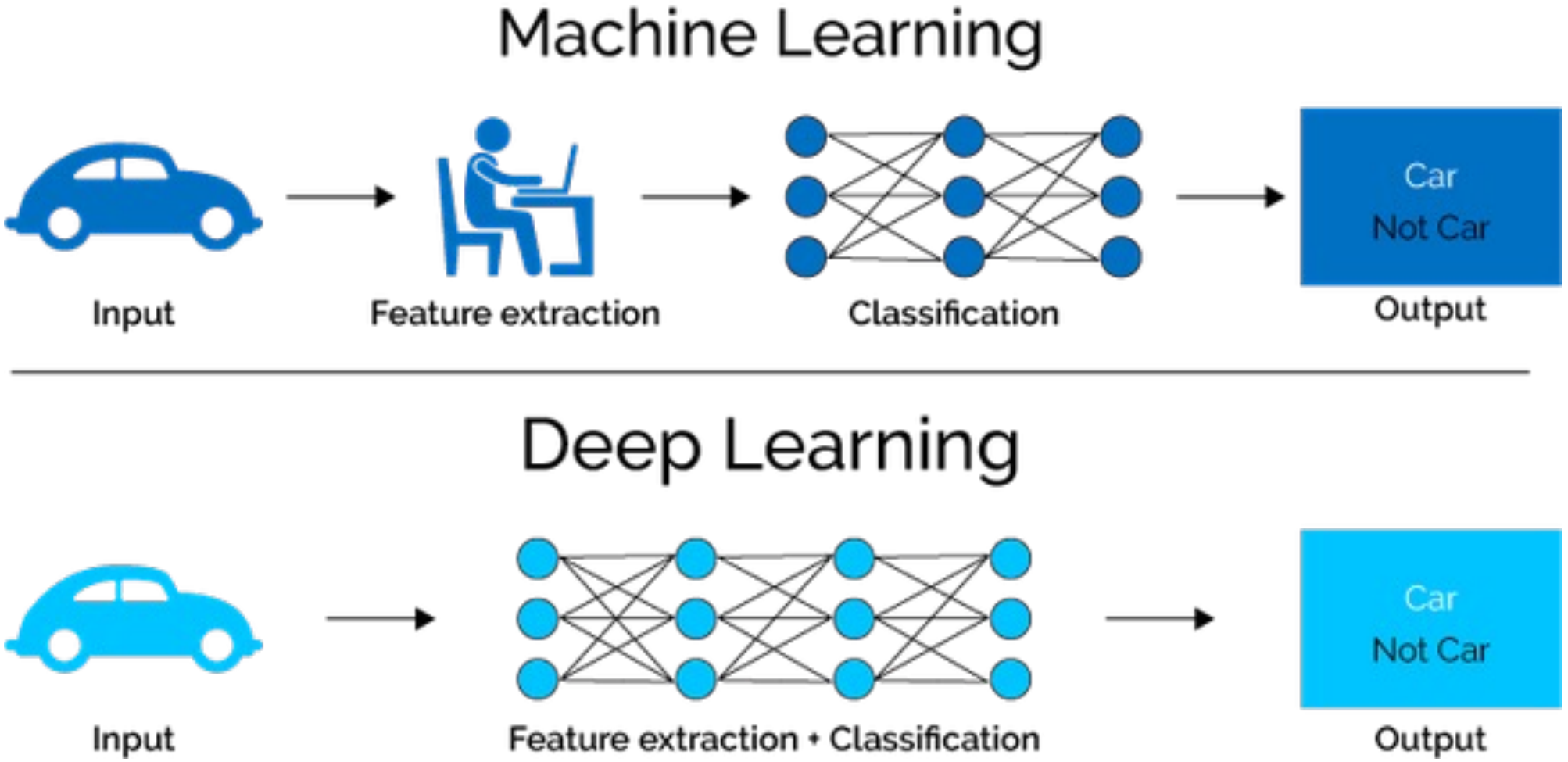


ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

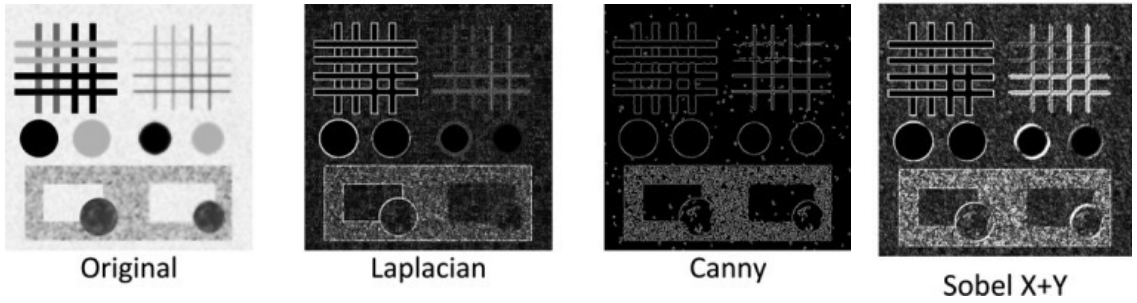
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

ML vs DL


Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output

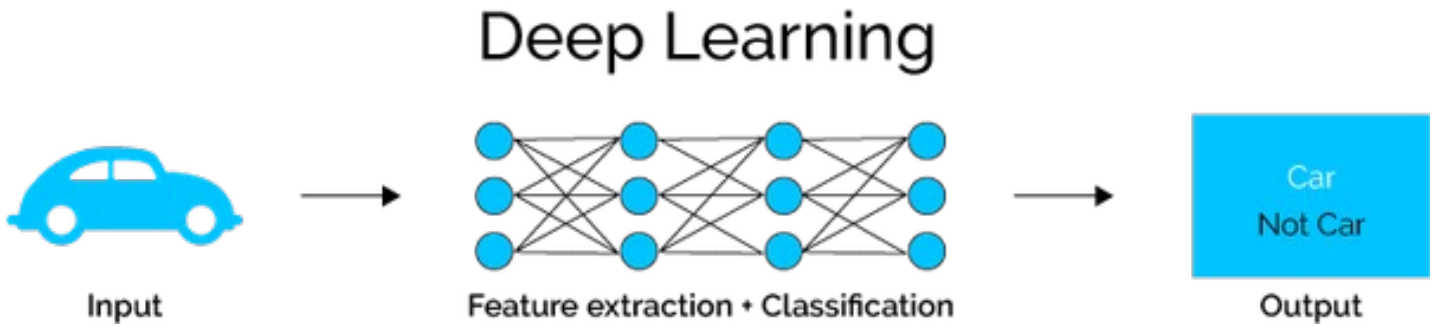
ML vs DL

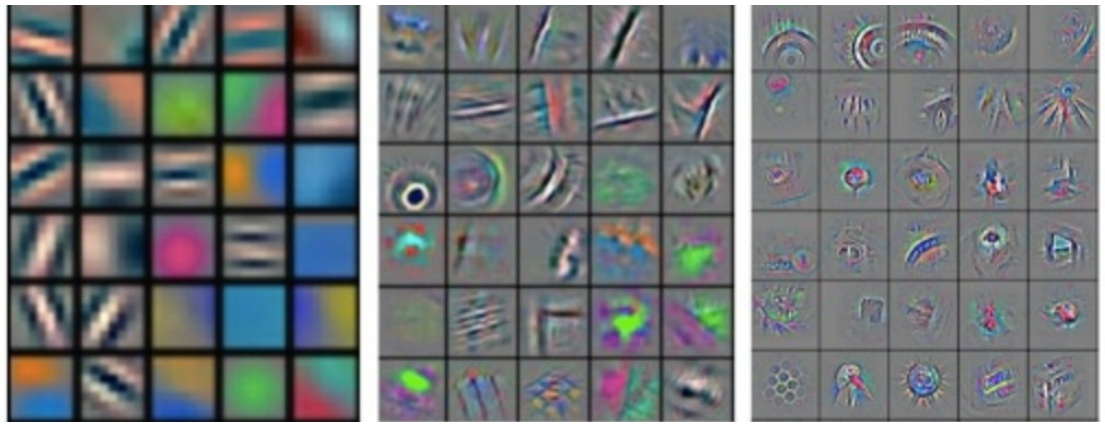
- ML
- 데이터의 어떤 특징을 어떻게 뽑아 냈는 가가 결과를 바꿈
  - Ex) 영상처리
    - 어떤 특징 : 외곽선 추출, 특징 점 추출.. 등
    - 어떻게
      - 외곽선 추출 : 어떤 필터를 썻는가
        - Laplace filter, sobel filter, Canny Edge
      - 특징 점 추출 : 어떤 특징 점 추출 방식을 사용했는가
        - SIFT, Harris Corner



ML vs DL

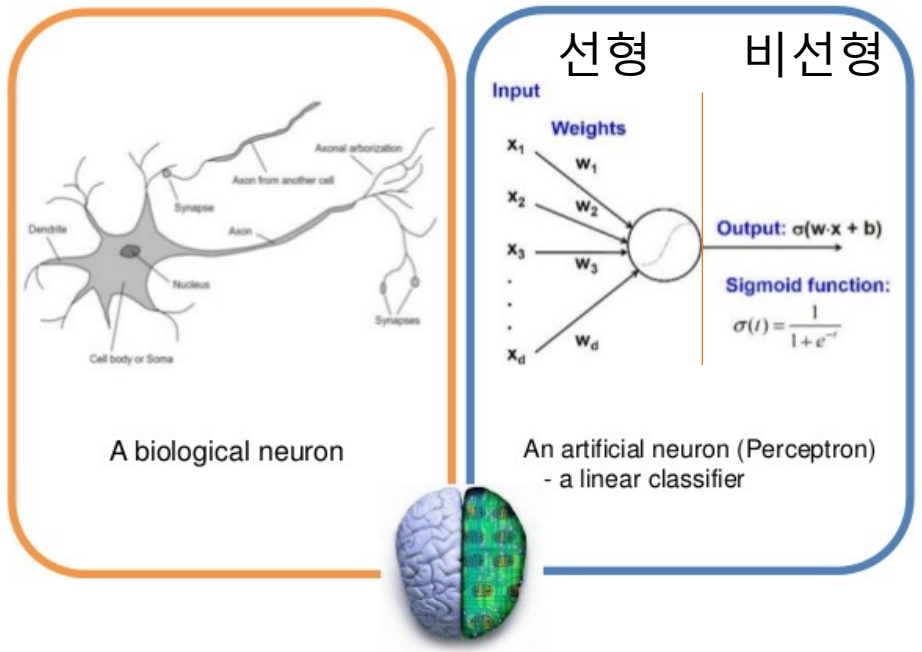
- DL


Deep Learning
Input
Feature extraction • Classification
Car
Not Car
Output



DL

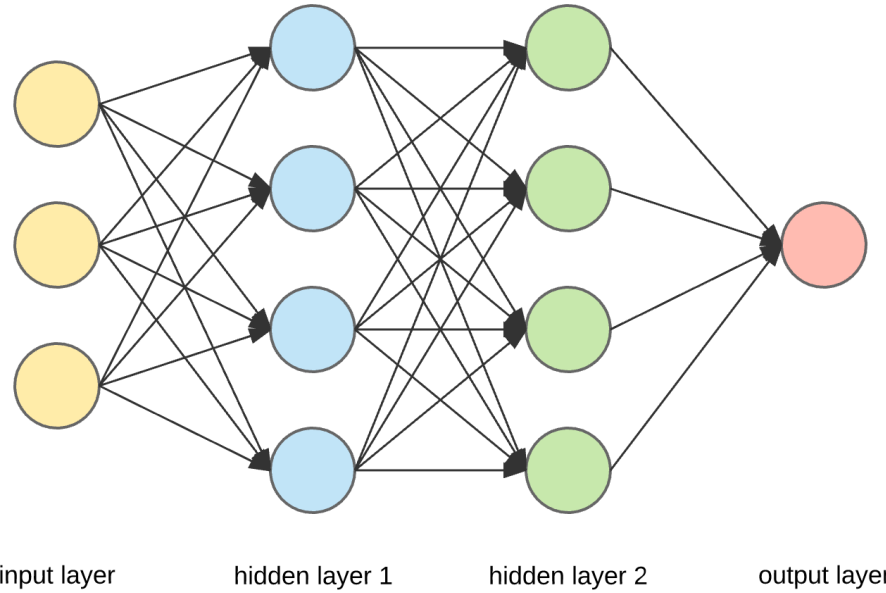
- Deep Learning(Deep Neural Network)이란?
  - Neural Network
    - 인간의 뇌를 모방한 알고리즘으로써 뇌의 최소 구성단위인 Neuron을 본 뜬 Perceptron으로 구성



DL

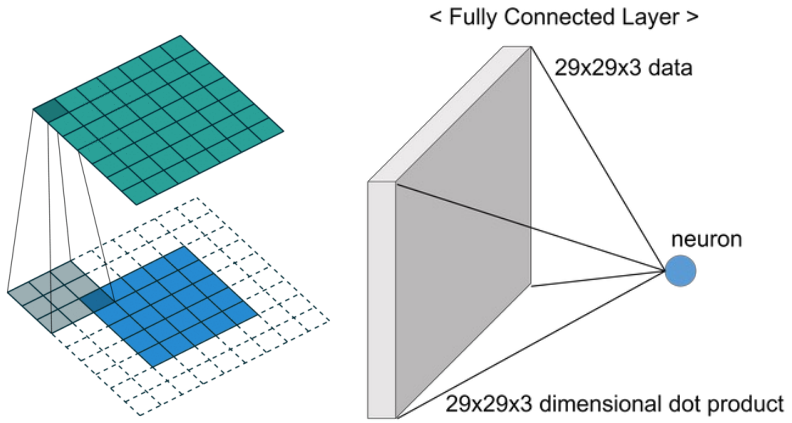
- Deep Learning(Deep Neural Network)이란?
  - Deep Neural Network

- Perceptron으로 이루어진 Hidden Layer가 두 개 층 이상 존재하면 Deep Neural Network라 칭함

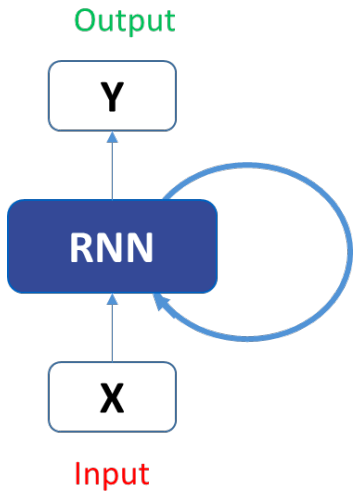


Components of DL

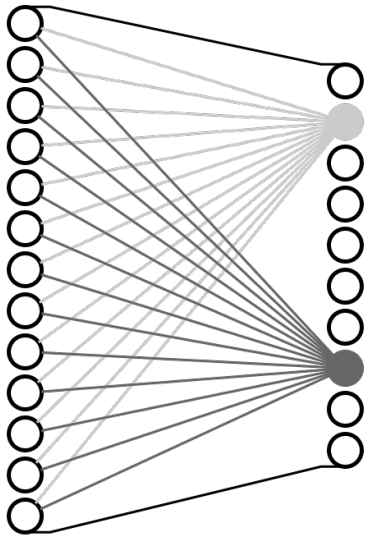
- Deep Learning은 Neuron으로 만들어진 다양한 Layer의 조합과 비선형 식의 조합으로 분류 능력이 뛰어난 알고리즘을 생성할 수 있음.



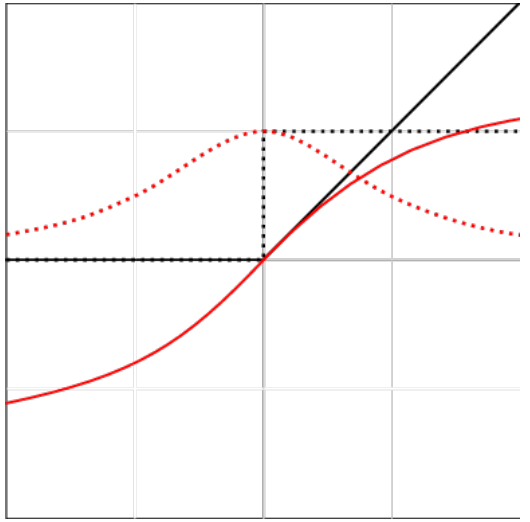
<Convolution Layer>
이미지 Feature 추출



<Recurrent Neural network>
시계열 data Feature 추출



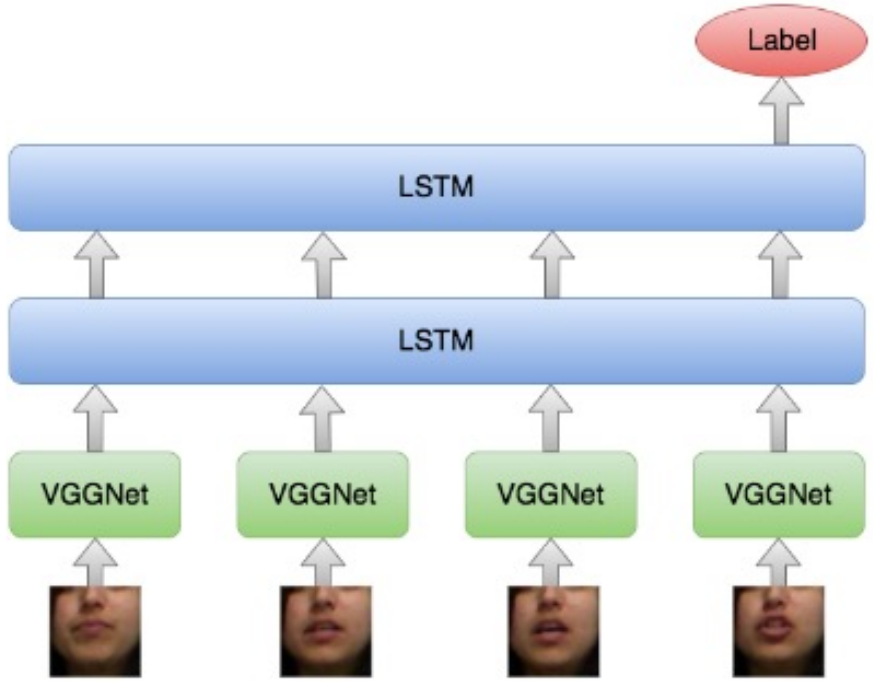
<Fully Connected Layer>
Input을 이용하여 분류



<Activation Function>
모델의 비선형성 증가

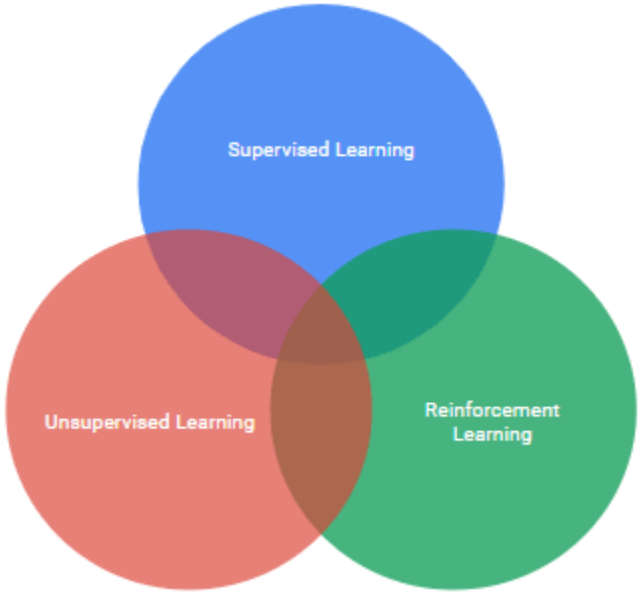
Components of DL

- Ex) Lip reading Model

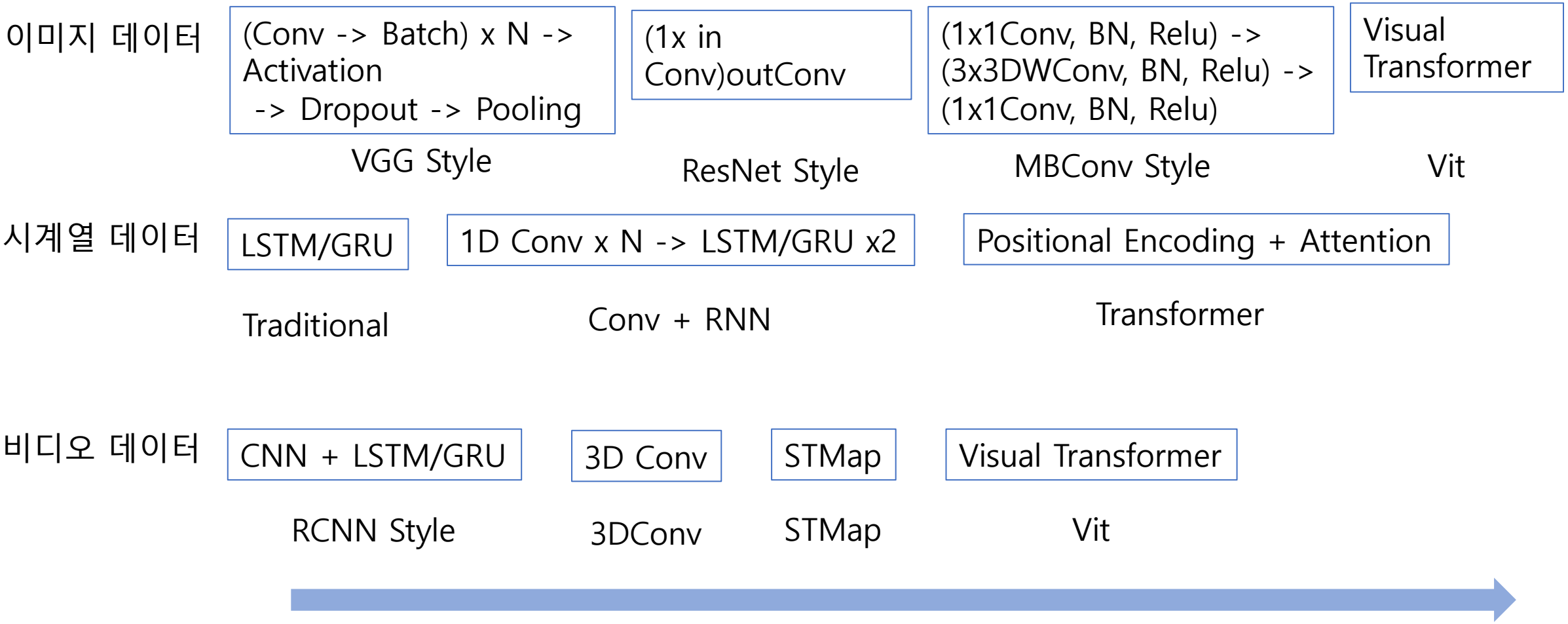


Types of ML

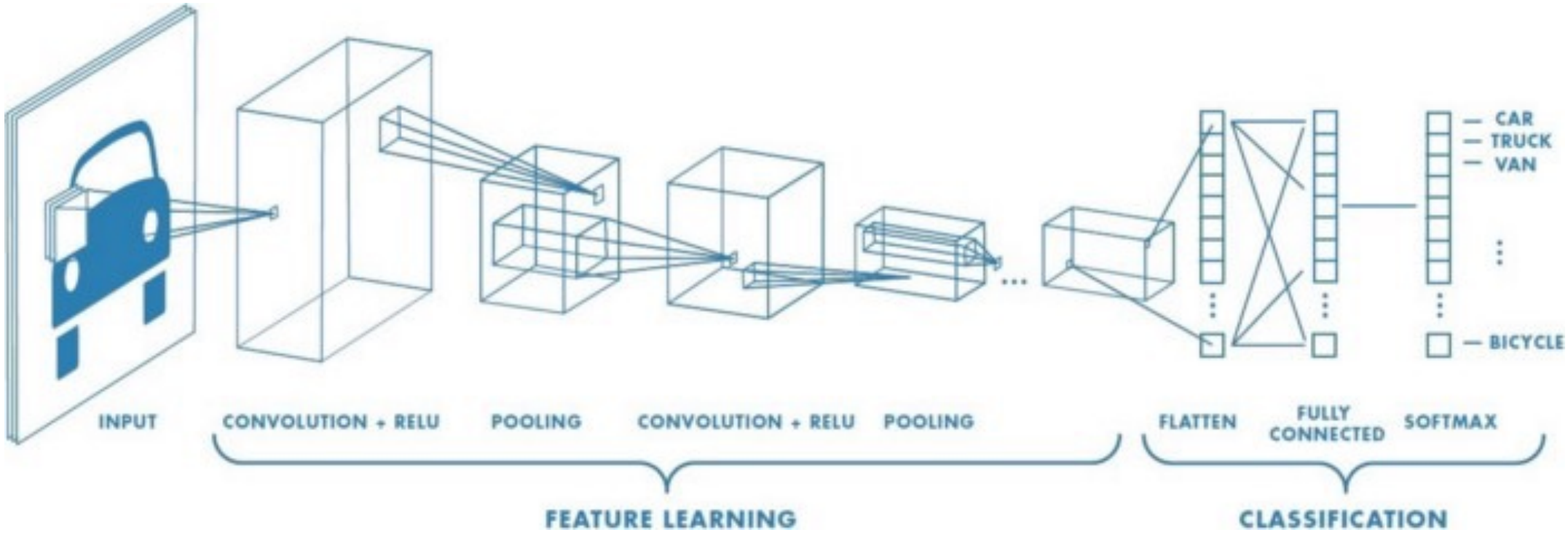
- 데이터에 따라 크게 3가지 분류의 학습 방법을 가진다
  - **Supervised Learning ← rPPG**
    - 데이터와 레이블이 있을 때 { X : Y }
  - Unsupervised Learning
    - 데이터만 있을때 {X}
  - Reinforcement Learning
    - 행동에 대한 결과(Reward)가 있을 때 { X : R }
- 추가적으로 데이터의 레이블이 일부 있을 때 Semi-supervised Learning 학습법을 이용한다.



Deep Learning Trend



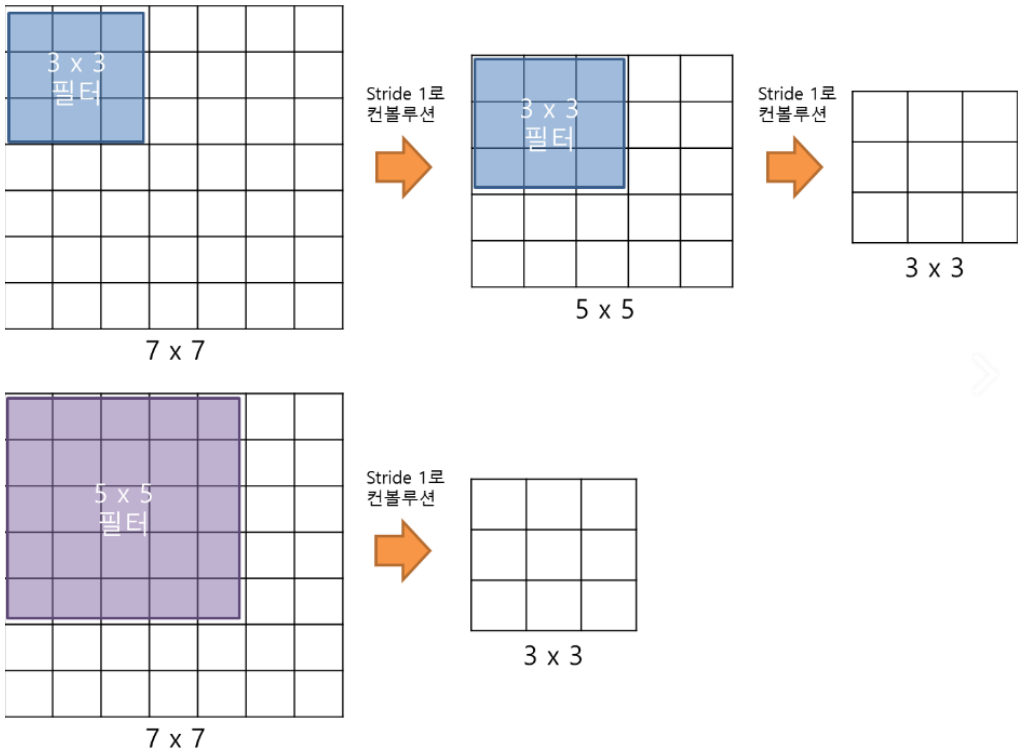
2D image processing history



2D image processing history

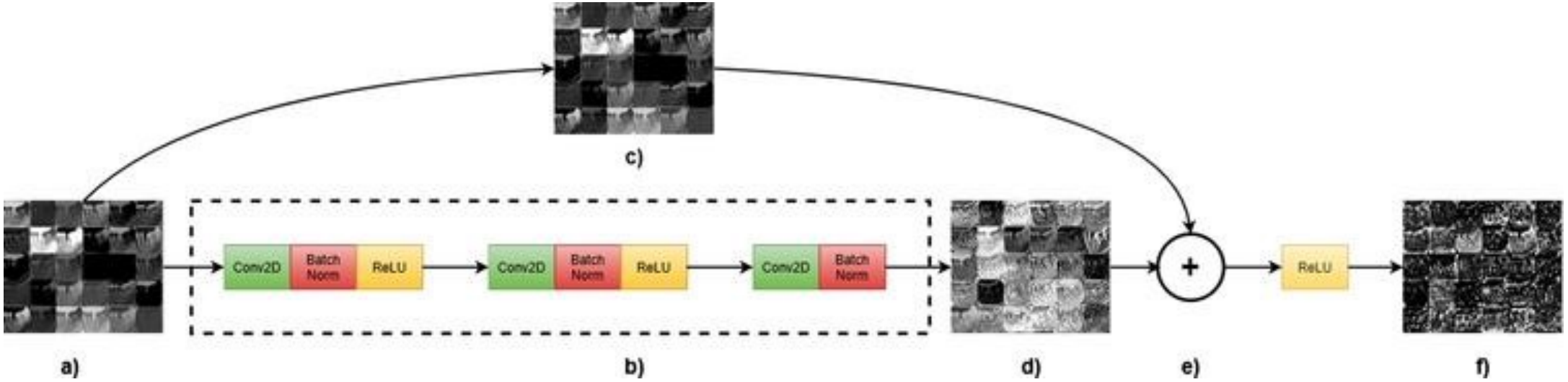
- Lenet style -> VGG style
  - LENET은 7x7 filter를 사용
    ➔ 많은 파라미터를 사용
  - VGG 는 7x7대신 3x3 세 번 사용
  - 7x7 filter의 49개의 파라미터
    ➔ 3x3 filter 3개는 9x3개의 파라미터

    49 ➔ 27 개로 획기적으로 파라미터 개선



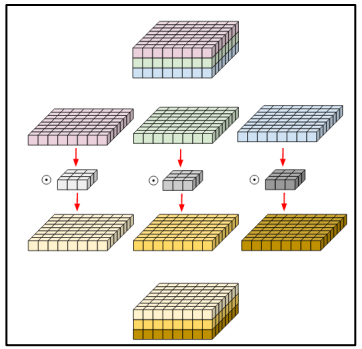
2D image processing history

- VGG style -> Resnet Style
  - VGG Style은 Deep 하게 쌓을 수록 학습이 안되는 문제를 갖고 있음
  - Conv 연산 전의 Feature를 유지 함으로써 Conv연산을 통해 소실된 데이터를 없애는 특징을 가짐



2D image processing history

- Resnet Style -> MB Conv Style
  - Depthwise Network 이용
    - Convolution이 모든 채널의 정보를 이용하는 점 때문에 Spatial한 정보를 취득하지 못하는 문제를 해결하기 위해 적용
  - SE(Squeeze & Excitation) Network를 이용
    - Squeeze
      - 전체 Filter를 대표하는 값을 추출 하는 역할
    - Excitation
      - one-hot activation이 아닌 다중 강조



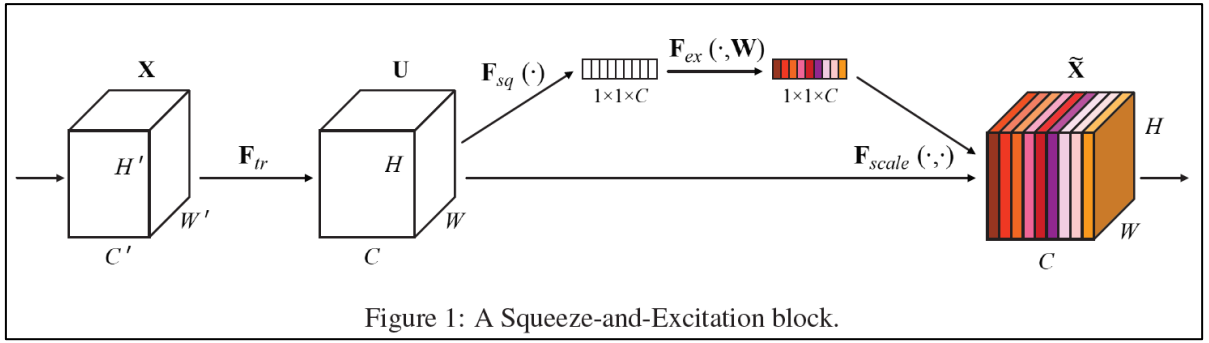
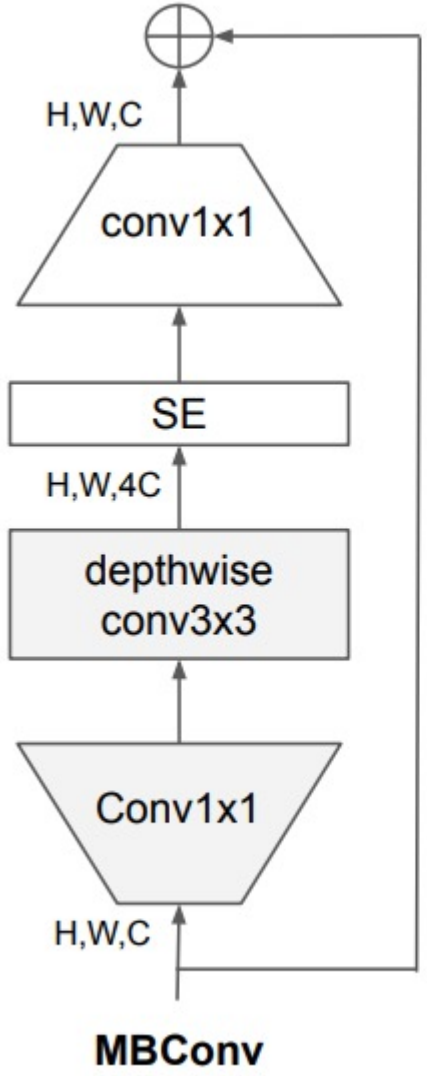


Figure 1: A Squeeze-and-Excitation block.

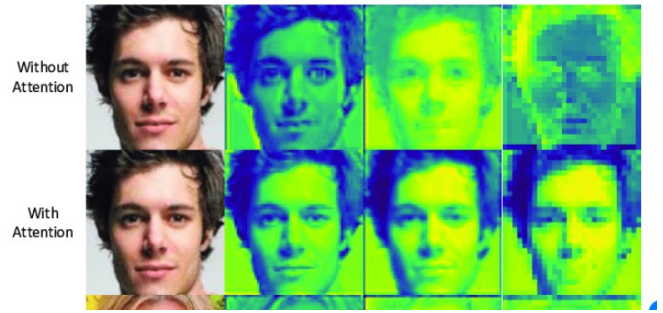


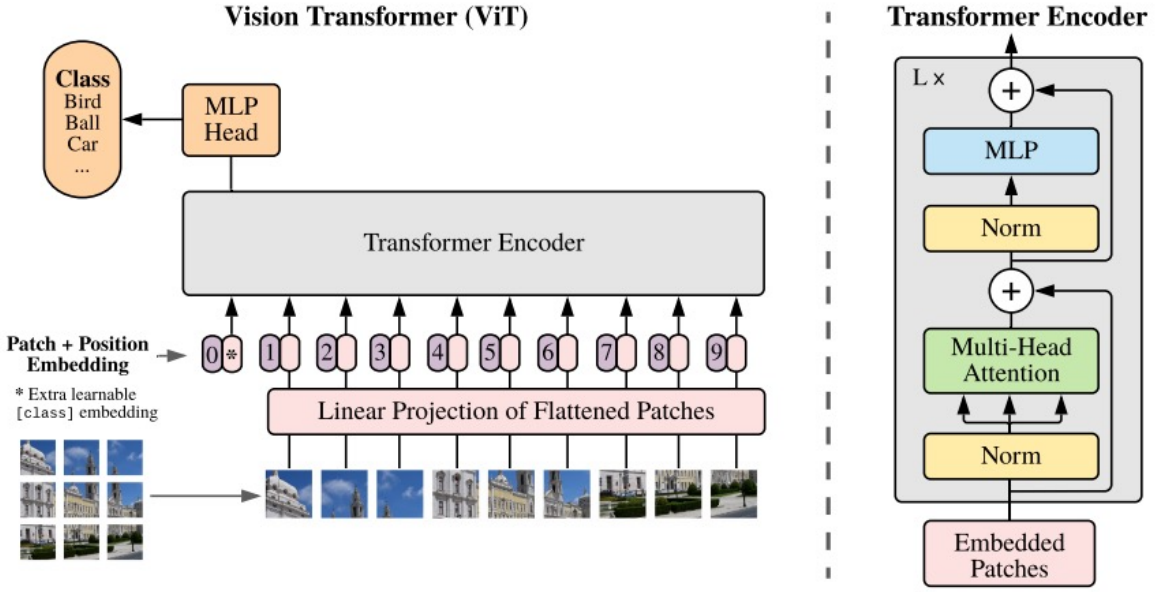
2D image processing history

- MB Conv Style -> ViT Style
  - ViT 이용
    - Convolution이 Local feature를 뽑기 때문에, 전체적인 특징을 뽑아내지 못하는 문제를 해결

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

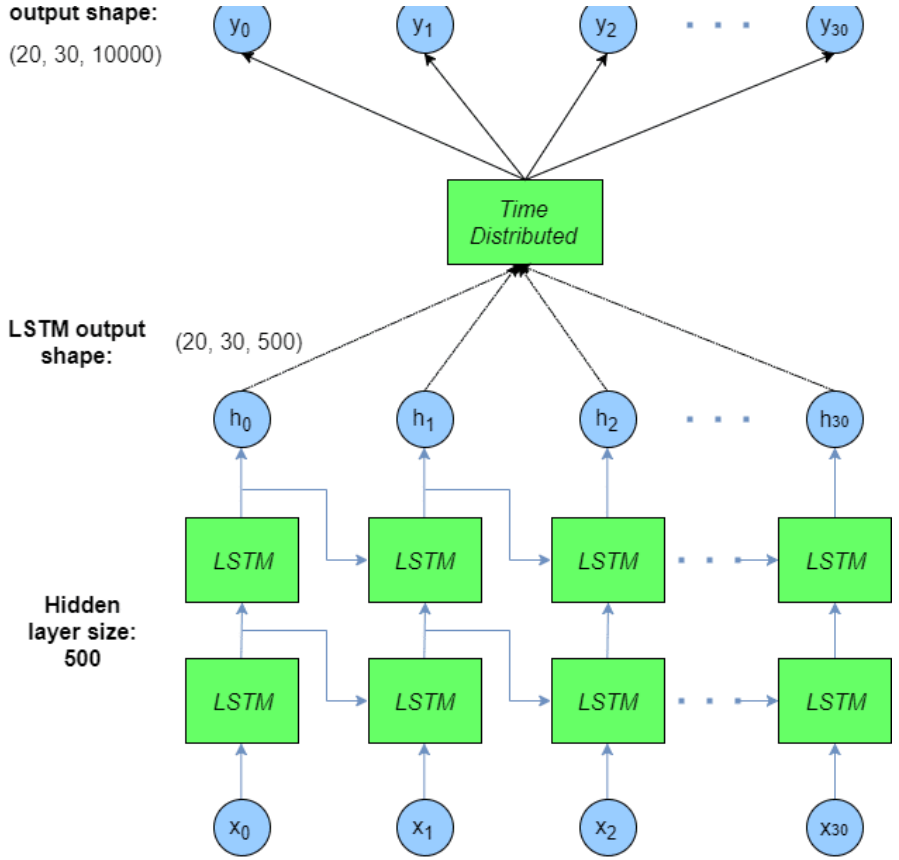
Table 1: Various relational inductive biases in standard deep learning components. See also Section 2.





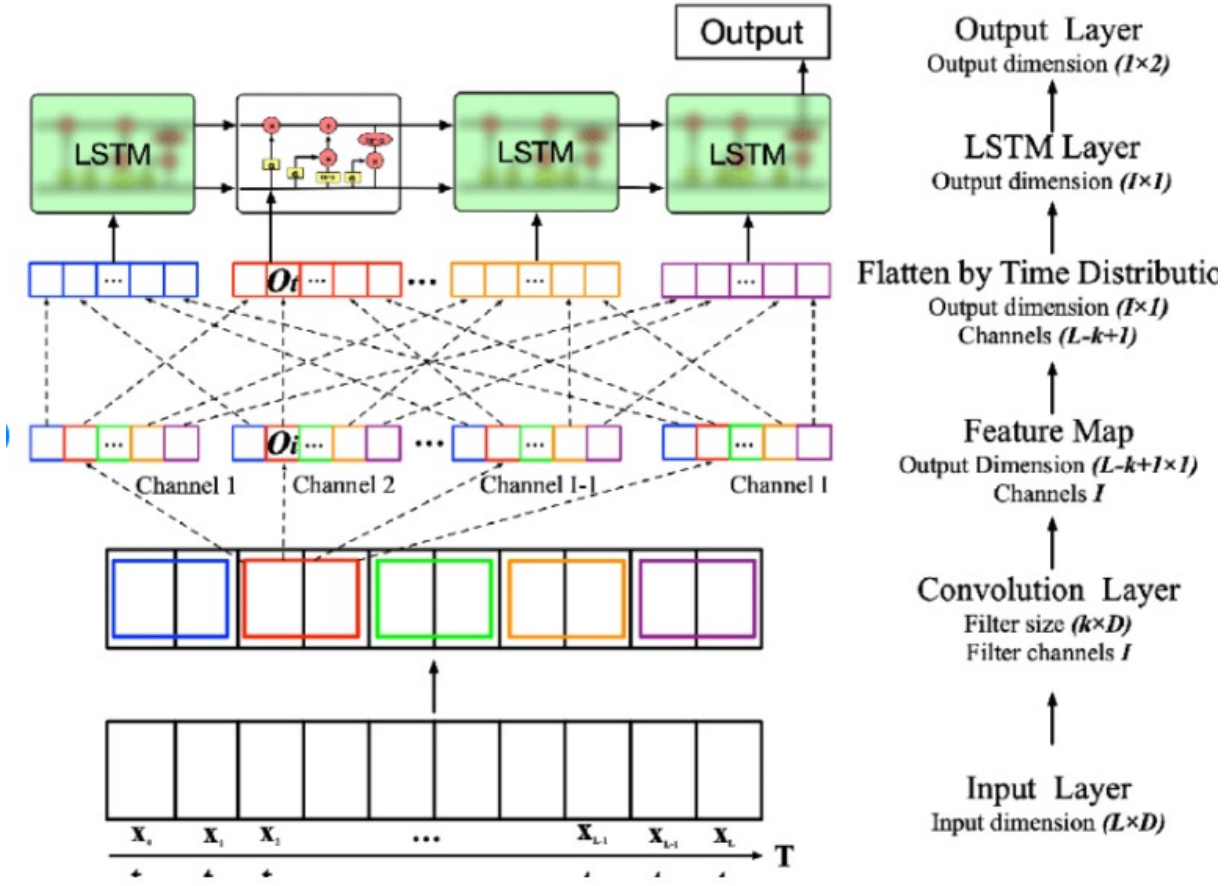
Timeseries processing history

- RNN Style
  - LSTM x2 + Dense 구조
    - 시계열 Feature를 구한 다음 Dense 구조
    - 기존의 LSTM 모델이 **이전의 데이터를 소실** 하는 특정이 있기 때문에 두 레이어를 쌓아 **양방향** 특징을 뽑아 내고지 함



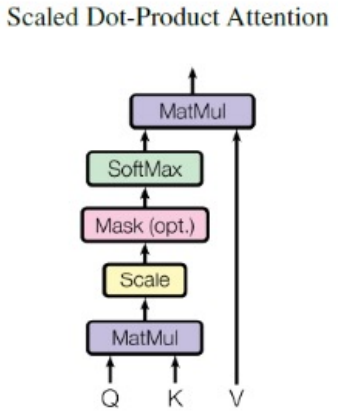
Timeseries processing history

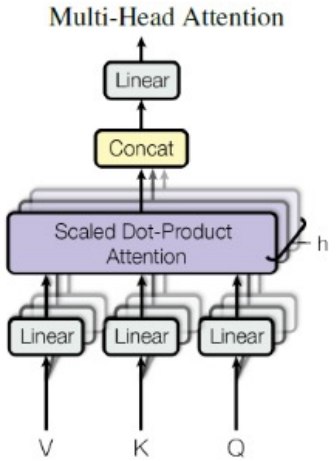
- RNN Style => Conv + RNN 구조
  - 1D CNN을 이용
    - Local Feature를 뽑아 여러 입력에 의해 값이 바뀔 수 있도록 구조 변경



Timeseries processing history

- Conv + RNN 구조 =>Positional Encoding + Attention
  - LSTM의 정보가 소실 되는 문제를 FC로 해결
  - Scaled Dot-product Attention을 통해 각 형태소의 특징 강화





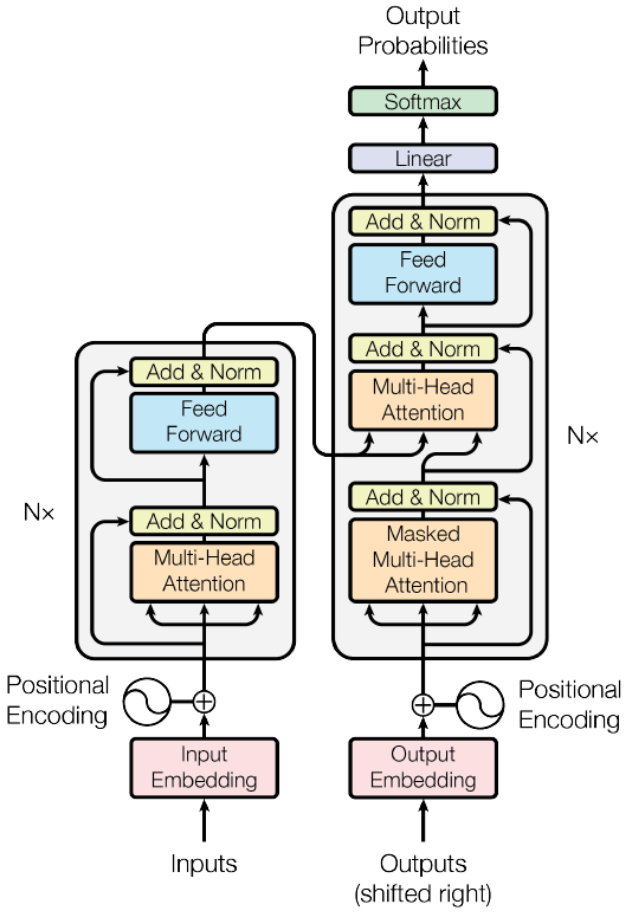


Figure 1: The Transformer - model architecture.

Video Data processing history

- Conv + LSTM
  - CNN을 통해 Feature Sequence 추출
  - Feature를 조합하여 LSTM으로 이미지 시퀀스 예측

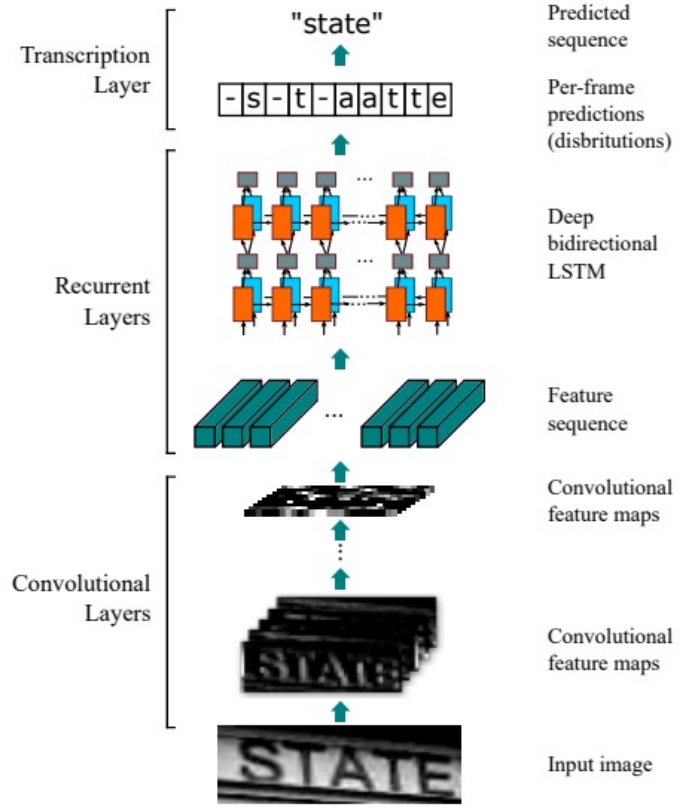
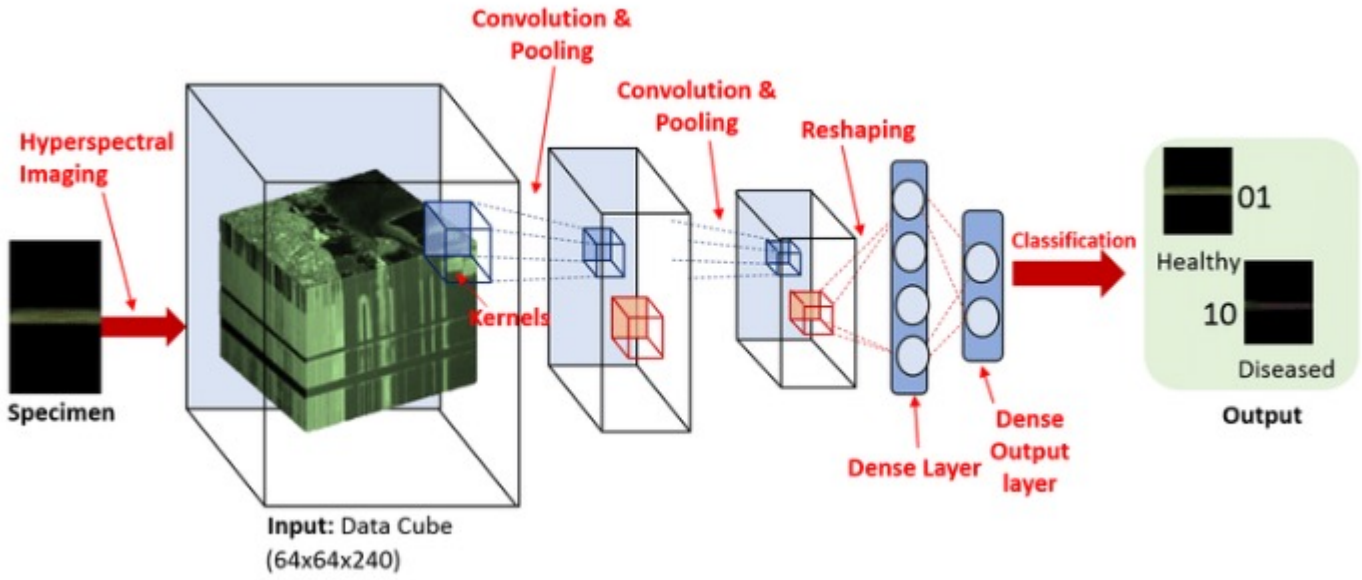


Figure 1. The network architecture. The architecture consists of three parts: 1) convolutional layers, which extract a feature sequence from the input image; 2) recurrent layers, which predict a label distribution for each frame; 3) transcription layer, which translates the per-frame predictions into the final label sequence.

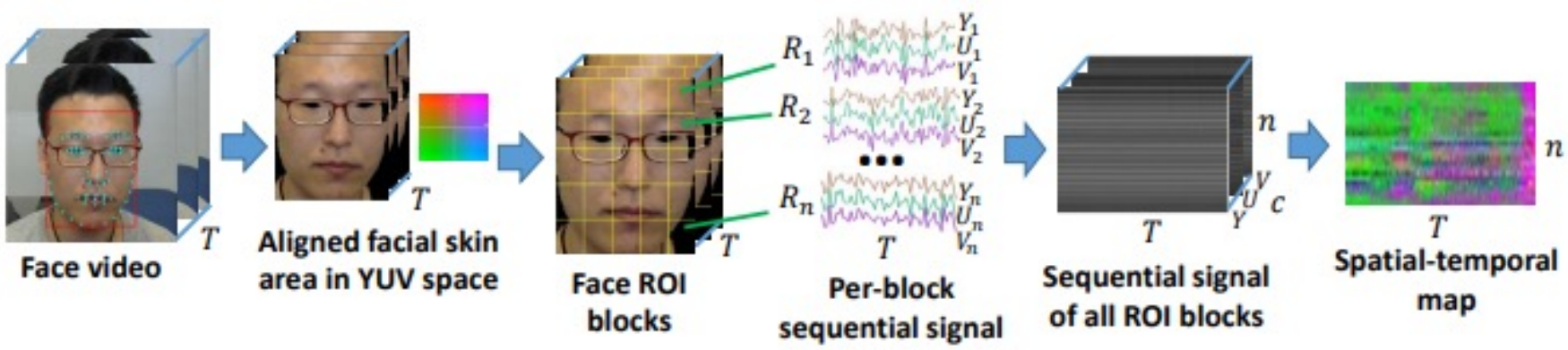
Video Data processing history

- Conv + LSTM => 3D CNN
  - LSTM의 Feature가 소실 되는 문제 해결
  - 2D(WxH)에 T(시간)을 추가하여 시 공간적인 특성을 해결하고자 함



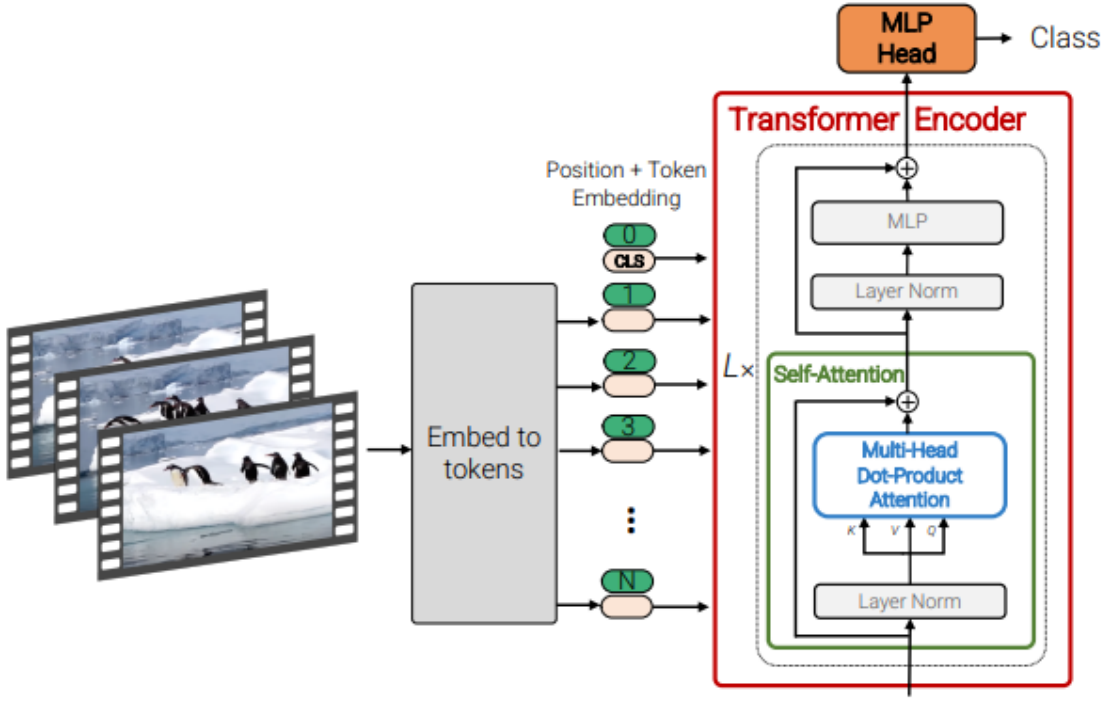
Video Data processing history

- 3D CNN => STMAP
  - 3D CNN의 파라미터가 많아 학습 속도가 느린 문제를 해결
  - 시간적인 특징에 더욱 집중하여, 시계열 정확도를 높임



Video Data processing history

- STMAP => ViT
  - 각 이미지를 position 토큰으로 변환하여 성능 향상



Deep Learning Trend

이미지 데이터

(Conv -> Batch) x N -> Activation -> Dropout -> Pooling

VGG Style

(1x in Conv)outConv

ResNet Style

(1x1Conv, BN, Relu) -> (3x3DWConv, BN, Relu) -> (1x1Conv, BN, Relu)

MBConv Style

Visual Transformer

Vit

시계열 데이터

LSTM/GRU

Traditional

1D Conv x N -> LSTM/GRU x2

Conv + RNN

Positional Encoding + Attention

Transformer

비디오 데이터

CNN + LSTM/GRU

RCNN Style

3D Conv

3DConv

STMap

STMap

Visual Transformer

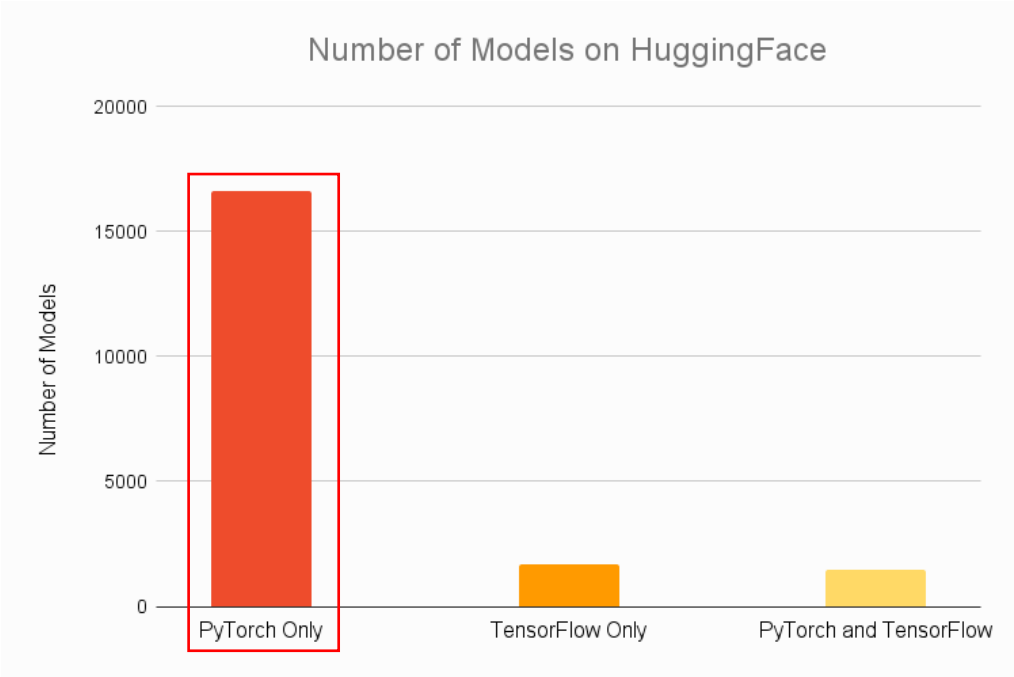
Vit

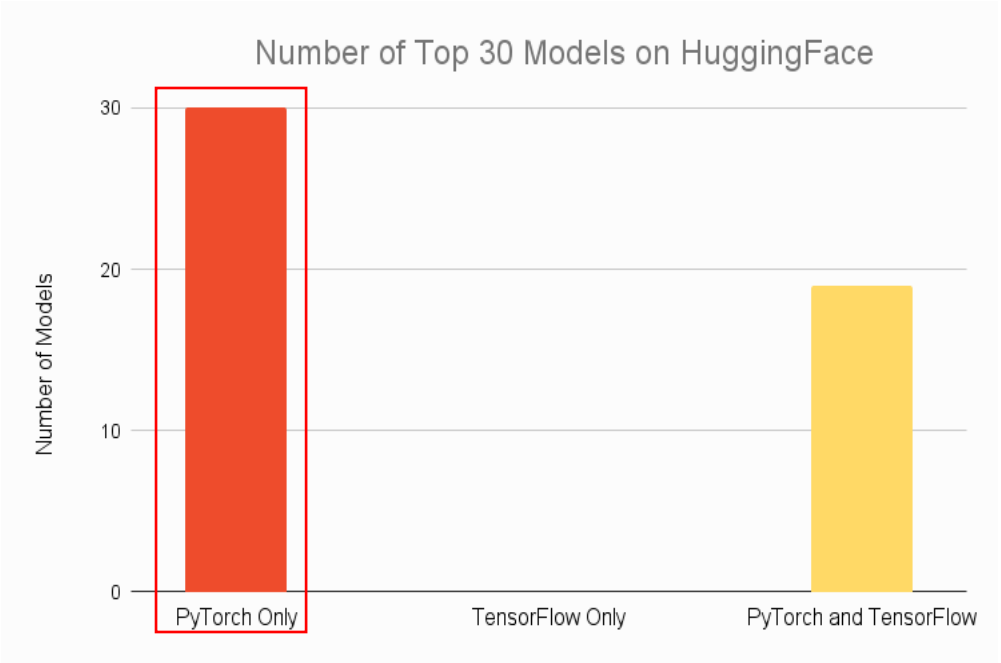
Deep Learning Framework

- Tensorflow vs PyTorch
  - https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/
- 고려사항
  - 모델 가용성
    - 새로운 SOTA 알고리즘들이 얼마나 활용 가능한가
  - 배포 인프라
    - 훈련된 모델을 얼마나 쉽고 사용 할 수 있다
  - 생태계
    - PC기반이 아닌 다른 특수한 생태계에서 통합 가능성이 있는가

Deep Learning Framework

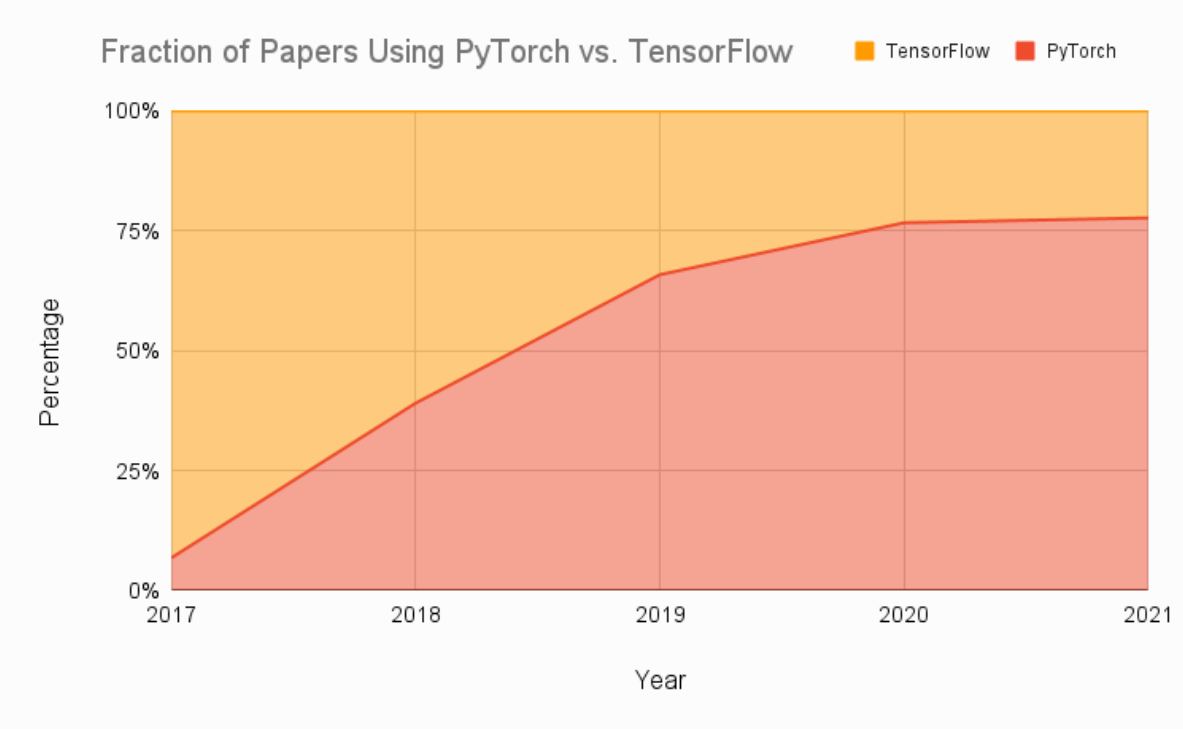
- 모델 가용성
  - Hugging Face :
    Transformer 기반으로 하는 다양한 모델의 학습 스크립트가 구현된 모듈

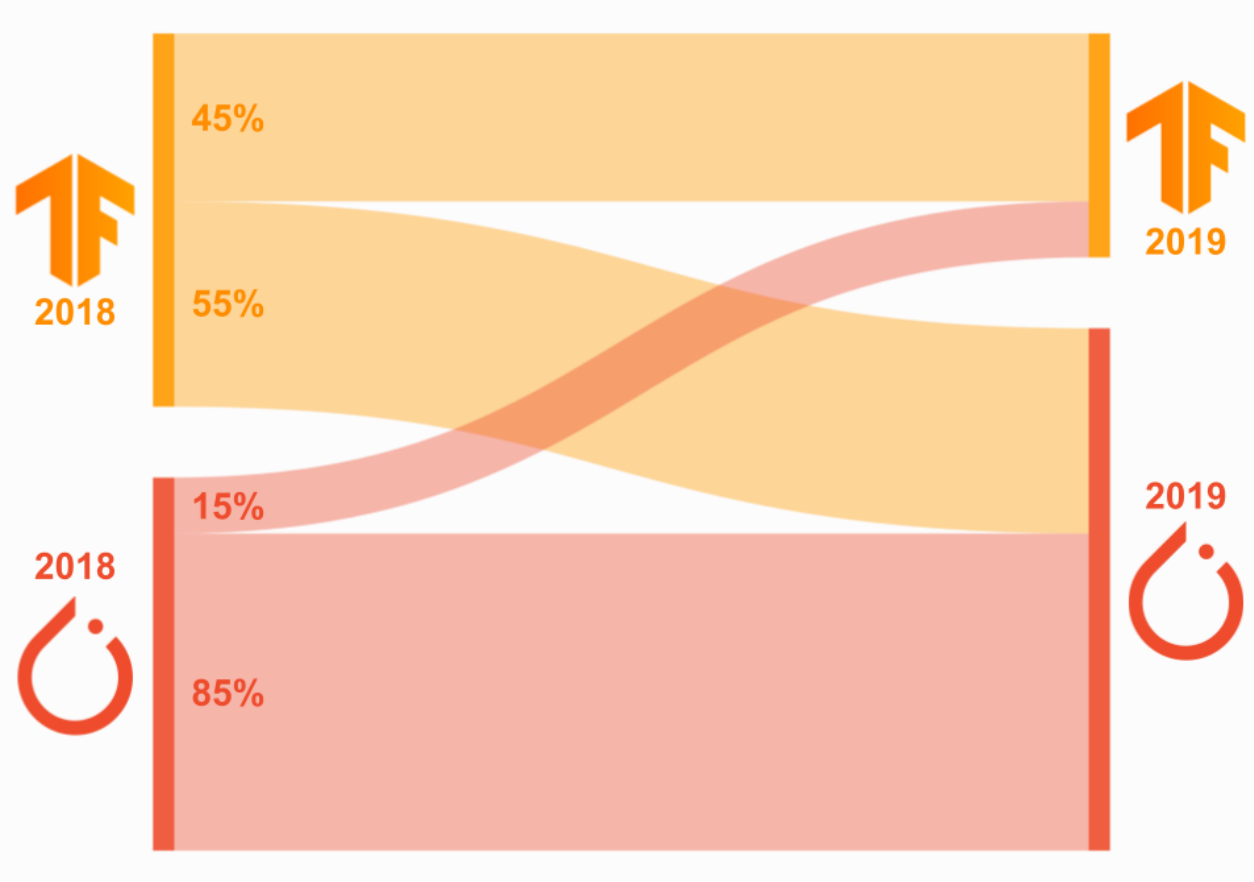




Deep Learning Framework

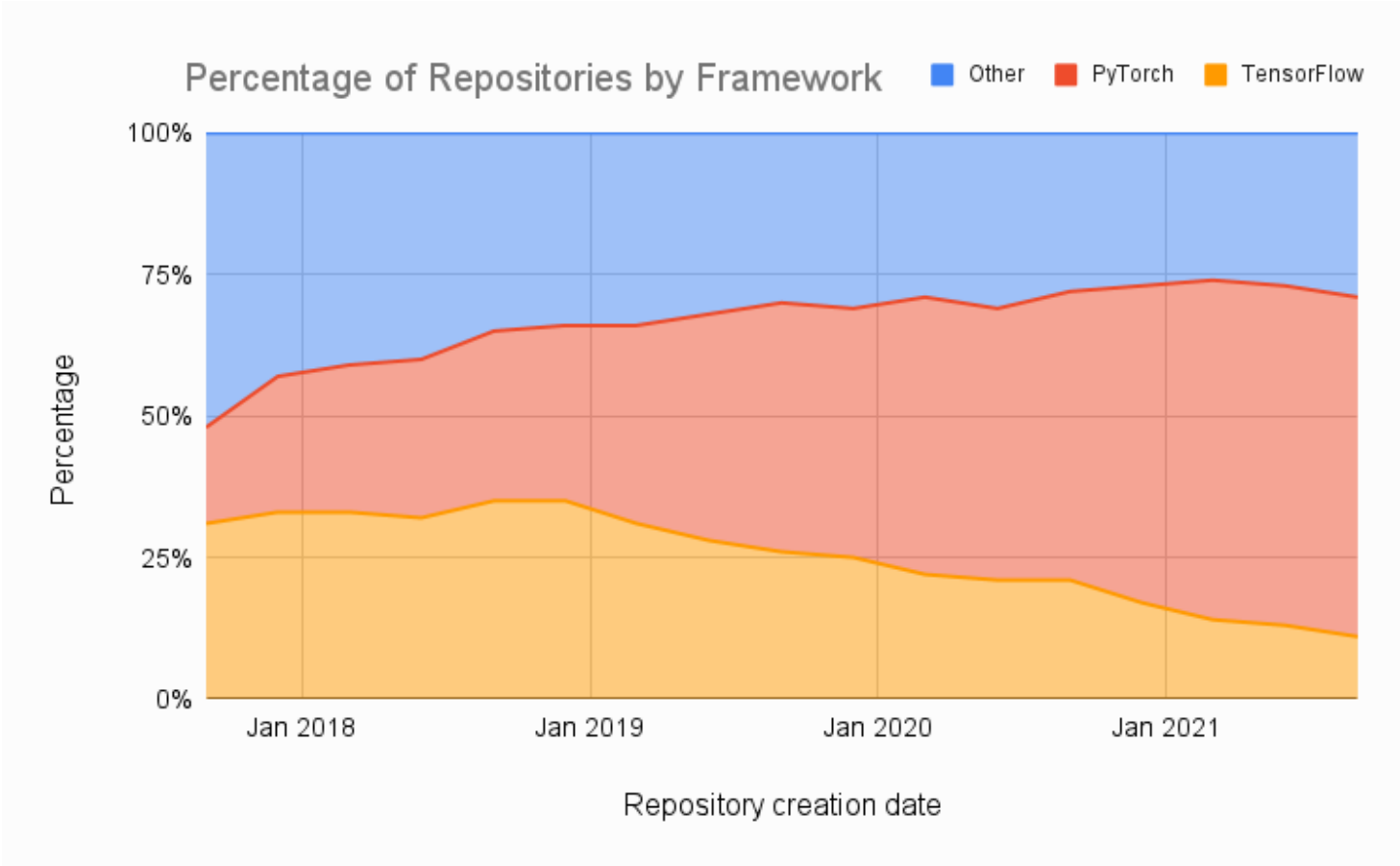
- 모델 가용성
  - Research Paper





Deep Learning Framework

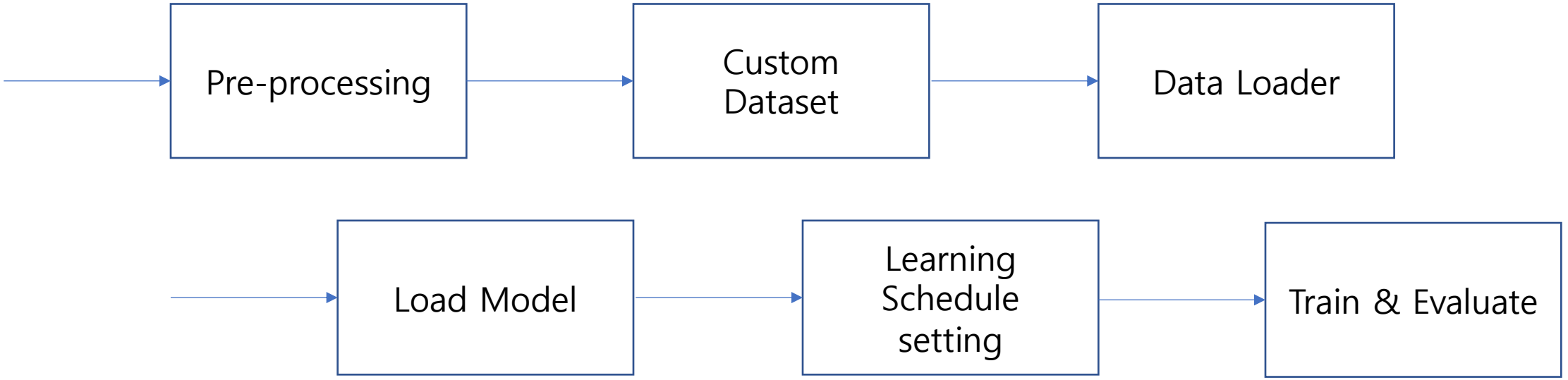
- 모델 가용성
  - Papers with Code



Deep Learning Framework

- 초보자라면..
  - TF v2(Keras)를 이용해서 딥러닝 연습 후 Pytorch 로 넘어 가는 것 추천

Deep Learning 학습 절차



Deep Learning 학습 절차

- Preprocessing
  - 데이터 preprocessing은 모델이 풀어야하는 문제를 좀더 쉽게 정의
  - 학습 수렴 속도/성능의 향상

  - Ex) data slicing, Normalization

```
def generate_Floatimage(frame):
    '''
    :param frame: roi frame
    :return: float value frame [0 ~ 1.0]
    '''
    dst = img_as_float(frame)
    dst = cv2.cvtColor(dst.astype('float32'), cv2.COLOR_BGR2RGB)
    dst[dst > 1] = 1
    dst[dst < 1e-6] = 1e-6
    return dst
```

Deep Learning 학습 절차

- Custom Dataset
  - 원하는 목적을 가지는 모델을 학습시키기 위해선 Dataset을 원하는 형태로 구성이 가능 하여야 함

  - 구성 요소
  - def __init__
  - def __getitem__
  - def __len__

```python
from torch.utils.data import Dataset
# torch.utils.data.Dataset 상속
class CustomDataset(Dataset):
  def __init__(self, x, y):
    self.x_data = x
    self.y_data = y

  def __getitem__(self,idx):
    x = torch.Tensor(self.x_data[idx])
    y = torch.Tensor(self.y_data[idx])
    if torch.cuda.is_available():
      return x.to('cuda'), y.to('cuda')
    return x,y

  def __len__(self):
    return len(self.x_data)
```

Deep Learning 학습 절차

- DataLoader
  - Dataset을 딥러닝 모델 학습에 편한 형태로 제공하기 위해 객체로 감쌈
  - Batch size, shuffle

  - 일반적으로 데이터셋은 trainset: testset = 80:20 으로 이용

```
train_dataset, test_dataset = random_split(dataset,
                                           [int(np.floor(len(dataset) * 0.8)),
                                            int(np.ceil(len(dataset) * (0.2)))])
train_loader = DataLoader(train_dataset, batch_size=300,shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=300,shuffle=True)
```

Deep Learning 학습 절차

- Load Model
  - 데이터셋의 입력/출력 형태에 맞게 모델 생성

  - torch.nn.Module 상속
  - def __init__(self):
    - 네트워크 레이어 등 각종 변수 선언
  - def forward(self, x):
    - 네트워크 레이어 연결

```
import torch
class Network(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.dense1 = torch.nn.Linear(2,10)
        self.dense2 = torch.nn.Linear(10,2)

    def forward(self, input)
        x = self.dense1(input)
        x = self.dense2(x)
        return x
```

Deep Learning 학습 절차

- Learning Schedule Setting
  - 모델의 optimizer/ lr 등 다양한 정보 세팅

```
import torch.nn.modules.loss as loss
import torch.optim as opt
import torch.optim import lr_scheduler

model = Network().to('cuda')
criterion = loss.MSELoss()
optimizer = opt.Adam(model.parameters, lr = 0.001, weight_decay = 0.0005)
scheduler = lr_scheduler.ExponentialLR(optimizer, gama=0.99)
```

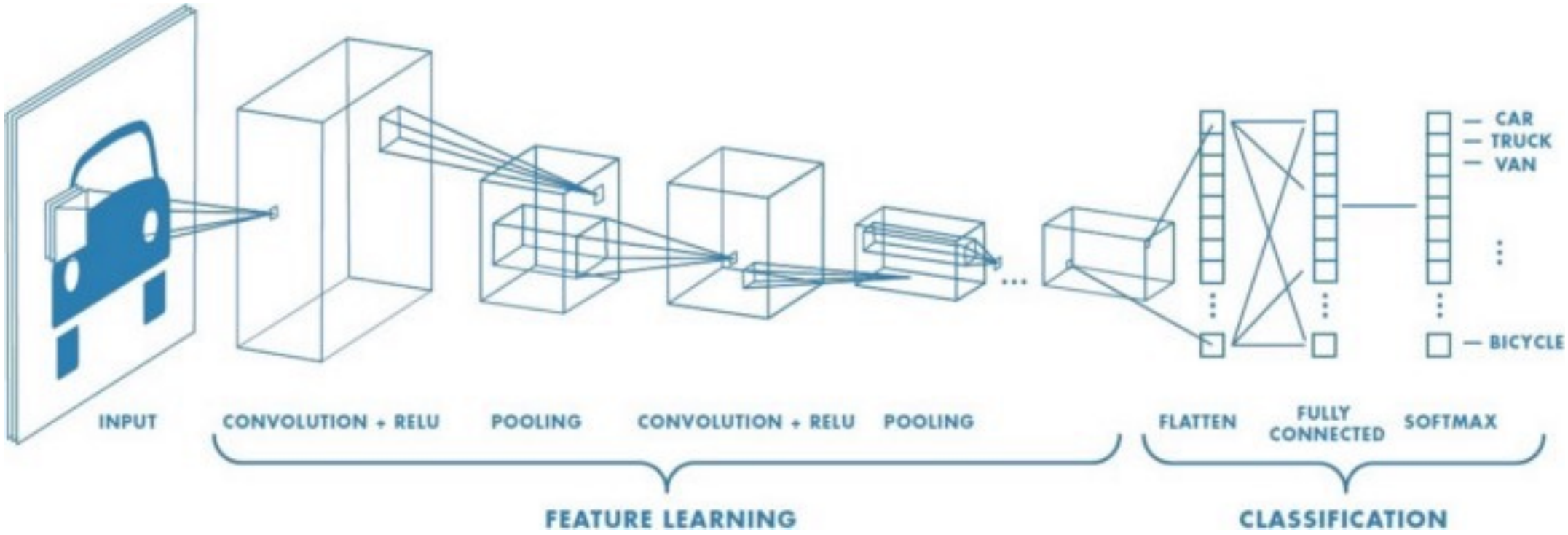
Deep Learning 학습 절차

- Train & evaluate

```
for e in range(epoch):
    with tqdm(train_loader, desc="Train",total=len(train_loader)) as tepoch:
      model.train()
      running_loss=0.0
      for inputs, label in tepoch:
        outputs = model(inputs)
        loss = criterion(outputs,label)
        optimizer.zero_grad()
        loss.backward()
        running_loss += loss.item()
        optimizer.step()
```

```
model.eval()
running_loss=0.0
for inputs, label in tepoch:
  outputs = model(inputs)
  loss = criterion(outputs,label)
```

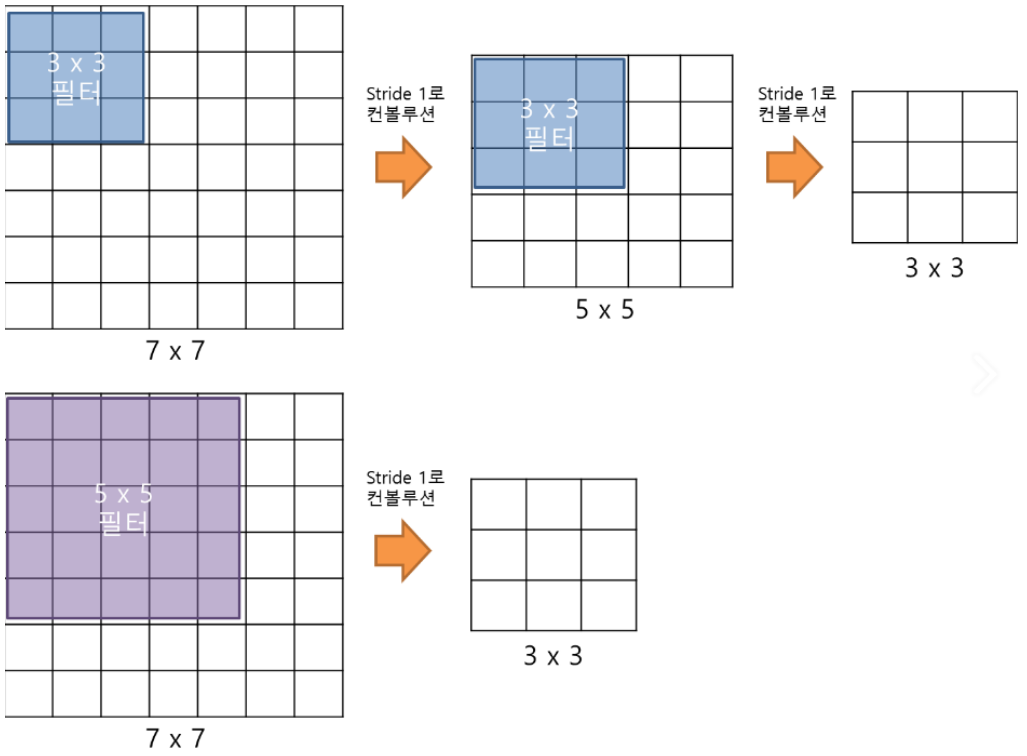
2D image processing history



2D image processing history

- Lenet style -> VGG style
  - LENET은 7x7 filter를 사용
    ➔ 많은 파라미터를 사용
  - VGG 는 7x7대신 3x3 세 번 사용
  - 7x7 filter의 49개의 파라미터
    ➔ 3x3 filter 3개는 9x3개의 파라미터

    49 ➔ 27 개로 획기적으로 파라미터 개선



2D image processing history

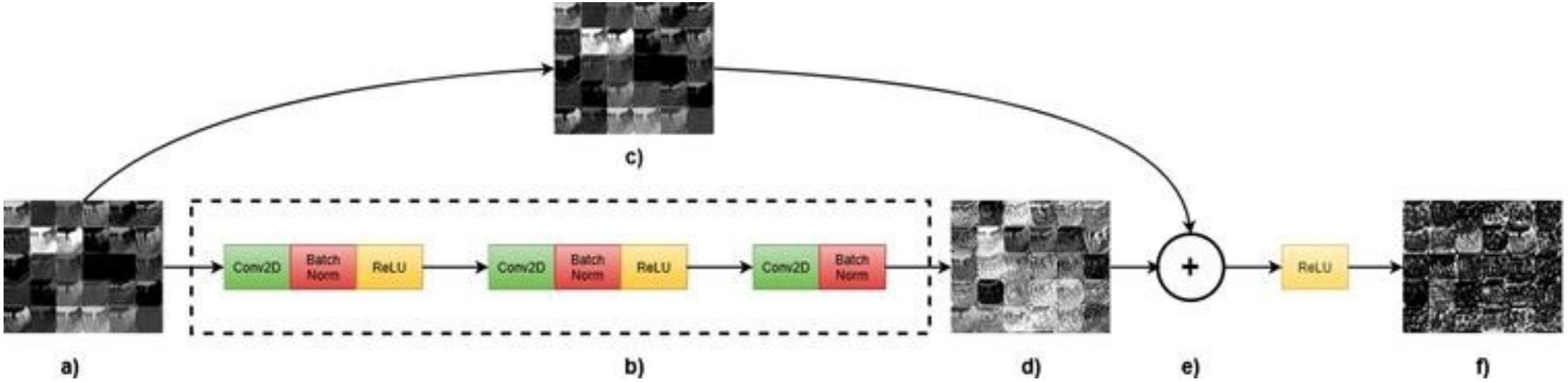
- Lenet style -> VGG style

```python
import torch
class LeNet(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.feature_block = torch.nn.Sequential(
            torch.nn.Conv2d( 1,   6,kernel_size = 5, stride=1),
            torch.nn.Tanh(),
            torch.nn.AvgPool2d(kernel_size=2,stride=2),
            torch.nn.Conv2d( 6,  16,kernel_size = 5, stride=1),
            torch.nn.Tanh(),
            torch.nn.AvgPool2d(kernel_size=2,stride=2),
            torch.nn.Conv2d(16, 120,kernel_siz e= 5, stride=1),
            torch.nn.Tanh(),
            torch.nn.Flatten()
            )
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(120, 84),
            torch.nn.Linear(84,10)
        )

    def forward(self,x):
        x = self.feature_block(x)
        x = self.dense(x)
        return x
```

```python
import torch
class VGGStyle(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.feature_block = torch.nn.Sequential(
            torch.nn.Conv2d( 1,   6,kernel_size = 3, stride=1),
            torch.nn.Conv2d( 6,   6,kernel_size = 3, stride=1),
            torch.nn.Tanh(),
            torch.nn.AvgPool2d(kernel_size=2,stride=2),
            torch.nn.Conv2d( 6,  16,kernel_size = 3, stride=1),
            torch.nn.Conv2d( 16, 16,kernel_size = 3, stride=1),
            torch.nn.Tanh(),
            torch.nn.AvgPool2d(kernel_size=2,stride=2),
            torch.nn.Conv2d(16,  120,kernel_siz e= 3, stride=1),
            torch.nn.Conv2d(120, 120,kernel_siz e= 3, stride=1),
            torch.nn.Tanh(),
            torch.nn.Flatten()
            )
        self.dense = torch.nn.Sequential(
            torch.nn.Linear(120, 84),
            torch.nn.Linear(84,10)
        )

    def forward(self,x):
        x = self.feature_block(x)
        x = self.dense(x)
        return x
```

2D image processing history

- VGG style -> Resnet Style
  - VGG Style은 Deep 하게 쌓을 수록 학습이 안되는 문제를 갖고 있음
  - Conv 연산 전의 Feature를 유지 함으로써 Conv연산을 통해 소실된 데이터를 없애는 특징을 가짐



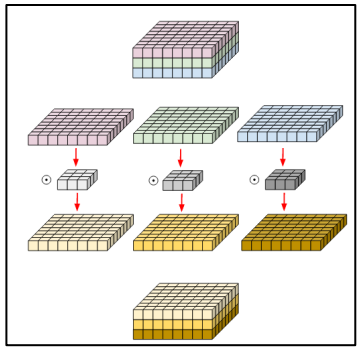
2D image processing history

- VGG style -> Resnet Style

```
import torch
class ResNetStyle(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.feature_block = torch.nn.Sequential(
            ...
            torch.nn.Conv2d(120,120, kernel_size=3, stride = 1)
        )
        self.block = torch.nn.Conv2d(120,120,kernel_size=1, stride = 1)
        self.dense = torch.nn.Sequential(
            ...
        )
    def forward(self,x):
        x = self.feature_block(x)
        x = self.block(x) + x
        x = self.dense(x)

        return x
```

2D image processing history

- Resnet Style -> MB Conv Style
  - Depthwise Network 이용
    - Convolution이 모든 채널의 정보를 이용하는 점 때문에 Spatial한 정보를 취득하지 못하는 문제를 해결하기 위해 적용
  - SE(Squeeze & Excitation) Network를 이용
    - Squeeze
      - 전체 Filter를 대표하는 값을 추출 하는 역할
    - Excitation
      - one-hot activation이 아닌 다중 강조



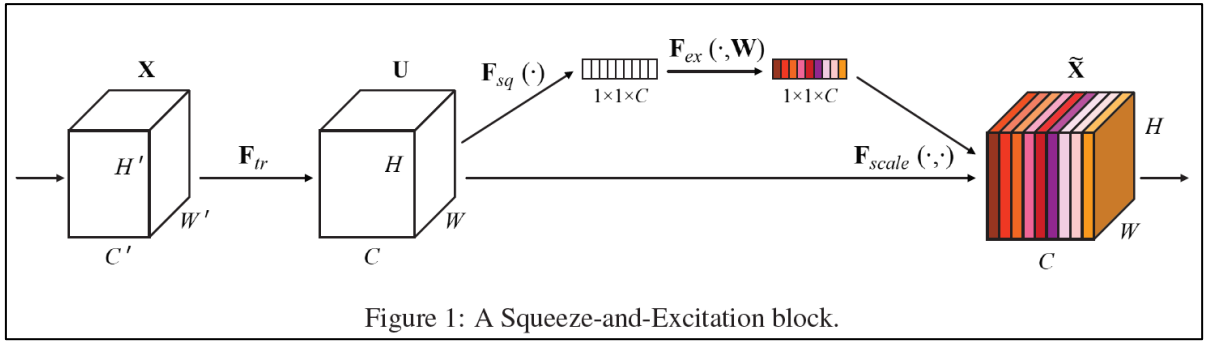
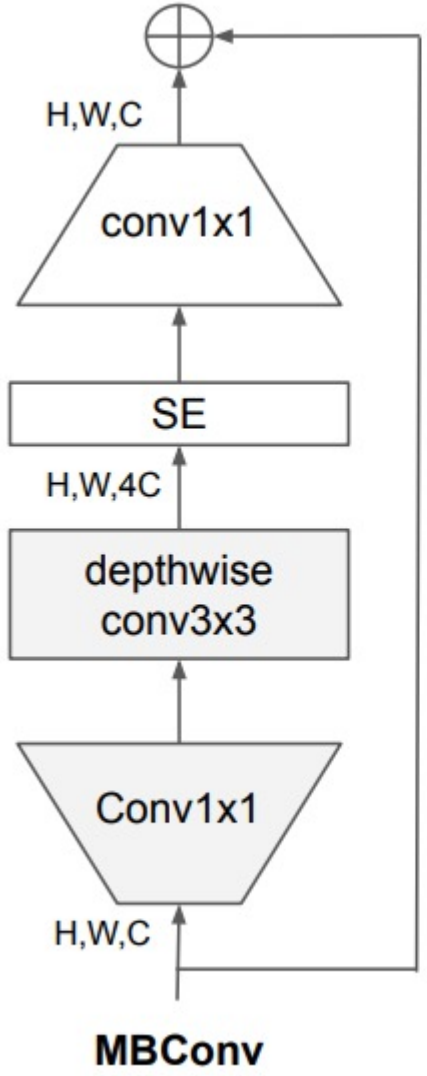


Figure 1: A Squeeze-and-Excitation block.



2D image processing history

- Resnet Style -> MB Conv Style

```python
import torch
from einops import rearrange
from einops.layers.torch import Rearrange, Reduce
class MBconvStyle(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.feature_block = torch.nn.Sequential(
            ...
            torch.nn.Conv2d(120,20, kernel_size=1)
            torch.nn.BatchNorm2d(20),
            torch.nn.SiLU(),
            torch.nn.Covn2d(20,20,kernel_size=3,stride=1),
            SEBlock(dim=20,shrinkage_rate=0.25),
            torch.nn.Conv2d(20,20, kernel_size=1),
            torch.nn.BatchNorm2d(20)

        )
        self.block = torch.nn.Conv2d(inchannel,20,kernel_size=1, stride = 1)
        self.dense = torch.nn.Sequential(
            ...
        )
    def forward(self,x):
        x = self.feature_block(x)
        x = self.block(x) + x
        x = self.dense(x)

        return x
```

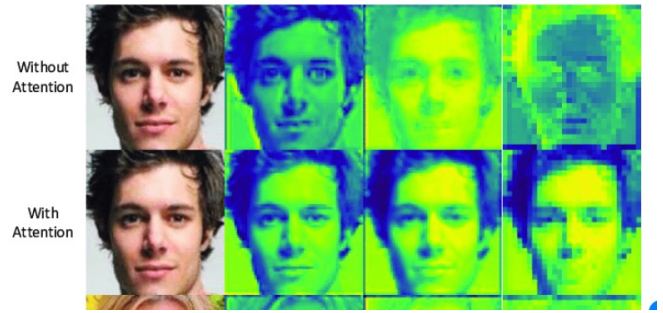
```python
class SEBlock(torch.nn.Module):
    def __init__(self, dim, shrinkage_rate = 0.25):
      super().__init__()
      hidden_dim = int(dim * shinkage_rate)

      self.gate = nn.Sequential(
        Reduce('b c h w -> b c', 'mean'),
        torch.nn.Linear(dim,hidden_dim, bias = False),
        nn.SiLU(),
        nn.Linear(hidden_dim, dim, bias = False),
        nn.Sigmoid(),
        Rearrange('b c -> b c 1 1')
      )

    def forward(self, x):
        return x * self.gate(x)
```

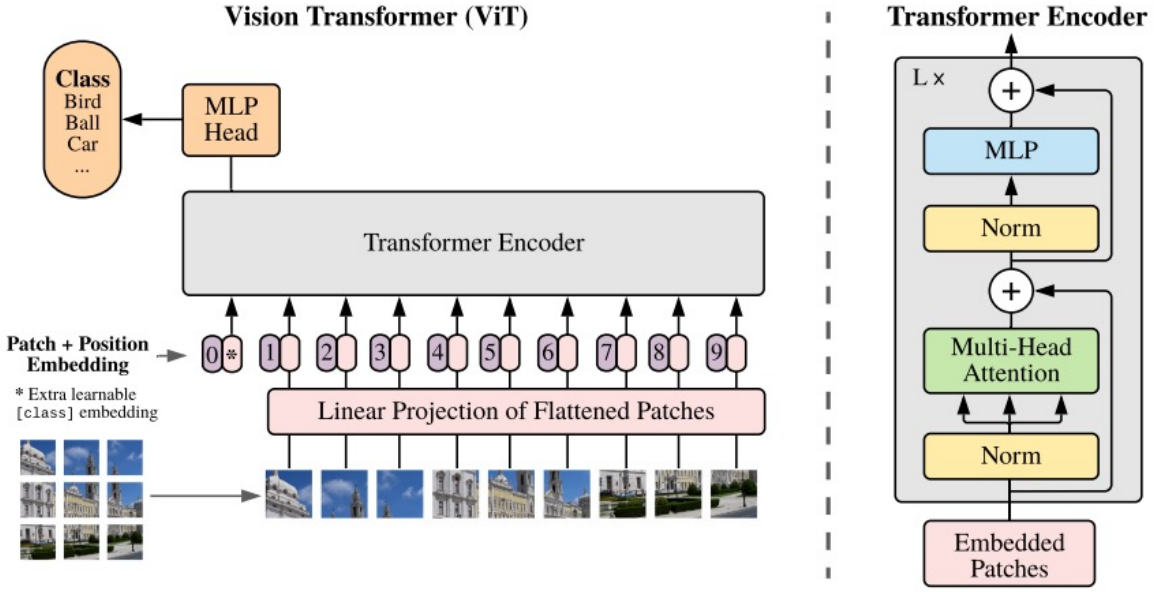
2D image processing history

- MB Conv Style -> ViT Style
  - ViT 이용
    - Convolution이 Local feature를 뽑기 때문에, 전체적인 특징을 뽑아내지 못하는 문제를 해결

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

Table 1: Various relational inductive biases in standard deep learning components. See also Section 2.





2D image processing history

- MB Conv Style -> ViT Style
  - ViT 이용

```python
class ViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads, mlp_dim, pool =
'cls', channels = 3, dim_head = 64, dropout = 0., emb_dropout = 0.):
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)

        num_patches = (image_height // patch_height) * (image_width // patch_width)
        patch_dim = channels * patch_height * patch_width

        self.to_patch_embedding = nn.Sequential(
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = patch_height, p2 = patch_width),
            nn.Linear(patch_dim, dim),
        )

        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)

        self.pool = pool
        self.to_latent = nn.Identity()

        self.mlp_head = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, num_classes)
        )
```

```python
def forward(self, img):
    x = self.to_patch_embedding(img)
    b, n, _ = x.shape

    cls_tokens = repeat(self.cls_token, '1 1 d -> b 1 d', b = b)
    x = torch.cat((cls_tokens, x), dim=1)
    x += self.pos_embedding[:, :(n + 1)]
    x = self.dropout(x)

    x = self.transformer(x)

    x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]

    x = self.to_latent(x)
    return self.mlp_head(x)
```

2D image processing history

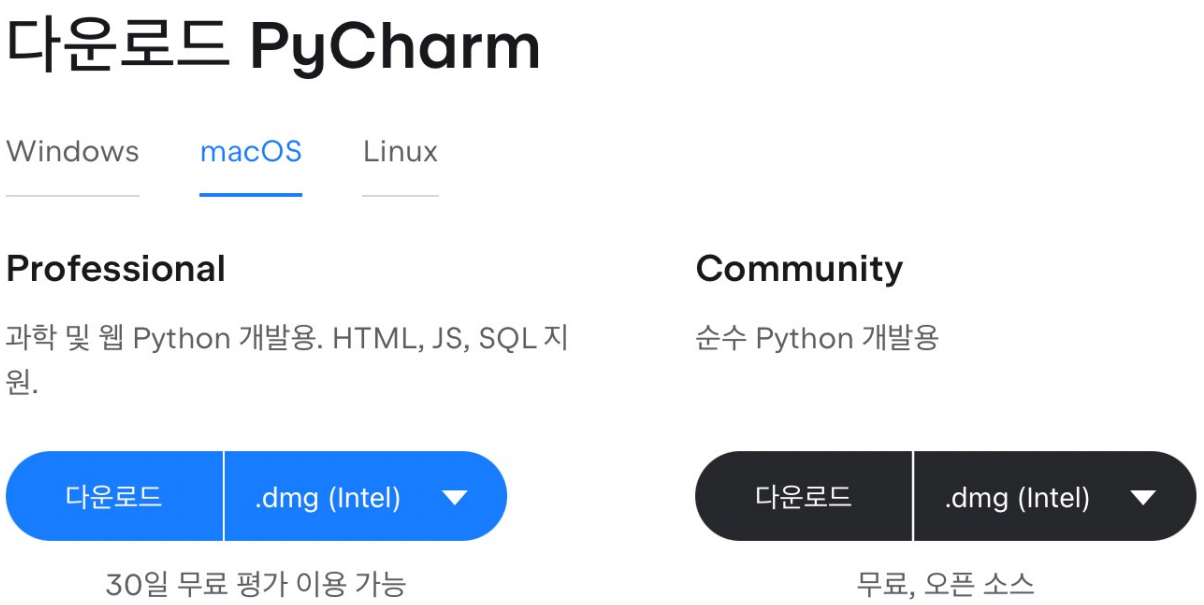
- MB Conv Style -> ViT Style

```python
class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64, dropout = 0.):
        super().__init__()
        inner_dim = dim_head *  heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.attend = nn.Softmax(dim = -1)
        self.dropout = nn.Dropout(dropout)

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        ) if project_out else nn.Identity()

    def forward(self, x):
        qkv = self.to_qkv(x).chunk(3, dim = -1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h = self.heads), qkv)

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

        attn = self.attend(dots)
        attn = self.dropout(attn)

        out = torch.matmul(attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        return self.to_out(out)
```

```python
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout = 0.):
        super().__init__()
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                PreNorm(dim, Attention(dim, heads = heads, dim_head = dim_head, dropout = dropout)),
                PreNorm(dim, FeedForward(dim, mlp_dim, dropout = dropout))
            ]))
    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x
        return x
```

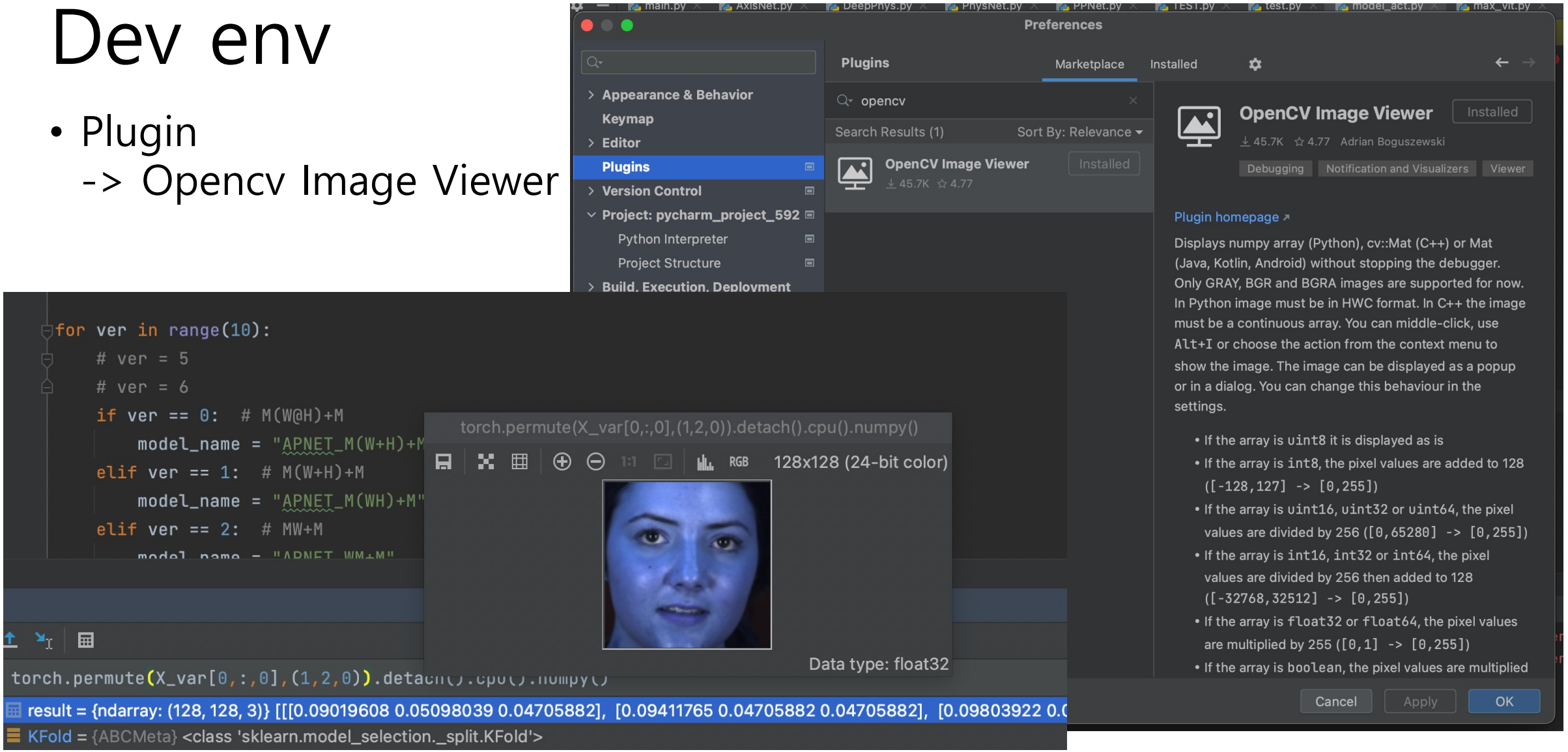
Dev env

- Pycharm professional
  - 학교 계정 활성화 및 사용



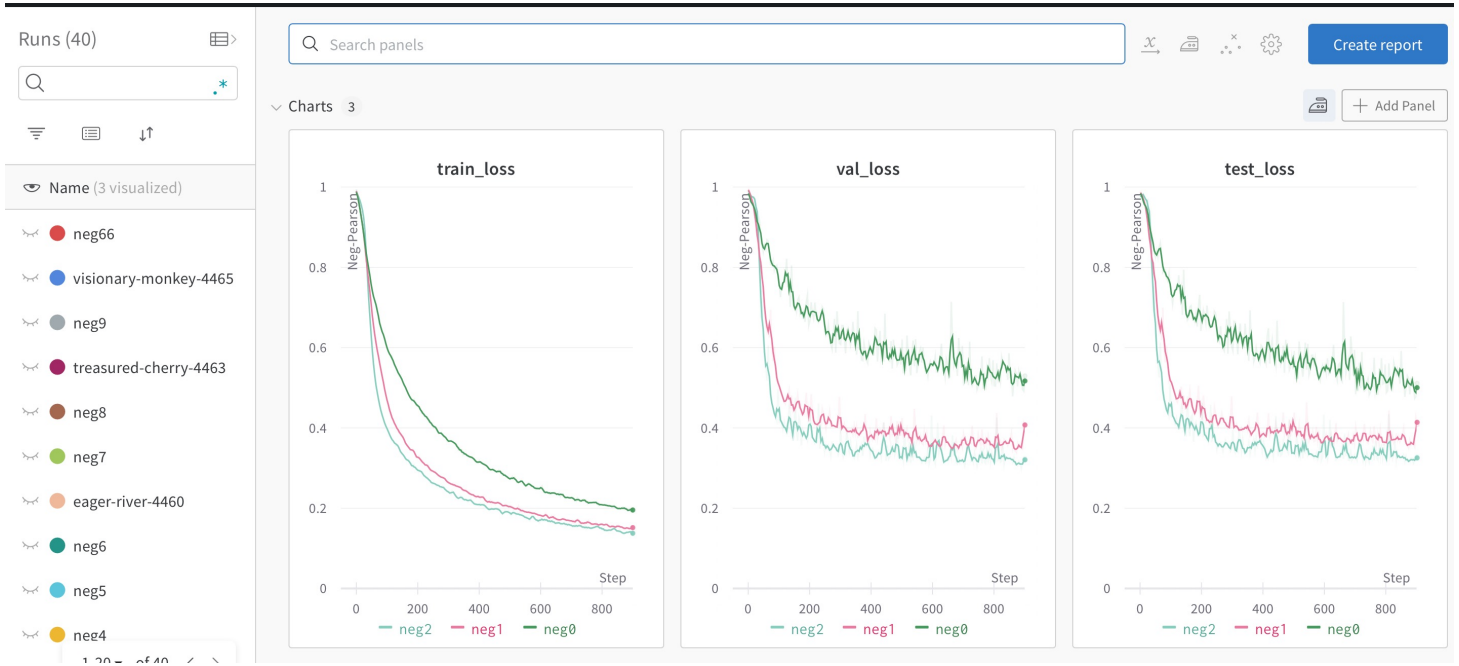
Dev env

- Plugin
  -> Opencv Image Viewer



Dev env

- Package
  - Eniops : torch.reshape, permute 대체 차원 변환이 쉽고 용이
  - Wandb : 학습 모니터링 툴, Hyperparmeter sweep, loss graph, model analysis 등 다양한 편리 기능 제공
  - Tqdm : 파이썬 프로그레스바, 학습 상황 모니터링에 편리
  - h5py : HDF 파일을 파이썬에서 사용 가능하도록 도와주는 파일. 파일의 계층 관리에 용이



Practice & Homework #1

- exp1
  - Linear layer mnist 학습 하기
  - Wandb로 로그 남기기

- Exp2
  - 학습에 유리한 learning_rate 찾기
- Exp3
  - 모델 학습이 잘 되고 있는지 확인하기
- Exp4
  - 모델 레이어 중 어느 부분이 문제가 있는지 확인하기
- Exp5 & Homework
  - Common.py 내의 블록/모델을 활용하여 우수한 성능의 network 만들어 보기
  - 사용한 블록/모델의 관련 논문 서치 및 장/단점 기록
  - Integrated_gradients 를 이용하여 exp3의 최종 산출물과 비교해보기
  - [자유 양식]

REF

- https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/
- https://mc.ai/machine-learning-vs-deep-learning-5/
- https://www.researchgate.net/figure/Effect-of-three-different-edge-detection-filters-Laplacian-Canny-and-Sobel-filters_fig5_236125496
- https://sacko.tistory.cohttps://sacko.tistory.com/10
- https://mc.ai/machine-learning-vs-deep-learning-5/
- https://dev.to/overridedeveloper/machine-learning-at-a-glance-1108
- https://github.com/remotebioseneing/rppgs