Deep Learning

오비컨(OBCon)

2023.07.11 ~ 2023.08.19

Date	Writer	Version	Description
2023.07.11	김계현	0.01	▪ Model 정리
2023.07.12	김계현	0.02	▪ 강화학습 등 추가 정리
2023.07.13	김계현	0.03	▪ RNN 등 추가 정리
2023.07.14	김계현	0.04	▪ AutoML 등 추가 정리
2023.07.23	김계현	0.05	▪ Deep Learning 추가 정리
2023.07.23	김계현	0.06	▪ CNN 추가 정리
2023.08.01	김계현	0.07	▪ Layer 정리
2023.08.02	김계현	0.08	▪ Layer 정리
2023.08.03	김계현	0.09	▪ Layer 정리
2023.08.05	김계현	0.10	▪ CNN 활용 정리
2023.08.07	김계현	0.11	▪ GAN 정리
2023.08.08	김계현	0.12	▪ AutoEncoder 정리
2023.08.10	김계현	0.13	▪ Dense, CNN 추가 정리
2023.08.11	김계현	0.14	▪ Word Embedding 정리
2023.08.15	김계현	0.15	▪ Layout overview
2023.08.17	김계현	0.16	▪ 강화학습 정리
2023.08.19	김계현	0.17	▪ Embedding 정리

목 차

목 차

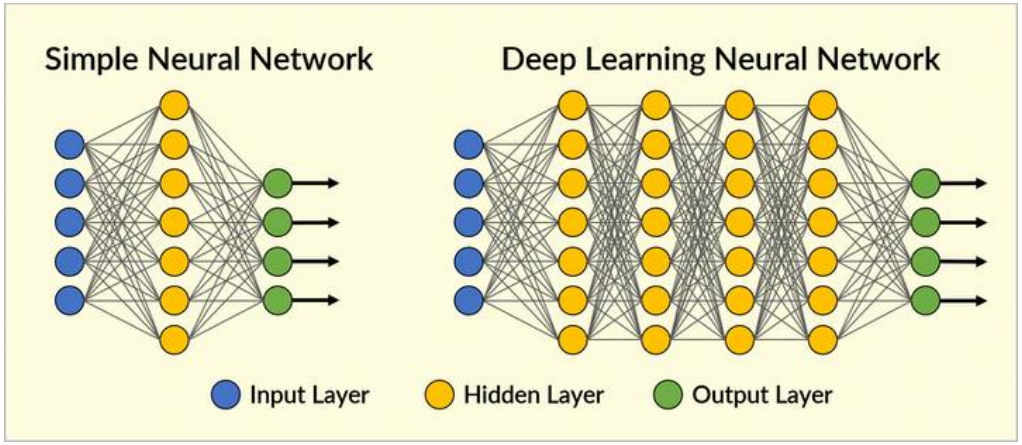
Introduction

Layers

개요

상상은 현실이 된다.

- Model을 구성하는 층
  - 하나 이상의 tensor을 입력 받아 하나 이상의 tensor를 출력 한다.



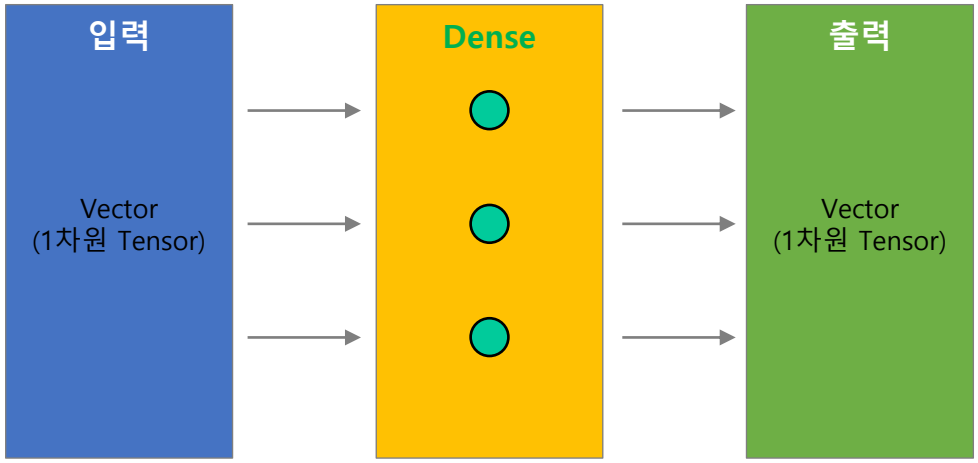
Graph에 비해 단순화된 모형

출처: https://gwoolab.tistory.com/46

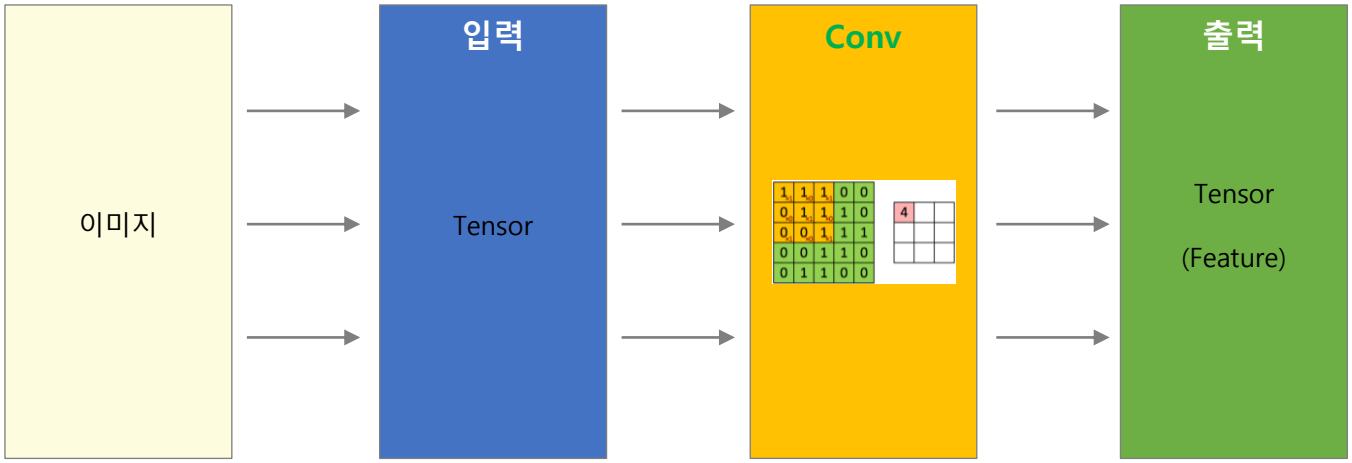
- Layer
  - y = f(x) : 함수
    - 데이터 변환
      - keras.layers.BatchNormalization : input data를 정규화. 학습 속도가 빨라짐
      - keras.layers.Conv2D : Convolution
      - keras.layers.MaxPooling2D : Max Pooling
    - 데이터 가공
      - keras.layers.Dropout : 샘플링 (일부 데이터 삭제)
      - keras.layers.Flatten : 다 차원 데이터를 1차원으로 가공

Layers

Layer Overview : Dense, Conv, RNN



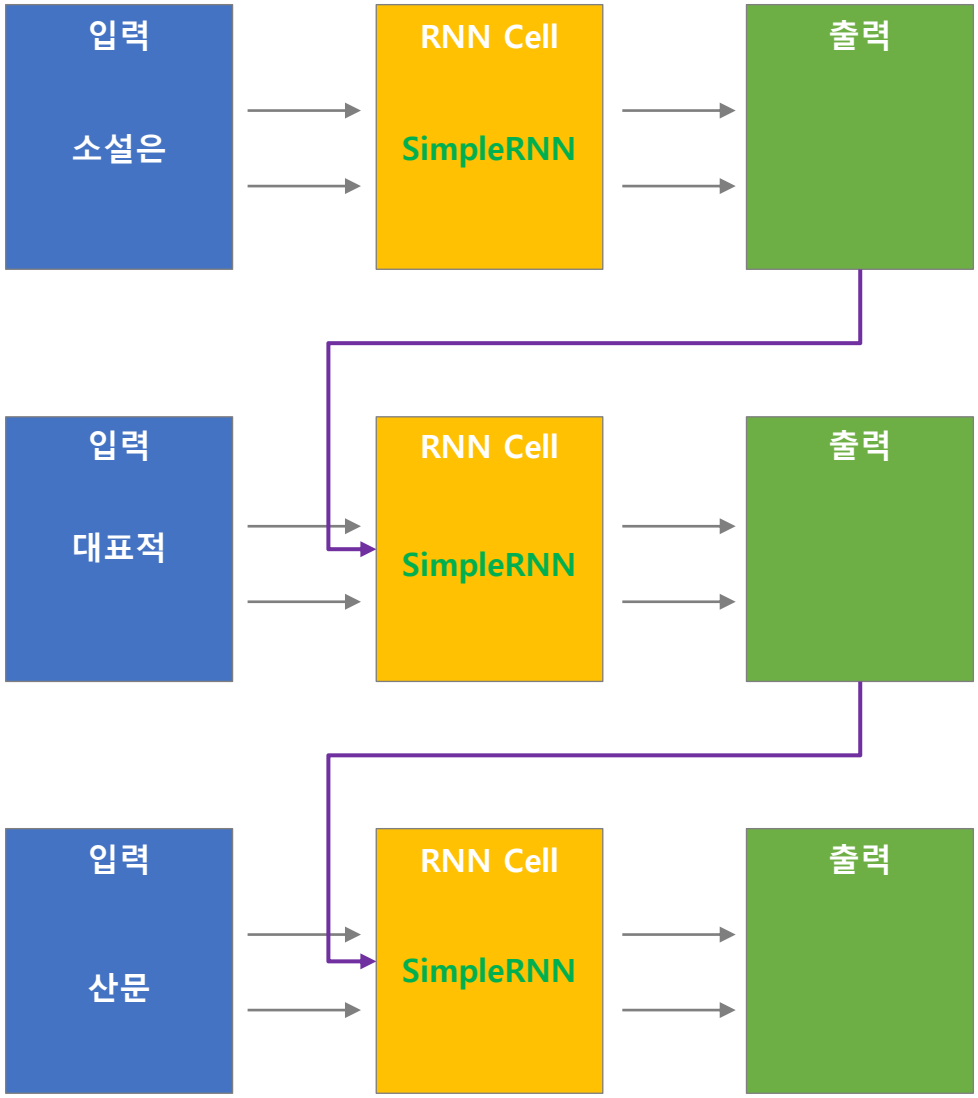
- Dense
  - NN (Neural Network)
  - Fully Connected Network
  - 밀집망
  - y = f(x)



- CNN
  - Convolutional Neural Network
  - 합성곱 신경망
- Conv
  - 입력 tensor에 filte를 적용하여 Feature (특징맵)을 생성
- Pooling
  - 입력을 요약

Layers

Layer Overview : Dense, Conv, RNN



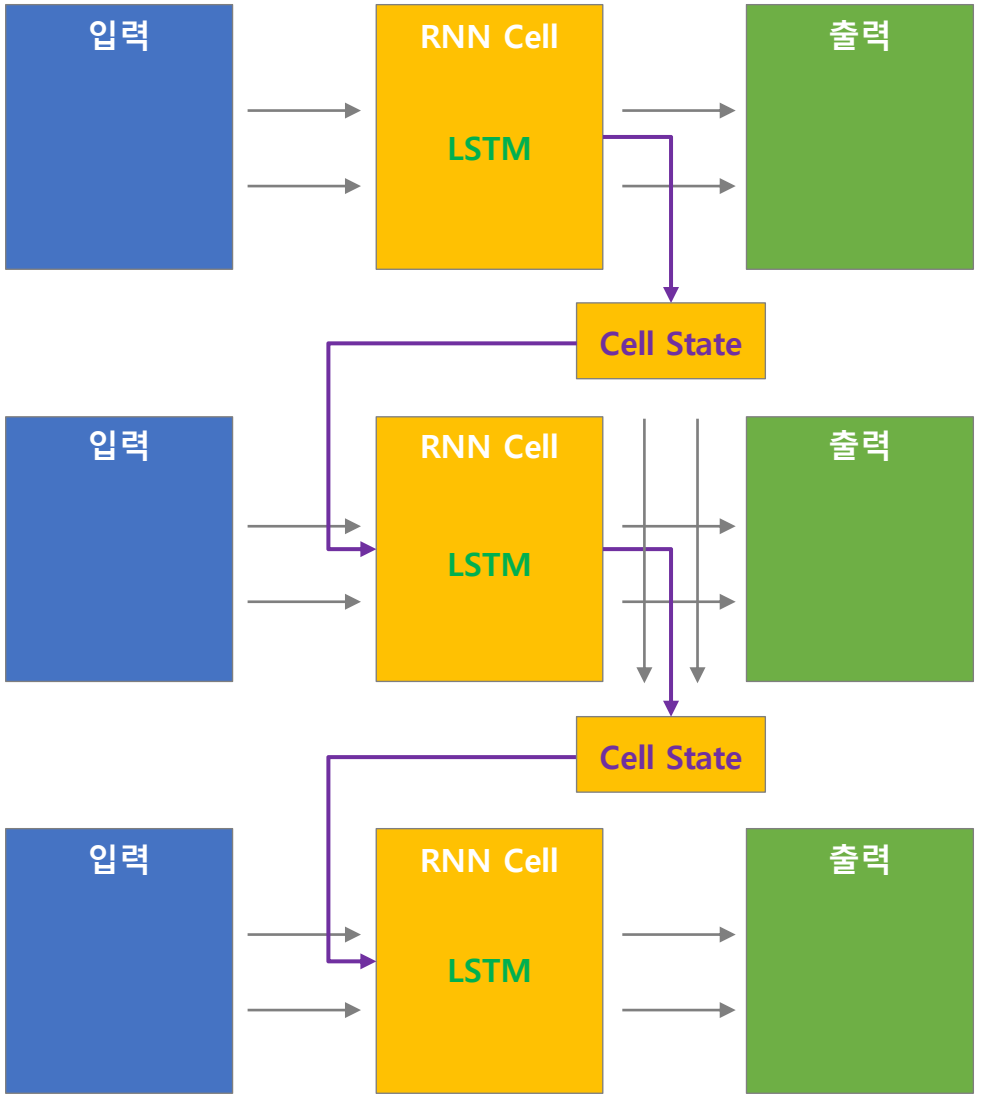
소설은 대표적 산문 문학으로서 근대 이후 많이 사랑받고 있는 문학의 장르이다.

그 종류만 해도 무척 다양하며, 그 만큼 읽고 즐기는 사람들의 계층도 매우 다양하다.

- RNN
  - Recurrent Neural Network
  - 순환 신경망
- SimpleRNN
  - Output을 다음 처리에 반영
- LSTM
  - Long Short-Term Memory
  - Cell State를 별도로 관리
- GRU
  - Gated Recurrent Unit
  - 복잡한 LSTM을 단순화함
  - 성능은 LSTM과 유사함
- Bidirectional
  - 양방향 RNN

Layers

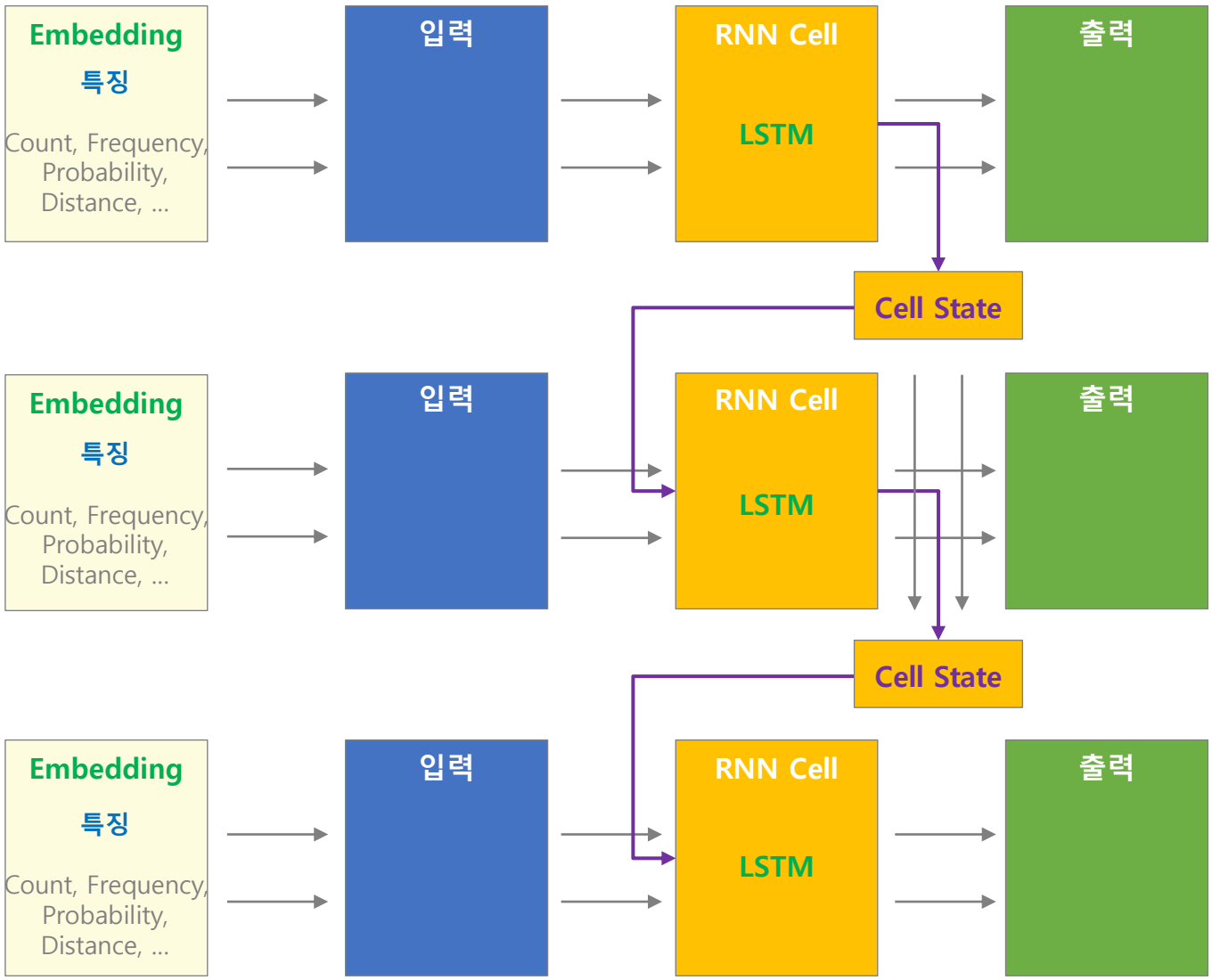
Layer Overview : Dense, Conv, RNN



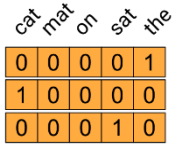
- RNN
  - Recurrent Neural Network
  - 순환 신경망
- SimpleRNN
  - Output을 다음 처리에 반영
- LSTM
  - Long Short-Term Memory
  - Cell State를 별도로 관리
- GRU
  - Gated Recurrent Unit
  - 복잡한 LSTM을 단순화함
  - 성능은 LSTM과 유사함
- Bidirectional
  - 양방향 RNN
  - 순방향 + 역방향

Layers

Layer Overview : Dense, Conv, RNN



- Embedding
  - Text에서 특징 추출
  - OHE (One-Hot Encoding)



출처: https://www.tensorflow.org/text/guide/word_embeddings

  - Count, Frequency, Probability, Distance, ...

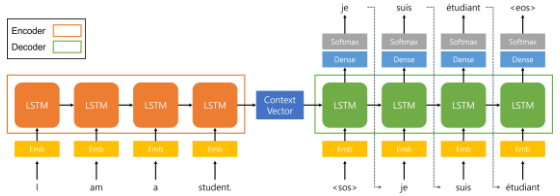
I like deep learning
I like NLP
I enjoy flying

카운트	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

출처: https://wikidocs.net/22885

  - Word2Vec
  - GloVe (Golbal Vectors for word representation, 단어 표현 전역 벡터)

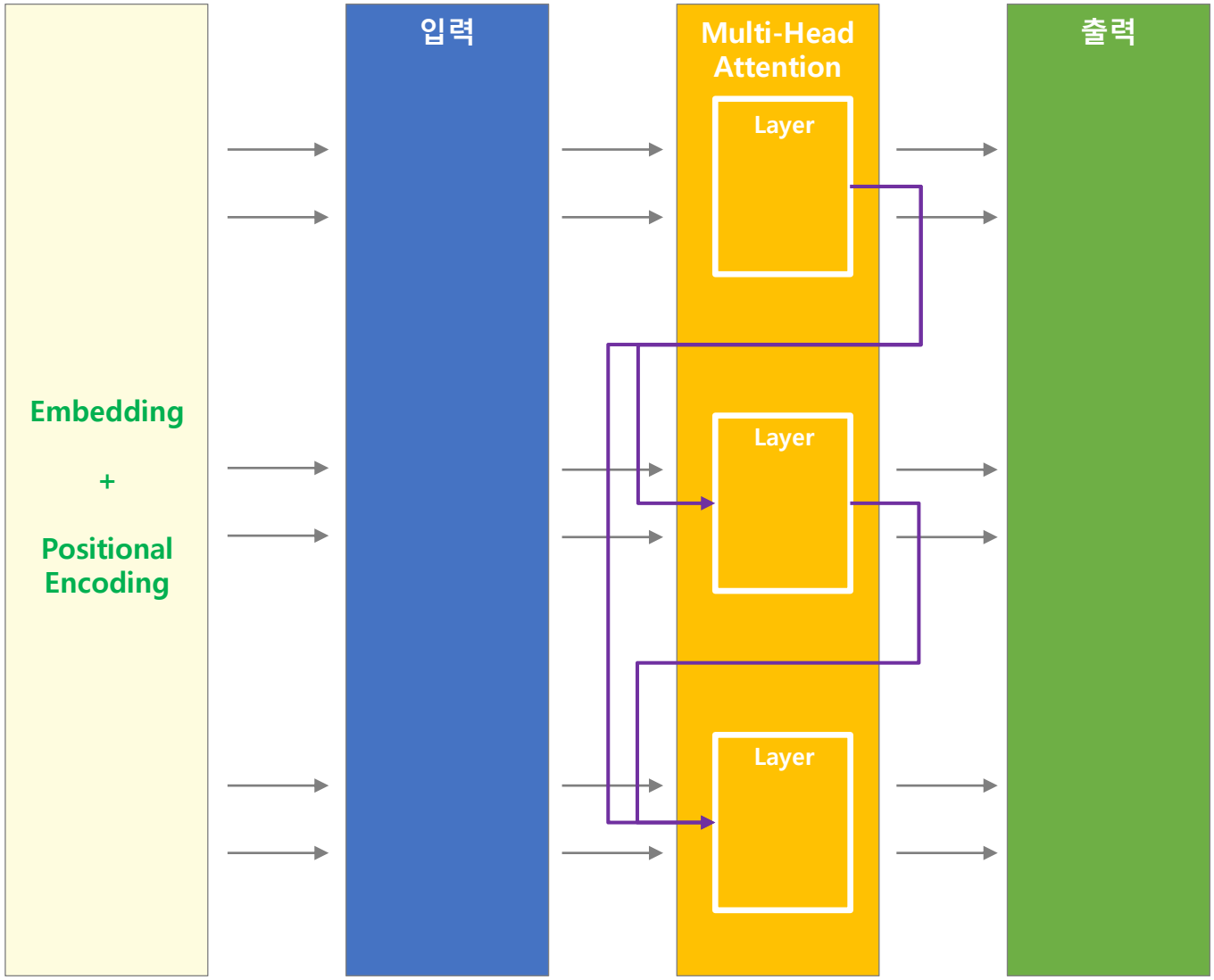
- Seq2Seq



출처: https://velog.io/@jhbale11/어텐션-매커니즘Attention-Mechanism이란-무엇인가

Layers

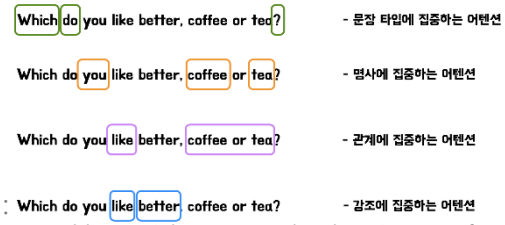
Layer Overview : Dense, Conv, RNN



- **Transfomer**
  - Positinal Encoding을 사용하여 순서를 지정 한다.
  - Attention을 사용하여 RNN을 제거 한다.
  - Encoder와 Decoder를 여러 개 사용

- Positional Encoding
  - 각 단어의 순서 정보

- Attention
  - 유사도를 반영한 가중치를 적용한 입력값
  - 입력과 출력의 차원이 동일
  - Multi-Head Attention



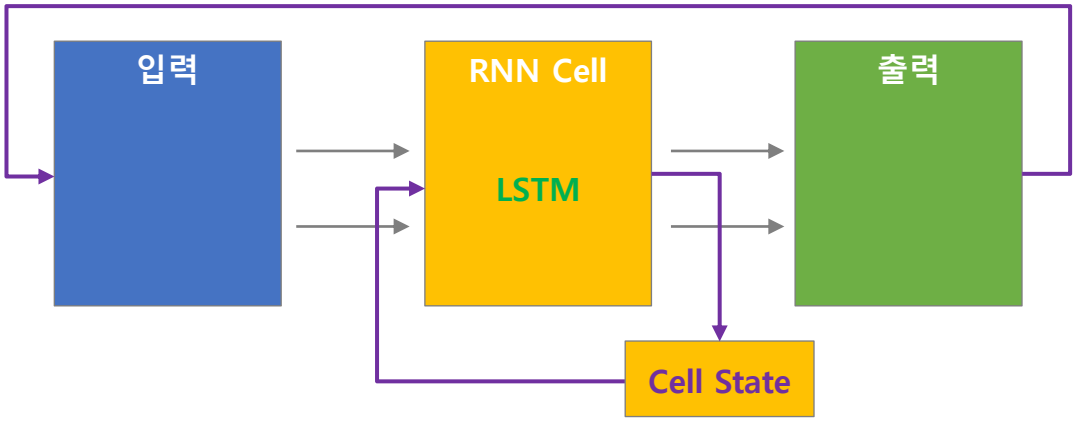
출처: https://www.blossominkyung.com/deeplearning/transformer-mha

- GPT: 순방향으로 Attention 적용
- BERT: 양방향으로 Attention 적용

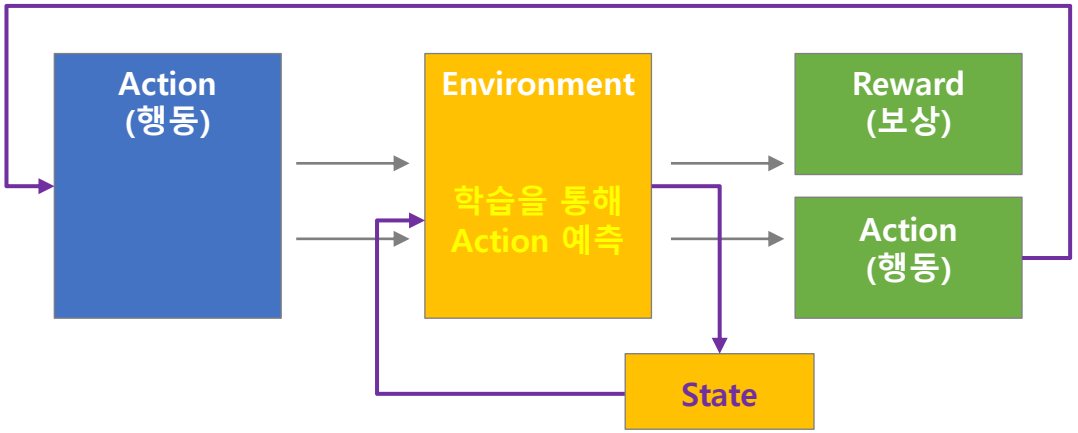
Layers

Layer Overview : Dense, Conv, RNN

- RNN in Decoder



- **DRL (Deep Reinforcement Learning, 심층 강화학습)**



- Environment Model (환경 모델)
  - State (상태)
  - Action (행동)
  - Reward (보상)
    - R (S_{n-1}, A_n, S_n)
- Policy (정책)
  - Policy-based 기법
  - $\pi(S)$ → A
- Return : G_t
  - Value-based 기법
  - 향후 가능한 모든 보상의 할인 총합
  - $V^{\pi}(S)$ → Return
  - $Q^{\pi}(S, A)$ → Return

Layers

Layer 종류 : Dense, Conv, RNN

- Layer
  - Dense : 밀집망

- Dense : 1차원 벡터 처리

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - Conv1D : Convolution 1D
  - Conv2D : Convolution 2D
  - Conv3D : Convolution 3D
  - Conv1DTranspose
  - Conv2DTranspose
  - Conv3DTranspose
  - DepthwiseConv1D
  - DepthwiseConv2D
  - SeparableConv1D
  - SeparableConv2D
  - ZeroPadding1D
  - ZeroPadding2D
  - ZeroPadding3D

- Conv : 공간 정보 추가

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - AveragePooling1D : 평균 Pooling 1D
  - AveragePooling2D : 평균 Pooling 2D
  - AveragePooling3D : 평균 Pooling 3D
  - MaxPooling1D : 최대 Pooling 1D
  - MaxPooling2D : 최대 Pooling 2D
  - MaxPooling3D : 최대 Pooling 3D

  - GlobalAveragePooling1D
  - GlobalAveragePooling2D
  - GlobalAveragePooling3D
  - GlobalMaxPooling1D
  - GlobalMaxPooling2D
  - GlobalMaxPooling3D

- Pooling : 축소

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - RNN
    - SimpleRNN
    - SimpleRNNCell
    - AbstractRNNCell
    - StackedRNNCells
  - LSTM
    - LSTMCell
    - ConvLSTM1D
    - ConvLSTM2D
    - ConvLSTM3D
  - GRU
    - GRUCell
  - Bidirectional : 양방향 RNN
  - TimeDistributed

- RNN : 과거 정보 반영
  - 시계열
- LSTM : RNN의 개선된 종류

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - RandomFlip : 이미지를 수평/수직으로 뒤집기
  - RandomRotation : 이미지를 회전 하기
  - RandomZoom : 이미지를 확대/축소 하기
  - Rescaling
  - Resizing

  - RandomWidth : 이미지 넓이 조정
  - RandomHeight : 이미지 높이 조정
  - RandomBrightness : 이미지 밝기 조정
  - RandomTranslation :
  - RandomContrast :
  - RandomCrop :

  - GaussianNoise : 잡음 추가
  - Reshape : 위상 변경

- Data Augment (데이터 증강)

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - BatchNormalization : Input data를 정규화. 학습 속도가 빨라짐
  - Normalization
  - LayerNormalization
  - UnitNormalization

  - Dropout : 샘플링 (일부 데이터 삭제)
  - AlphaDropout
  - SpatialDropout1D
  - SpatialDropout2D
  - SpatialDropout3D
  - GaussianDropout

  - UpSampling1D
  - UpSampling2D
  - UpSampling3D

  - Flatten : 평탄화. 다 차원 데이터를 1차원으로 가공

- Dense : 1차원 벡터 처리

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - Embedding : Word Embedding
  - TextVectorization

  - Activation : Activation 실행
  - LeakyReLU

  - LocallyConnected1D
  - LocallyConnected2D
  - Cropping1D
  - Cropping2D
  - Cropping3D

  - IntegerLookup
  - StringLookup

Layers

작성중

Layer 종류 : Dense, Conv, RNN

- Layer
  - EinsumDense
  - Lambda
  - Masking
  - CategoryEncoding
  - Discretization
  - Hashing
  - CenterCrop
  - ActivityRegularization
  - Permute
  - RepeatVector
  - Wrapper

Layers

작성중

Layer 종류 : Dense, Conv, RNN

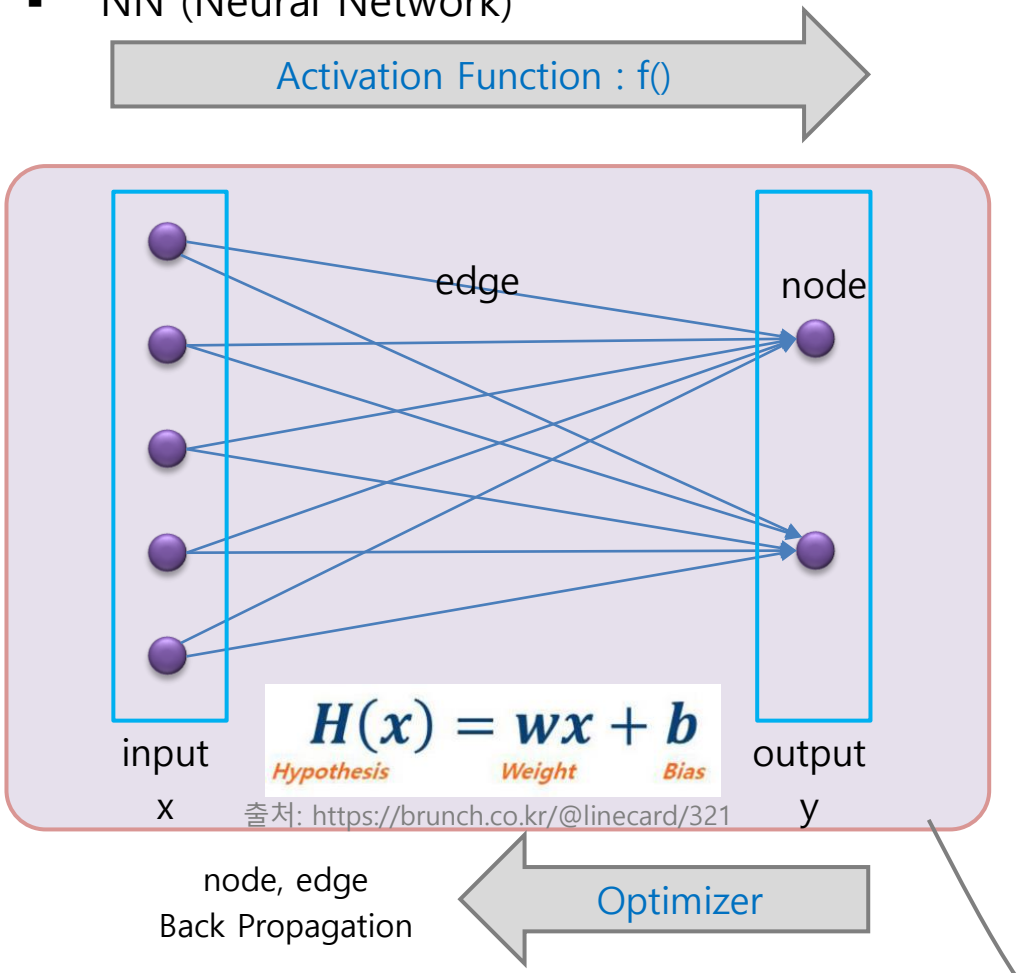
- Layer
  - Add
  - Subtract
  - Multiply
  - Maximum
  - Minimum
  - Average

  - Concatenate
  - Dot

Layers

Layer 종류 : keras.layers.Dense (밀집망)

- NN (Neural Network)



- Parameter 갯수
  - edge, node
  - x * y + y
- 작업량
  - Optimizer
  - Loss Function
  - Activation Function
  - Predict
- Model 비교 기준
  - 속도
  - 가중치 크기
- x = f'(y) : 전치 함수
  - 역행렬
- 장점
  - Loss Function을 사용하여 학습이 가능함
- 단점
  - 이미지 분류시 위상 정보를 활용하지 못함

Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Activation Functions (활성화 함수)
  - https://reniew.github.io/12/

  - sigmoid : Sigmoid, [0, 1] : 이진 분류
  - hard_sigmoid

  - tanh : TanH, [-1, 1] ← Sigmoid 개선 : 이진 분류
  - softmax : Softmax, Output의 총합은 1 (확률) : 다중 분류

  - relu : ReLU (Rectified Linear Unit), [0, x] : 연산이 빠름
  - LeakyReLU : [0.01x, x] ← ReLU 개선
  - PReLU : [ax, x] ← Leakly ReLU 개선
  - elu : ELU (Exponential Linear Unit), [$a(e^x - 1)$, x] ← TanH + ReLU
  - ThresholderdReLU

  - linear : Linear, 입력값을 그대로 출력

Layers

작성중

Layer 종류 : keras.layers.Dense (밀집망)

- Activation Functions (활성화 함수)
  - exponential
  - gelu
  - selu
  - softplus
  - softsign
  - swish

Layers

작성중

Layer 종류 : keras.layers.Dense (밀집망)

- Loss Functions (손실 함수)
  - binary_crossentropy
  - binary_focal_crossentropy
  - categorical_crossentropy : 분류시 사용
  - sparse_categorical_crossentropy

  - mean_squared_error : MSE : 예측시 사용
  - mean_squared_logarithmic_error : MSLE
  - mean_absolute_error : MAE
  - mean_absolute_percentage_error : MAPE

  - categorical_hinge
  - cosine_similarity
  - hinge
  - huber
  - kl_divergence
  - log_cosh
  - log_cosh as logcosh
  - poisson
  - squared_hinge

Layers

작성중

Layer 종류 : keras.layers.Dense (밀집망)

- Optimizer (최적화)
  - experimental
  - legacy
  - schedules

adam	: Adam	: 예측시 사용
nadam	: Nadam	
adamax	: Adamax	
adadelta	: Adadelta	
adagrad	: Adagrad	: 학습률을 조정하여 학습
ftrl	: Ftrl	
gradient_descent	: SGD (Stochastic Gradient Descent, 확률적 경사 하강법)	
rmsprop	: RMSprop	

Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Matric (척도)
  - https://ek-koh.github.io/data analysis/evaluation/

  - Accuracy : 정확도 (예측과 실제값이 같은 비율)
  - BinaryAccuracy
  - CategoricalAccuracy
  - binary_accuracy
  - categorical_accuracy
  - sparse_categorical_accuracy
  - sparse_top_k_categorical_accuracy
  - top_k_categorical_accuracy
  - SparseCategoricalAccuracy
  - SparseTopKCategoricalAccuracy
  - TopKCategoricalAccuracy

  - Precision : 정밀도 (예측이 a인 경우, 예측과 실제값이 a이 비율)
  - PrecisionAtRecall

  - Recall : 재현율 (실제값이 a인 경우, 예측과 실제값이 a이 비율)
  - RecallAtPrecision

Layers

작성중

Layer 종류 : keras.layers.Dense (밀집망)

- Matric (척도)
  - Mean
  - MeanMetricWrapper
  - MeanTensor
  - Metric
  - Sum

  - AUC
  - BinaryCrossentropy
  - BinaryIoU
  - CategoricalCrossentropy
  - CategoricalHinge
  - CosineSimilarity
  - FalseNegatives
  - FalsePositives
  - Hinge
  - IoU
  - KLDivergence

Layers

작성중

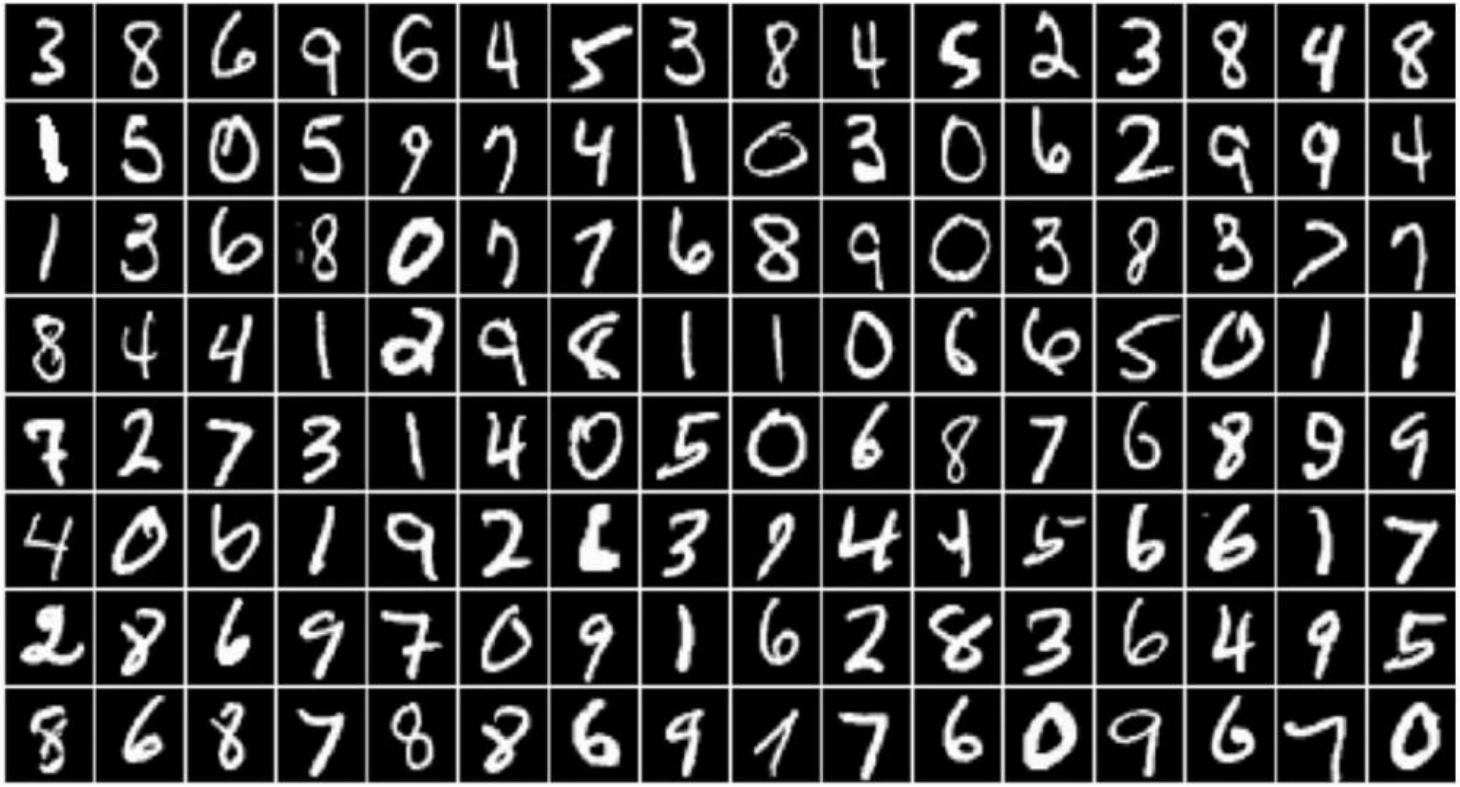
Layer 종류 : keras.layers.Dense (밀집망)

- Matric (척도)
  - LogCoshError
  - MeanAbsoluteError
  - MeanAbsolutePercentageError
  - MeanIoU
  - MeanRelativeError
  - MeanSquaredError
  - MeanSquaredLogarithmicError
  - OneHotIoU
  - OneHotMeanIoU
  - Poisson
  - RootMeanSquaredError
  - SensitivityAtSpecificity
  - SparseCategoricalCrossentropy
  - SpecificityAtSensitivity
  - SquaredHinge
  - TrueNegatives
  - TruePositives

Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Input
  - 숫자 손글씨
  - 28 * 28



출처: https://learnopencv.com/implementing-mlp-tensorflow-keras/

Layers

Layer 종류 : keras.layers.Dense (밀집망)

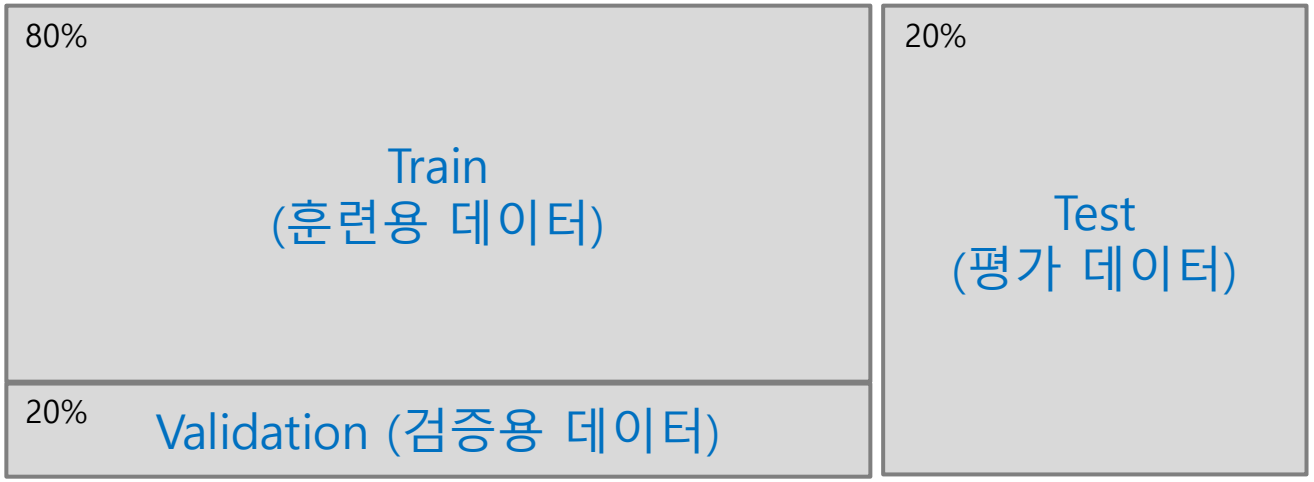
- 데이터 가공
  - Nomalization
  - Layer에서 데이터 가공과 다른 점은 "**한번만 실행된다**"는 것 이다.

- 훈련 데이터 : train
  - 훈련용 : train : 80%
  - 검증용 : validation : 20%

- 평가 데이터 : test

- ephochs
  - 훈련 집합 횟수

- batch_size
  - 훈련 집합당 훈련 횟수



Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Model

```
model = keras.models.Sequential()
#--- nb_classes = 28 * 28
model.add(keras.layers.Dense(self.nb_classes, input_shape=(reshaped,), activation='softmax'))

model.compile(
    optimizer=self.optimizer,                    #--- Adam
    loss=self.loss_function,                     #--- categorical_crossentropy
    metrics=[ self.metrics ]                     #--- accuracy
)
```

Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Output

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                7850                #--- (28 * 28) * 10 + 10

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
```

Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Output

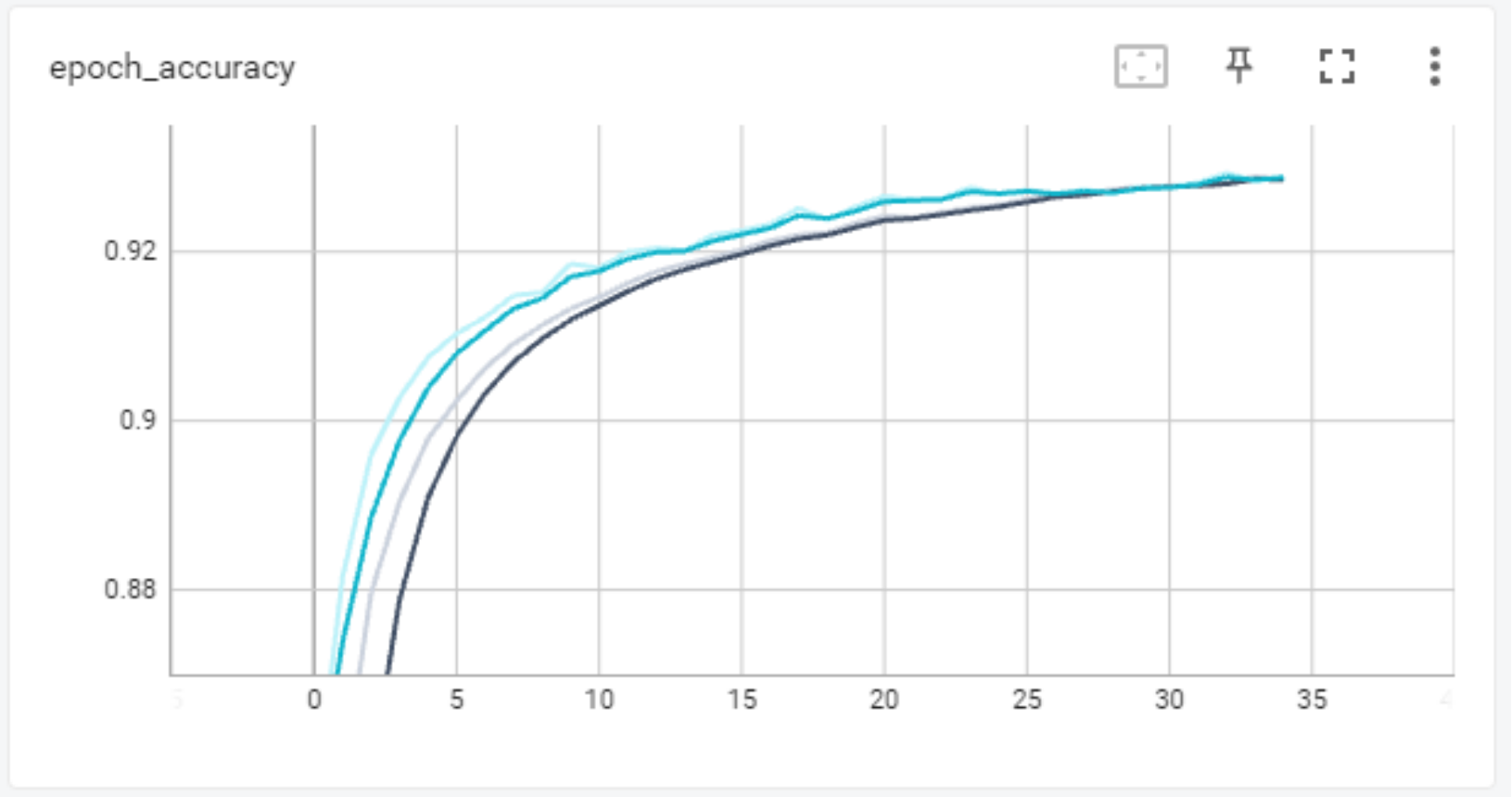
```
Epoch 1/35
94/94 [==============================] - 1s 4ms/step - loss: 1.2428 - accuracy: 0.6838 - val_loss: 0.7018 - val_accuracy: 0.8438
Epoch 2/35
94/94 [==============================] - 0s 3ms/step - loss: 0.6172 - accuracy: 0.8523 - val_loss: 0.4953 - val_accuracy: 0.8803
... 생략 ...
Epoch 34/35
94/94 [==============================] - 0s 3ms/step - loss: 0.2570 - accuracy: 0.9283 - val_loss: 0.2617 - val_accuracy: 0.9294
Epoch 35/35
94/94 [==============================] - 0s 3ms/step - loss: 0.2555 - accuracy: 0.9287 - val_loss: 0.2627 - val_accuracy: 0.9287


313/313 [==============================] - 0s 1ms/step - loss: 0.2669 - accuracy: 0.9269
Test accuracy: 0.9269000291824341
```

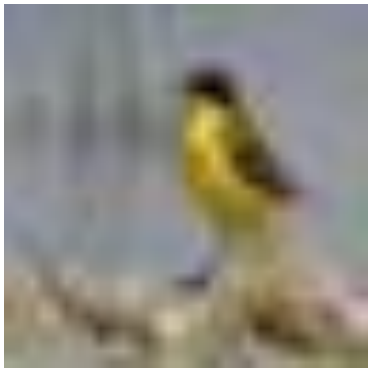
Layers

Layer 종류 : keras.layers.Dense (밀집망)

- Graph



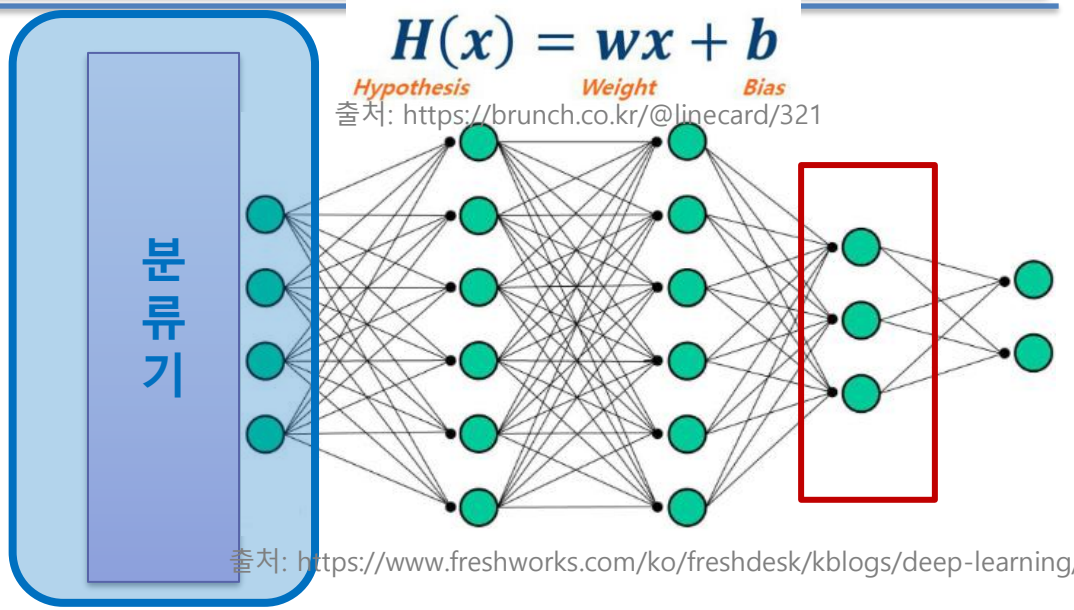
Deep Learning 응용



60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

출처: https://www.cs.toronto.edu/~kriz/cifar.html



출처: https://brunch.co.kr/@linecard/321

출처: https://www.freshworks.com/ko/freshdesk/kblogs/deep-learning/

생성기

y = f(x)
= softmax(H(x))
= softmax(wx + b)

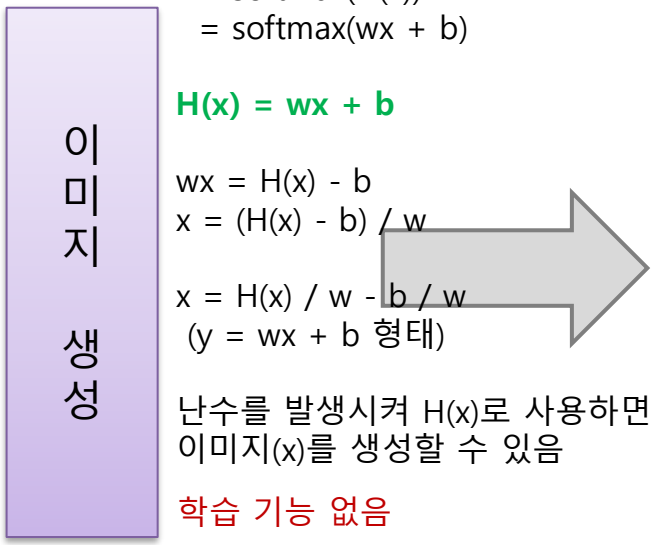
H(x) = wx + b

wx = H(x) - b
x = (H(x) - b) / w

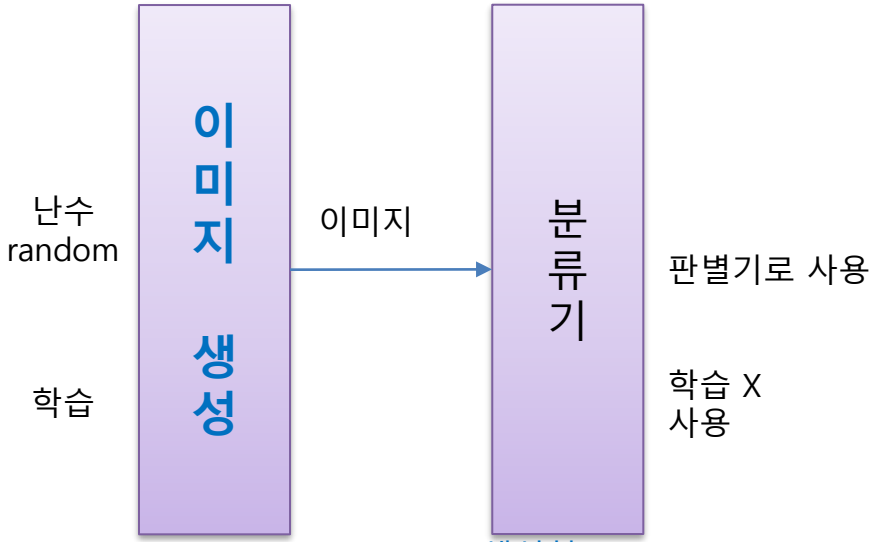
x = H(x) / w - b / w
(y = wx + b 형태)

난수를 발생시켜 H(x)로 사용하면
이미지(x)를 생성할 수 있음

학습 기능 없음



생성기 → 분류기



Generative AI (생성형 AI)

Deep Learning 응용

60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

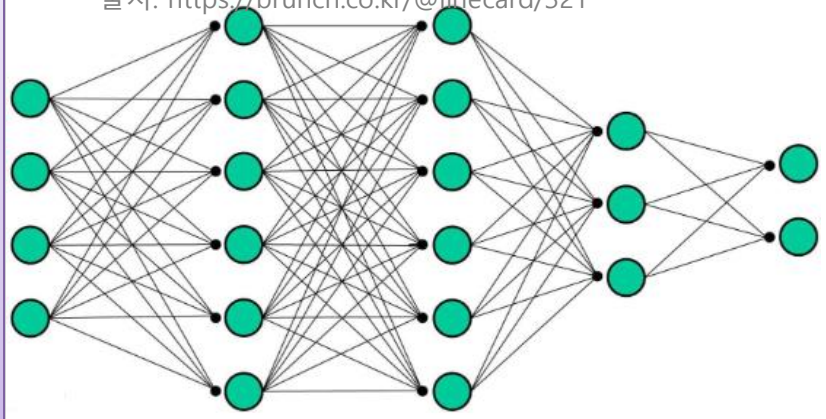
출처: https://www.cs.toronto.edu/~kriz/cifar.html

분류기

$$H(x) = wx + b$$

Hypothesis Weight Bias

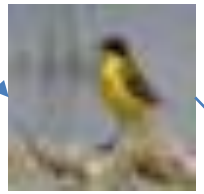
출처: https://brunch.co.kr/@linecard/321



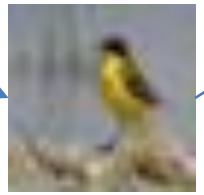
출처: https://www.freshworks.com/ko/freshdesk/kblogs/deep-learning/

생성기 → 판별기

생성기



이미지



생성한 이미지

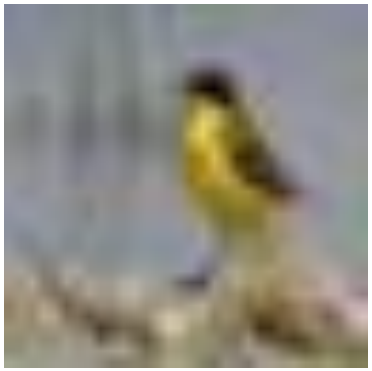
판별기

이미지를 생성하는 모델과
이미지를 판별하는 모델을
모두 학습 시킨다

x : 이미지 또는 생성한 이미지
y : 사진, 기계가 생성한 이미지

GAN (Generative Adversarial Networks, 생성적 적대 신경망)

Deep Learning 응용



60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

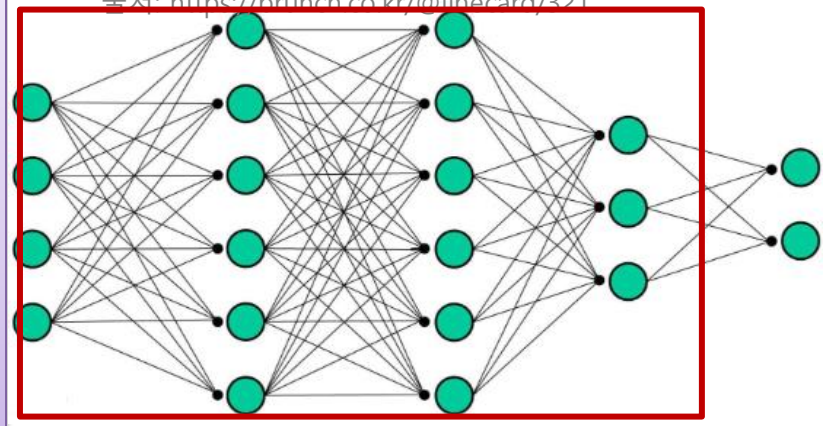
출처: https://www.cs.toronto.edu/~kriz/cifar.html

분류기

$$H(x) = wx + b$$

Hypothesis Weight Bias

출처: https://brunch.co.kr/@linecard/321



출처: https://www.freshworks.com/ko/freshdesk/kblogs/deep-learning/

생성기

이미지 생성

y = f(x)
= softmax(H(x))
= softmax(wx + b)

H(x) = wx + b

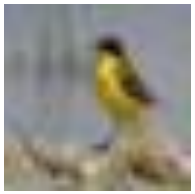
wx = H(x) - b
x = (H(x) - b) / w

x = H(x) / w - b / w
(y = wx + b 형태)

난수를 발생시켜 H(x)로 사용하면 이미지(x)를 생성할 수 있음

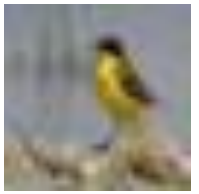
학습 기능 없음

Encoder → Decoder



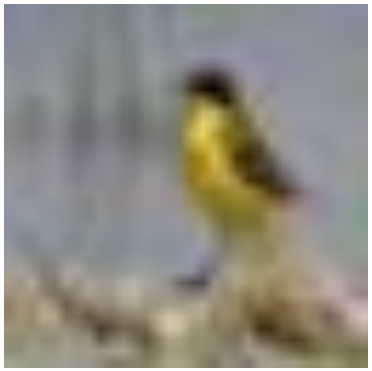
인코더

디코더



원본 이미지와 생성된 이미지의 유사성으로 학습 가능

AutoEncoder



60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

출처: https://www.cs.toronto.edu/~kriz/cifar.html

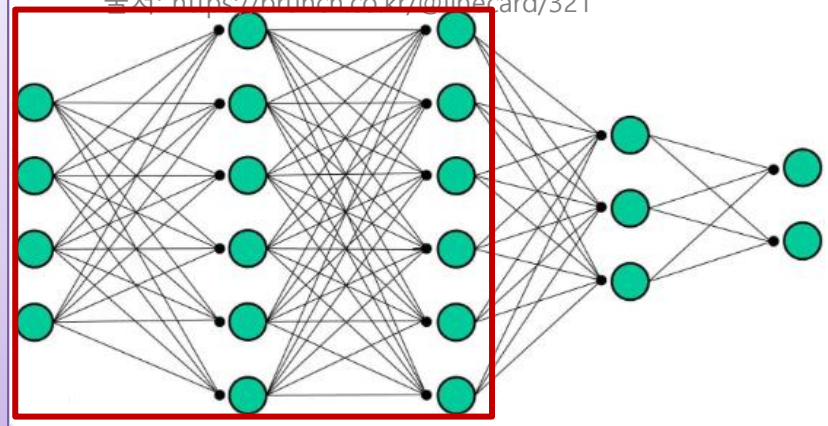
$$H(x) = wx + b$$

Hypothesis Weight Bias

출처: https://brunch.co.kr/@linecard/321

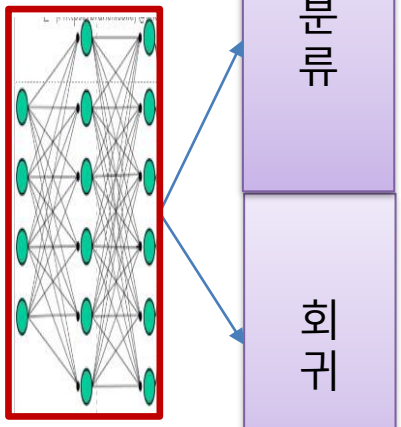
회귀

x, y, width, height



출처: https://www.freshworks.com/ko/freshdesk/kblogs/deep-learning/

분류와 지역화



분류

회귀

x, y, width, height

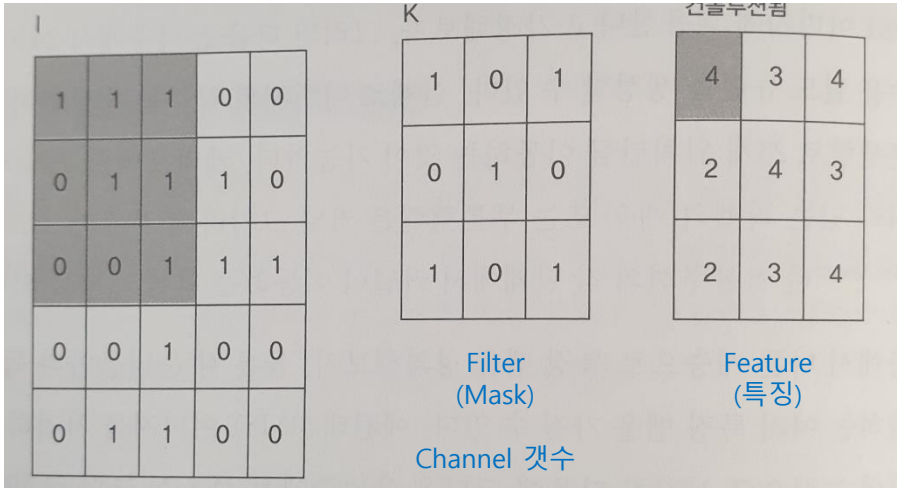
- 객체 탐지 (분류 + 회귀)
  - 동물, 동물의 위치

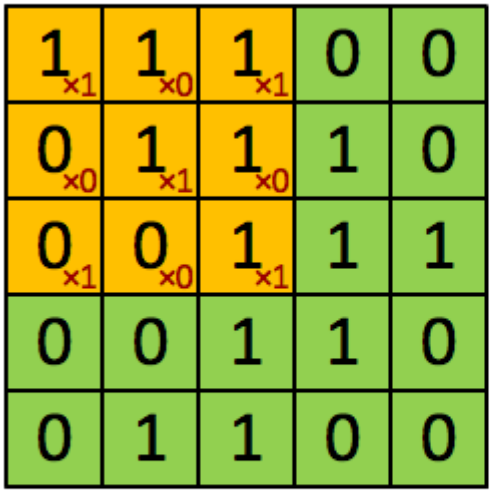
Layers

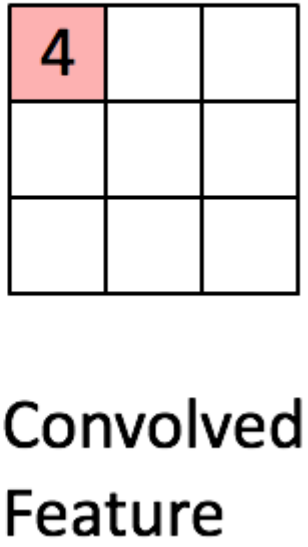
Layer 종류 : keras.layers.Conv2D

- Convolution : 관계 정보를 활용하는 방법
  - Dense : 선형 (1차원 배열)
  - Convolution : n차원 부분 행열
    - 행열에 Convolution을 적용하여 특징맵을 생성
    - 2차원 : 이미지, 오디오, 텍스트
    - 3차원 : 비디오

- Transpose Convolution
  - 전치 컨볼루션
  - 입력과 출력을 반전





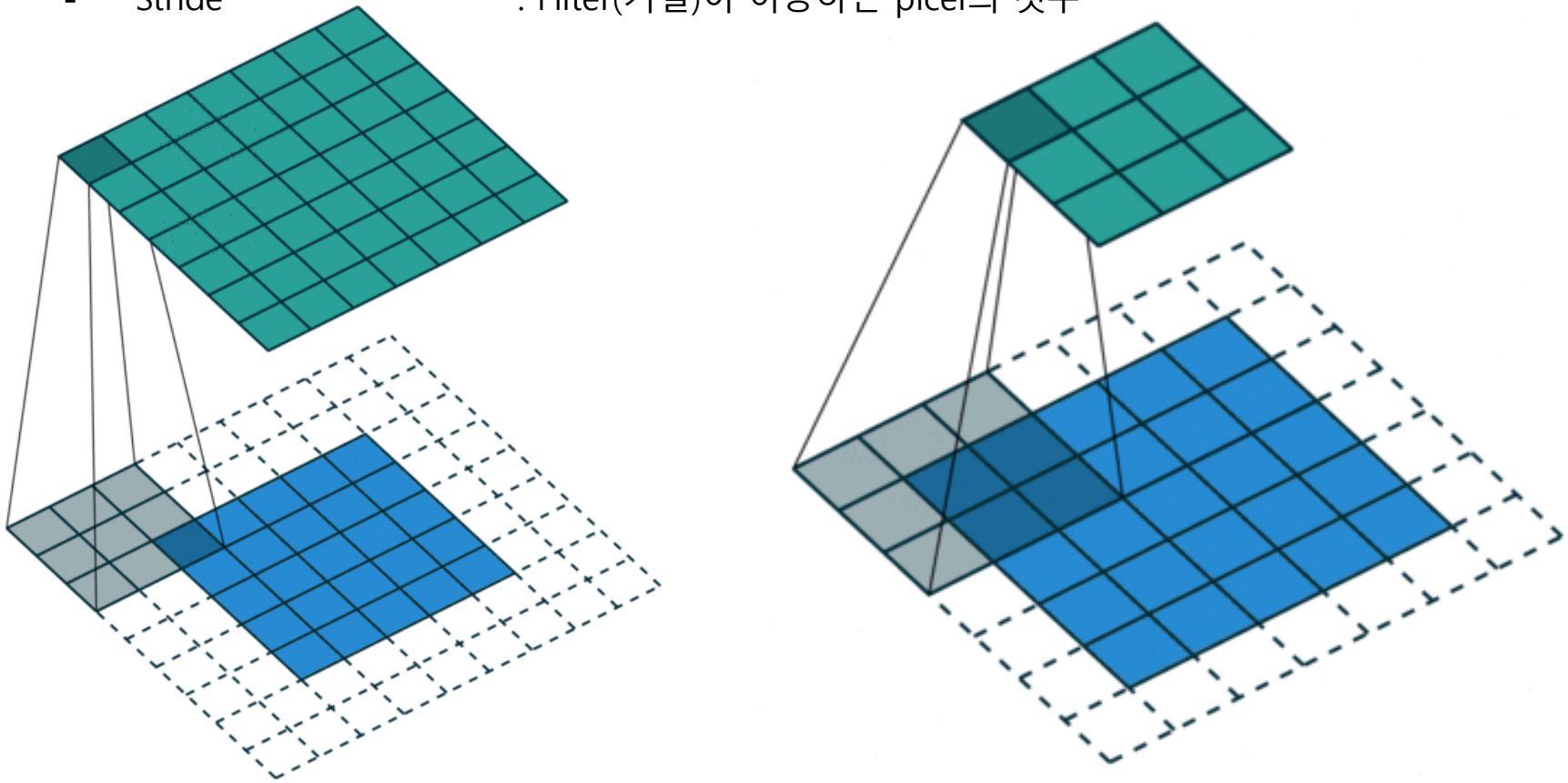


출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

Layers

Layer 종류 : keras.layers.Conv2D

- Convolution
  - Padding : 주로 입력과 출력의 크기를 동일하도록 조정
  - Stride : Filter(커널)이 이동하는 picel의 갯수



출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

Layers

Layer 종류 : keras.layers.Conv2D

- Convolution

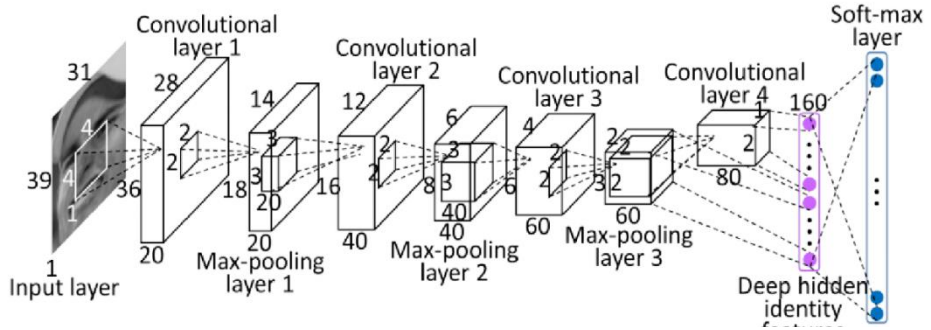
- Filter
  - 특정 효과를 가지는 filter에 대한 학습이 가능 하다.
  - 특정 이미지만을 위한 filter가 생성 된다.

출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

Layers

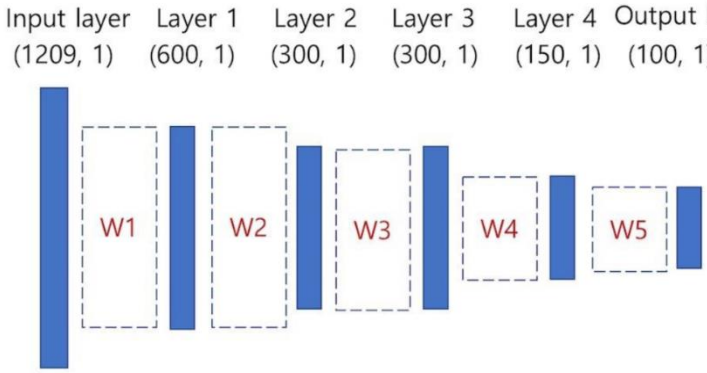
Layer 종류 : keras.layers.Conv2D

- 4개의 컨볼루션 레이어, 39x31x1 크기의 입력 데이터, 100개의 클래스로 분류
  - 20만개 parameter → Dense에 비해 학습이 쉽고 처리 속도가 빠르다.



출처: https://github.com/sooftware/Speech-Recognition-Tutorial/blob/master/seminar/CNN.pdf

- 4개의 dmsslrcmd, 1209x1(39x31x1) 크기의 입력 데이터, 100개의 클래스로 분류
  - 100만개 parameter. 은닉층이 깊어질 수록 급격히 늘어남

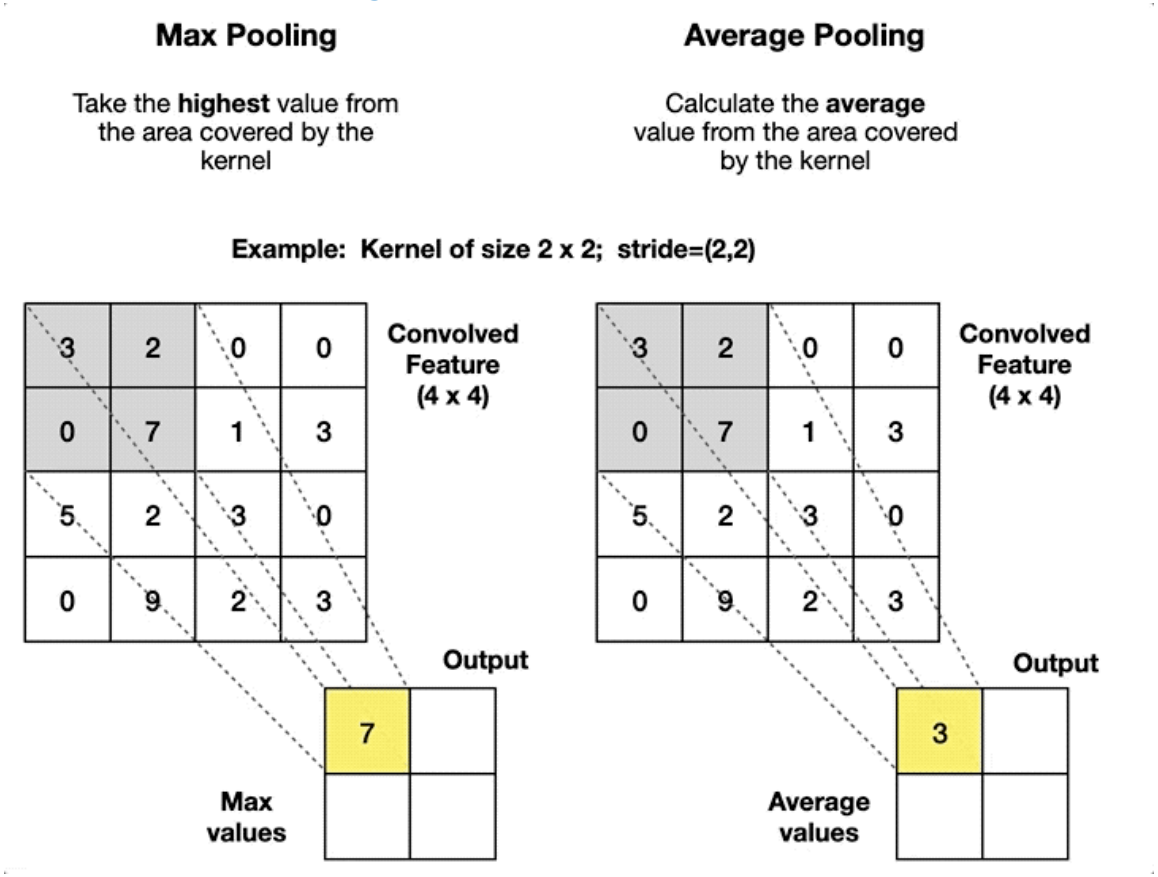


출처: https://github.com/sooftware/Speech-Recognition-Tutorial/blob/master/seminar/CNN.pdf

Layers

Layer 종류 : keras.layers.MaxPooling2D

- Pooling
  - n차원 부분 행열을 사용하여 특징맵을 요약
  - 주로 Max Pooling을 사용 한다.



출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

Layers

Layer 종류 : keras.layers.MaxPooling2D

- Pooling

- 효과
  - 오류 보완
  - 확대

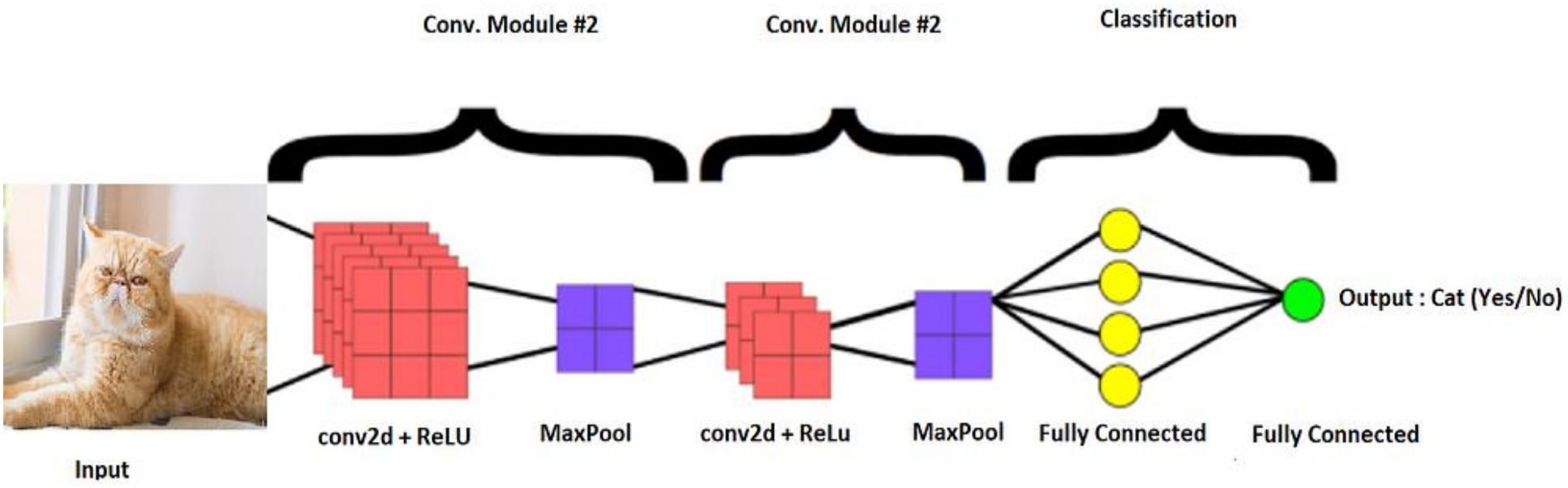
출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

Layers

Layer 종류 : keras.layers.Conv2D, keras.layers.MaxPooling2D

- CNN (Convolutional Neural Networks, 합성곱 신경망)
  - DCNN (Deep CNN, 심층 합성곱 신경망)
  - Convolution layer
  - ReLU (Rectified Linear Unit)
  - Pooling layer
  - FC (Fully connected) layer

- 데이터 변환 계층
- Receptive field (로컬 수용 필드)
- 가중치 공유

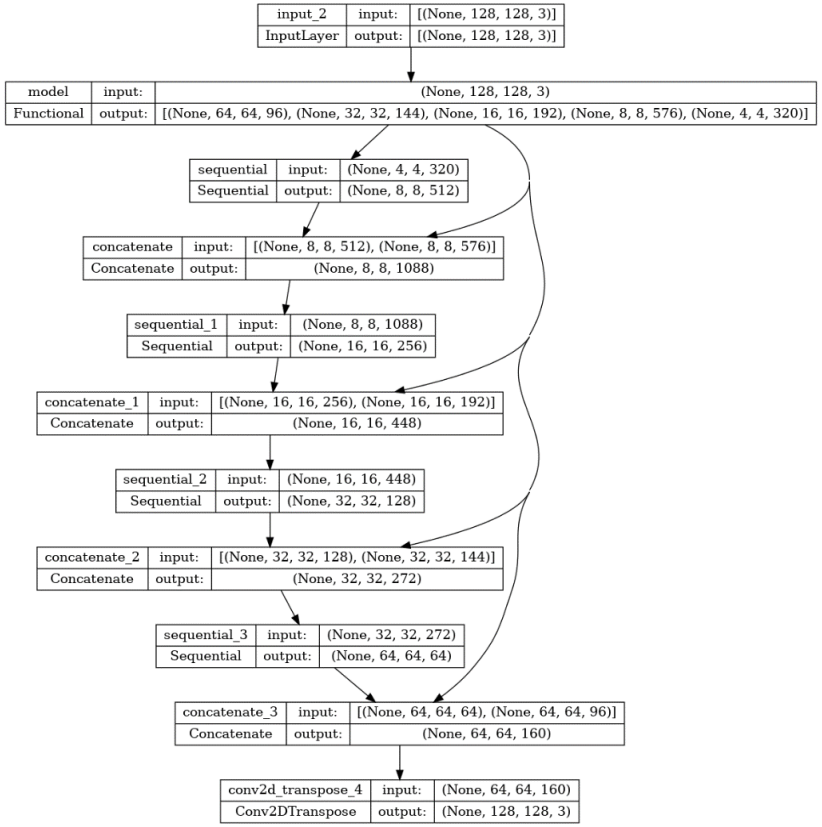


출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

Layers

Layer 종류 : keras.layers.Conv2D, keras.layers.MaxPooling2D

- CNN (Convolutional Neural Networks, 합성곱 신경망)
  - Filter의 크기에 따라 서로 다른 Feature가 생성 한다.
    - 외부에서 다양한 종류의 Feature를 생성하여 조합 한다.



원본 이미지 : 128 * 128

Conv를 적용한 다양한 종류의 output 생성
- 4 * 4
- 8 * 8
- 16 * 16
- 32 * 32
- 64 * 64

→ Conv한 output을 연산

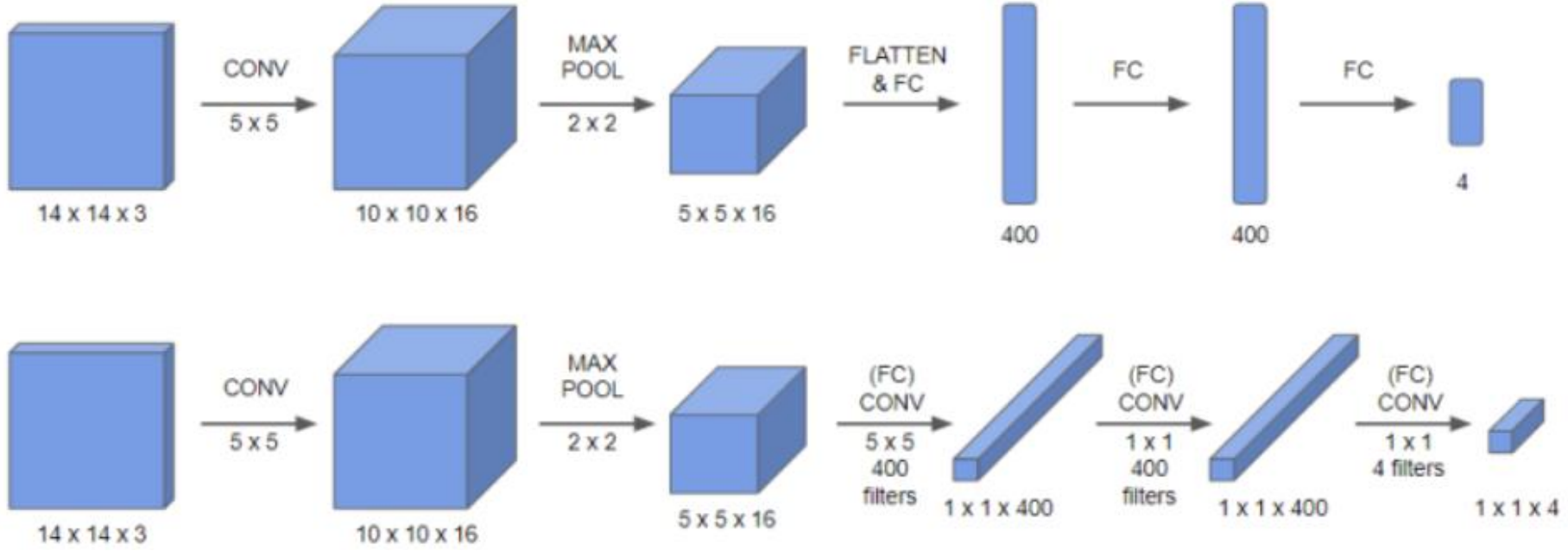
출처: https://www.tensorflow.org/tutorials/images/segmentation?hl=ko

Layers

Layer 종류 : keras.layers.Conv2D, keras.layers.MaxPooling2D

- CNN (Convolutional Neural Networks, 합성곱 신경망)
  - Dense (Fully Connected) layer를 Conv layer로 대체하여 성능 향상

  - Convolutional Fully Connected 사용시 장점
    - 속도가 빠름
    - 다양한 크기의 이미지를 입력 받을 수 있음

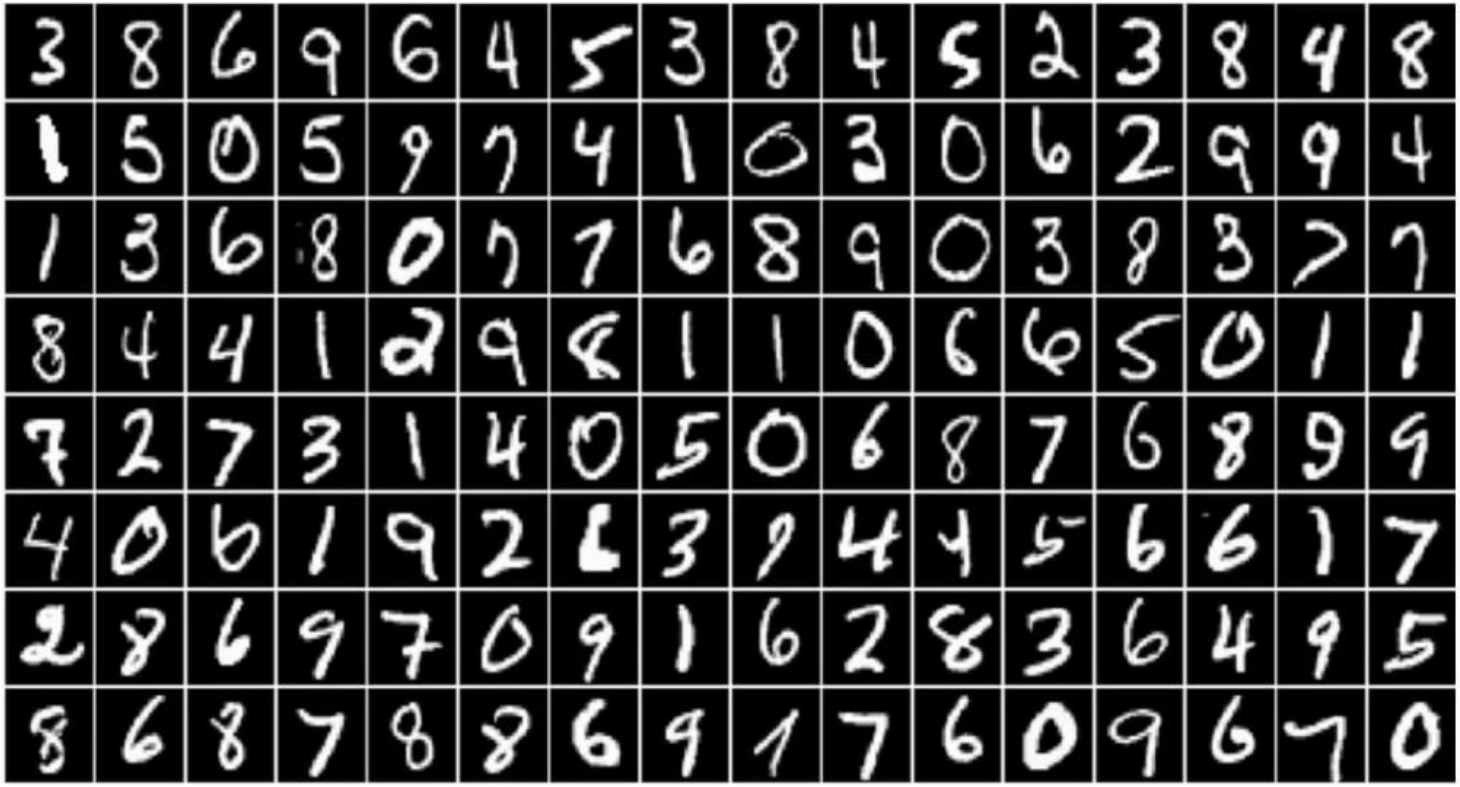


출처: https://velog.io/@pabiya/OverFeatIntegrated-Recognition-Localization-and-Detectionusing-Convolutional-Networks

Layers

CNN (Convolutional Neural Networks, 합성곱 신경망)

- Input
  - 숫자 손글씨
  - 28 * 28



출처: https://learnopencv.com/implementing-mlp-tensorflow-keras/

Layers

CNN (Convolutional Neural Networks, 합성곱 신경망)

- 데이터 가공
  - Nomalization
  - Layer에서 데이터 가공과 다른 점은 "**한번만 실행된다**"는 것 이다.

- 훈련 데이터 : train
  - 훈련용 : train : 80%
  - 검증용 : validation : 20%

- 평가 데이터 : test

- ephochs
  - 훈련 집합 횟수

- batch_size
  - 훈련 집합당 훈련 횟수

Layers

CNN (Convolutional Neural Networks, 합성곱 신경망)

- Model

```
IMG_ROWS, IMG_COLS = 28, 28
input_shape = (IMG_ROWS, IMG_COLS, 1)

model = keras.models.Sequential()
model.add(keras.layers.Conv2D(20, (5, 5), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(keras.layers.Conv2D(50, (5, 5), activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(500, activation="relu"))
model.add(keras.layers.Dense(self.nb_classes, activation="softmax"))

model.compile(
    optimizer=keras.optimizers.Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy' ]
)
```

Layers

CNN (Convolutional Neural Networks, 합성곱 신경망)

- Output

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 24, 24, 20)        520
 max_pooling2d (MaxPooling2D)  (None, 12, 12, 20)      0
 conv2d_1 (Conv2D)           (None, 8, 8, 50)          25050
 max_pooling2d_1 (MaxPooling2D)  (None, 4, 4, 50)      0
 flatten (Flatten)           (None, 800)               0
 dense (Dense)               (None, 500)               400500
 dense_1 (Dense)             (None, 10)                5010
=================================================================
Total params: 431,080
Trainable params: 431,080
Non-trainable params: 0
```

Layers

CNN (Convolutional Neural Networks, 합성곱 신경망)

- Output

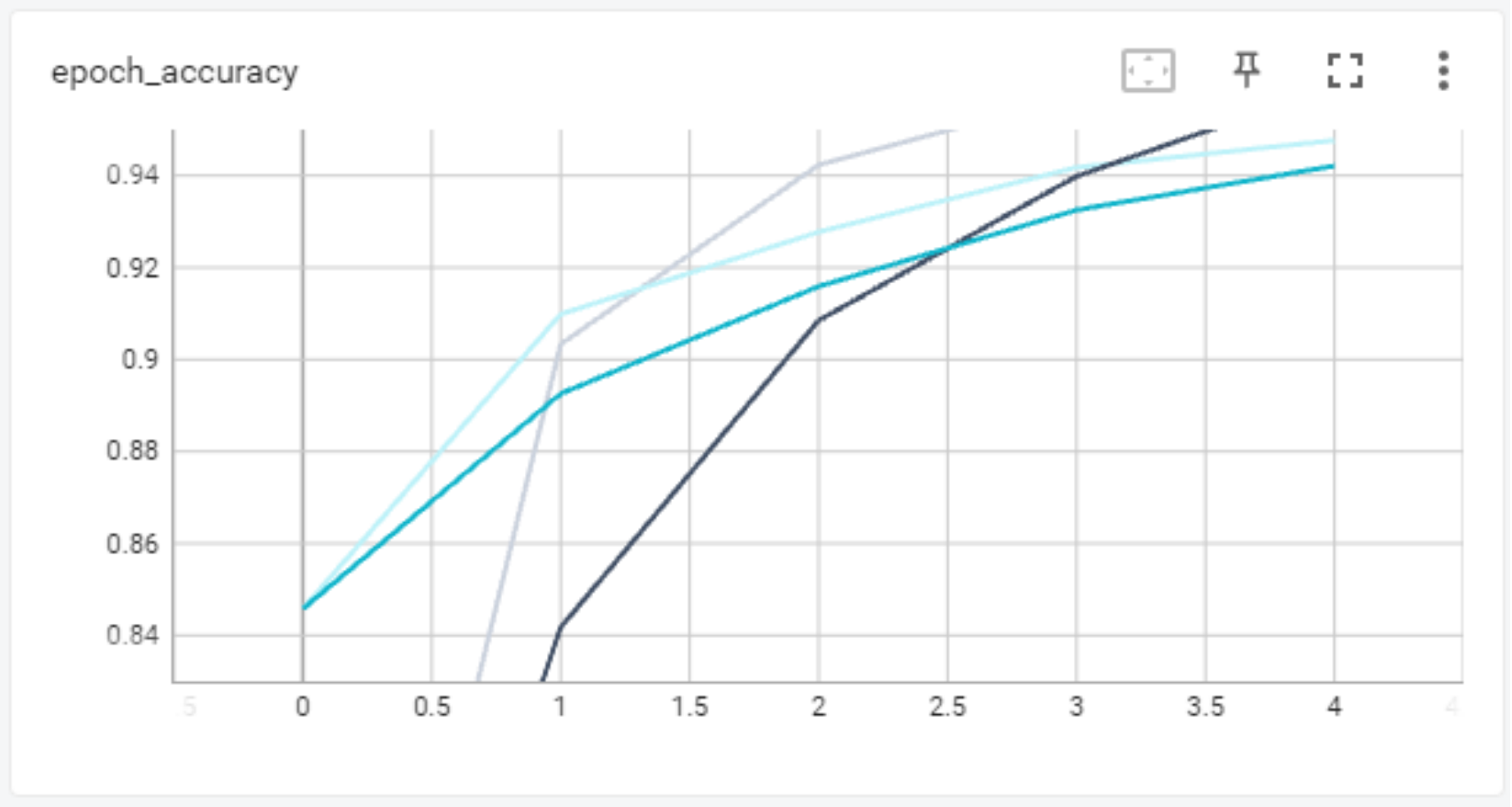
```
Epoch 1/5
2023-08-03 14:06:54.469518: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8600
24/24 [==============================] - 5s 52ms/step - loss: 1.1939 - accuracy: 0.7053 - val_loss: 0.4708 - val_accuracy: 0.8558
Epoch 2/5
24/24 [==============================] - 1s 39ms/step - loss: 0.3677 - accuracy: 0.8913 - val_loss: 0.3250 - val_accuracy: 0.9014
Epoch 3/5
24/24 [==============================] - 1s 38ms/step - loss: 0.2287 - accuracy: 0.9403 - val_loss: 0.2305 - val_accuracy: 0.9316
Epoch 4/5
24/24 [==============================] - 1s 38ms/step - loss: 0.1650 - accuracy: 0.9547 - val_loss: 0.1785 - val_accuracy: 0.9457
Epoch 5/5
24/24 [==============================] - 1s 38ms/step - loss: 0.1240 - accuracy: 0.9697 - val_loss: 0.1629 - val_accuracy: 0.9504

313/313 [==============================] - 1s 2ms/step - loss: 0.1474 - accuracy: 0.9541
```

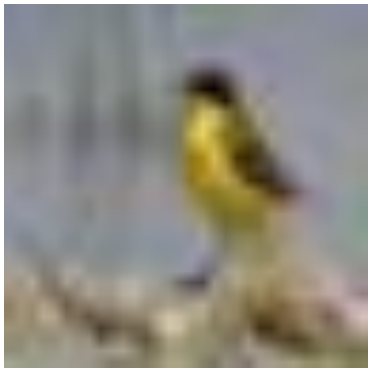
Layers

CNN (Convolutional Neural Networks, 합성곱 신경망)

- Graph



Deep Learning 응용

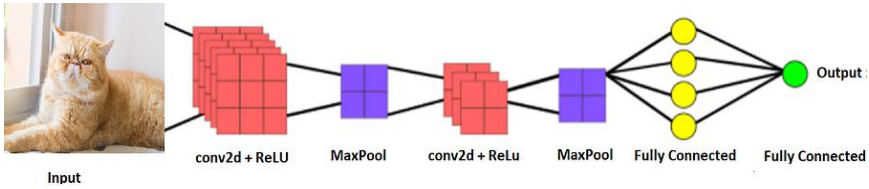


60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

출처: https://www.cs.toronto.edu/~kriz/cifar.html

분류



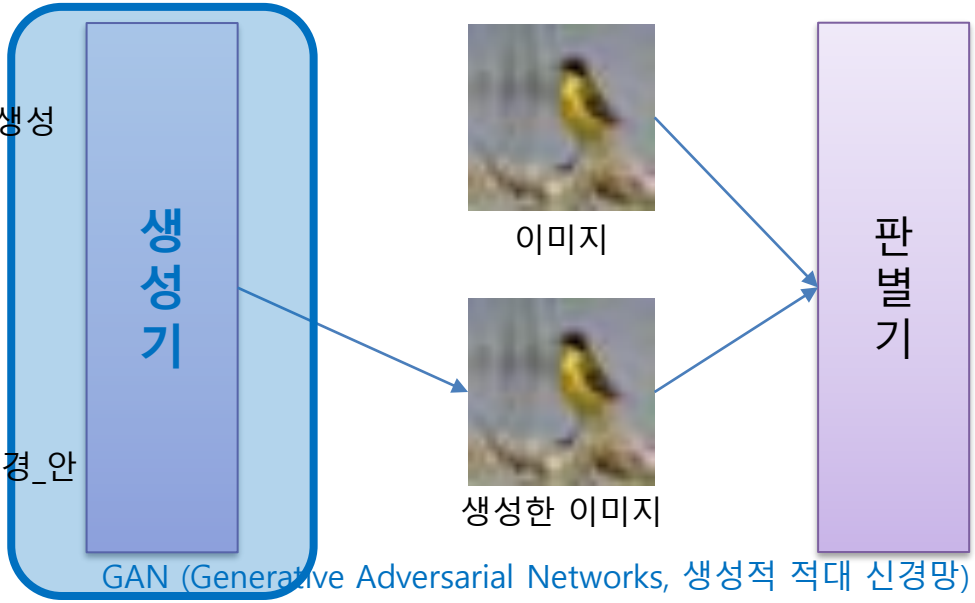
출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

1

생성기 → 판별기

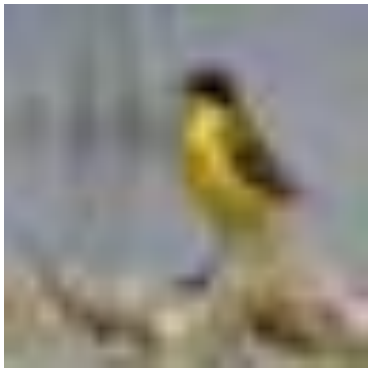
- SRGAN (Super Resolution GAN)
  - 저해상도 이미지로 고해상도 이미지 생성
- CycleGAN : 이미지 변환
  - 풍경을 반고호가 그린 풍경으로 변환
  - 여름_사진 → 겨울_사진
- InfoGAN
  - 생성된 이미지의 다양한 속성을 제어

- GAN은 산술 연산 가능
  - 안경_쓴_남자 - 안경_안_쓴_남자 + 안경_안_쓴_여자 → 안경_쓴_여자



GAN (Generative Adversarial Networks, 생성적 적대 신경망)

Deep Learning 응용

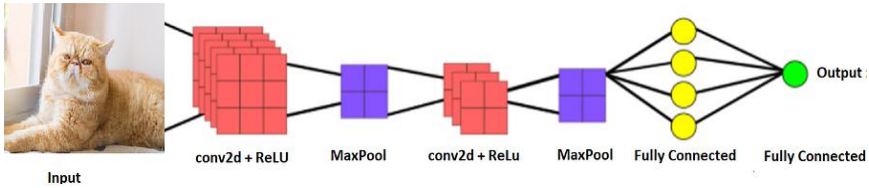


60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

출처: https://www.cs.toronto.edu/~kriz/cifar.html

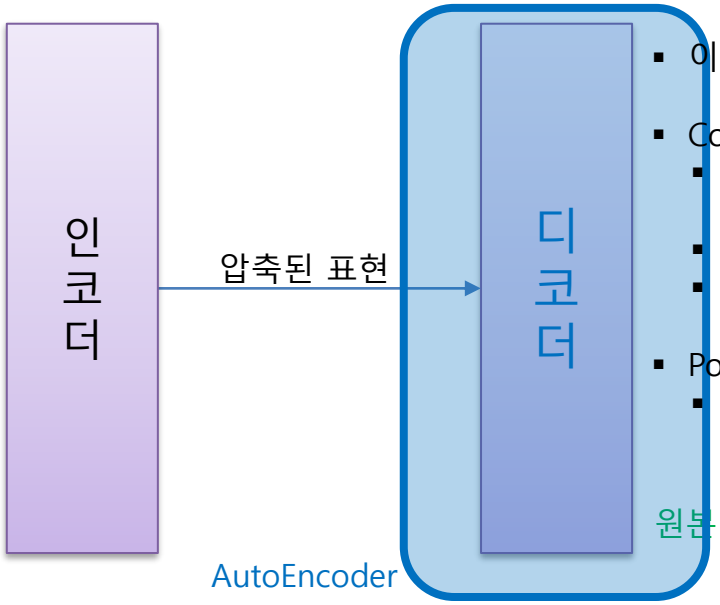
분류



출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

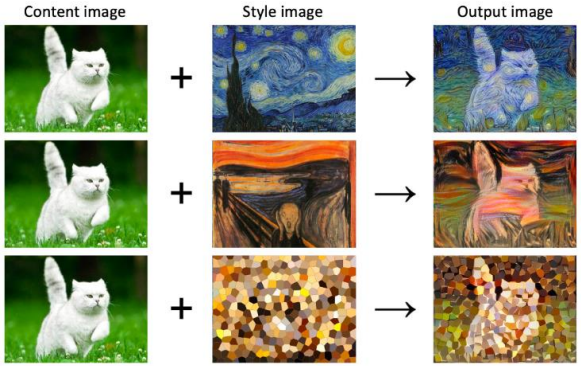
2 Encoder → Decoder

y = h(x) r = f(y) = f(h(x)) r = f(y)

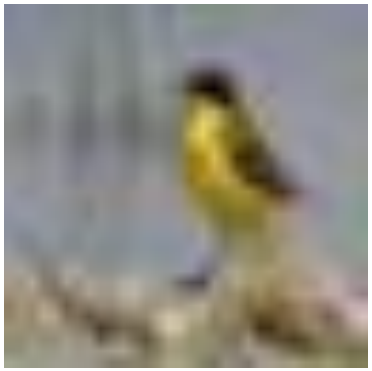


- 이미지 재구성, 군집화, 기계 번역
- Convolution 효과
  - Style transfer : 학습한 스타일(Filter)로 스타일 변환
  - 습득한 스타일(Filter)로 효과 추가
  - 이미지 해상도 확대
- Pooling 효과
  - 손상된 부분 복원
    - 이미지에서 사람 지우기

원본 이미지와 비교하여 학습



출처: https://gm-note.tistory.com/entry/머신러닝-Style-transfer스타일-변환



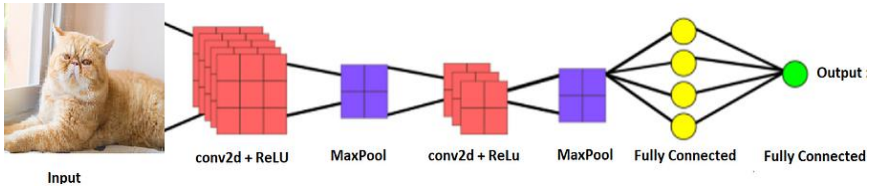
60,000 이미지
100개 분류
분류당 600개 이미지

32 * 32 picels

출처: https://www.cs.toronto.edu/~kriz/cifar.html

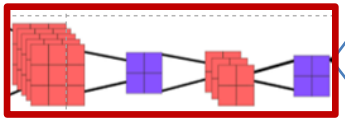
회귀

x, y, width, height



출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

3 분류와 지역화



분류

회귀

x, y, width, height

- 객체 탐지 (분류 + 회귀)
  - 동물, 동물의 위치

RNN

회귀

상태

다음 입력으로 사용

학습 기능 없음

RNN (Recurrent Neural Network, 순환 신경망)

GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- DCGAN (Deep Convolution GAN)
  - images_real : tensorflow.keras.datasets.mnist에서 가져온 진짜 손글씨 이미지
  - noise : 난수
  - image_fake : noise (난수)로 생성한 가짜 손글씨 이미지

```
def train(self, dataset, epochs, batch_size=256, save_interval=50):
        binary_cross_entropy =keras.losses.BinaryCrossentropy()
        generator_optimizer = keras.optimizers.Adam(1e-4)
        discriminator_optimizer = keras.optimizers.Adam(1e-4)

        for epoch in tqdm(range(epochs)):
            for images_real in dataset:
                noise = tf.random.normal([batch_size, self.latent_dim])                    #--- 난수 발생

                with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
                    images_fake = self.generator(noise, training=True)                     #--- 생성기 실행

                    real_output = self.discriminator(images_real, training=True)          #--- 판별기 실행
                    fake_output = self.discriminator(images_fake, training=True)          #--- 판별기 실행
```

GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- DCGAN (Deep Convolution GAN)

```
# with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
```

Loss 계산

```
gen_loss = binary_cross_entropy(tf.ones_like(fake_output), fake_output)

real_loss = binary_cross_entropy(tf.ones_like(real_output), real_output)
fake_loss = binary_cross_entropy(tf.zeros_like(fake_output), fake_output)
disc_loss = real_loss + fake_loss
```

Optimizer

Loss를 사용하여 최적화

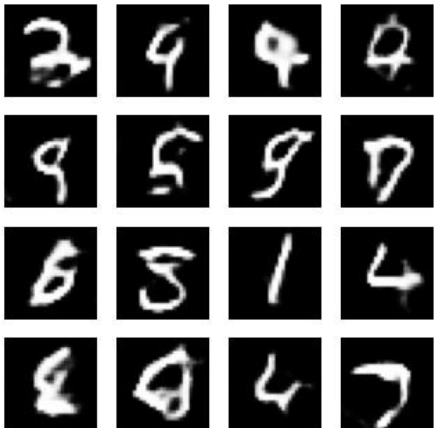
```
gradients_of_generator = gen_tape.gradient(gen_loss, self.generator.trainable_variables)
gradients_of_discriminator = disc_tape.gradient(disc_loss, self.discriminator.trainable_variables)

generator_optimizer.apply_gradients(zip(gradients_of_generator,
self.generator.trainable_variables))
discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
self.discriminator.trainable_variables))
```

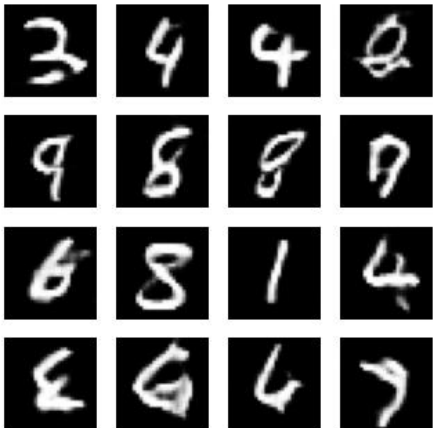
GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- DCGAN (Deep Convolution GAN)

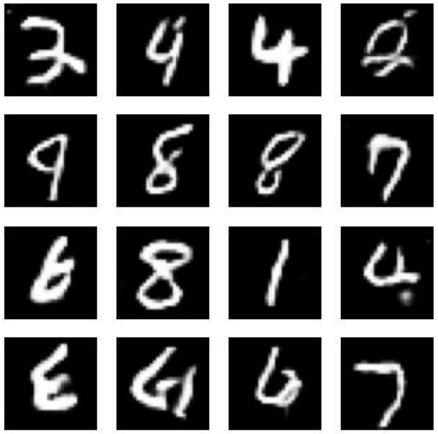
epoch 10



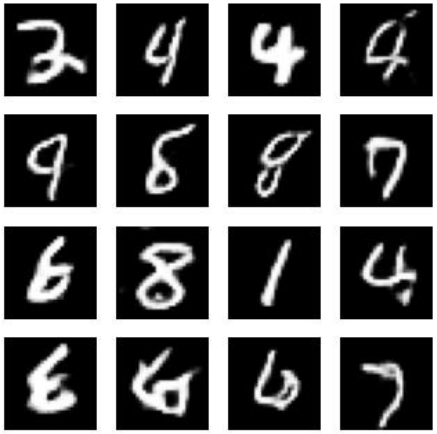
epoch 20



epoch 30



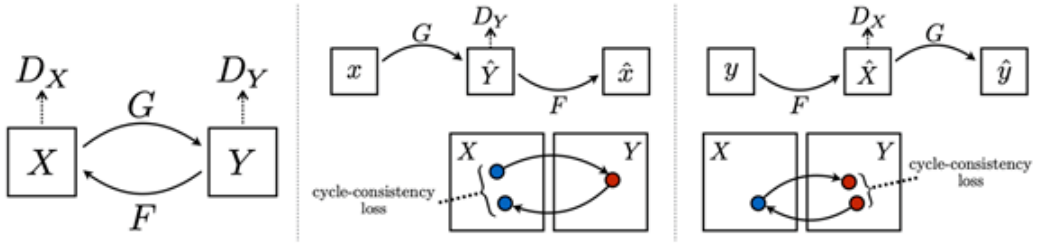
epoch 40



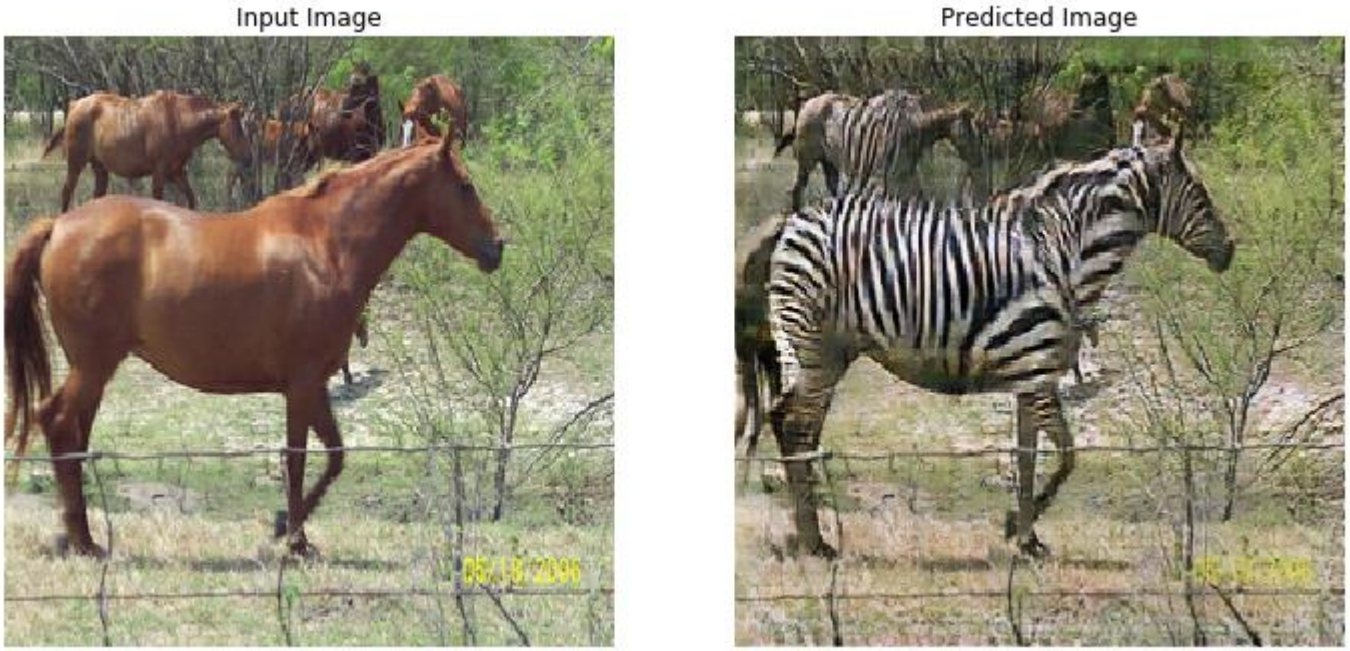
epoch 50

GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- CycleGan
  - 다양한 스타일의 이미지간 변환을 학습



출처: https://neptune.ai/blog/6-gan-architectures

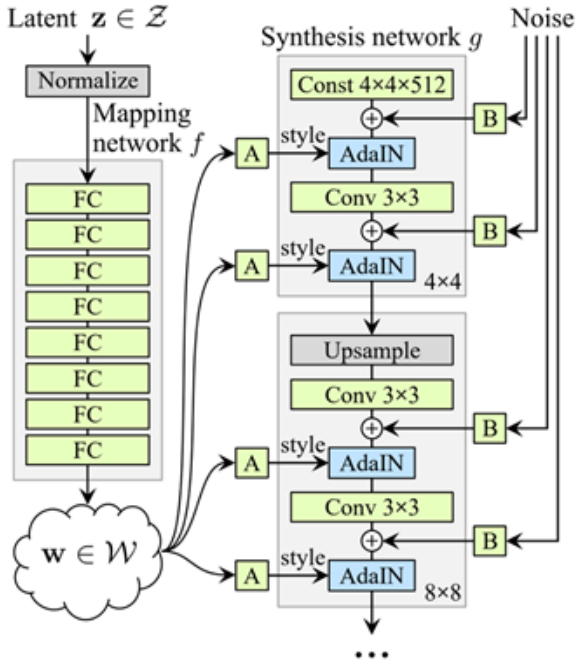


출처: https://www.tensorflow.org/tutorials/generative/cyclegan?hl=ko

GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- StyleGAN
  - 인간 얼굴의 특징으로 새로운 인간의 얼굴 이미지 생성
  - 2,620만개 매개변수

출처: https://thispersondoesnotexist.com/

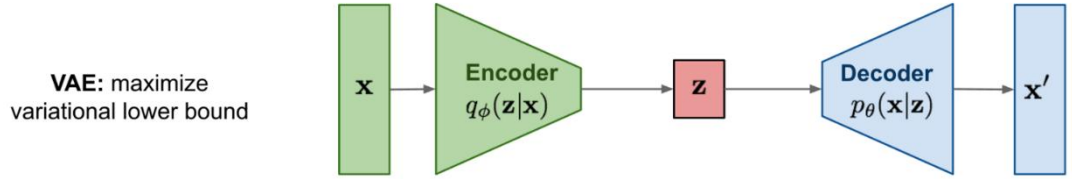


출처: https://neptune.ai/blog/6-gan-architectures

- pixelRNN
- text-2-image
- DiscoGAN
- IsGAN

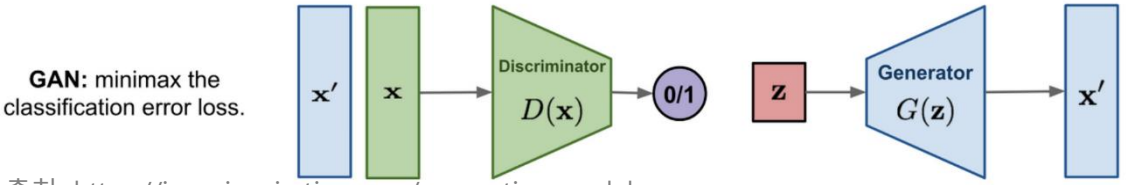
GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- Generative AI 모델의 종류
  - VAE (Variational Autoencoder, 변이형 오토인코더)



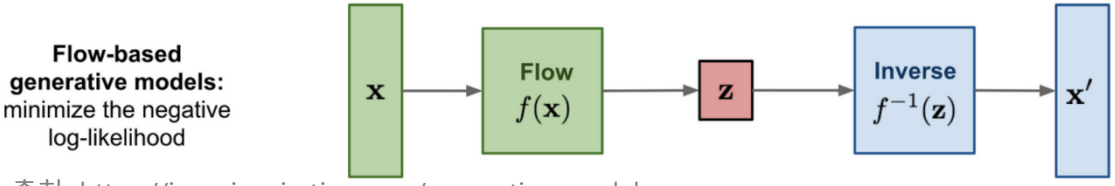
출처: https://jang-inspiration.com/generative-model

  - GAN (Generative Adversarial Network, 생성적 적대 신경망)



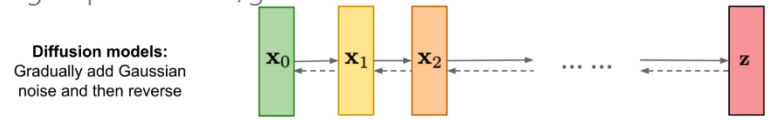
출처: https://jang-inspiration.com/generative-model

  - Likelihood Based Model
    - Autoregressive Models (자기 회귀 모델) : Regression 함수 사용
    - Flow-based Models (플로우 기반 모델) : 확률 밀도 함수 사용



출처: https://jang-inspiration.com/generative-model

  - Diffusion Model



출처: https://jang-inspiration.com/generative-model

AutoEncoder

- Encoder

```
class Encoder(keras.layers.Layer):
    def __init__(self, hidden_dim):
        super(Encoder, self).__init__()
        self.hidden_layer = keras.layers.Dense(units=hidden_dim, activation=tf.nn.relu)

    def call(self, input_features):
        activation = self.hidden_layer(input_features)
        return activation
```

- Decoder

```
class Decoder(keras.layers.Layer):
    def __init__(self, hidden_dim, original_dim):
        super(Decoder, self).__init__()
        self.output_layer = keras.layers.Dense(units=original_dim, activation=tf.nn.relu)

    def call(self, encoded):
        activation = self.output_layer(encoded)
        return activation
```

AutoEncoder

- AutoEncoder

```
class Autoencoder(keras.Model):
    def __init__(self, hidden_dim, original_dim):
        super(Autoencoder, self).__init__()
        self.loss = []
        self.encoder = Encoder(hidden_dim=hidden_dim)
        self.decoder = Decoder(hidden_dim=hidden_dim, original_dim=original_dim)

    def call(self, input_features):
        encoded = self.encoder(input_features)
        reconstructed = self.decoder(encoded)
        return reconstructed
```

AutoEncoder

- Train

```
def train(model, loss, opt, dataset, epochs=20):
    for epoch in range(epochs):
        epoch_loss = 0
        for images in dataset:
            with tf.GradientTape() as tape:
```

AutoEncoder 실행

```
                preds = model(images)
```

Loss 계산

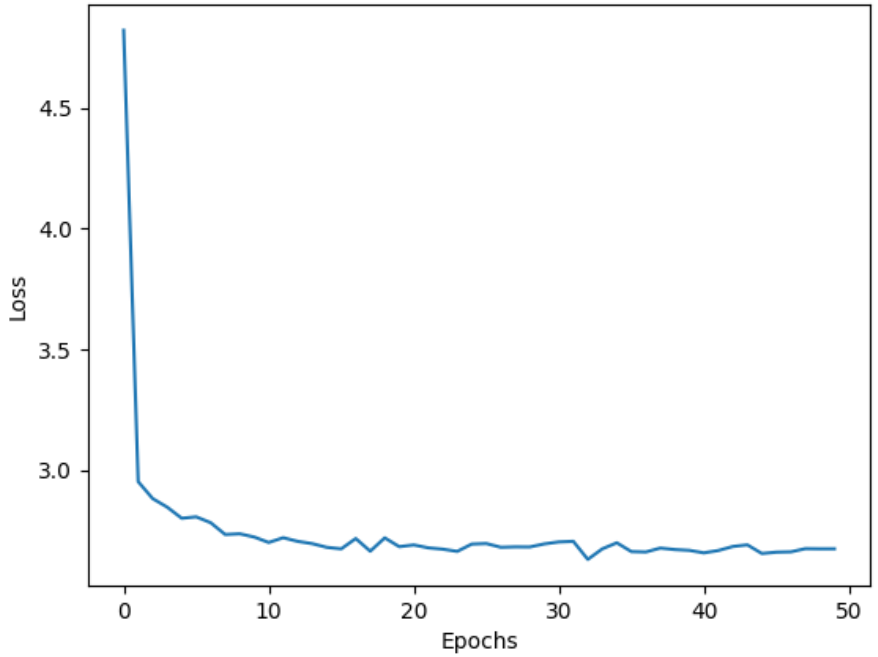
```
                loss_values = loss(preds, images)
```

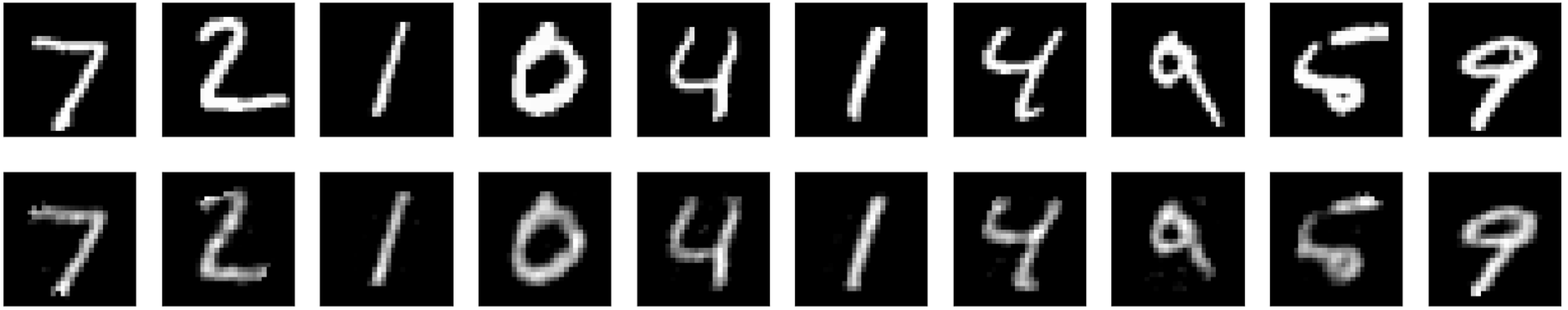
Loss를 사용하여 최적화

```
                gradients = tape.gradient(loss_values, model.trainable_variables)
                opt.apply_gradients(zip(gradients, model.trainable_variables))

            epoch_loss = epoch_loss + loss_values
        model.loss.append(epoch_loss)
```

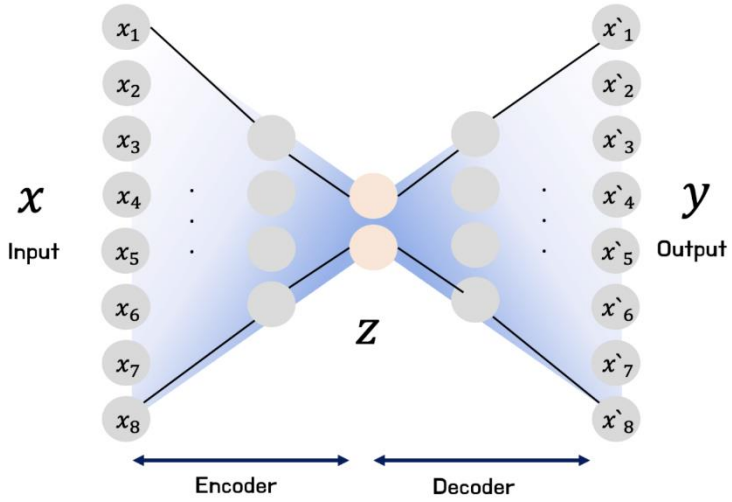
AutoEncoder





AutoEncoder

- AutoEncoder 모델
  - r = f(y) = f(h(x))
    - y = h(x) : Encoder
      - 일반적으로 y는 x, r보다 차원이 낮다.
      - y는 x의 특징으로 볼 수 있다.
    - r = f(y) : Decoder
    - e = x - r : 오차
  - 활용
    - AutoEncoder를 계층적으로 구성할 수 있다.
    - h(x)를 가공하여 여러가지 효과를 줄 수 있다.
    - r을 x와 관련 있지만 다른 것을 배치하여 학습 가능
      - 예) x. 영어, r. 프랑스어



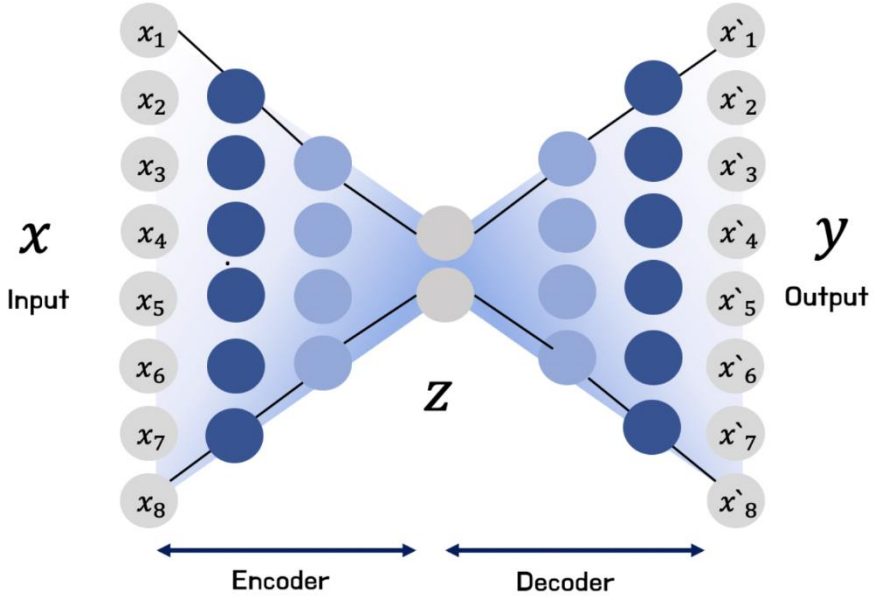
출처:
https://velog.io/@jochedda/%EB%94%A5%EB%9F%AC%EB%8B%9D-Autoencoder-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A2%85%EB%A5%98

- 사용 layer
  - Dense : CF (Collaborative Filtering, 협업 필터링) 모델 구축에 유리
  - Conv
    - 얼굴에서 안경을 제거
  - RNN, LSTM, GRU : 텍스트 데이터 처리에 유리

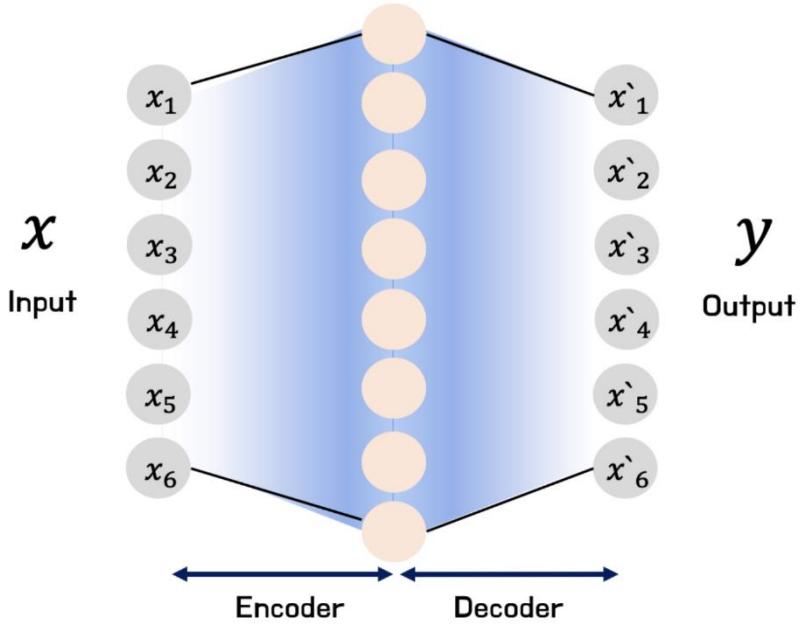
AutoEncoder

- AutoEncoder 모델

Stacked AutoEncoder

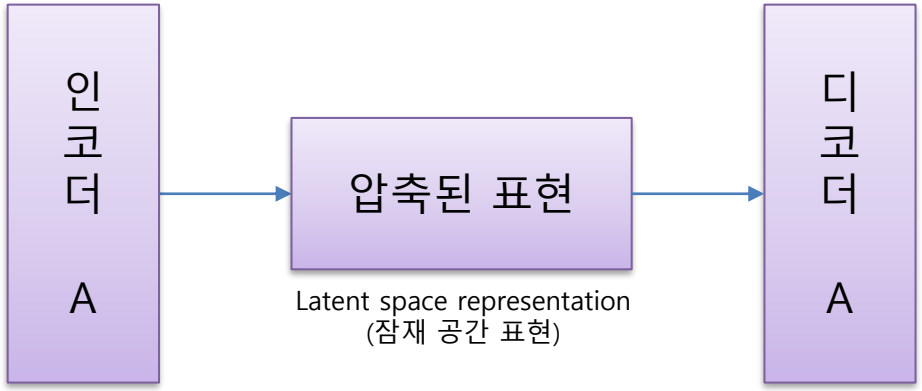


Sparse AutoEncoder



출처: https://velog.io/@jochedda/%EB%94%A5%EB%9F%AC%EB%8B%9D-Autoencoder-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A2%85%EB%A5%98

AutoEncoder

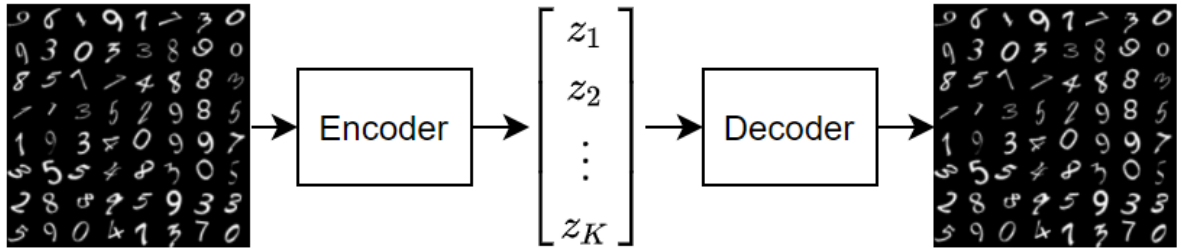


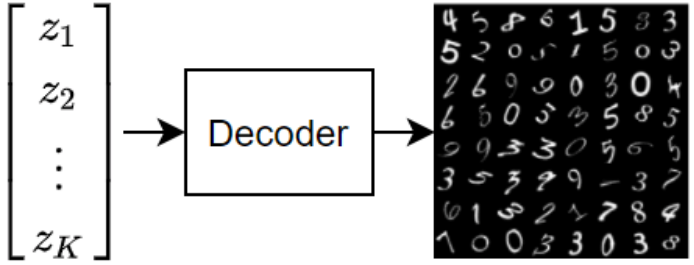
- Denoising images (이미지 잡음 제거)
  - 원본 이미지의 손상이 AutoEncoder를 거치면 복원 된다.
  - 손상된 이미지 복원 → 이미지에서 사람 지우기
  - 손상된 텍스트에서 누락된 단어 예측

- Anommaly Detection (이상 탐지)
  - 정상 이미지로 AutoEncoder를 사용하여 학습 한다. (디코딩 후 Loss Function 값 확인)
  - 비정상 이미지로 AutoEncoder를 하면 Loss Function값이 확연하게 다를 것 이다.
  - 신용카드 이상 거래 탐지: https://github.com/KerasKorea/KEKOxTutorial/blob/master/20_Keras의 Autoencoder를 활용해 신용카드 이상 거래 탐지하기.md

AutoEncoder

- Image generation (이미지 생성) : VAE (Variational AutoEncoder)
  - 인코더 A로 확률 분포를 따르는 압축된 표현을 생성 한다.
  - 확률 분포 (압축된 표현)으로 디코더를 통해 이미지를 생성 한다.





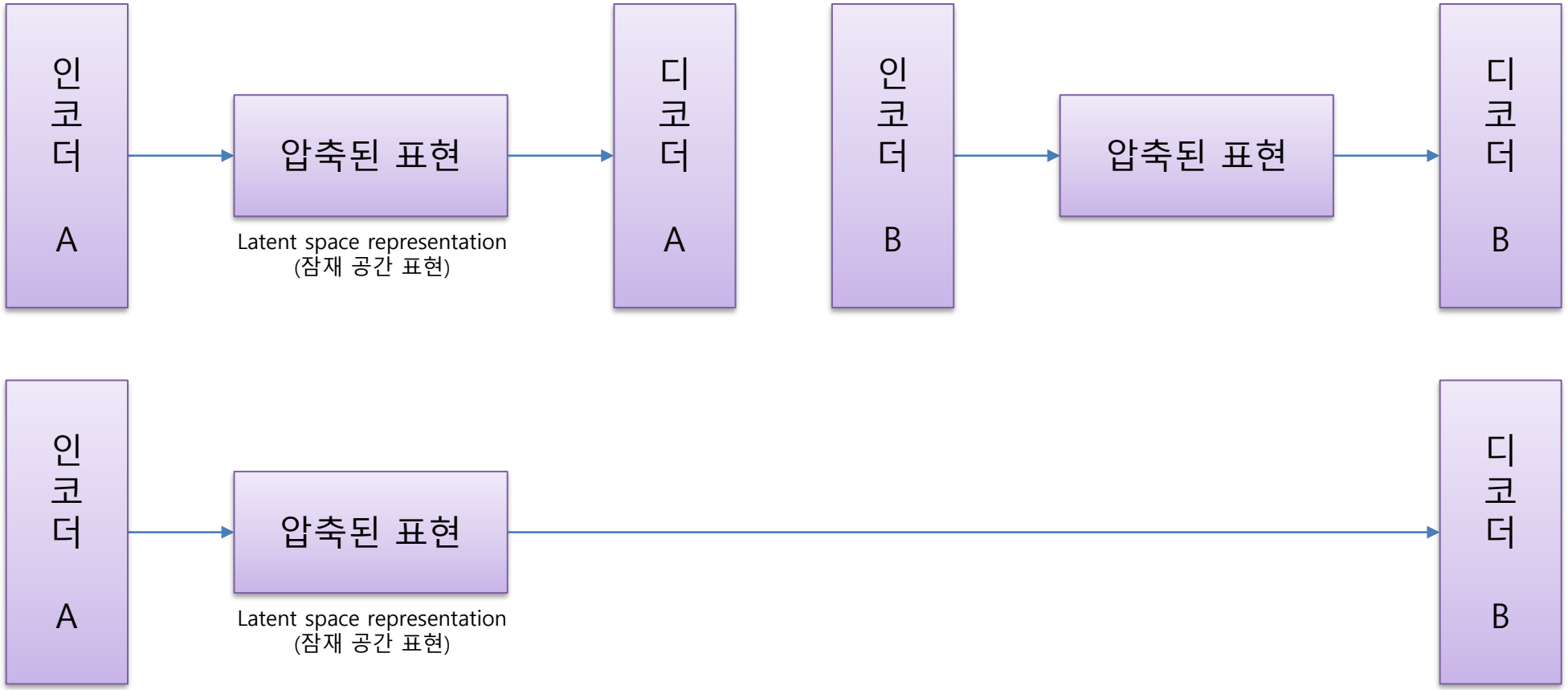
출처: https://kr.mathworks.com/help/deeplearning/ug/train-a-variational-autoencoder-vae-to-generate-images.html

AutoEncoder

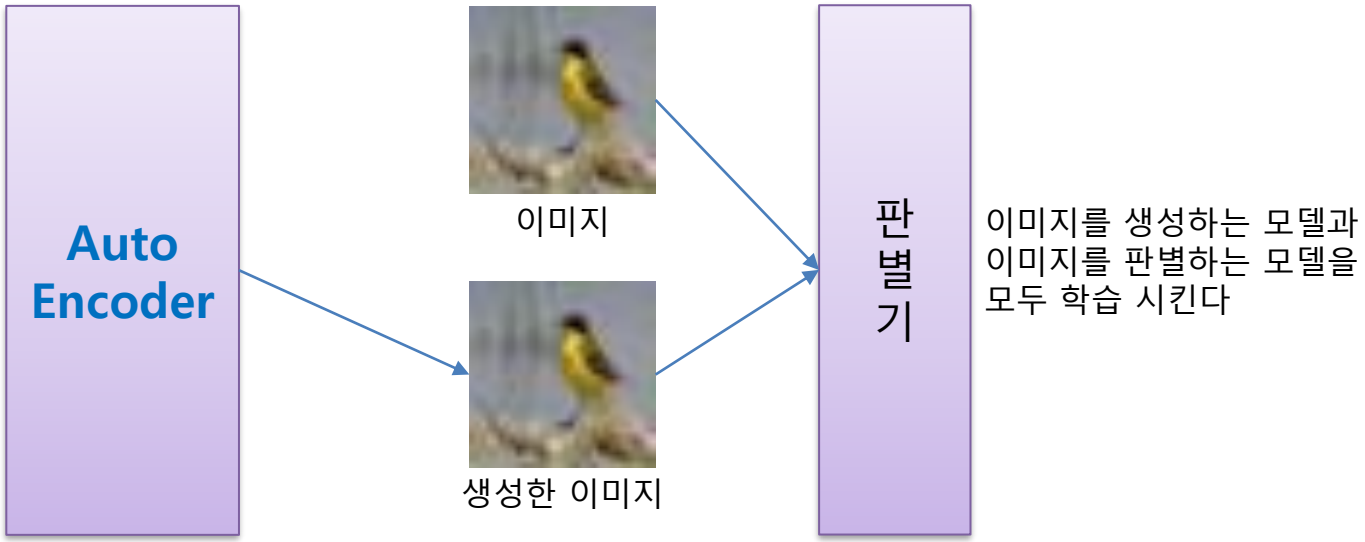
- Dimensionally Reduction (차원 감소)
  - PCA (Principal Component Analysis, 주성분 분석)과 유사
  - PCA는 선형적으로 차원 감소. AutoEncoder는 비선형적으로 차원 감소
- Recommendation Engines (추천 엔진)
- Image Recognnition (이미지 인식)

AutoEncoder

- Style Transfer : 학습한 스타일로 스타일 변환
  - 인코더 A로 압축한 것을 디코더 B로 복원 한다.



AutoEncoder + GAN (Generative Adversarial Networks)



ppp

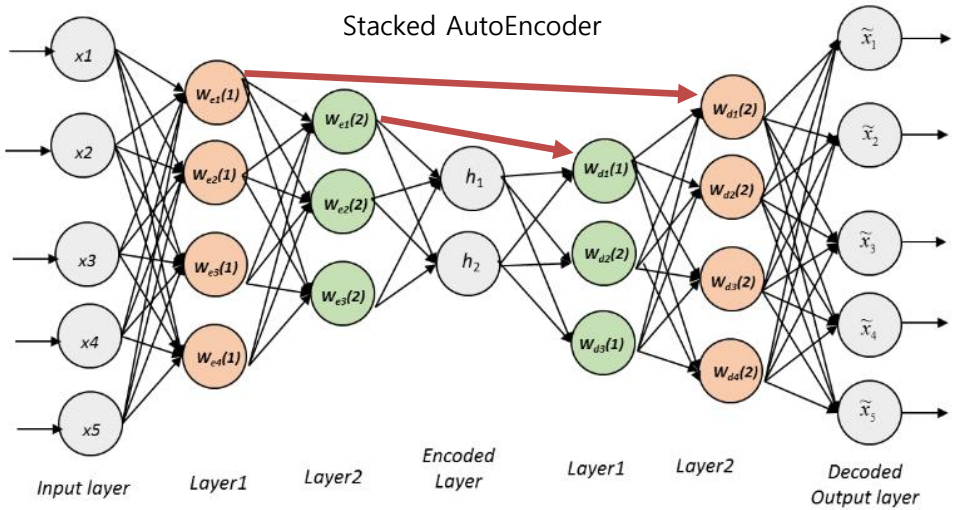
AutoEncoder

- 원본 이미지 (Pixel Space)

- Pixel Space에서 이미지 보간 (Linearly interpolate) → Fading overlay

- Latent space representation에서 보간한 후 Decoder 적용

AutoEncoder 응용



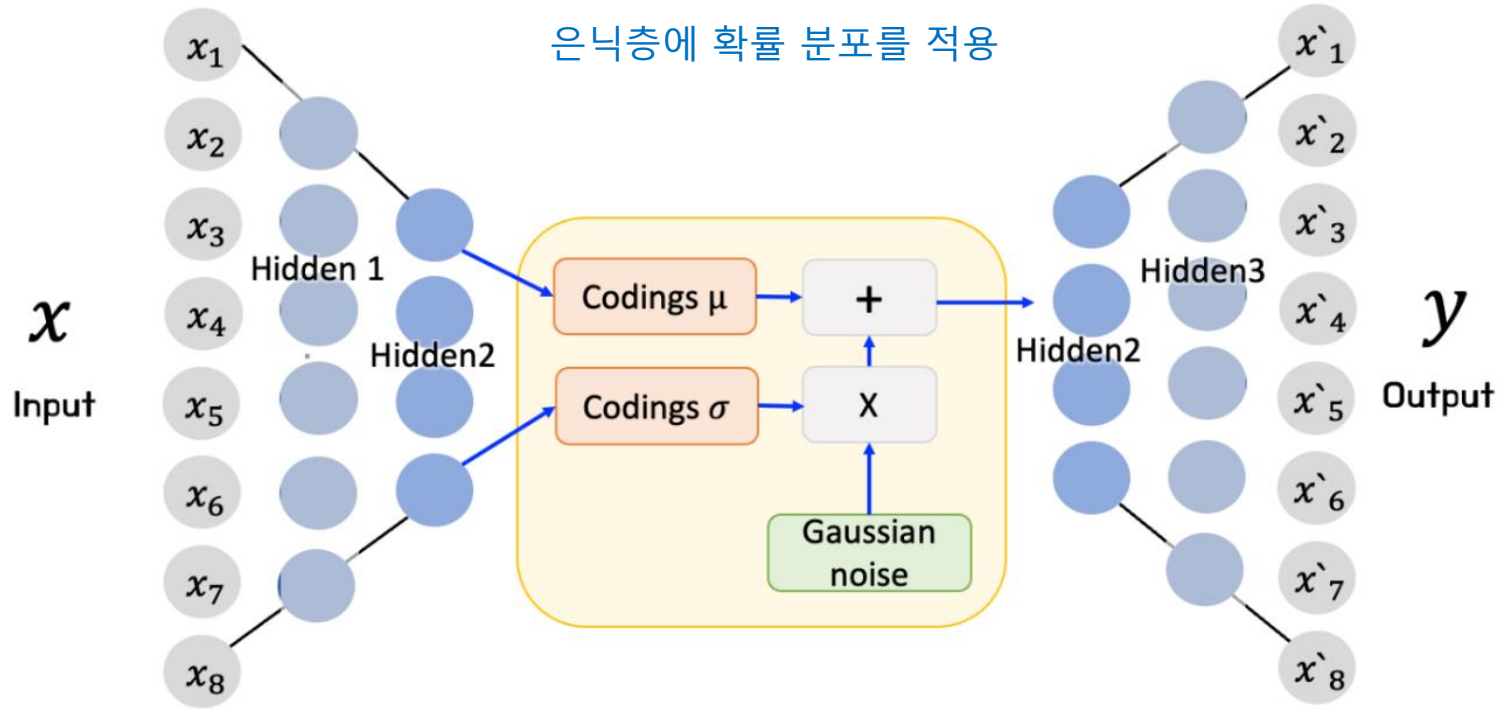
출처: https://techblog-history-younghunjo1.tistory.com/130

- 인코더와 디코더간의 같은 계층이 서로 관련이 있다.

AutoEncoder

- AutoEncoder 모델
  - μ : 평균
  - σ : 표준편차

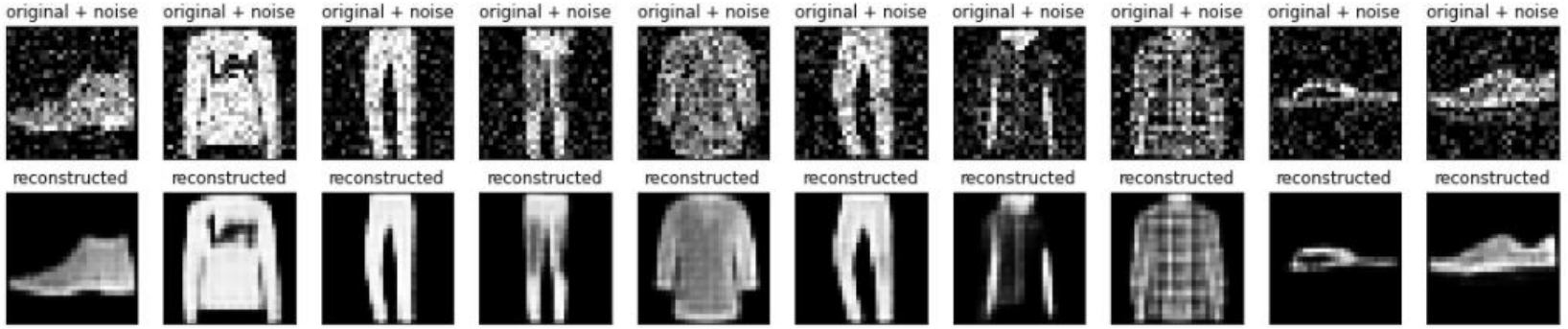
VAE (Variational AutoEncoder, 변형 오토인코더)



출처: https://velog.io/@jochedda/%EB%94%A5%EB%9F%AC%EB%8B%9D-Autoencoder-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A2%85%EB%A5%98

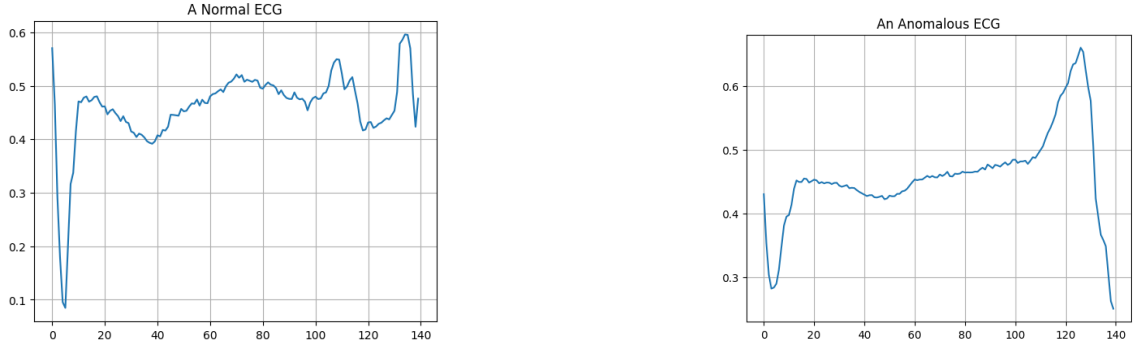
AutoEncoder 활용

- 이미지에서 노이즈 제거



출처: https://www.tensorflow.org/tutorials/generative/autoencoder?hl=ko

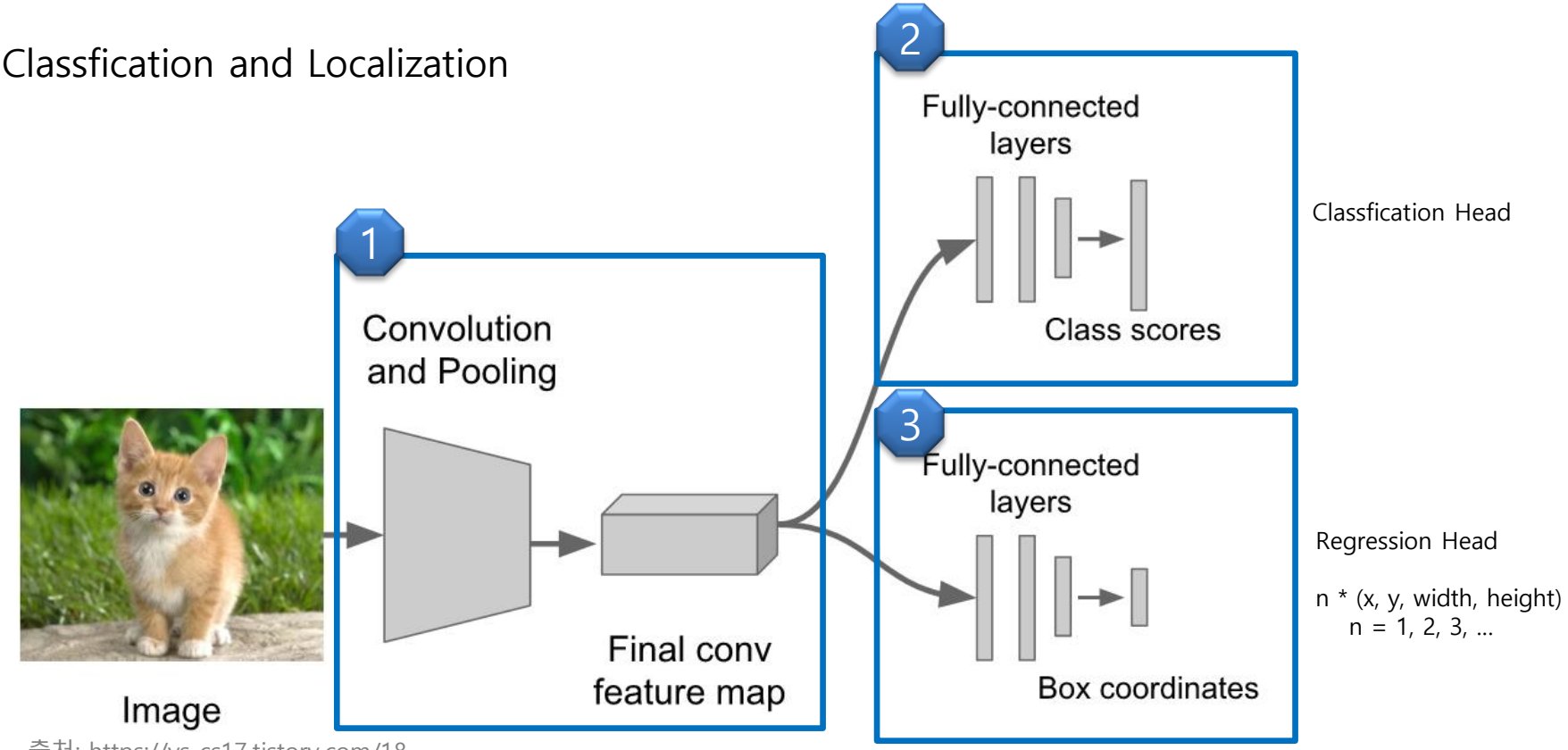
- 이상 감지
  - 정상 심전도 리듬으로 AutoEncoder 훈련
  - 심전리 리듬으로 AutoEncoder 실행
    - 비정상 리듬의 경우 재구성 오류가 클 것이다 → 이상 감지



출처: https://www.tensorflow.org/tutorials/generative/autoencoder?hl=ko

분류와 지역화

- Classfication and Localization



출처: https://ys-cs17.tistory.com/18

- 1번의 끝이
  - Feature map인 경우 : Overfeat, VGG
  - Dense인 경우 : DeepPose, R-CNN

Layers

현재까지 학습한 사항

- Layer
  - Dense < Conv, Pooling

- Model
  - Classfication (분류) : 분류기
  - Regression (회기) : Localization v1

Layers

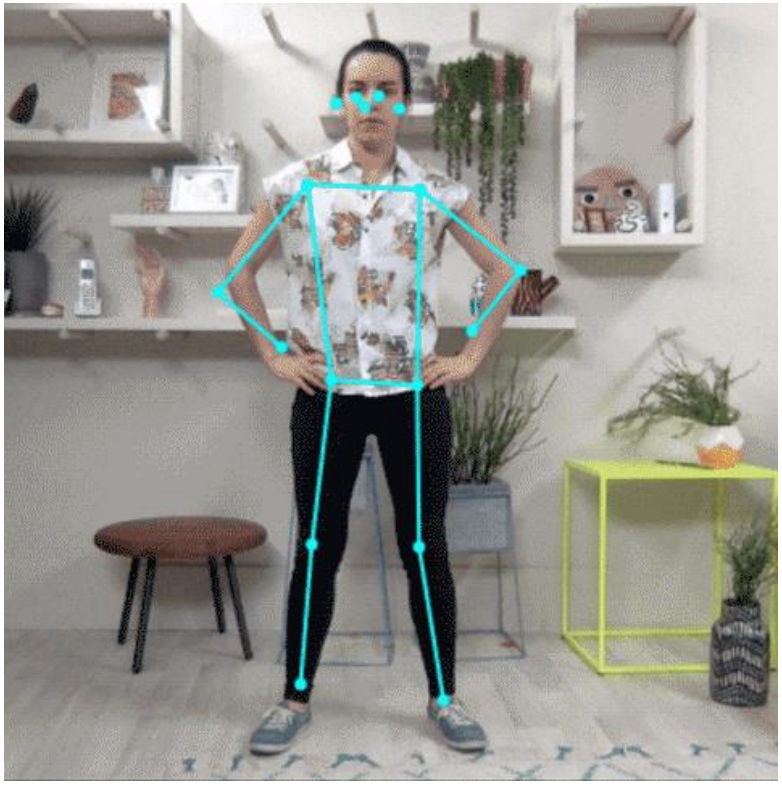
현재까지 학습한 사항

- 1차 응용
  - Generative AI : 생성기 v1
  - GAN (Generative Adversarial Networks) : 생성기 v2, 판별기
  - AutoEncoder (Encoder > Decoder) : 디코더
  - Regression
    - Localization → n * (x, y, width, height)

출처: https://medium.com/ml-research-lab/what-is-object-detection-51f9d872ece7

- Human Pose Estimation (자세 추정)

출처: https://kemtai.com/blog/the-complete-guide-to-human-pose-estimation/



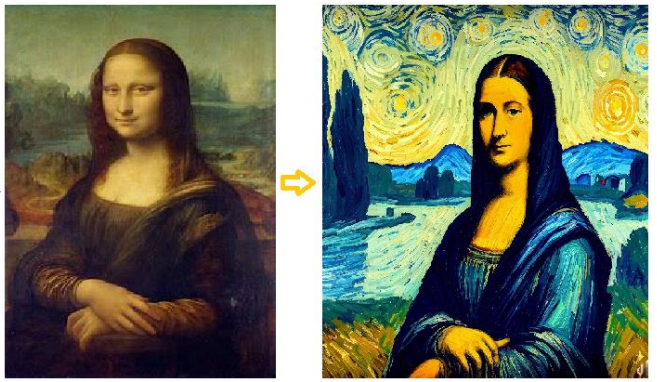
출처: https://www.tensorflow.org/lite/examples/pose_estimation/overview?hl=ko

Layers

현재까지 학습한 사항

출처: https://magazine.hankyung.com/business/article/202009230709b

- 2차 응용
  - 생성기 → ~~Data Augmentation~~
    - Data Augmentation (데이터 증강)
      - 출처: https://www.cubox.ai/board/blog/board_view.php?&page=1&num=625
      - 이동, 회전 반전, 확대, 축소, 필터(색조정 등), 투명도 / 여러 이미지 조합
      - AutoAugment (2018년. 강화학습), Fast AutoAugment (2019년. 강화학습 + 최적화)
      - RandAugment 2019년), UniformAugment (2020년)
  - Style transfer (Generative AI)
  - Filter 효과 추가
  - 이미지 해상도 확대
  - Image inpainting (손상된 이미지 복원) → 불필요한 부분 지우기



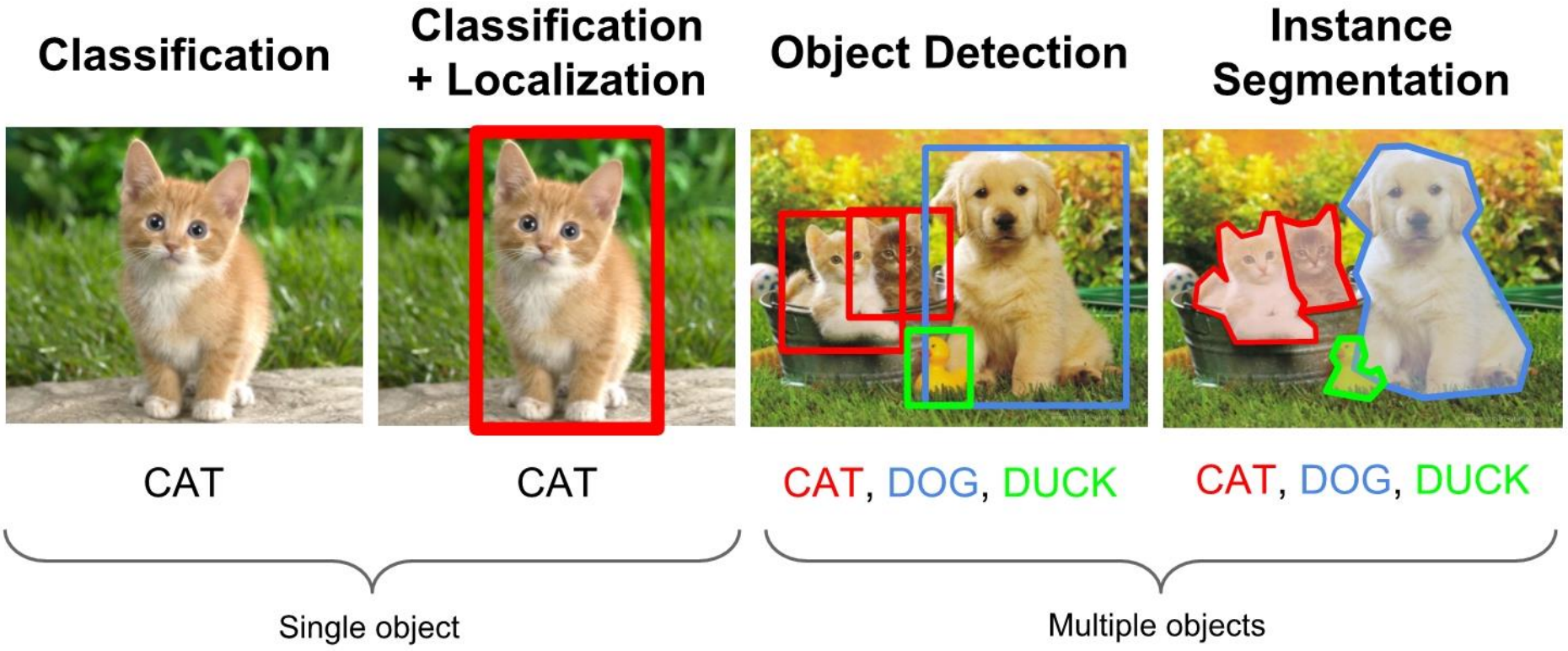
출처: https://futures-studies.tistory.com/entry/%EC%83%9D%EC%84%B1AI-%EC%83%9D%EC%84%B1AI-%EA%B8%B0%EC%B4%88-%EC%8A%A4%ED%83%80%EC%9D%BC-%ED%8A%B8%EB%9E%9C%EC%8A%A4%ED%8D%BCStyle-Transfer%EB%9E%80

출처: https://stopoverhere.tistory.com/entry/%EC%98%9B%EB%82%A0-%EC%82%AC%EC%A7%84-%EB%B3%B5%EC%9B%90-%EC%95%B1-%EC%82%AC%EC%9D%B4%ED%8A%B8

Layers

해결하여야 할 사항

- Classfication (분류)
- Localization
- Object Detection (객체 탐지)
- Instance Segmentation (영역 분할)



출처: https://ys-cs17.tistory.com/18

Layers

해결하여야 할 사항

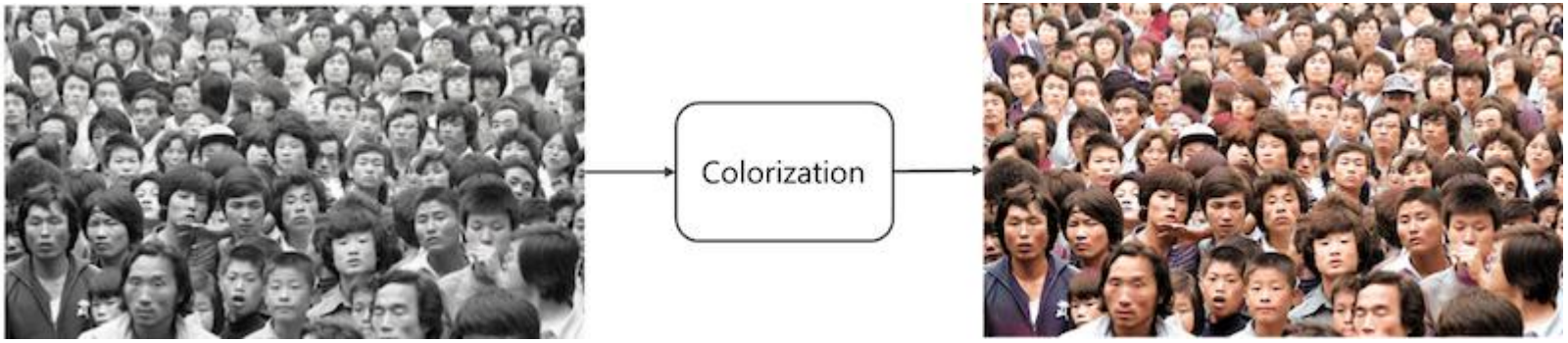
- Image recognition (얼굴 인식)
  - Verification : 동일인 여부 판단 (1:1)
  - Identification : 유사한 인물 찾기 (1:n)

출처: https://www.cubox.ai/board/blog/board_view.php?&page=1&num=121

Layers

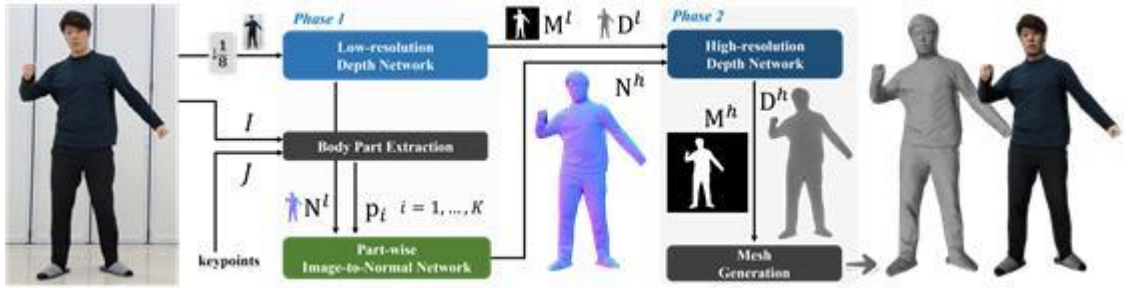
해결하여야 할 사항

- 영상 복원
  - 흑백 사진을 칼라화



출처: https://devocean.sk.com/blog/techBoardDetail.do?ID=164235

  - 평면 사진으로 3D 객체 생성

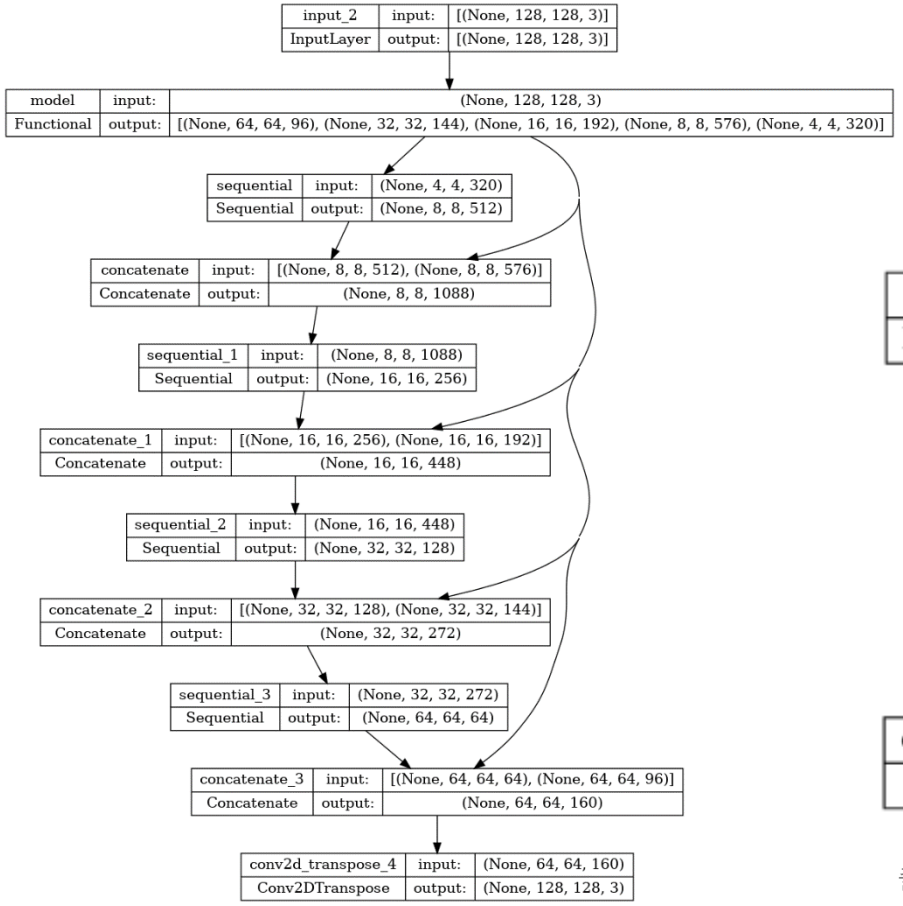


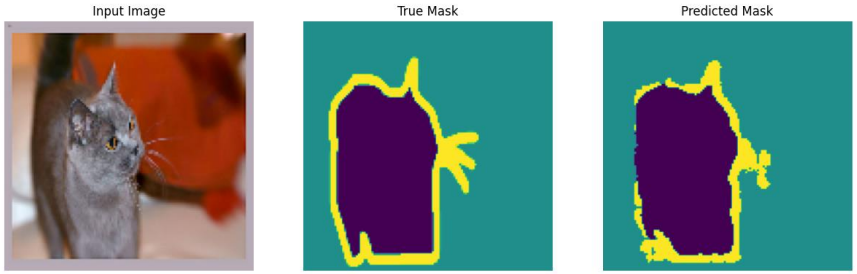
출처: http://www.mtnews.net/m/view.php?idx=16323

Layers

Image Segmentation (의미 세그먼테이션)

- 이미지를 여러 segment로 분할하는 process
  - 각 픽셀이 특정 class와 연관되어 있는지 예측 ←→ 객체 탐지 (직사각형 영역에서 객체를 탐지)
- U-Net : https://github.com/jakeret/tf_unet





input_2	input:	[(None, 128, 128, 3)]
InputLayer	output:	[(None, 128, 128, 3)]

```
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
```

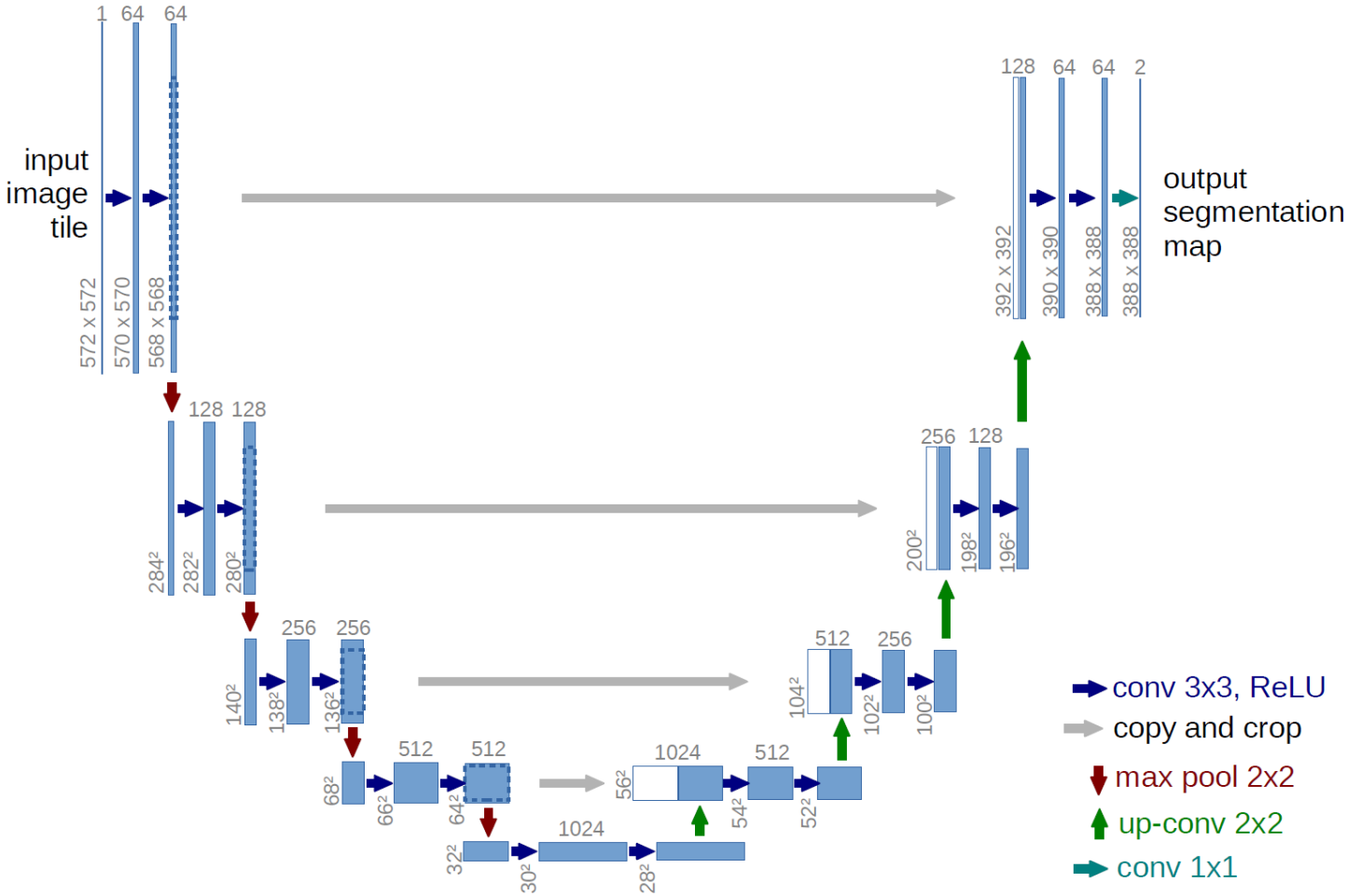
conv2d_transpose_4	input:	(None, 64, 64, 160)
Conv2DTranspose	output:	(None, 128, 128, 3)

출처: https://www.tensorflow.org/tutorials/images/segmentation?hl=ko

Layers

Image Segmentation (의미 세그먼테이션)

- U-Net : https://github.com/jakeret/tf_unet



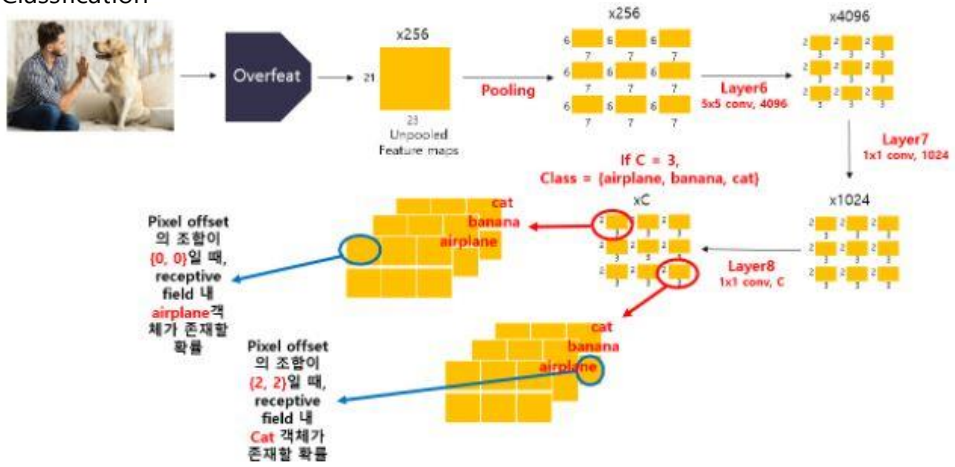
출처: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

Layers

Object Detection

- OverFeat : https://arxiv.org/abs/1312.6229
  - Sliding Window와 확률을 사용

Classfication



출처: https://velog.io/@kangtae/%EB%85%BC%EB%AC%B8%EB%A6%AC%EB%B7%B0-OverFeat

R-CNN (2013.11) → OverFeat (ICLR' 14) → MultiBox (CVPR' 14) → SPP-Net (ECCV' 14) → MR-CNN (ICCV' 15) → DeepBox (ICCV' 15) → AttentionNet (ICCV' 15)

Fast R-CNN (ICCV' 15) → DeepProposal (ICCV' 15) → Faster R-CNN (NIPS' 15) → OHEM (CVPR' 16) → YOLO v1 (CVPR' 16) → G-CNN (CVPR' 16) → AZNet (CVPR' 16)

Inside-OutsideNet(ION) (CVPR' 16) → HyperNet (CVPR' 16) → CRAFT (CVPR' 16) → MultiPathNet(MPN) (BMVC' 16) → SSD (ECCV' 16) → GBDNet (ECCV' 16)

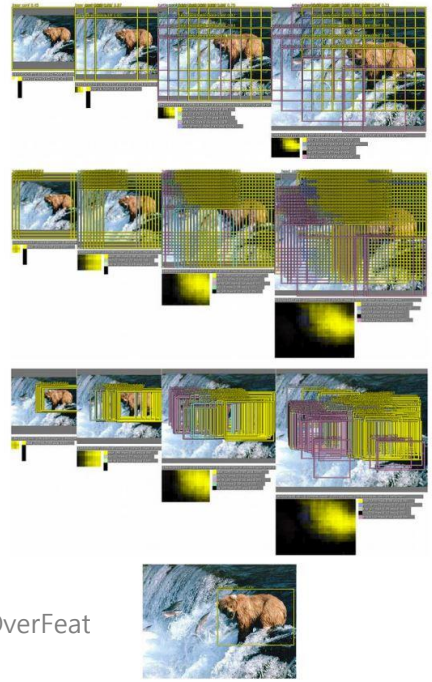
CPF (ECCV' 16) → MS-CNN (ECCV' 16) → R-FCN (NIPS' 16) → PVANET (NIPSW' 16) → DeepID-Net (PAMI' 16) → NoC (TPAMI' 16) → DSSD (arXiv' 17) → TDM (CVPR' 17) → YOLO v2 (CVPR' 17)

Feature Pyramid Net(FPN) (CVPR' 17) → RON (CVPR' 17) → DCN (ICCV' 17) → DeNet (ICCV' 17) → CoupleNet (ICCV' 17) → RetinaNet (ICCV' 17) → DSOD (ICCV' 17)

Mask R-CNN (ICCV' 17) → SMN (ICCV' 17) → YOLO v3 (arXiv' 18) → SIN (CVPR' 18) → STDN (CVPR' 18) → RefineDet (CVPR' 18) → MLKP (CVPR' 18) → Relation-Net (CVPR' 18)

Cascade R-CNN (CVPR' 18) → RFBNet (ECCV' 18) → CornetNet (ECCV' 18) → Pelee (NIPS' 18) → MethAnchor (NIPS' 18) → SNIPER (NIPS' 18) → M2Det (AAAI' 19) ...

출처: https://velog.io/@pabiya/OverFeatIntegrated-Recognition-Localization-and-Detectionusing-Convolutional-Networks



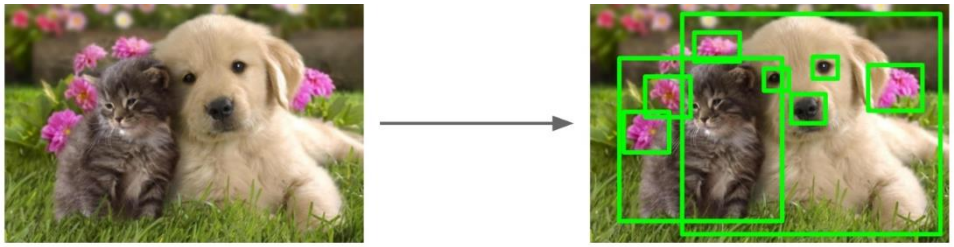
Localization

다양한 해상도의 사진 사용

voting 방식으로 경계상자와 분류를 최종적으로 수행

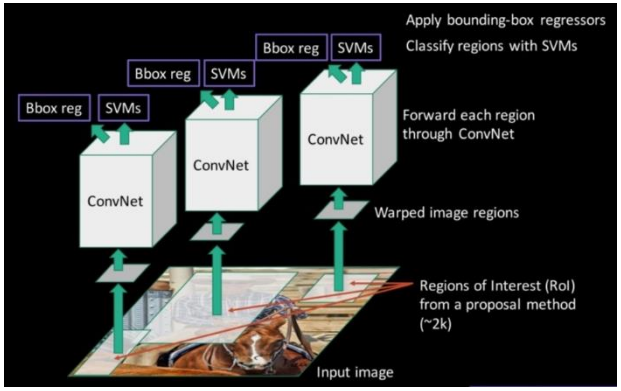
Object Detection

- R-CNN (Region Proposals CNN)
  - R-CNN 이전에는, 해상도 크기별로 이미지를 분할하여 Classfication & Localization 진행
  - R-CNN의 개요
    - Region Proposals (영역 제안) : 전통적인 광학 기법으로 객체가 있을 만한 영역을 추천
      - Edge box 알고리즘 사용을 추천



출처: https://ys-cs17.tistory.com/18

    - 추천 받은 영역에 대해서 Classfication & Localization 진행



출처: https://ys-cs17.tistory.com/18

Layers

Object Detection

- R-CNN (Region Proposals CNN)
  - Region Proposals (영역 제안) : 전통적인 광학 기법으로 객체가 있을 만한 영역을 추천
  - 이미지를 각 영역으로 분할
  - 각 영역에 대해서 CNN 적용

- Fast R-CNN : R-CNN보다 약 25배 빠름
  - Region Proposals (영역 제안) : 전통적인 광학 기법으로 객체가 있을 만한 영역을 추천
  - 원본 이미지에 CNN을 적용하여 특징 맵 생성
  - 특징 맵에 Region Proposals 결과를 반영하여 특징 맵을 pooling
  - Pooling한 것으로 객체를 탐지

이미지 분할 대신에
특징맵 Pooling

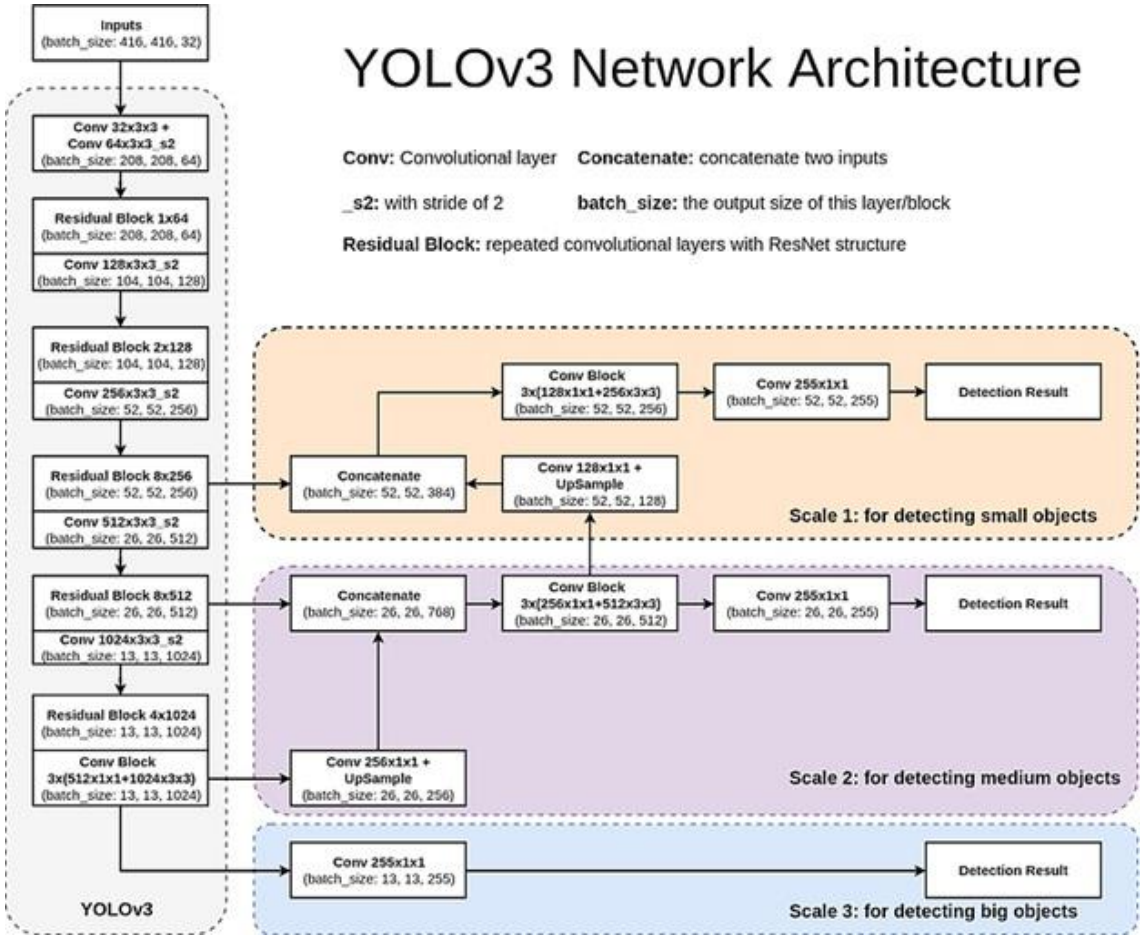
- Faster R-CNN : R-CNN보다 약 250배 빠름
  - https://github.com/tensorpack/tensorpack/tree/master/examples/FasterRCNN
  - 이미지에 CNN을 적용하여 특징 맵 생성
  - 특징 맵에 RPN (Region Proposal Network)을 적용하여 영역 제안
  - 특징 맵에 RPN 결과를 반영하여 특징 맵을 pooling
  - Pooling한 것으로 객체를 탐지

Region Proposals 대신에
RPN (Region Proposal Network) 사용

Layers

Object Detection

- YOLO (You Only Look Once) v3 : https://arxiv.org/pdf/1804.02767.pdf
  - 속도가 빨라 실시간에 적합 (초당 YOLO 45 frames)



- https://herbwood.tistory.com/21
- Feature map
  - 52 * 52
  - 26 * 26
  - 13 * 13

출처: https://rubber-tree.tistory.com/119

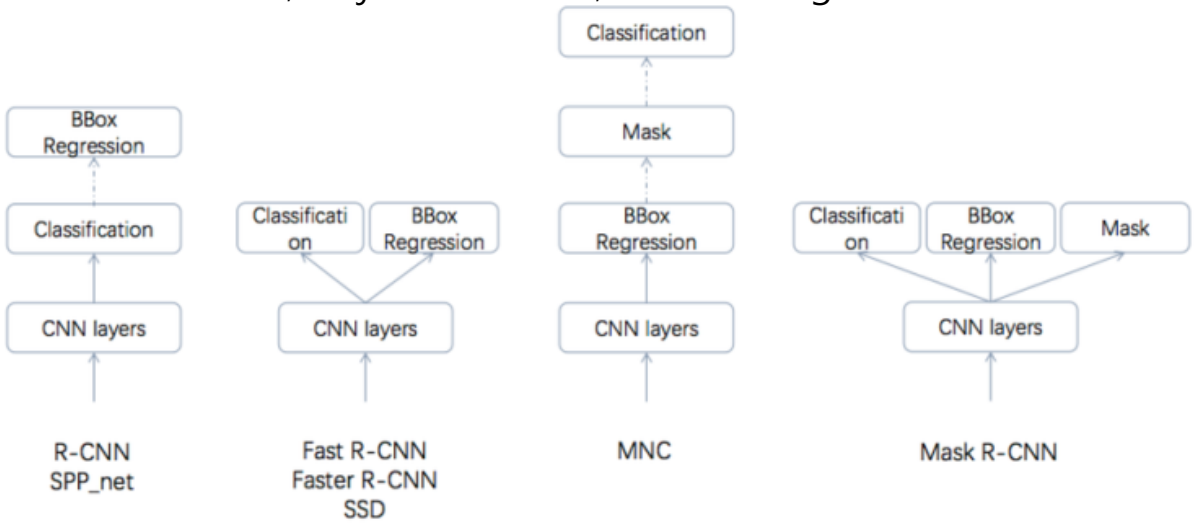
출처:
https://d33wubrfki0l68.cloudfront.net/c6fd049f28b66dbd35faed6965905ec6281f7d7d/c0399/assets/images/yolo/yolo-architecture.webp

Layers

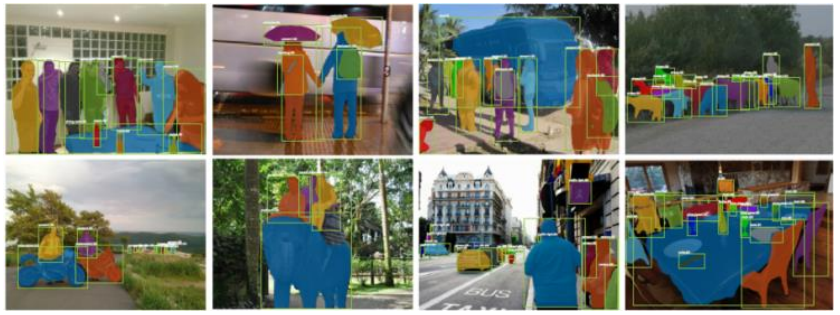
Instance Segmentation

- Mask R-CNN : https://herbwood.tistory.com/20
  - Classfication, Object Detection, Instance Segmentation을 동시에 진행

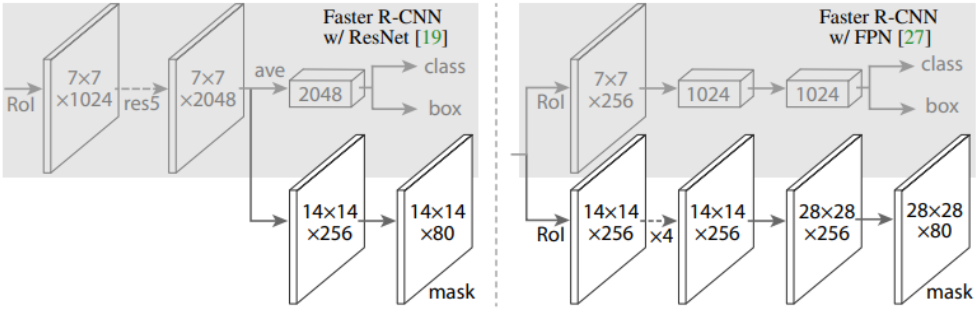
BBox (Bounding Box)
FPN (Feature Pyramid Networks)
FCN (Fully Convolutional Network)



출처: https://m.blog.naver.com/sogangori/221012300995



출처: https://m.blog.naver.com/sogangori/221012300995



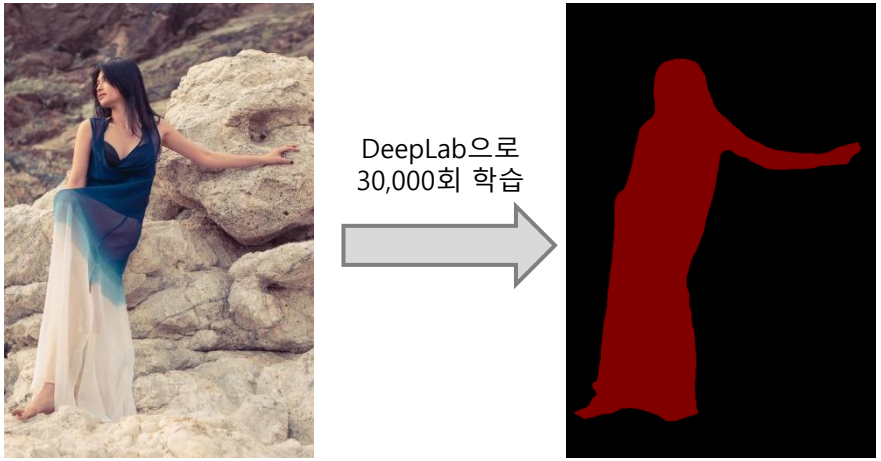
출처: https://wikidocs.net/163945

Layers

Instance Segmentation

- DeepLab-v3+ : https://github.com/tensorflow/models/tree/master/research/deeplab
  - https://www.tensorflow.org/lite/examples/segmentation/overview?hl=ko

출처: https://github.com/tensorflow/models/tree/master/research/deeplab



출처: https://m.blog.naver.com/PostView.naver?blogId=skyshin0304&logNo=221591774871&targetKeyword=&targetRecommendationCode=1

Image외의 데이터로 확장

- 1차원 Tensor
  - OHE (One-Hot Encoding)에서 word
  - Text : Word2Vec, GloVe
  - Music : 음악
- 2차원 Tensor
  - 흑백 이미지 : x, y
  - OHE (One-Hot Encoding)에서 phrase
- 3차원 Tensor
  - 컬러 이미지 : x, y, rgb
  - 흑백 동영상 (x, y, time)
- 4차원 Tensor
  - 컬러 동영상 (x, y, rgb, time)
- 5차원 Tensor

Layers

Layer 종류 : keras.layers.RNN, SimpleRNN

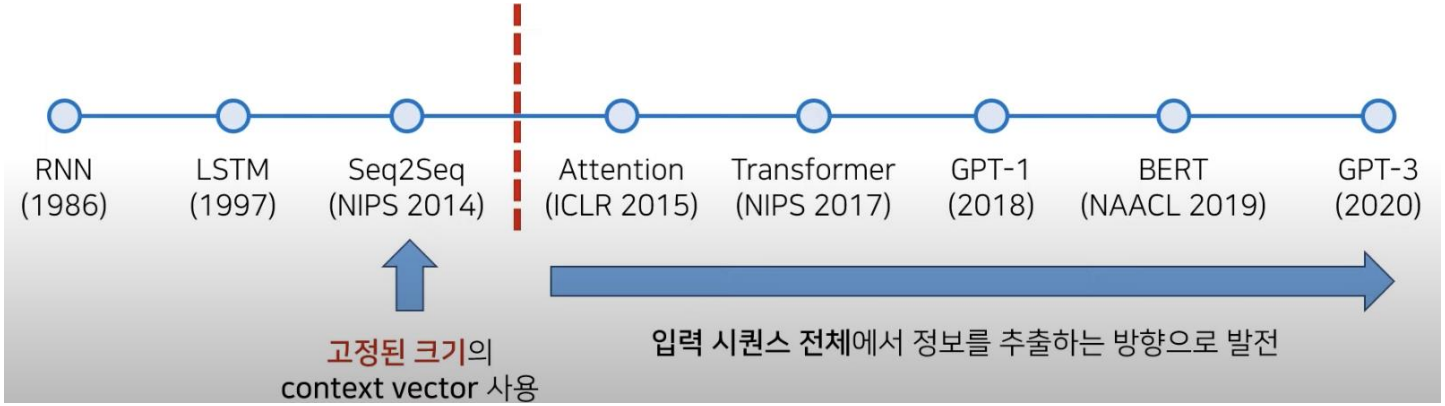
- RNN (Recurrent Neural Networks, 순환 신경망)
  - 순서가 있는 데이터를 입력 받아 변화하는 입력에 대한 출력을 얻는다.
  - 순차 입력 : 텍스트, 음성, 시계열 등

  - 음성 인식, 언어 모델링, 기계 번역, 감정 분석, 이미지 캡션 등
  - Hidden Layer (은닉층) + Hidden Status (은닉 상태)
  - BPTT (BackPropagation Through Time, 시간에 따른 역전파)

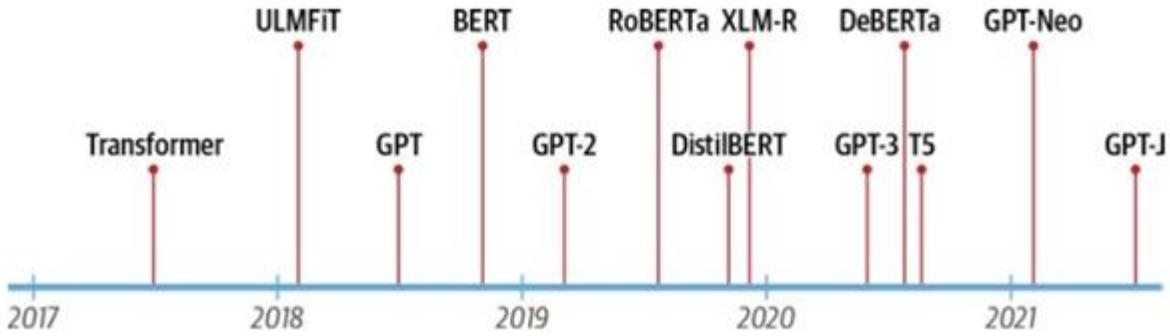
Layers

Layer 종류 : keras.layers.RNN, SimpleRNN

- RNN의 종류
  - SimpleRNN
  - LSTM (Long Short-Term Memory)
  - GRU (Gated Recurrent Unit)



출처: https://www.youtube.com/watch?v=AA621UofTUA

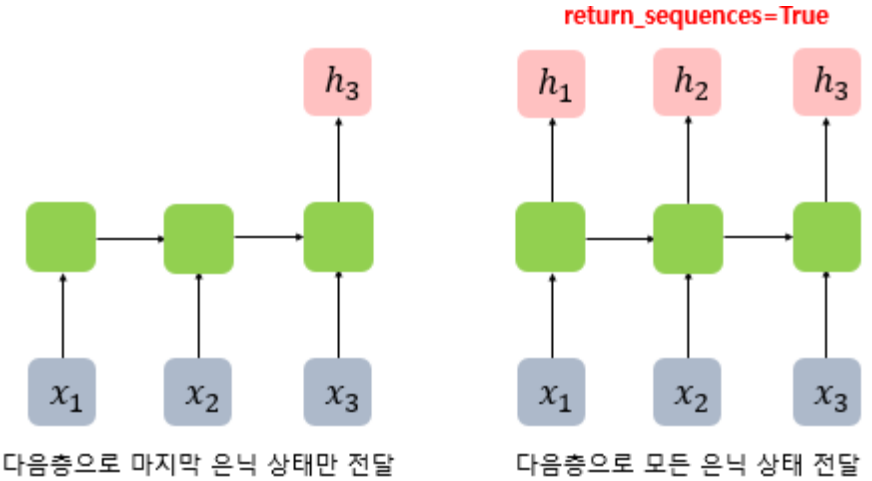


출처: https://m.hanbit.co.kr/channel/category/category_view.html?cms_code=CMS5215583920

Layers

Layer 종류 : keras.layers.RNN, SimpleRNN

- RNN (Recurrent Neural Networks, 순환 신경망)
  - x를 입력 받아 h (Hidden state)를 출력 한다.
    - h는 다음 h를 계산할 때, 입력과 함께 사용 한다.
  - $h_t = \emptyset(h_{t-1}, x_t)$: Hidden state

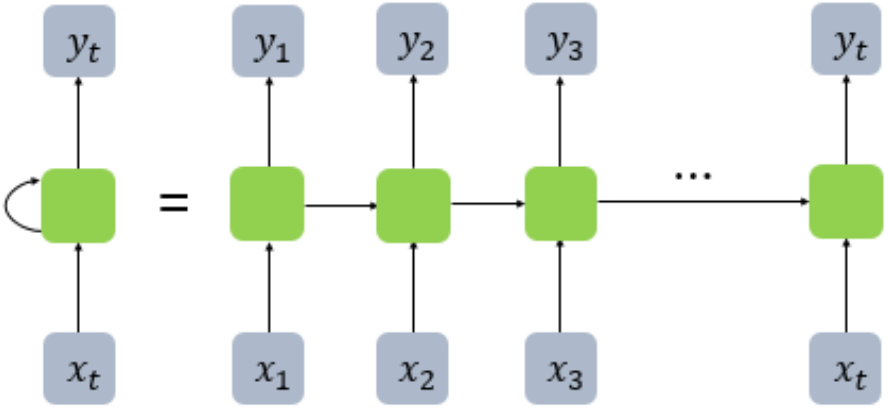


출처: https://wikidocs.net/22886

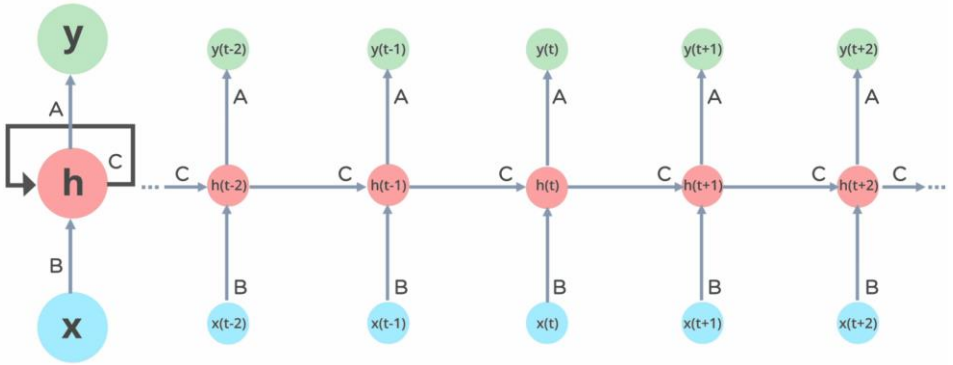
```
last_state = SimpleRNN(3)(train_x)

last_state, last_state = SimpleRNN(3, return_state=True)(train_x)
hidden_states = SimpleRNN(3, return_sequences=True)(train_x)

hidden_states, last_state = SimpleRNN(3, return_sequences=True, return_state=True)(train_x)
```



출처: https://wikidocs.net/22886

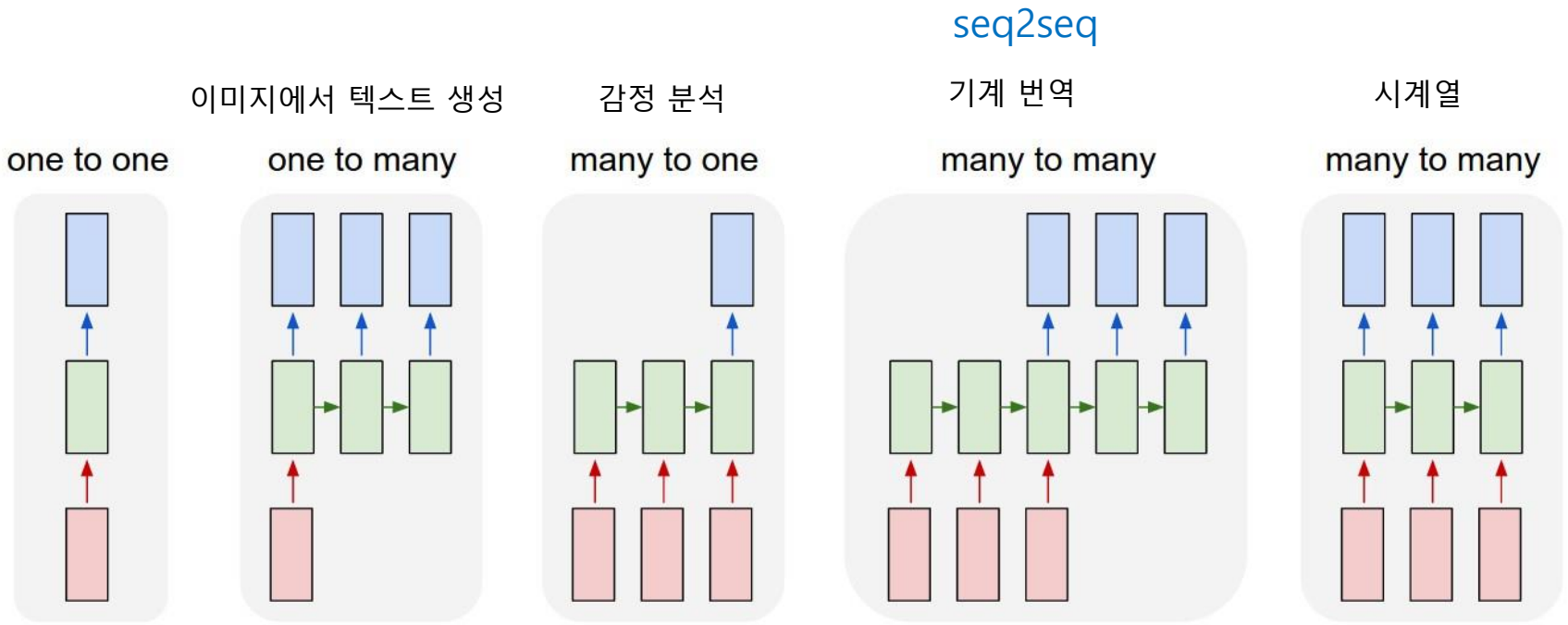


출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

Layers

Layer 종류 : keras.layers.RNN, SimpleRNN

- RNN (Recurrent Neural Networks) 위상
  - NLP (Natural Language Processing, 자연어 처리)
    - 언어 모델: 이전 단어가 주어질 때 텍스트 내에서 어떤 단어의 확률을 예측하는 모델



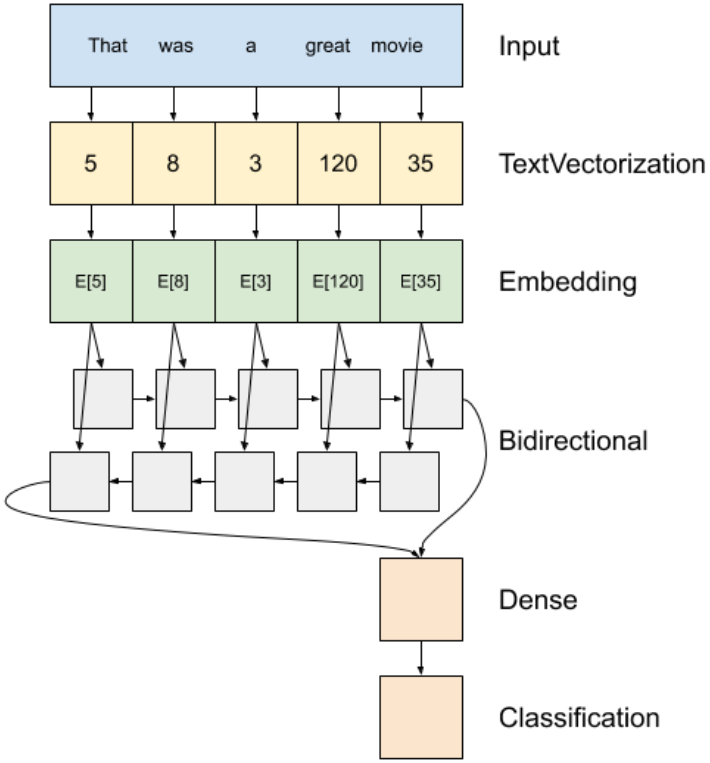
출처: https://wandb.ai/wandb_fc/korean/reports/Keras-LSTM---VmlldzoxNTEzMTg3

Layers

Layer 종류 : keras.layers.RNN, SimpleRNN

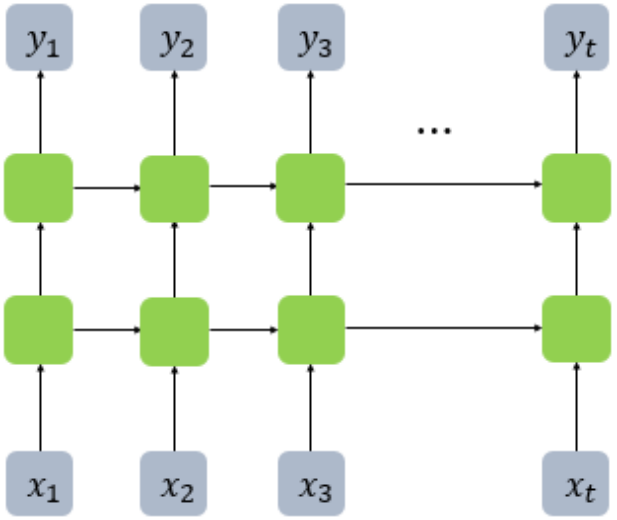
- Bidirectional
  - Forward, Backward 양방향으로 RNN 실행

```
bilstm = Bidirectional(LSTM(3, return_sequences=False, return_state=True)
hidden_states, forward_h, forward_c, backward_h, backward_c = bilstm(train_X)
```



출처: https://www.tensorflow.org/text/tutorials/text_classification_rnn

Deep RNN



출처: https://wikidocs.net/22886

Layers

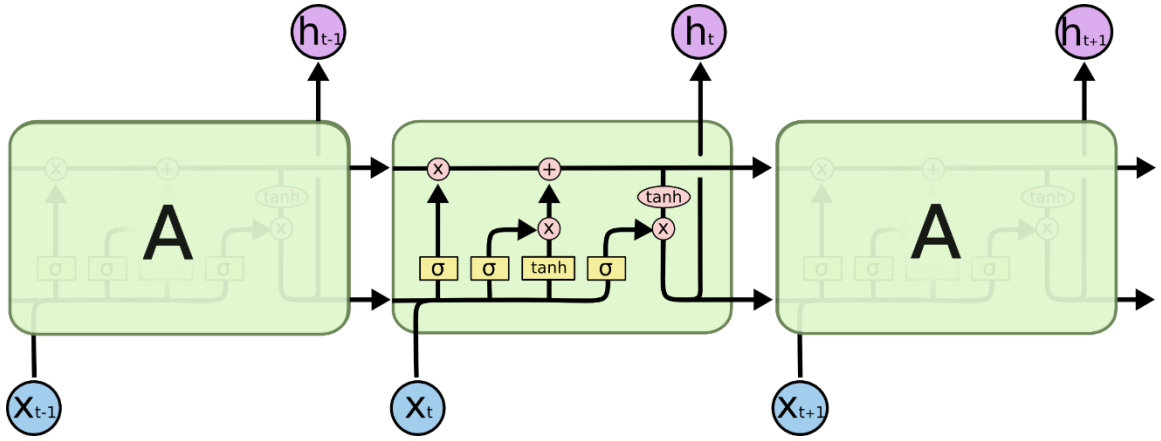
Layer 종류 : keras.layers.LSTM

- LSTM (Long Short-Term Memory)
  - x를 입력 받아 h (Hidden state)를 출력 한다.
    - c (Cell state)를 별도로 관리 한다.
    - x로 c를 업데이트 한다.
    - c는 다음 h를 계산할 때, 입력과 함께 사용 한다.

```
last_state = LSTM(3)(train_x)

last_state, last_state, last_cell_state = LSTM(3, return_state=True)(train_x)
hidden_states, last_cell_state = LSTM(3, return_sequences=True)(train_x)

hidden_states, last_state, last_cell_state = LSTM(3, return_sequences=True, return_state=True)(train_x)
```

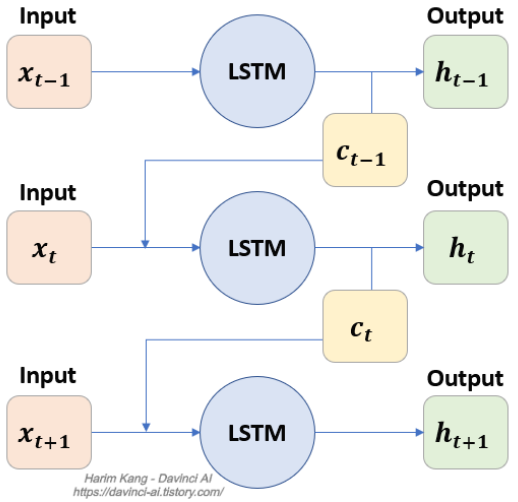


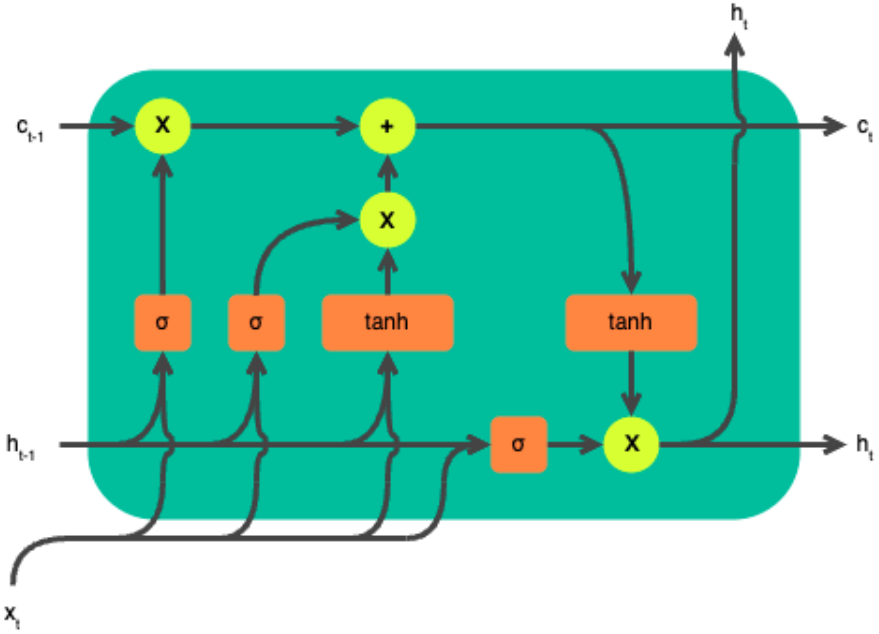
출처: https://www.shiksha.com/online-courses/articles/rnn-vs-gru-vs-lstm/

Layers

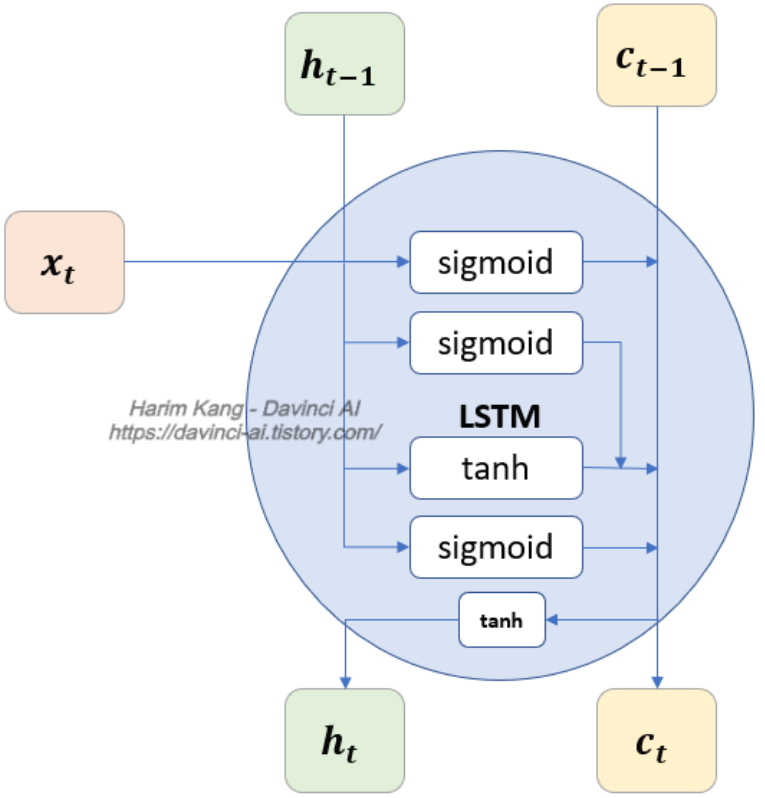
Layer 종류 : keras.layers.LSTM

- LSTM (Long Short-Term Memory)
  - Cell의 상태 c를 보존
  - 바로 직전 상태에 대한 의존성을 줄임





출처: https://www.machinelearningnuggets.com/tensorflow-lstm/

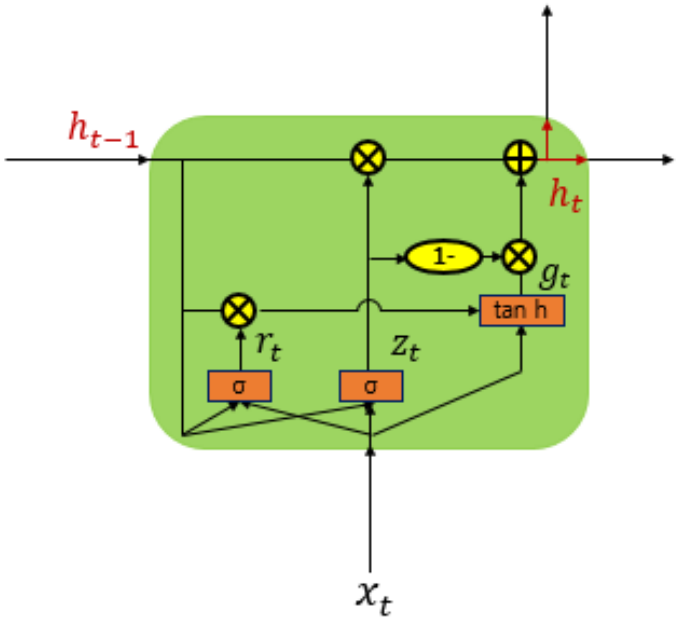


출처: https://davinci-ai.tistory.com/30

작성중

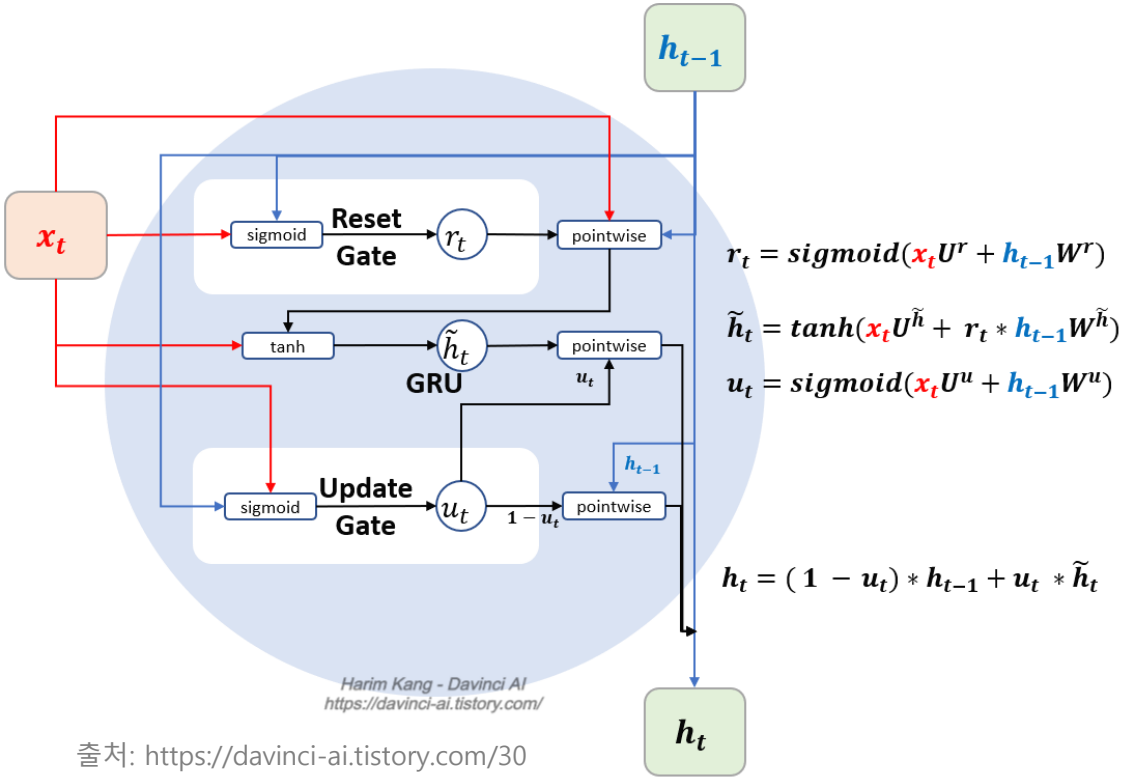
Layer 종류 : keras.layers.GRU

- GRU (Gated Recurrent Unit)
  - 복잡한 LSTM을 단순화함
  - 성능은 LSTM과 유사함



출처: https://wikidocs.net/22889

- NLP process
  - Embedding
  - Encoding
  - Attention
  - Predict



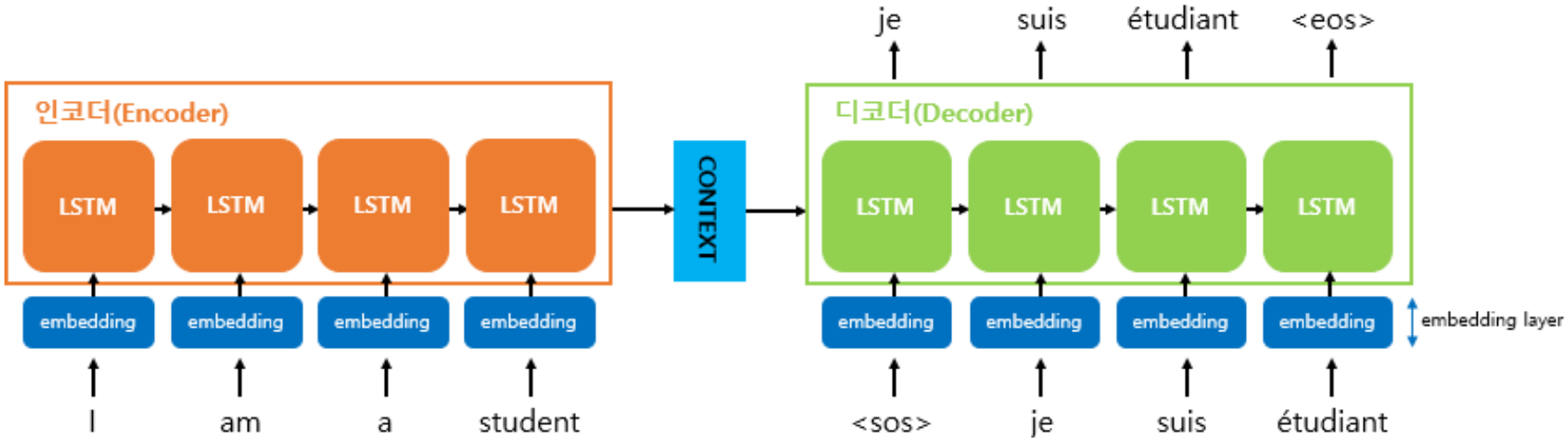
출처: https://davinci-ai.tistory.com/30

Layers

Layer 종류 : keras.layers.RNN

- Seq2Seq
  - Encoder-Decoder 구조
  - 입력된 sequence로부터 다른 domain의 sequence를 출력
    - Chat bot : 입력. 질문, 출력. 답변
    - Machine Translation : 입력. 영어, 출력. 프랑스어
    - Text Summarization, STT (Speech to Text)

  - Input tensor for embedding : (batch_size, number_of_encoder_timesteps)
  - Output tensor for embedding : (batch_size, number_of_encoder_timesteps, encoder_embedding_dim)

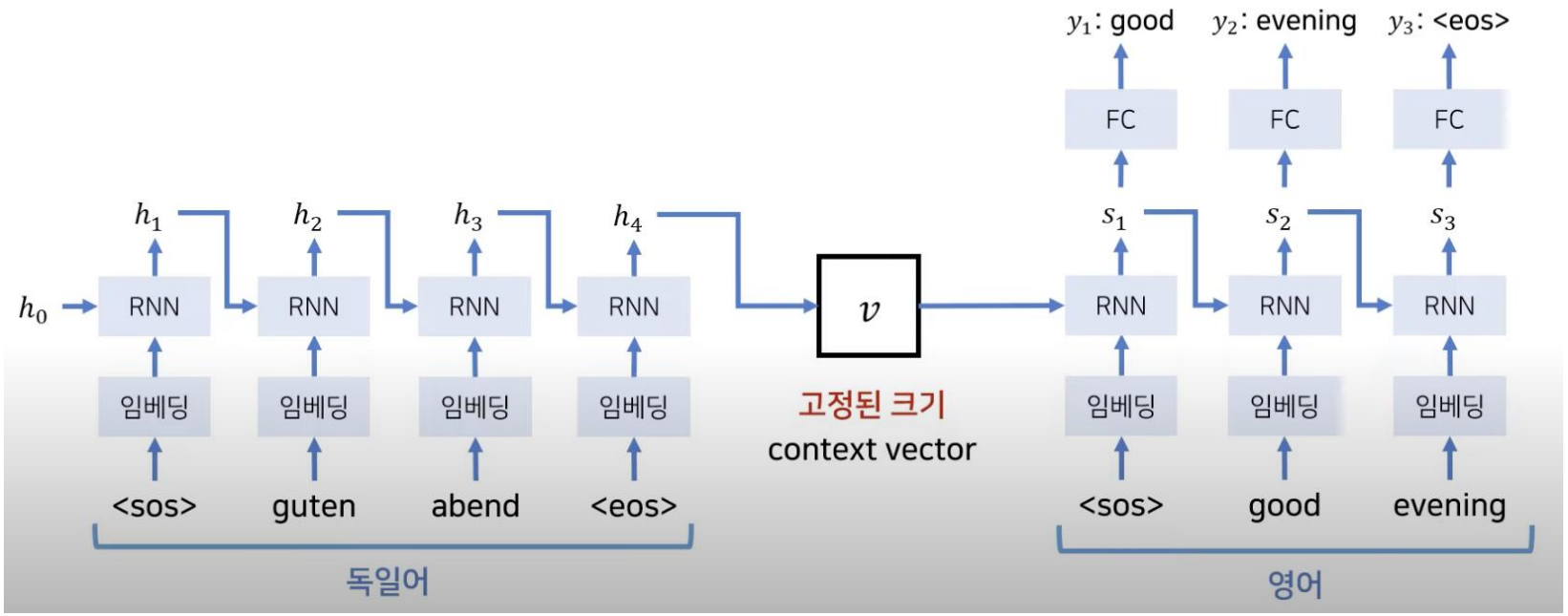


출처: https://wikidocs.net/24996

Layers

Layer 종류 : keras.layers.RNN

- Seq2Seq의 단점
  - Encoder의 모든 정보를 context vector로 압축 한다.
  - 가장 마지막의 RNN cell의 내용이 가장 많이 반영 된다.
  - 마지막 hidden state를 제외하고, hidden states가 전혀 사용되지 않는다.



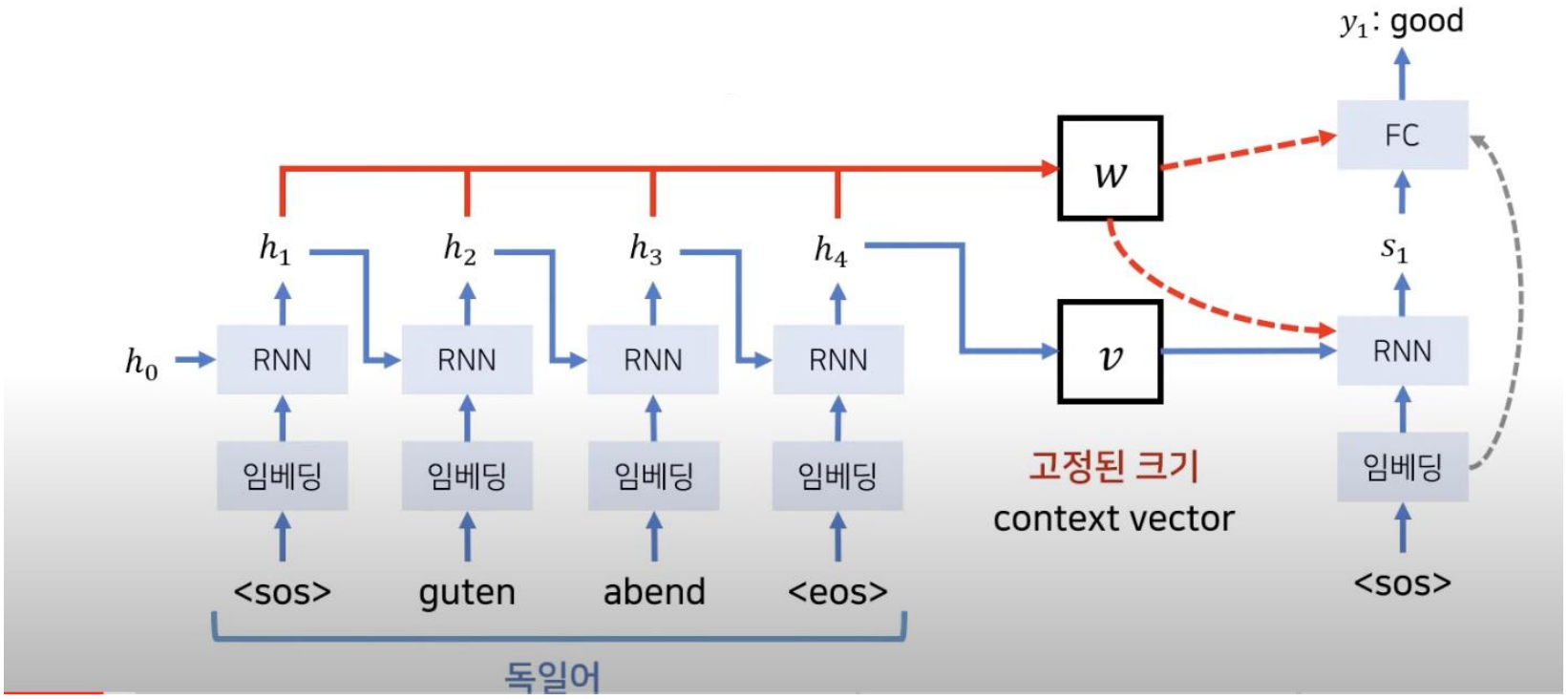
출처: https://www.youtube.com/watch?v=AA621UofTUA

Layers

Layer 종류 : keras.layers.RNN

- Seq2Seq + Attention
  - Encoder의 hidden states를 활용 한다.
  - Decoder의 모든 time step에 encoder의 hidden states를 반영 한다.

  - h4는 w에 반영된다. 따라서 v의 필요성이 없다. → Transfomer



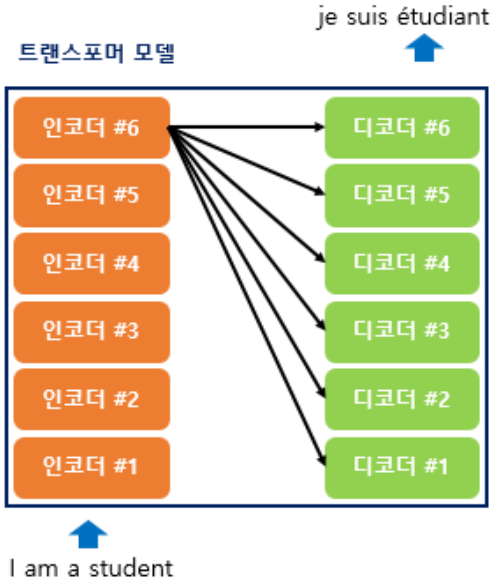
출처: https://www.youtube.com/watch?v=AA621UofTUA

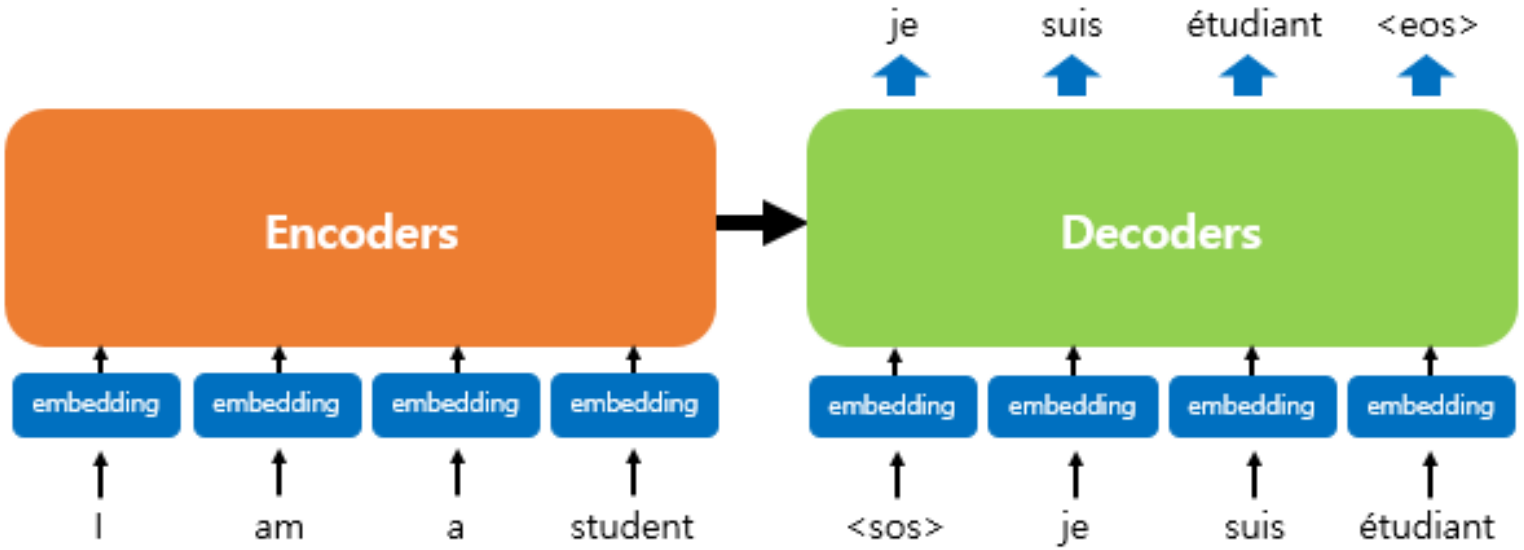
Layers

작성중

Layer 종류 : keras.layers.RNN

- Transformer
  - Attention만 사용 한다.
    - 이미 context vector 정보가 attention에 반영되어 있다.
  - Positional Encoding (각 단어의 순서 정보)를 사용 한다.
    - 순서 정보가 반영되어 있으므로 RNN을 사용할 필요가 없다.
  - Multi-Head Attention을 사용 한다.
    - 입력과 출력의 차원이 동일 하다.
    - 따라서, Encoder와 Decoder를 여러개 중첩하여 사용할 수 있다.



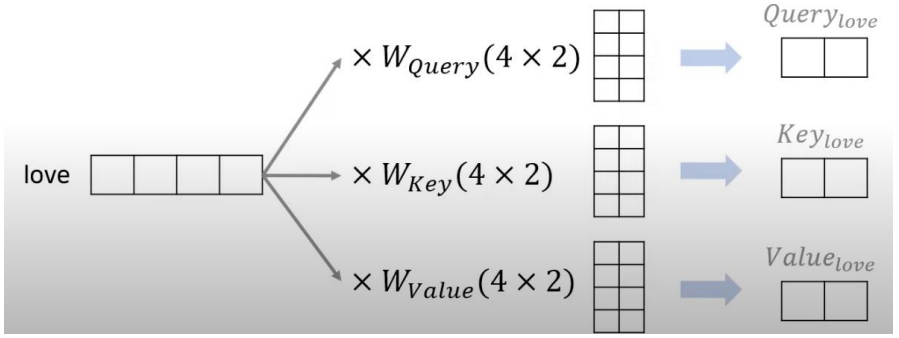


출처: https://wikidocs.net/31379

작성중

Layer 종류 : keras.layers.RNN

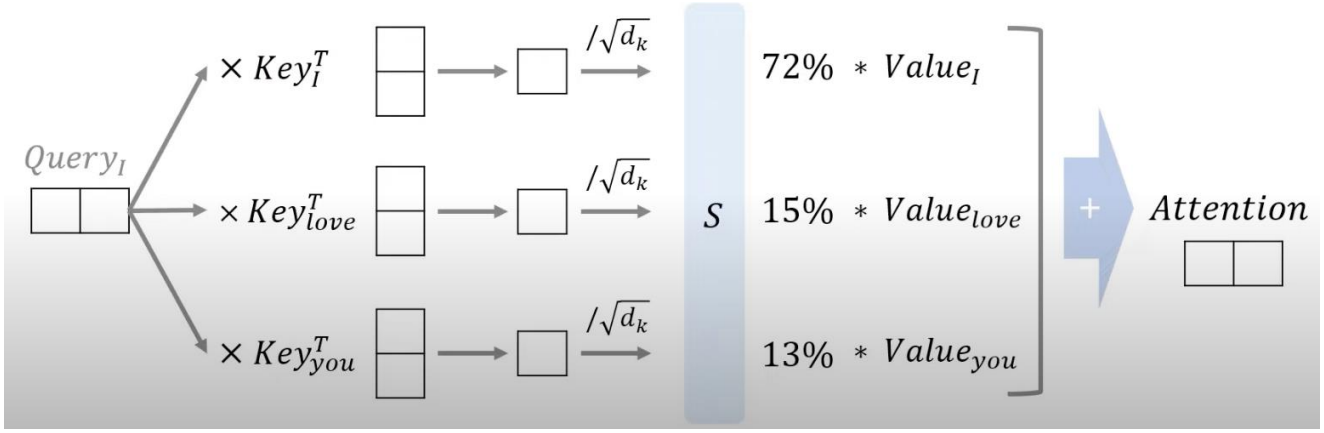
- Multi-Head Attention
  - Scaled Dot-Product Attention 방식 사용
  - 입력과 출력의 차원이 동일 하다.



출처: https://www.youtube.com/watch?v=AA621UofTUA

- Attention Function
  - Attention Value = Attention(Q, K, V)
  - x를 사용하여 Query, Key, Value를 생성 한다.

  - Query에 대해서 모든 Key와의 유사도를 계산 한다.
  - 이 유사도를 키와 맵핑되어있는 각각의 Value에 반영 한다.
  - 각각의 value를 모두 합산하여 반환 한다.



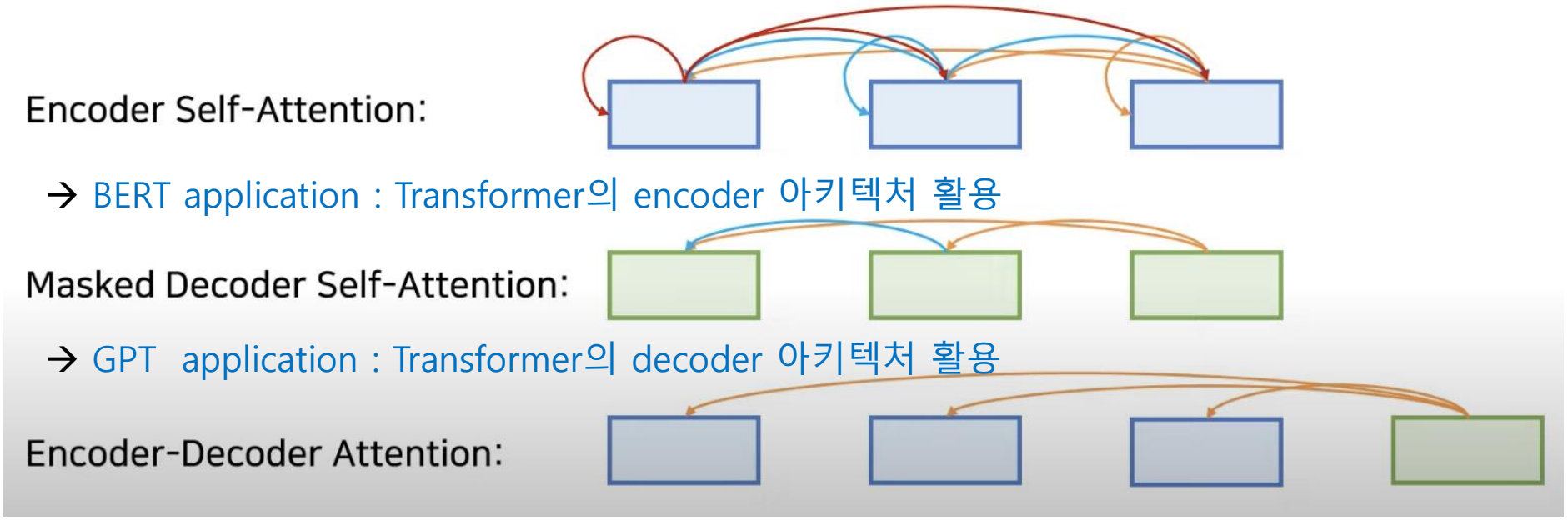
출처: https://www.youtube.com/watch?v=AA621UofTUA

작성중

Layer 종류 : keras.layers.RNN

- Attention의 종류
  - Self-Attention : key == value == query
  - Cross-Attention : key == value != query

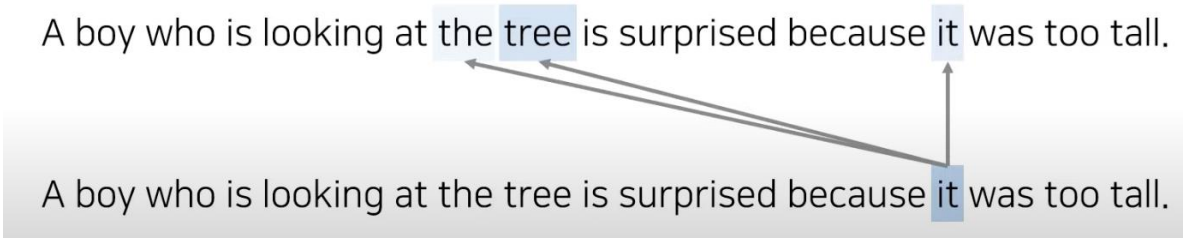
  - Masked Decoder Self-Attion : 미래의 단어를 학습하지 않는다.



작성중

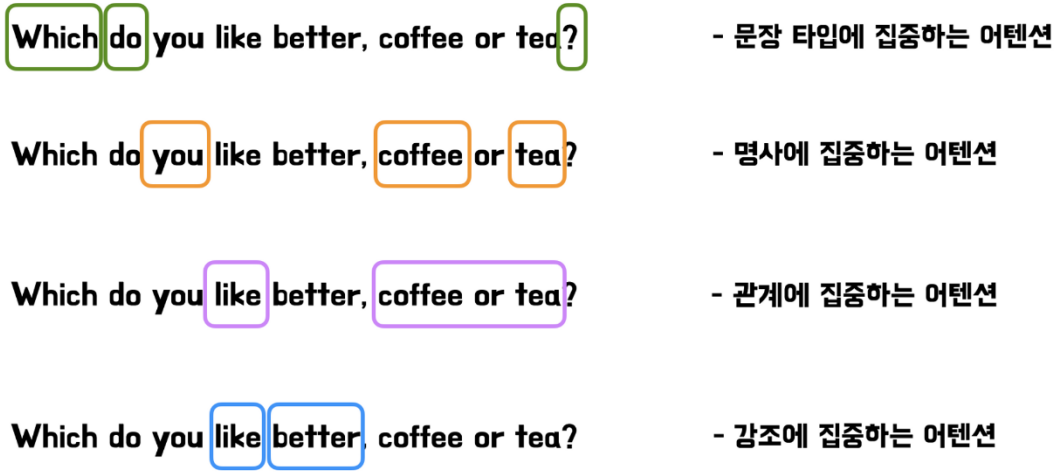
Layer 종류 : keras.layers.RNN

- Self-Attention
  - 입력 문장에서 각 단어가 다른 어떤 단어와 연관성이 높은지 계산 한다.



출처: https://www.youtube.com/watch?v=AA621UofTUA

- Multi-Head Attention

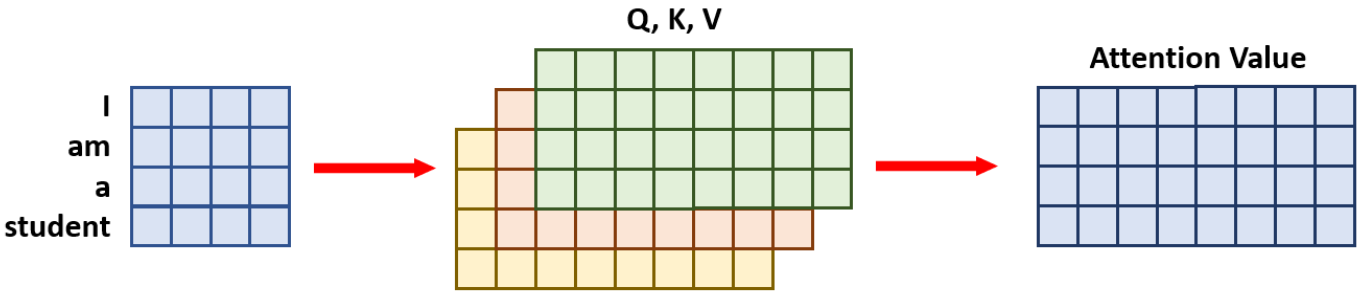


출처: https://www.blossominkyung.com/deeplearning/transformer-mha

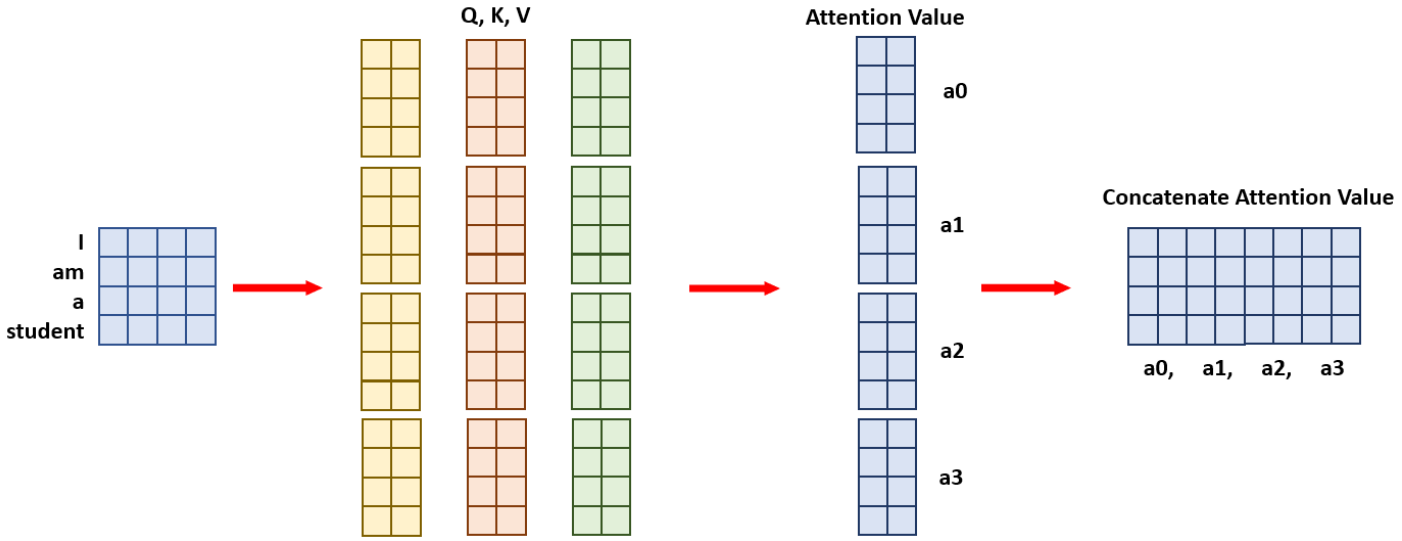
작성중

Layer 종류 : keras.layers.RNN

- Attention



- Multi-Head Attention

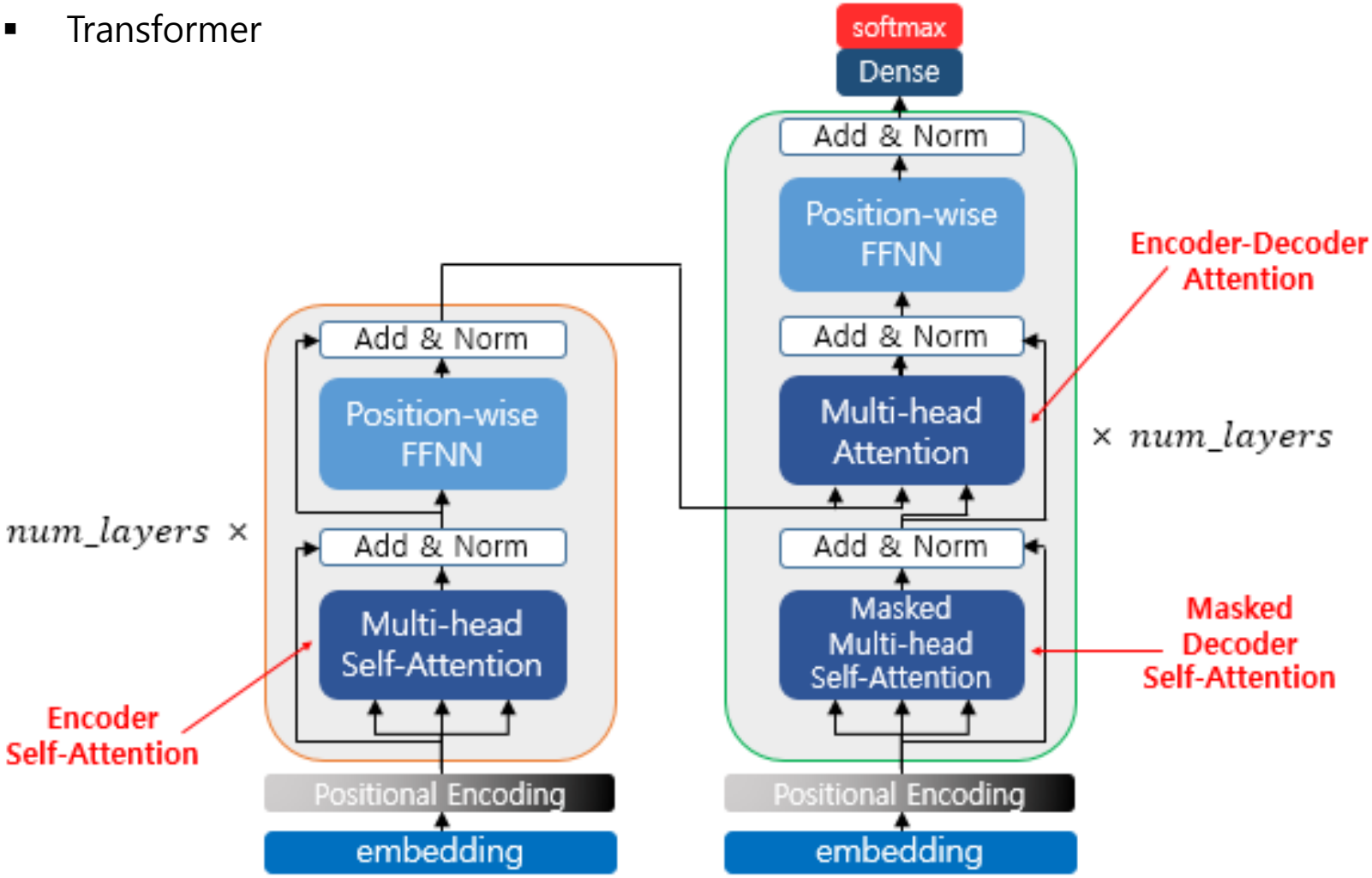


출처: https://codingopera.tistory.com/44

작성중

Layer 종류 : keras.layers.RNN

- Transformer

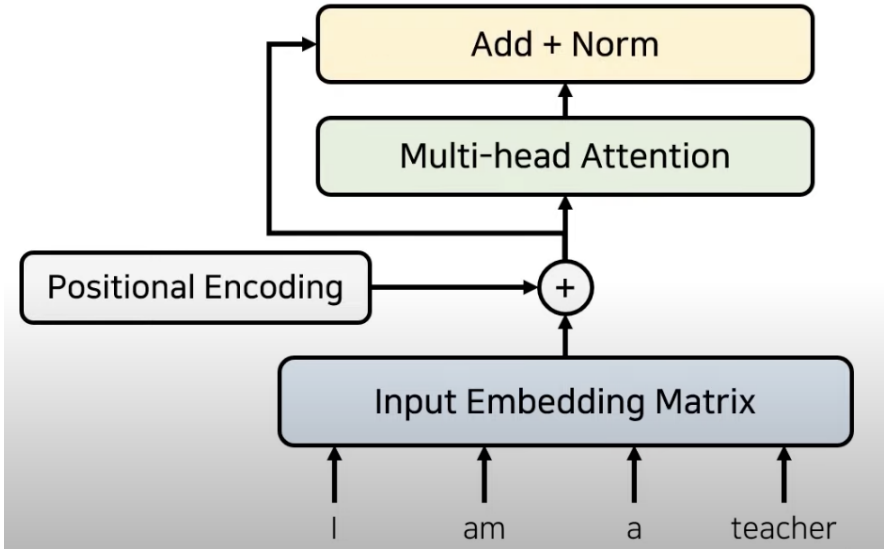


출처: https://wikidocs.net/31379

작성중

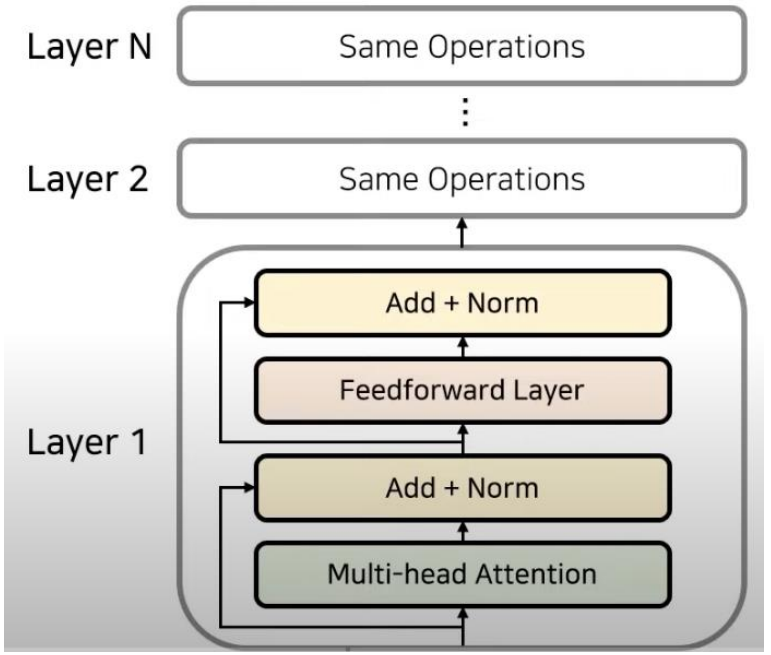
Layer 종류 : keras.layers.RNN

- Encoder in Transformer
  - Attention은 입력과 출력의 차원이 동일하기 때문에 여러개를 사용할 수 있다.



출처: https://wikidocs.net/31379

- Residual Learning
  - y = f(x) + x
  - 결과에 x를 반영하면 학습 성능이 좋아 진다.

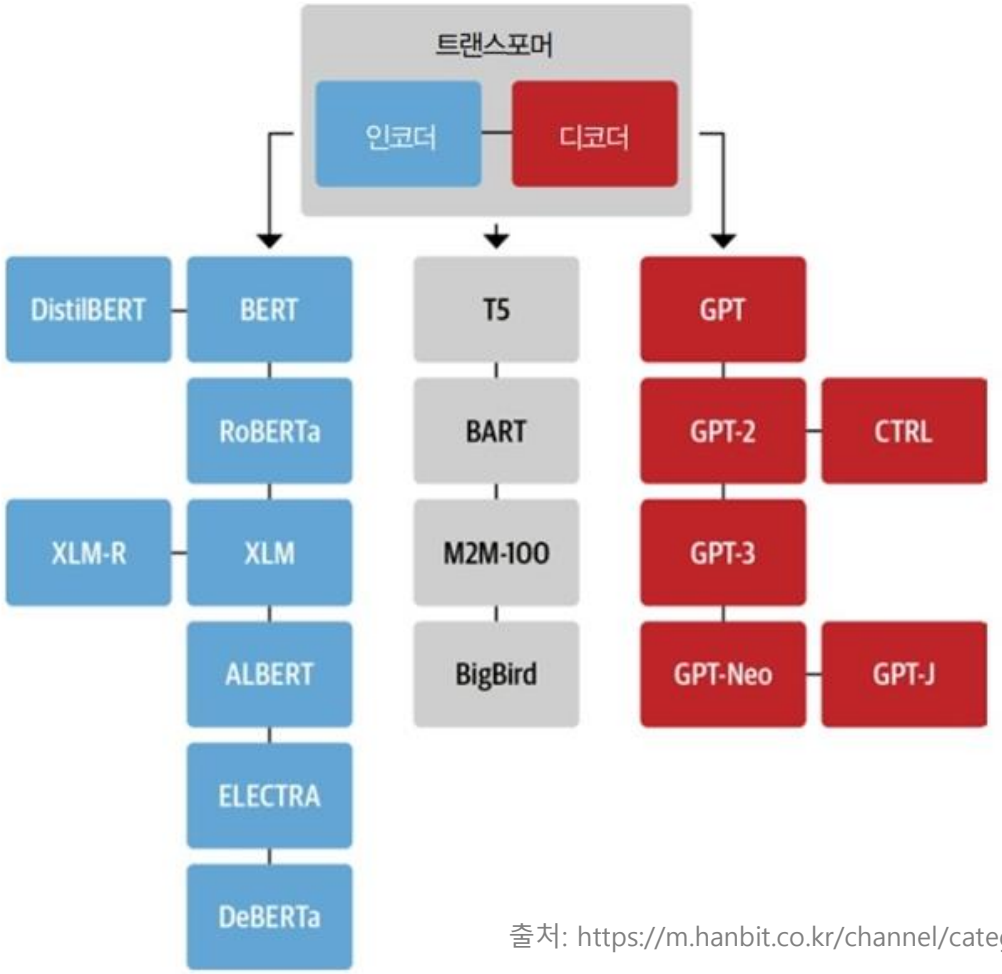


출처: https://wikidocs.net/31379

작성중

Layer 종류 : keras.layers.RNN

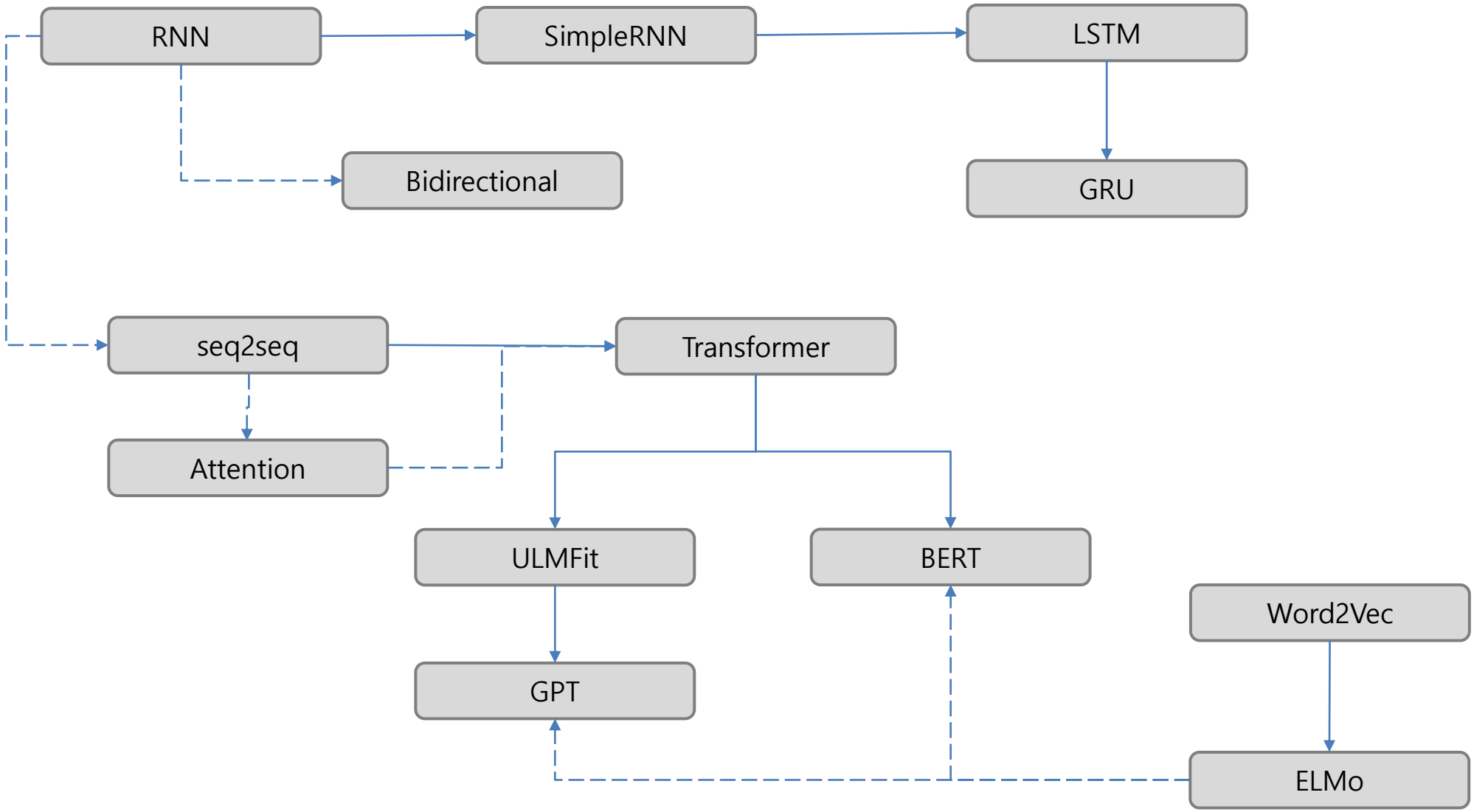
- Transformer 가계도



출처: https://m.hanbit.co.kr/channel/category/category_view.html?cms_code=CMS5215583920

Layers

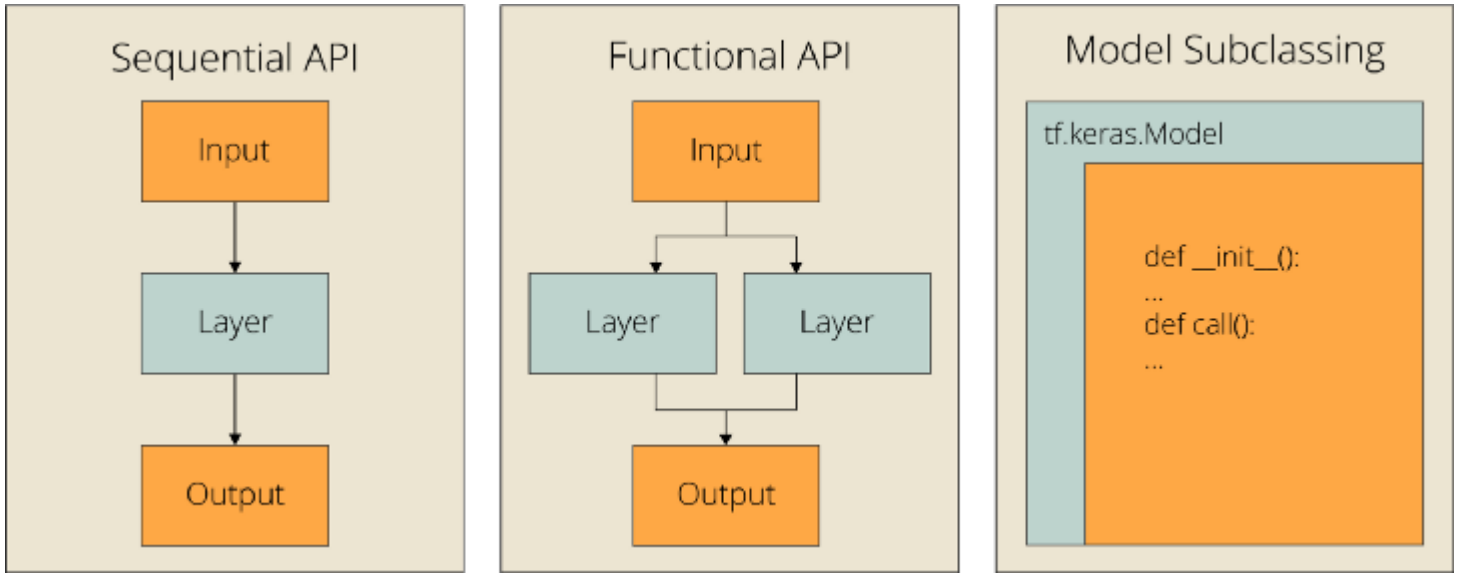
Layer 종류 : keras.layers.RNN



Model 구현 방식

Model 구현 방식

- Sequential API : 간단한 모델 구현
- Functional API : 복잡한 모델 구현
- Model Subclassing
  - Function API로 구현하기 힘든 모델 구현
  - 자유도가 제일 높은 모델 구축 방법



출처: https://wikidocs.net/106897

Model 구현 방식

Sequential API

- CNN

```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(20, (5, 5), activation='relu', input_shape=input_shape))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(keras.layers.Dropout(self.dropout))

model.add(keras.layers.Conv2D(50, (5, 5), activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(keras.layers.Dropout(self.dropout))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(self.nb_classes, activation="softmax"))

model.compile(
    optimizer=self.optimizer,
    loss=self.loss_function,
    metrics=[ self.metrics ]
)
```

Model 구현 방식

Functional API

- Peephole LSTM 샘플

```
inputs = keras.Input(shape=(num_timesteps, embedding_dim))
peephole_lstm_cell = keras.experimental.PeepholeLSTMCell(hidden_dim)
rnn_layer = keras.layers.RNN(peephole_lstm_cell)
outputs = rnn_layer(inputs)

model = keras.models.Model(inputs=[inputs], outputs=[outputs])
model.compile(loss="categorical_crossentropy", optimizer="adam")
```

- 샘플 2

```
x = keras.layers.Embedding(vocab_size, max_seqlen)(x)
x = keras.layers.Bidirectional(tf.keras.layers.LSTM(max_seqlen))(x)
x = keras.layers.Dense(64, activation="relu")(x)
x = keras.layers.Dense(1, activation="sigmoid")(x)
```

Model 구현 방식

Model Subclassing

- AutoGraph
  - Graph : tensorflow.Operation 객체의 집합을 포함하고 있는 데이터 구조
  - 연산의 단위와 텐서 객체, 연산간에 흐르는 데이터의 단위

```
@tf.function
def dense_layer(x, w, b):
    return tf.matmul(x, w) + b
```

dense_layer.pyton_function(x, w, b) : 원본 Python 함수 호출

Model 구현 방식

Model Subclassing

- Custom (사용자 정의)

```
class CustomModel(keras.Model):

class CustomLayer(keras.layers.Layer):

@tf.function
def CustomActivationFunction(x, axis=-1):

class CustomLossFunction(keras.losses.Loss):

class CustomOptimizer(keras.optimizers.Optimizer):

class CustomMetric(keras.metrics.Metric):

class CustomCallback(keras.callbacks.Callback):
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Model

```
class CustomAutoencoderClass(keras.Model):
    def __init__(self, hidden_dim, original_dim):
        super(Autoencoder, self).__init__()
        self.loss = []
        self.encoder = CustomEncoderClass(hidden_dim=hidden_dim)
        self.decoder = CustomDecoderClass(hidden_dim=hidden_dim, original_dim=original_dim)

    def call(self, input_features):                    #--- input  >  output
        encoded = self.encoder(input_features)
        reconstructed = self.decoder(encoded)
        return reconstructed

autoencoder = CustomAutoencoderClass(hidden_dim=hidden_dim, original_dim=original_dim)
for epoch in range(epochs):
    epoch_loss = 0
    with tf.GradientTape() as tape:
        preds = autoencoder(images)                                  #--- Model 실행
        loss_values = loss_function(preds, images)                   #--- Loss 계산
        gradients = tape.gradient(loss_values, model.trainable_variables)
        opt.apply_gradients(zip(gradients, model.trainable_variables))   #--- Optimizer에 적용
        epoch_loss = epoch_loss + loss_values
    autoencoder.loss.append(epoch_loss)                              #--- Loss 저장
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Layer

```
class CustomEncoderClass(keras.layers.Layer):
    def __init__(self, hidden_dim):
        super(Encoder, self).__init__()
        self.hidden_layer = keras.layers.Dense(units=hidden_dim, activation=tf.nn.relu)

    def call(self, input_features):                    #--- input data  >  output data
        activation = self.hidden_layer(input_features)
        return activation
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : ActivationFunction

```
def customActivationFunction(x):
    return x * keras.backend.tanh(keras.backend.softplus(x))

x = keras.layers.Activation(customActivationFunction)(x)

#--- Custom 2
class Mish(Activation):
    def __init__(self, activation, **kwargs):
        super(Mish, self).__init__(activation, **kwargs)
        self.__name__ = 'Mish'

def mish(x):
    return x * K.tanh(K.softplus(x))
get_custom_objects().update({'mish': Mish(mish)})
x = Activation('mish')(x)

#--- Custom 3
@tf.function
def CustomActivationFunction(x, axis=-1):
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Loss

```
class CustomLossClass(keras.losses.Loss):
    def call(self, y_true, y_pred):
        return tf.reduce_mean(tf.math.square(y_pred - y_true), axis=-1)

#---
def loss_function(preds, real):
    return tf.reduce_mean(tf.square(tf.subtract(preds, real)))
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Optimizer

```
class CustomOptimizerClass(keras.optimizers.Optimizer):

#000
class GradientDescent(tensorflow.Module):
  def __init__(self, learning_rate=1e-3):
    # Initialize parameters
    self.learning_rate = learning_rate
    self.title = f"Gradient descent optimizer: learning rate={self.learning_rate}"

  def apply_gradients(self, grads, vars):
    # Update variables
    for grad, var in zip(grads, vars):
      var.assign_sub(self.learning_rate*grad)
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Metric

```
class CustomMetricClass(keras.metrics.Metric):
    def __init__(self, name="categorical_true_positives", **kwargs):
        super(CategoricalTruePositives, self).__init__(name=name, **kwargs)
        self.true_positives = self.add_weight(name="ctp", initializer="zeros")

    def update_state(self, y_true, y_pred, sample_weight=None):
        y_pred = tf.reshape(tf.argmax(y_pred, axis=1), shape=(-1, 1))
        values = tf.cast(y_true, "int32") == tf.cast(y_pred, "int32")
        values = tf.cast(values, "float32")
        if sample_weight is not None:
            sample_weight = tf.cast(sample_weight, "float32")
            values = tf.multiply(values, sample_weight)
        self.true_positives.assign_add(tf.reduce_sum(values))

    def result(self):
        return self.true_positives

    def reset_state(self):
        # The state of the metric will be reset at the start of each epoch.
        self.true_positives.assign(0.0)
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Callback

```
class CustomCallbackClass(keras.callbacks.Callback):
    def on_train_begin(self, logs=None):
        print(f"Starting training; got log keys: {list(logs.keys())}")

    def on_train_end(self, logs=None):
        print(f"Stop training; got log keys: {list(logs.keys())}")

    def on_epoch_begin(self, epoch, logs=None):
        print(f"Start epoch {epoch} of training; got log keys: {list(logs.keys())}")

    def on_epoch_end(self, epoch, logs=None):
        print(f"End epoch {epoch} of training; got log keys: {list(logs.keys())}")

    def on_test_begin(self, logs=None):
        print(f"Start testing; got log keys: {list(logs.keys())}")

    def on_test_end(self, logs=None):
        print(f"Stop testing; got log keys: {list(logs.keys())}")

    def on_predict_begin(self, logs=None):
        print(f"Start predicting; got log keys: {list(logs.keys())}")
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의) : Callback

```
def on_predict_end(self, logs=None):
    print(f"Stop predicting; got log keys: {list(logs.keys())}")

def on_train_batch_begin(self, batch, logs=None):
    print(f"...Training: start of batch {batch}; got log keys: {list(logs.keys())}")

def on_train_batch_end(self, batch, logs=None):
    print(f"...Training: end of batch {}; got log keys: {list(logs.keys())}")

def on_test_batch_begin(self, batch, logs=None):
    print(f"...Evaluating: start of batch {batch}; got log keys: {list(logs.keys())}")

def on_test_batch_end(self, batch, logs=None):
    print(f"...Evaluating: end of batch {batch}; got log keys: {list(logs.keys())}")

def on_predict_batch_begin(self, batch, logs=None):
    print(f"...Predicting: start of batch {batch}; got log keys: {list(logs.keys())}")

def on_predict_batch_end(self, batch, logs=None):
    print("...Predicting: end of batch {batch}; got log keys: {list(logs.keys())}")
```

Model 구현 방식

Model Subclassing

- Custom (사용자 정의)

```
model = tf.keras.Sequential([])

model.save('my_model')
keras.models.load_model("my_model", custom_objects={ "CustomModel": CustomModel })



with open('my_model.json', 'w') as json_file:
    json_file.write(model.to_json())

my_model = keras.models.model_from_json(
                                open('my_model.json').read(),
                                custom_objects={ "CustomModel": CustomModel }
)
```

Model 구현 방식

Estimator (추정기) : Deprecated

- Estimator : 사전 제작된 모델
  - v1.Session 스타일 코드를 실행
    - Migration 가이드 : https://www.tensorflow.org/guide/migrate?hl=ko&_gl=1*1jfz0h3*_ga*ODY4NzU1NjA0LjE2ODg4MDU3MTk.*_ga_W0YLR4190T*MTY5MTQ4NzE5Ni4zMy4xLjE2OTE0ODcxOTYuMC4wLjA.

```
estimator = keras.estimator.model_to_estimator(model, config=config)
result = estimator.train(input_fun=lambda:input_fn())
estimator.evaluate(lambda:input_fun())
```

Application

Application

- Application
  - Model + Weight
  - 최적화
  - AutoML
  - TFLite for TensorFlow Lite
  - tensorflow.js

```
json = model.to_json()
model keras.models.model_from_json(json)

model.save_weights('~.h5)
model.load_weights(filename)
```

Application

Application

- json model

```
{
    class_name: 'Sequential',
    config: {
        name: 'sequential',
        layers: [
            { class_name: 'InputLayer', config: { 생략 } },
            { class_name: 'Conv2D', config: { 생략 } },
            { class_name: 'MaxPooling2D', config: { 생략 } },
            ……
            { class_name: 'Dropout', config: { 생략 } },
            { class_name: 'Dense', config: { 생략 } }
        ]
    },
    keras_version: '2.10.0',
    backend: 'tensorflow'
}
```

- 인공지능 IDE : https://www.knime.com/

Application

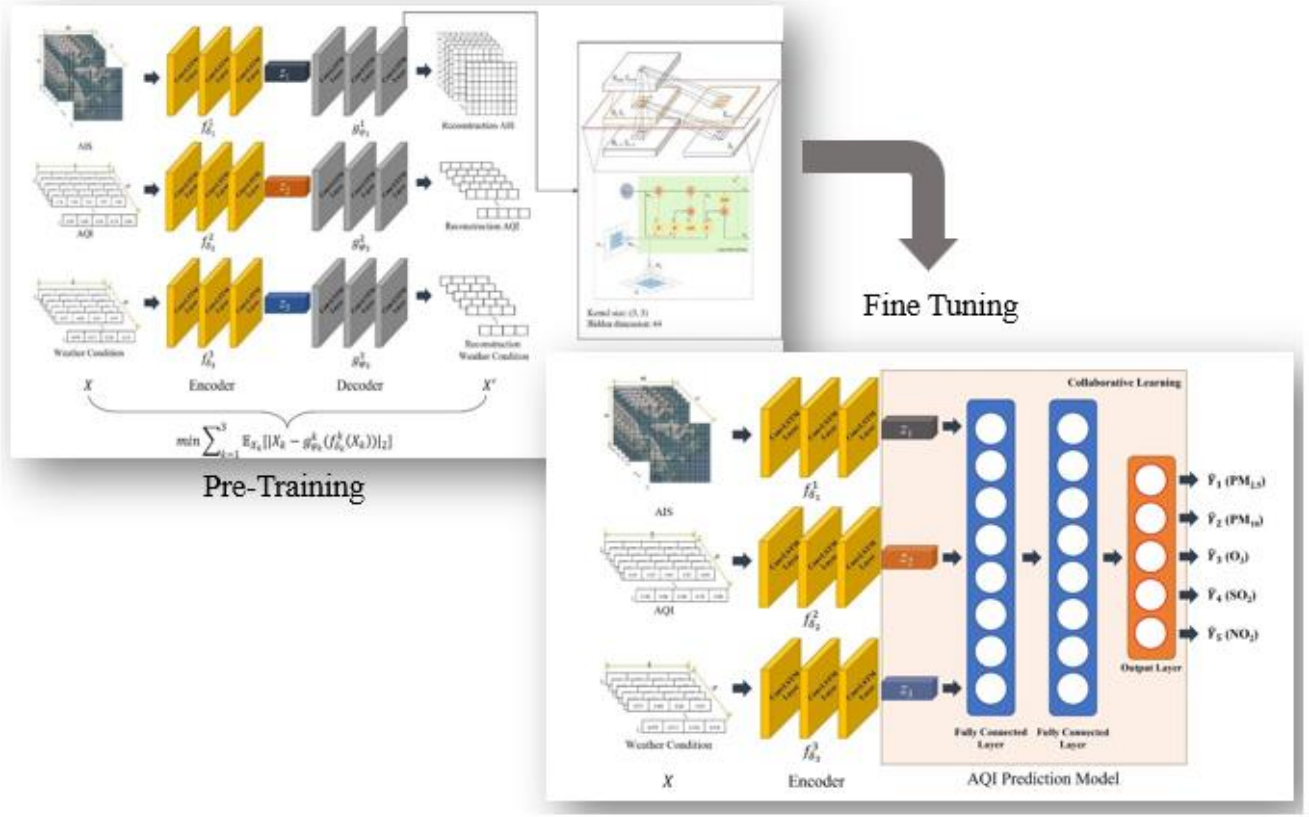
- weight
  dense/kernel:0 : Array<float32>
  dense/bias:0 : Array<float32>
  dense_1/kernel:0 : Array<float32>
  dense_1/bias:0 : Array<float32>
  dense_2/kernel:0 : Array<float32>
  dense_2/bias:0 : Array<float32>

$$H(x) = wx + b$$

Hypothesis Weight Bias

출처: https://brunch.co.kr/@linecard/321

- Transfer Learning (전이 학습)
  - Pre-trained Model ← Application (Model + Weight)
  - Fine Tuning (미세 조정)
  - 사례: ResNet, YOLO, BERT



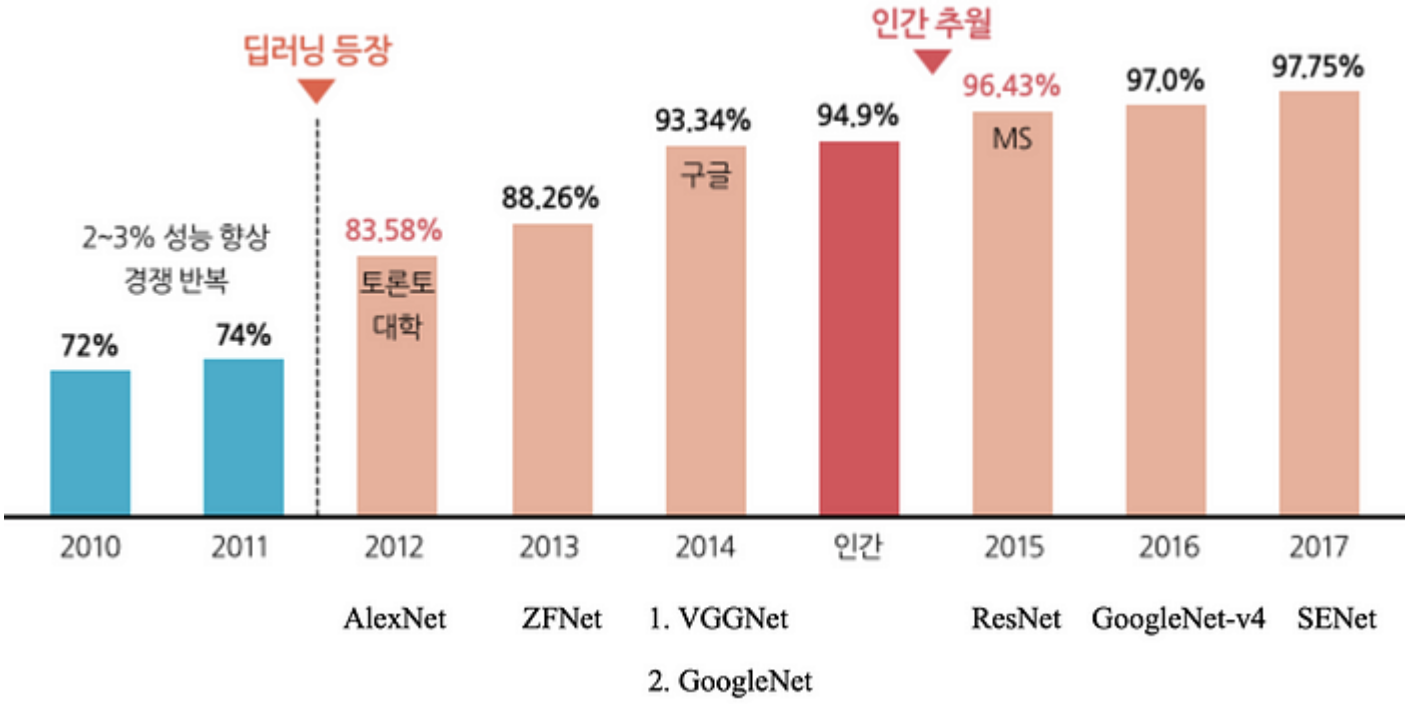
- AutoML

출처: https://onestoria.tistory.com/56

작성중

Application

- Application
  - resnet
  - resnet50
  - resnet_rs
  - resnet_v2
  - ResNet101
  - ResNet152
  - ResNet50
  - ResNetRS101
  - ResNetRS152
  - ResNetRS200
  - ResNetRS270
  - ResNetRS350
  - ResNetRS420
  - ResNetRS50
  - ResNet101V2
  - ResNet152V2
  - ResNet50V2



출처: https://medium.com/ddiddu-log/%EC%9D%B4%EB%AF%B8%EC%A7%80-%EC%9D%B8%EC%8B%9D%EC%9D%98-%EC%A0%95%EC%9D%98%EC%99%80-%EC%A3%BC%EC%9A%94-%EB%AA%A8%EB%8D%B8-%EB%B9%84%EA%B5%90-0-%EC%9D%B4%EB%AF%B8%EC%A7%80-%EC%9D%B8%EC%8B%9D-visual-recognition-a939d10c0ff7

작성중

Application

- Application
  - regnet
  - RegNetX002
  - RegNetX004
  - RegNetX006
  - RegNetX008
  - RegNetX016
  - RegNetX032
  - RegNetX040
  - RegNetX064
  - RegNetX080
  - RegNetX120
  - RegNetX160
  - RegNetX320

작성중

Application

- Application
  - RegNetY002
  - RegNetY004
  - RegNetY006
  - RegNetY008
  - RegNetY016
  - RegNetY032
  - RegNetY040
  - RegNetY064
  - RegNetY080
  - RegNetY120
  - RegNetY160
  - RegNetY320

작성중

Application

- Application
  - efficientnet
  - efficientnet_v2
  - EfficientNetB0
  - EfficientNetB1
  - EfficientNetB2
  - EfficientNetB3
  - EfficientNetB4
  - EfficientNetB5
  - EfficientNetB6
  - EfficientNetB7

  - EfficientNetV2B0
  - EfficientNetV2B1
  - EfficientNetV2B2
  - EfficientNetV2B3
  - EfficientNetV2L
  - EfficientNetV2M
  - EfficientNetV2S

작성중

Application

- Application
  - MobileNet
  - MobileNetV2
  - MobileNetV3Large
  - MobileNetV3Small
  - mobilenet
  - mobilenet_v2
  - mobilenet_v3

  - vgg16
  - vgg19
  - VGG16
  - VGG19

작성중

Application

- Application
  - convnext
  - ConvNeXtBase
  - ConvNeXtLarge
  - ConvNeXtSmall
  - ConvNeXtTiny
  - ConvNeXtXLarge

  - densenet
  - DenseNet121
  - DenseNet169
  - DenseNet201

  - nasnet
  - NASNetLarge
  - NASNetMobile

작성중

Application

- Application
  - inception_v3
  - InceptionV3
  - inception_resnet_v2
  - InceptionResNetV2

  - xception
  - Xception

  - imagenet_utils

Application

작성중

Model과 Application

- Application 전체를 재사용
  - Capsule : Application을 하나의 layout로 사용

작성중

Model과 Application

- Application 일부를 재활용
  - 유사한 문제를 해결하는데 기존에 존재하는 application을 재활용
    - model의 일부를 삭제 후 추가
    - weigh의 일부를 삭제 후 추가

  - 학습에 필요한 충분한 데이터가 없는 경우 사용 가능
  - 이미지를 100가지로 분류하는 application이 있는데 이를 재활용하여 200가지로 분류하는 application을 만들려고 하는 경우

작성중

AutoML

- Optimizer : 방식, 매개변수 변동폭
- epochs * batch_size
- hidden 개수
- model

- AutoML
  - HyperParameter
    - 좌측 항목 등
  - Loss Function

- 모델 선택 기준
  - Parameter 개수 최소화

Logistic Regression

Logistic Regression

- Activation Functions
  - Y = A * W + b : 수식. X = A
    - W (Weight, 가중치)
    - b (Bias, 오차)
- Loss Functions
  - Least squares : 오차 제곱합
- Optimizers
  - 미분을 사용하여 W와 b를 추정
- Metrics
- Back Propagation (오차 역전파)

Algorithms

Algorithms

작성중

Algorithms의 종류

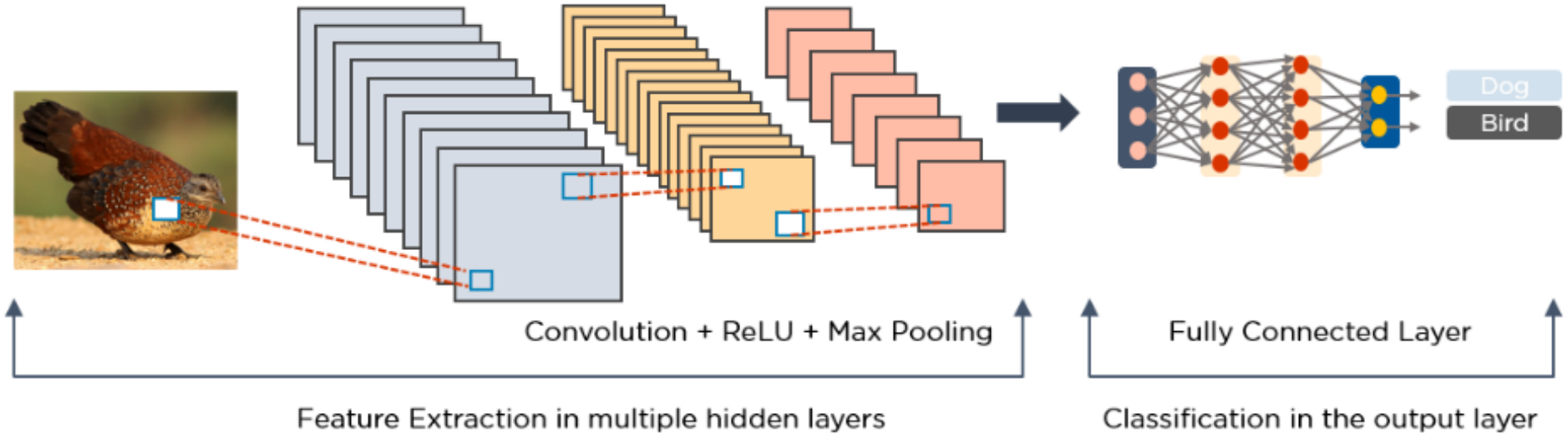
- CNN (Convolutional Neural Network)
- LSTM (Long Short Term Memory Network)
- RNN (Recurrent Neural Network)
- GAN (Generative Adversarial Network)
- RBFN (Radial Basis Function Network)
- MLP (Multilayer Perceptron)
- SOM (Self Organizing Map)
- DBN (Deep Belief Network)
- RBM (Restricted Boltzmann Machine)
- AutoEncoder

- https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm
- https://www.projectpro.io/article/deep-learning-algorithms/443

작성중

CNN (Convolutional Neural Network)

- 이미지 처리
- 객체 탐지



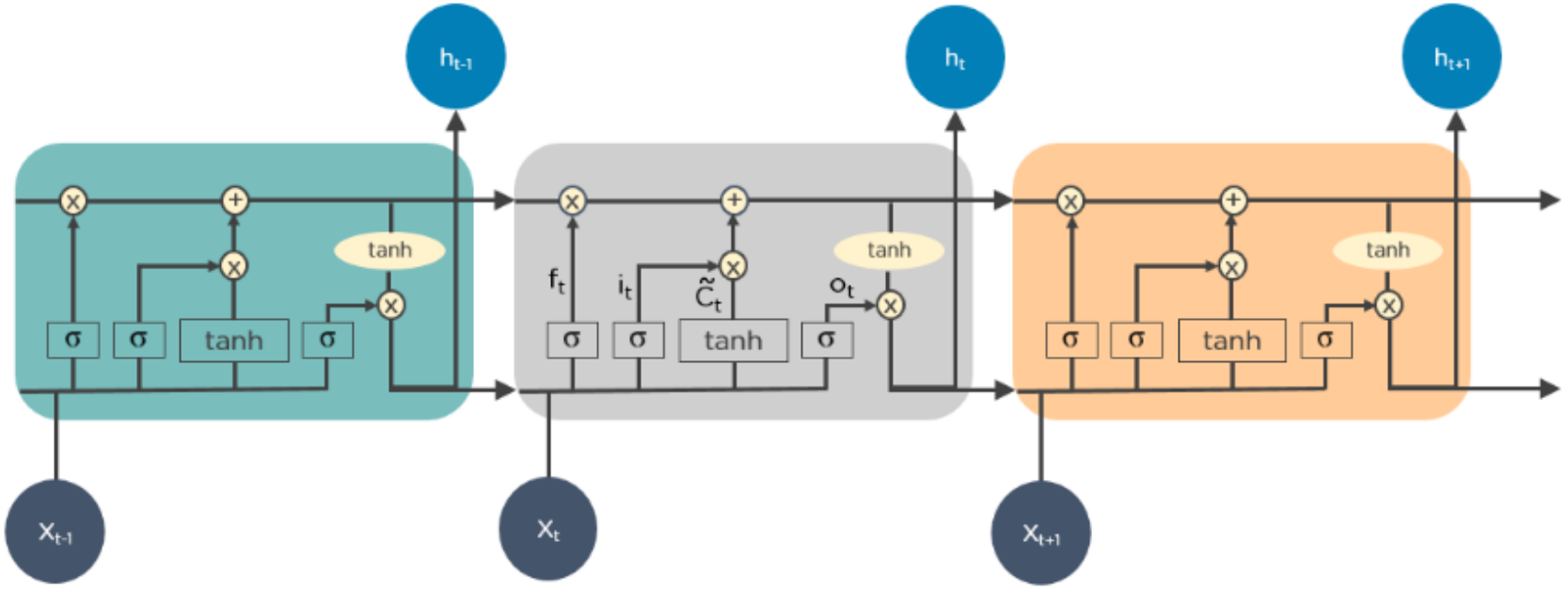
출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

Algorithms

작성중

LSTM (Long Short Term Memory Network)

- 시계열 예측
- 음악 작곡. 음성 인식

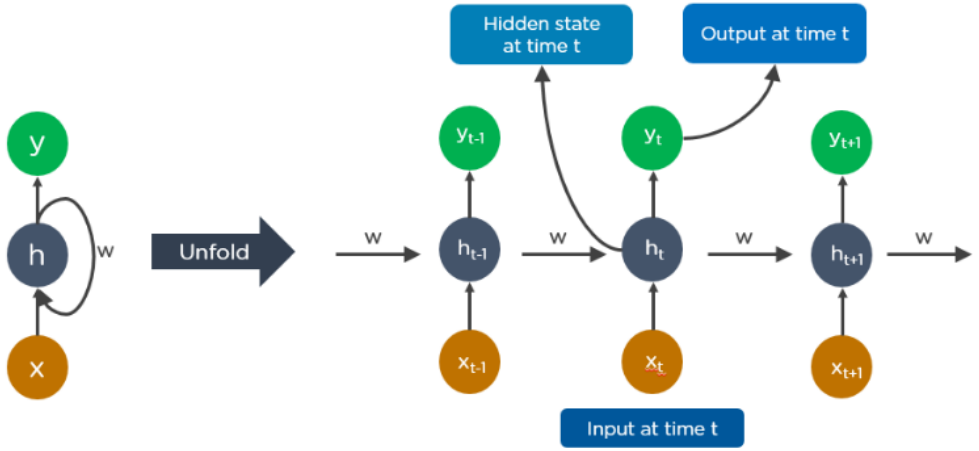


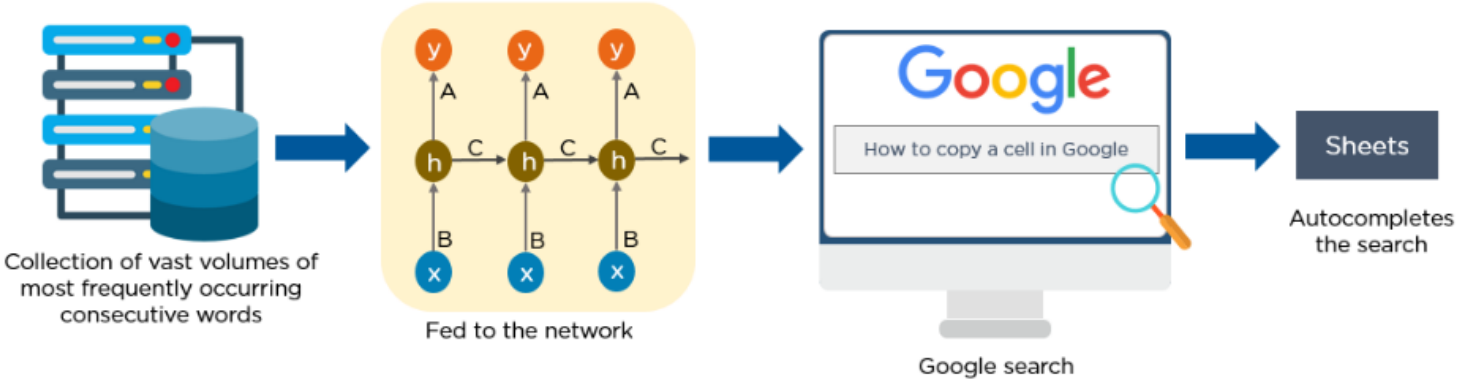
출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

작성중

RNN (Recurrent Neural Network)

- 시계열 분석
- 자연어 처리
- 필기 인식
- 기계 번역
- 이미지 캡션



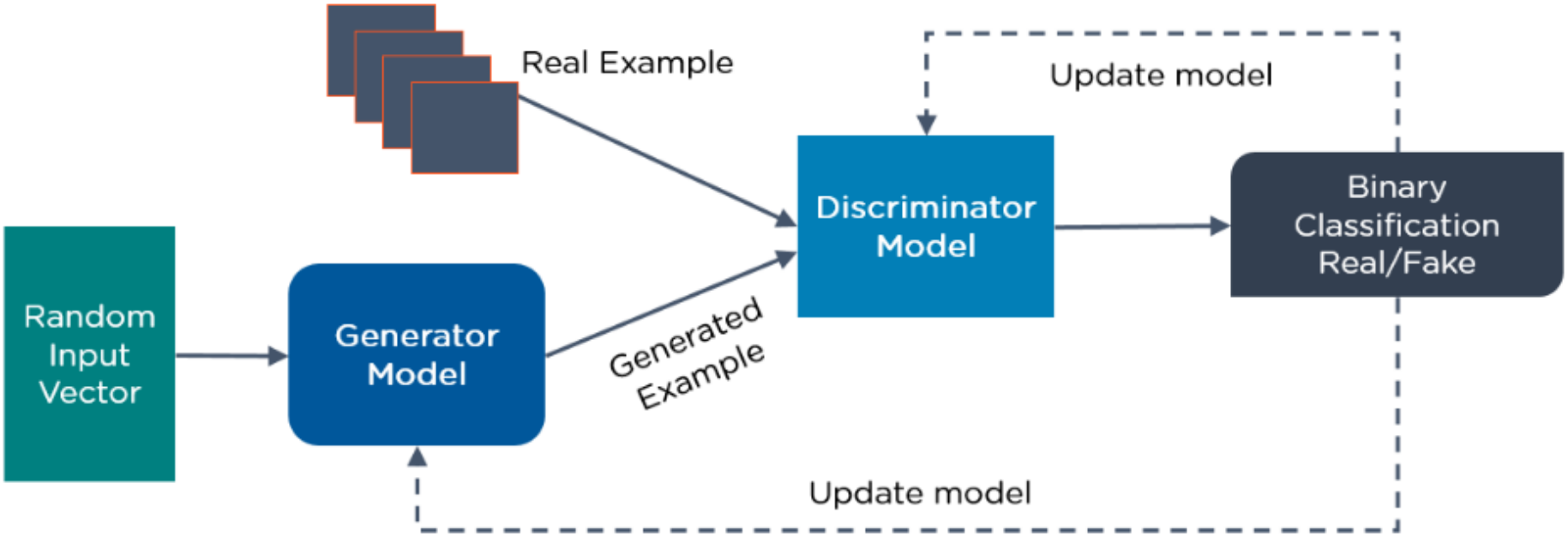


출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

작성중

GAN (Generative Adversarial Network)

- 이미지 생성

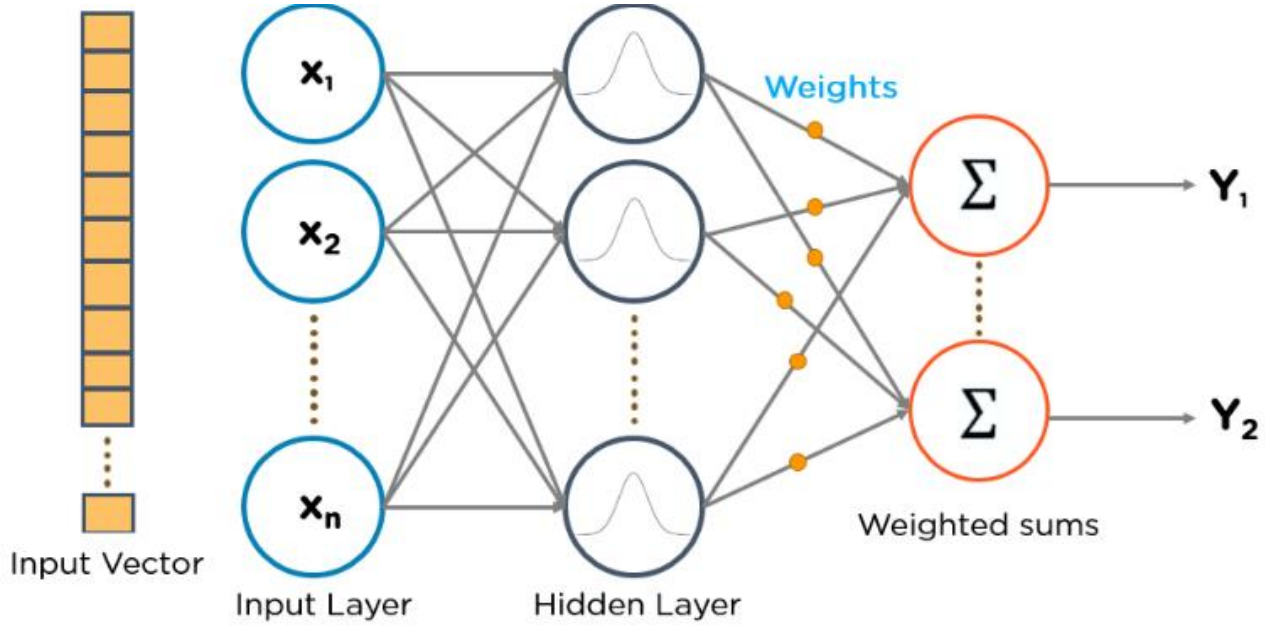


출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

작성중

RBFN (Radial Basis Function Network)

- 분류, 회귀, 시계열 예측



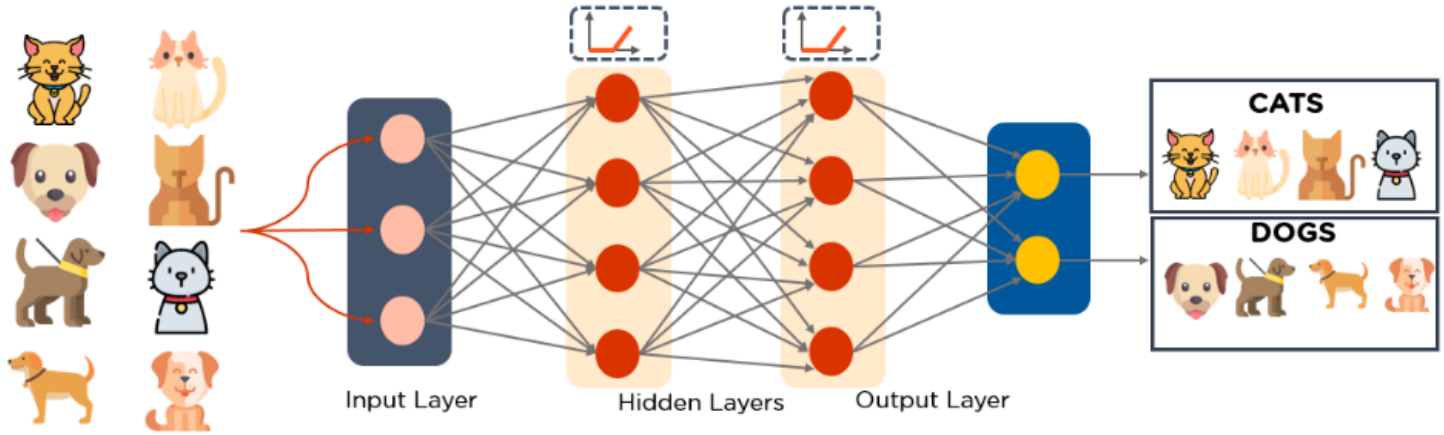
출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

Algorithms

작성중

MLP (Multilayer Perceptron)

- 음성 인식
- 이미지 인식
- 기계 번역
- 소프트웨어 구축

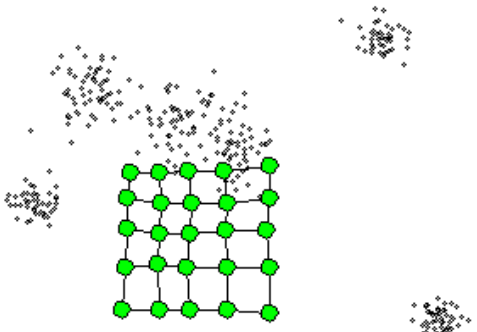


출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

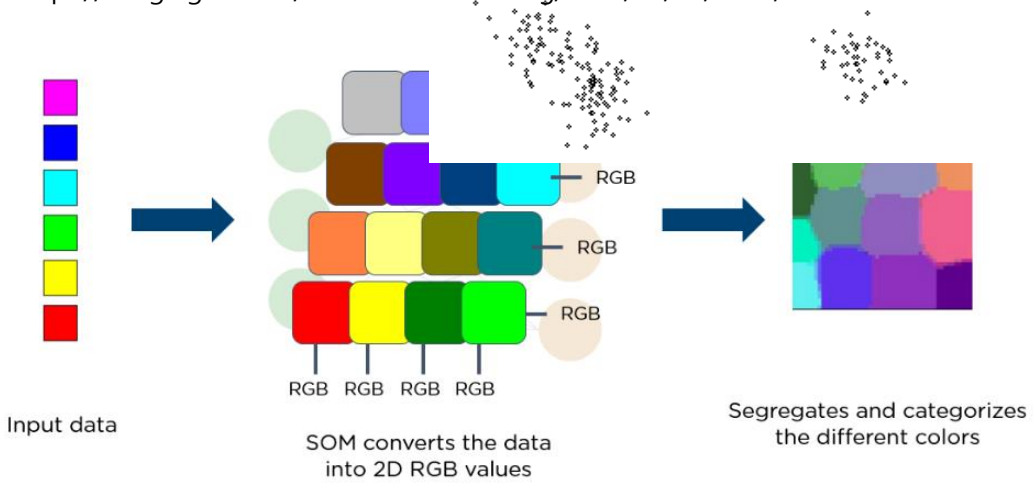
작성중

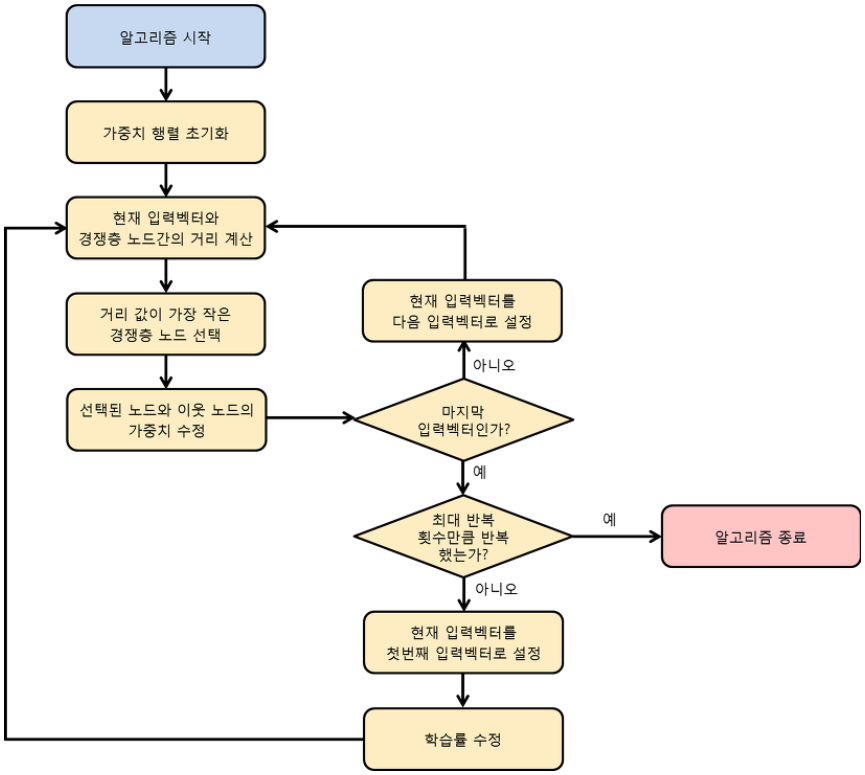
SOM (Self Organizing Map)

- 고차원의 데이터를 저차원의 격자 형태로 변환
  - 입력 데이터의 분포를 보존하면서 비슷한 패턴이나 특징을 가진 데이터를 인접한 노드에 매핑
- 구조 탐색 (분류)
- 차원 축소
- 시각화



출처: https://ratsgo.github.io/machine%20learning/2017/05/01/SOM/





출처: https://untitledtblog.tistory.com/5

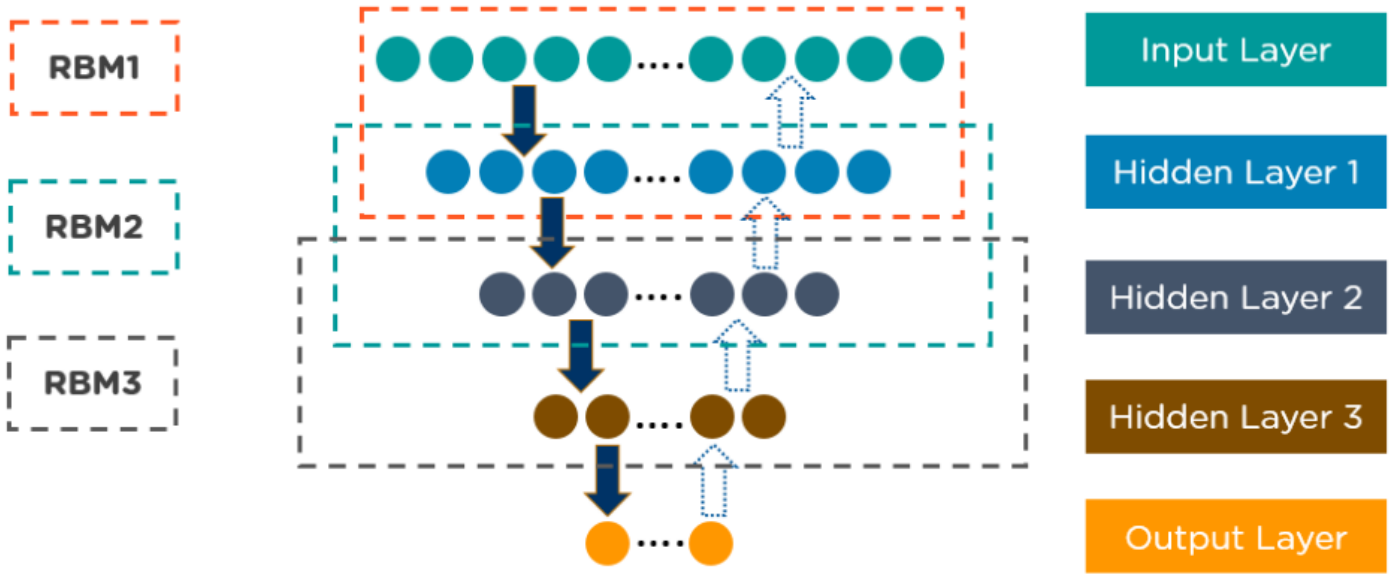
출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

Algorithms

작성중

DBN (Deep Belief Network) ← RBM

- 이미지 인식
- 비디오 인식
- 모션 캡쳐



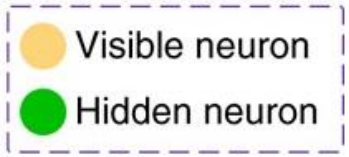
출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

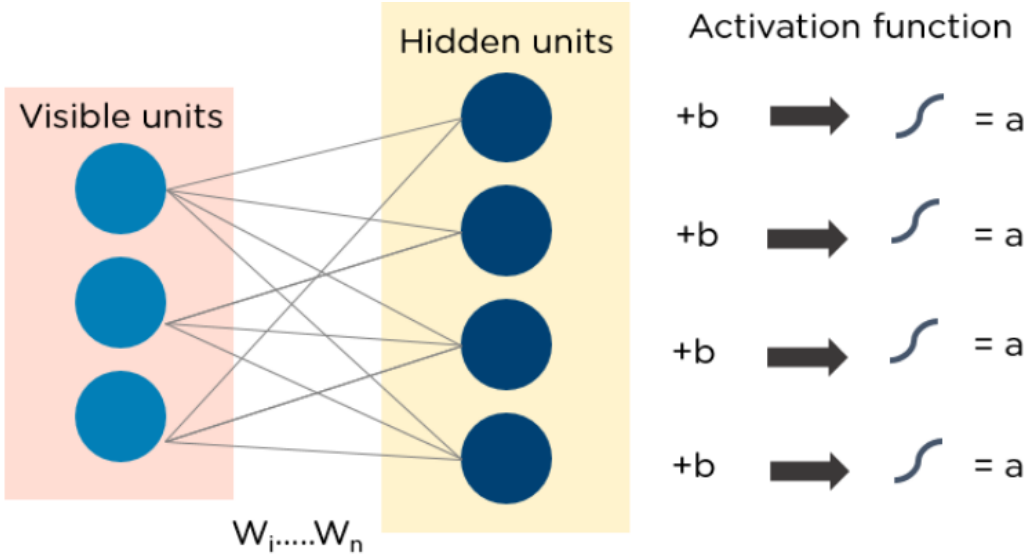
Algorithms

작성중

RBM (Restricted Boltzmann Machine) : 생성 AI

- Boltzmann Distribution를 기반으로한 확률 모형
  - Visible units : 특징 데이터
  - Hidden units : 확률 분포
- 학습 : 샘플 데이터와 유사한 데이터 생성
  - 분류, 협업, 특징값 학습
  - 선형 회귀 분석, 협업 필터링, 주제 모델링
  - 차원 감소



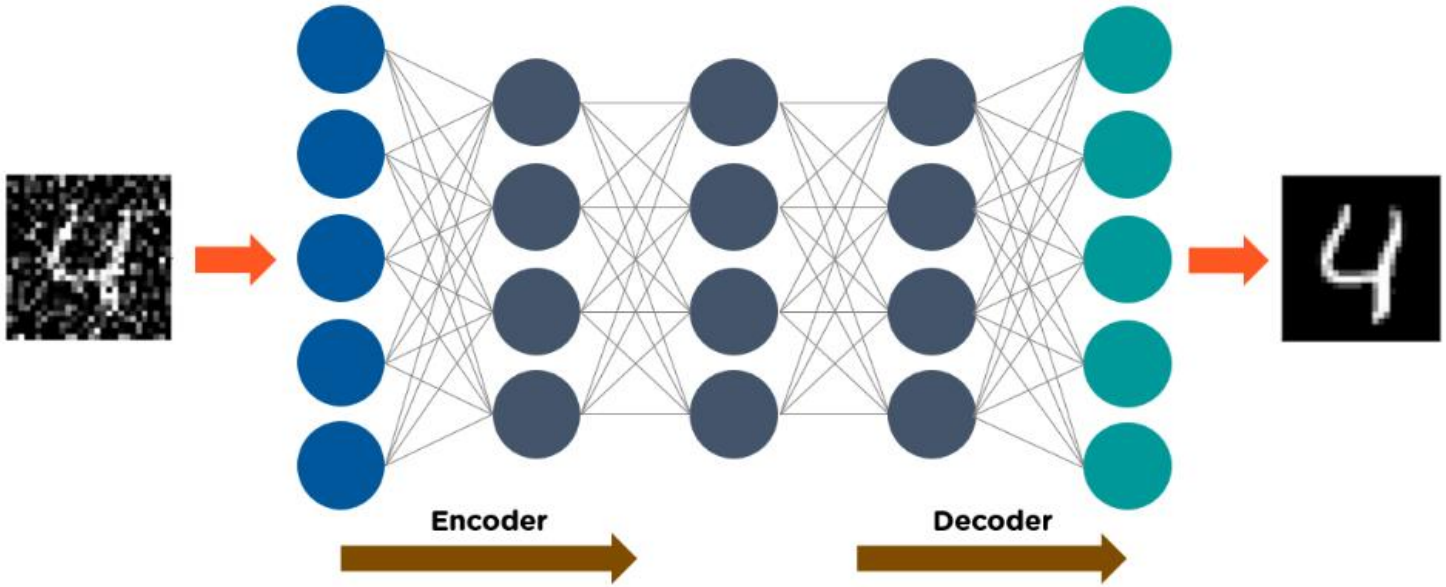


출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

작성중

AutoEncoder : 생성 AI

- 입력 데이터의 분포를 학습하고, 분포에서 랜덤하게 샘플링하여 새로운 콘텐츠 생성
  - 이미지 복원과 생성. 이미지의 스타일 변경
  - 음악 생성. 텍스트 생성
  - 추천 시스템에서 상품을 추천



출처: https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm

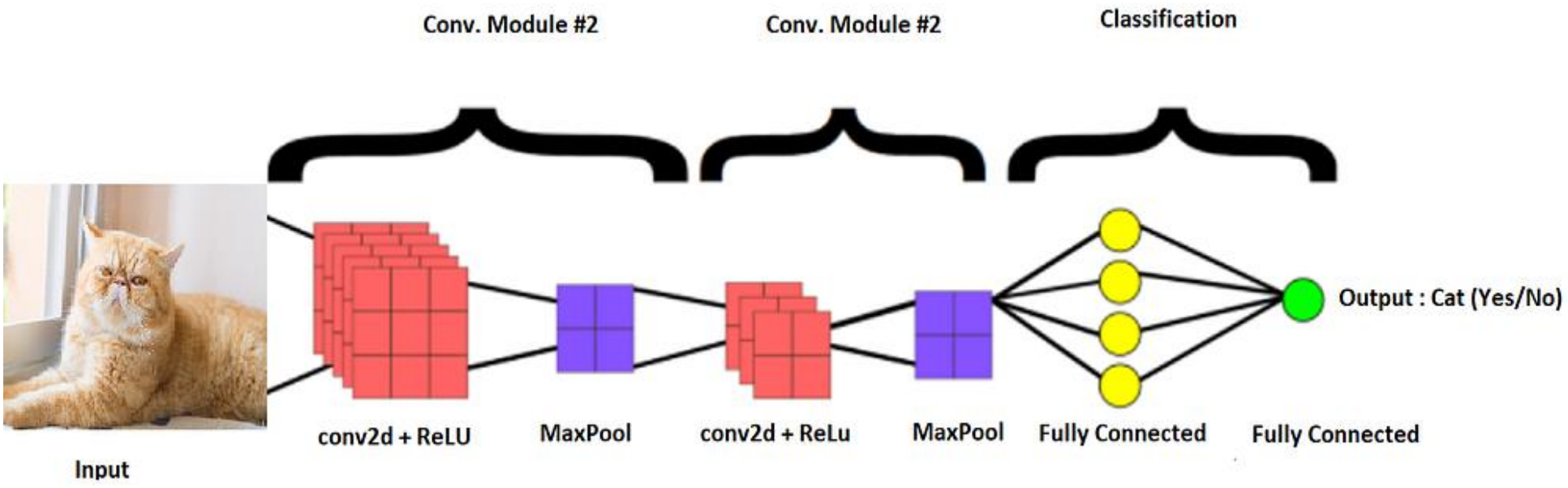
CNN (Convolutional Neural Networks)

CNN

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- CNN (Convolutional Neural Networks)
  - DCNN (Deep CNN, 심층 합성곱 신경망)
  - Convolution layer
  - ReLU (Rectified Linear Unit)
  - Pooling layer
  - FC (Fully connected) layer

- 데이터 변환 계층
- Receptive field (로컬 수용 필드)
- 가중치 공유

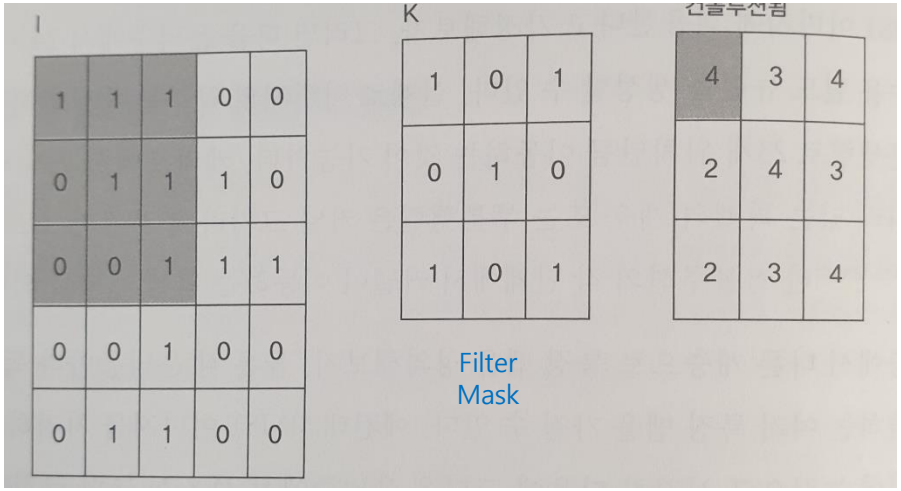


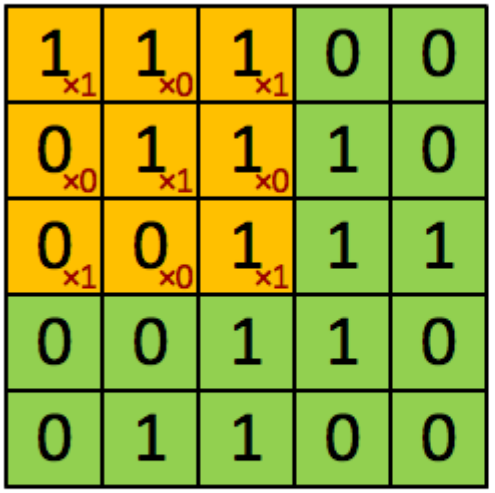
출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

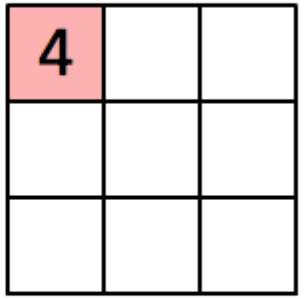
CNN

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Convolution : 관계 정보를 활용하는 방법
  - Dense : 선형
  - Convolution : n차원 부분 행열
    - 행열에 Convolution을 적용하여 특징맵을 생성
    - 2차원 : 이미지, 오디오, 텍스트
    - 3차원 : 비디오





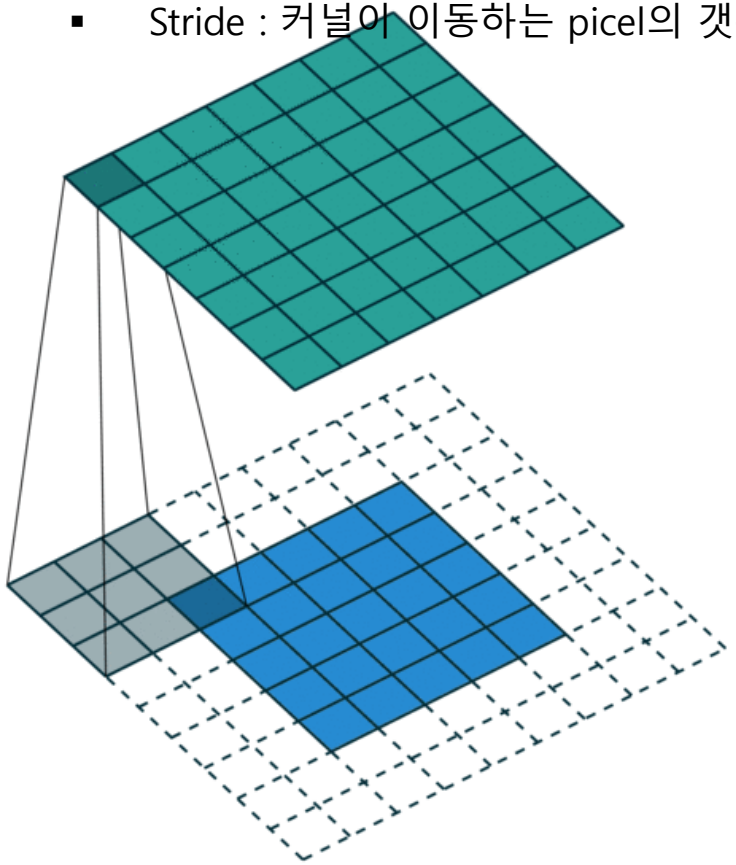


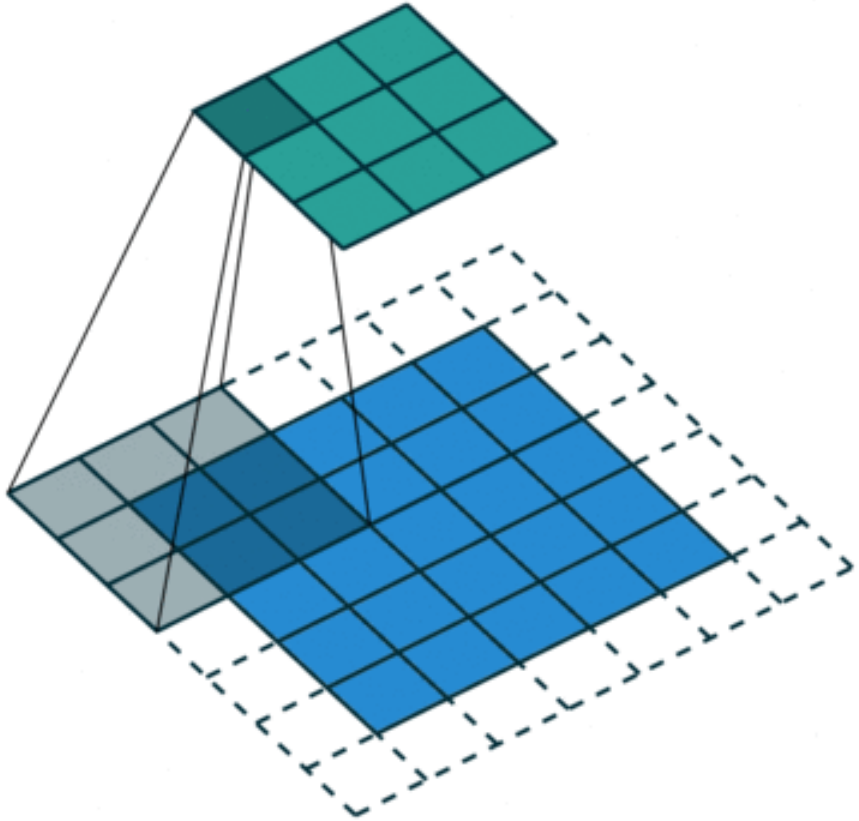
출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

CNN

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Convolution
  - padding
  - Stride : 커널이 이동하는 picel의 갯수





출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

CNN

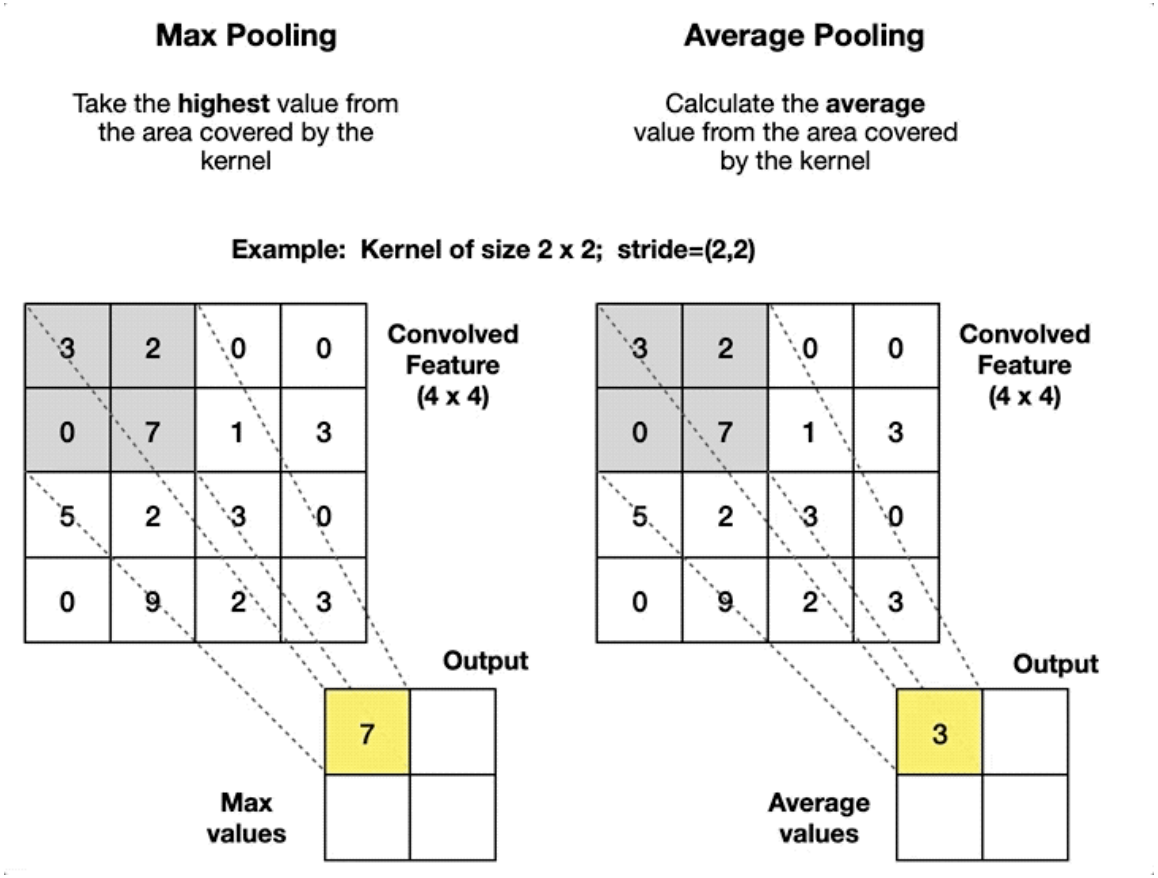
CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Convolution

출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Pooling
  - n차원 부분 행열을 사용하여 특징맵을 요약



출처: https://towardsai.net/p/l/introduction-to-pooling-layers-in-cnn

CNN

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Pooling

출처: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

CNN

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- CNN

```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(20, (5, 5), activation='relu', input_shape=input_shape))
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(keras.layers.Conv2D(50, (5, 5), activation='relu')
model.add(keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(500, activation="relu"))
model.add(keras.layers.Dense(self.nb_classes, activation="softmax"))

model.compile(
    optimizer=self.optimizer,
    loss=self.loss_function,
    metrics=[ self.metrics ]
)
```

CNN

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- CNN

```
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 24, 24, 20)        520
 max_pooling2d (MaxPooling2D)  (None, 12, 12, 20)       0

 conv2d_1 (Conv2D)            (None, 8, 8, 50)          25050
 max_pooling2d_1 (MaxPooling2D)  (None, 4, 4, 50)         0

 flatten (Flatten)            (None, 800)               0
 dense (Dense)                (None, 500)               400500
 dense_1 (Dense)              (None, 10)                5010

=================================================================
Total params: 431,080
Trainable params: 431,080
Non-trainable params: 0
```

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- CNN

```
Epoch 14/15
47/47 [==============================] - 1s 19ms/step - loss: 0.0045 - accuracy: 0.9993 - val_loss: 0.0908 - val_accuracy: 0.9755
Epoch 15/15
47/47 [==============================] - 1s 19ms/step - loss: 0.0038 - accuracy: 0.9995 - val_loss: 0.1002 - val_accuracy: 0.9751


313/313 [==============================] - 1s 2ms/step - loss: 0.0734 - accuracy: 0.9799
Test accuracy: 0.9799000024795532
```

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- 활용
  - 분류 ← 분류 헤드
  - 의미 세그먼테이션 ← 회귀 헤드
    - 로컬화
    - Object detection (객체 탐지)
    - 인스턴스 세그먼테이션
  - 전이학습
  - Style transfer

- **전치 컨볼루션**

- Capsule networks (캡슐망, CapsNets)

CNN

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- CNN 적용 분야
  - 영상
  - 동영상, 오디오, 텍스트
  - 객체 검출
  - 합성 데이터 생성 for GAN (생성적 적대 신경망)

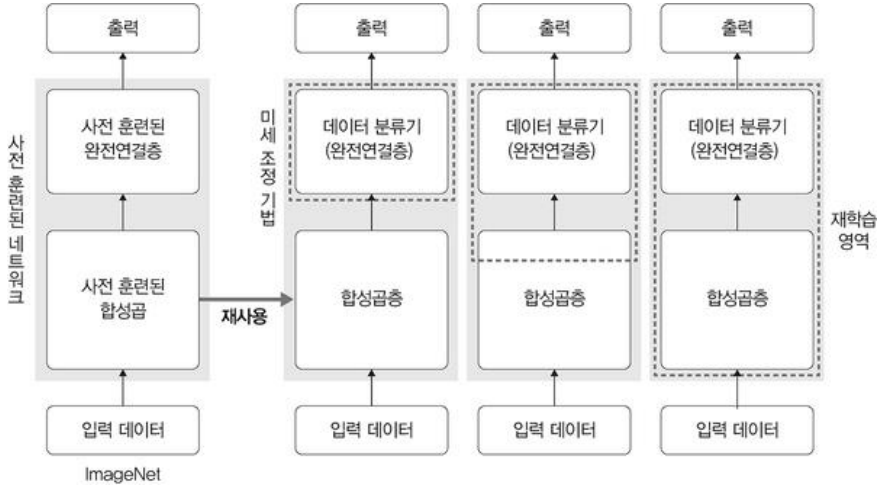
- 비즈니스 어플리케이션
  - 이미지 분류
  - 추천 시스템
  - 이미지 검색
  - 얼굴 인식
  - 광학 문자 인식

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Transfer Learning (전이 학습)
  - 사전 학습된 application을 사용하여 새로운 작업에 대한 representation (표현)을 만든다.
  - Pre-trained Model ← Application
  - Fine Tuning (미세 조정)

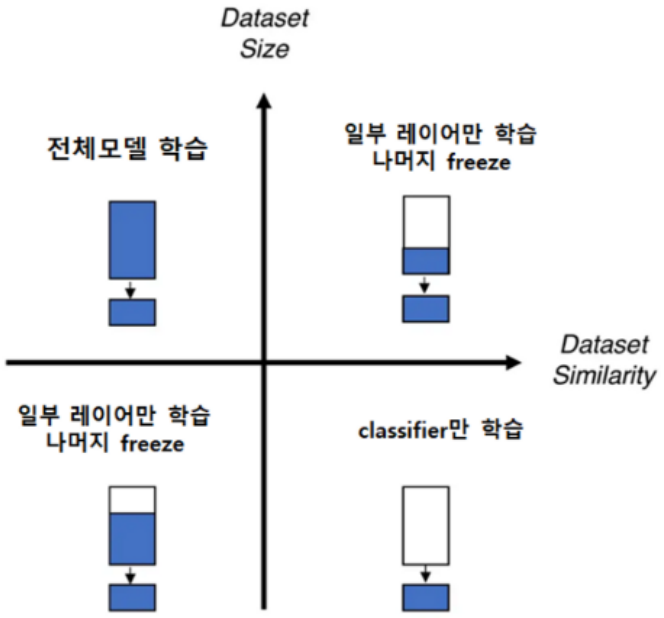
- 전이 학습 workflow
  - 기본 모델을 인스턴스화하고 사전 훈련된 가중치를 여기에 로드합니다.
  - trainable = False를 설정하여 기본 모델의 모든 레이어를 동결합니다.
  - 기본 모델에서 하나 이상의 레이어 출력 위에 새 모델을 만듭니다.
  - 새 데이터세트에서 새 모델을 훈련합니다.



출처: https://thebook.io/080289/0277/

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Fine Tuning (미세 조정)
  - Pre-trained 모델을 새로운 Task에 적용 시키는 과정
  - Layer 속성
    - trainable : True. 훈련. 이 속성이 변경되면 다시 compile을 하여야 합니다.
    - weights : 가중치 변수 목록 (get_weights())
    - trainable_weights : 훈련 대상이 되는 가중치 변수 목록
    - non_trainable_weights : 훈련되지 않은 가중치 변수 목록



출처: https://yeko90.tistory.com/entry/deep-learning-Fine-Tuning-tip

CNN

작성중

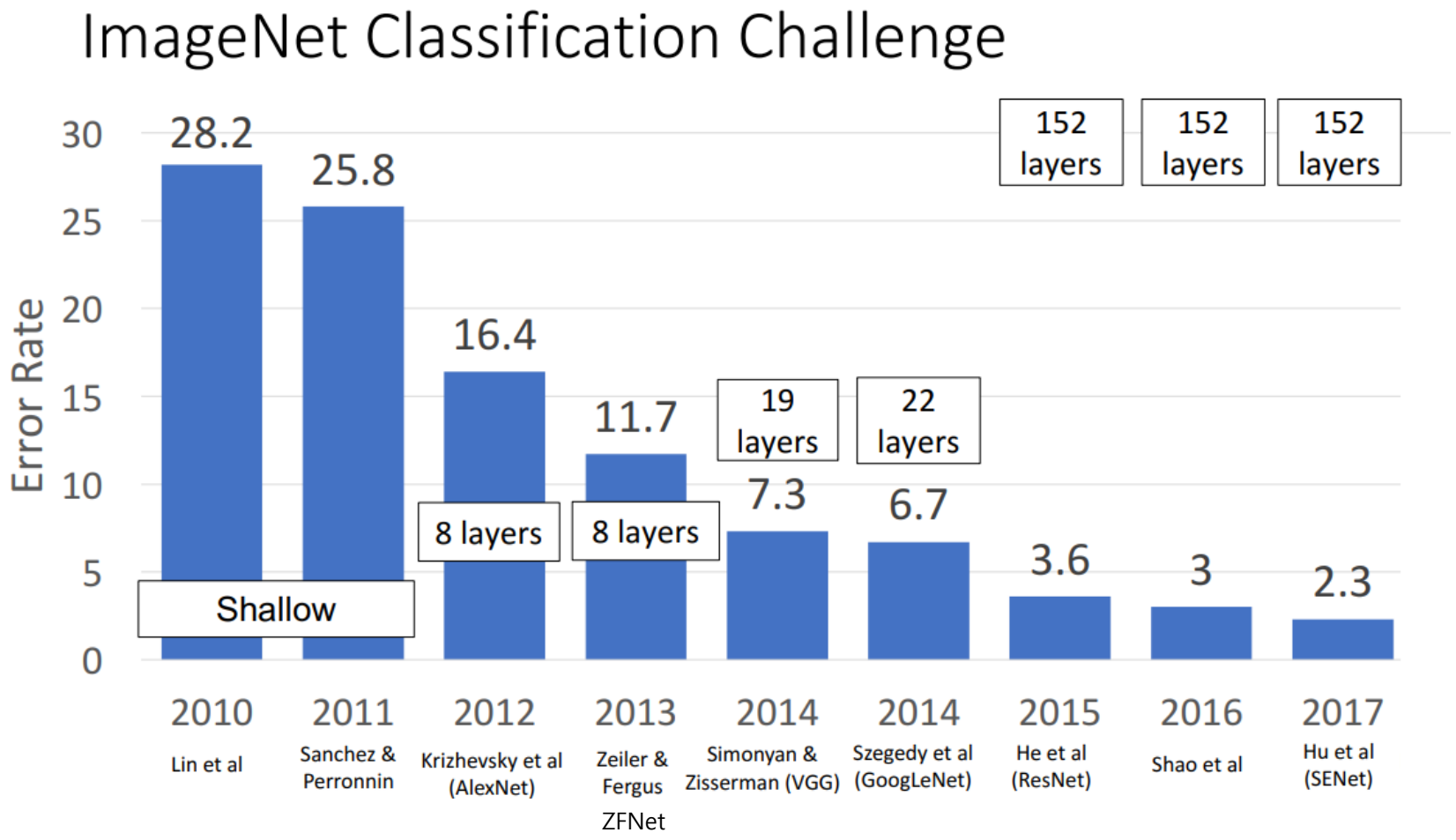
CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Keras application : tensorflow.keras.applications
- TensorFlow Hub : https://www.tensorflow.org/hub

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

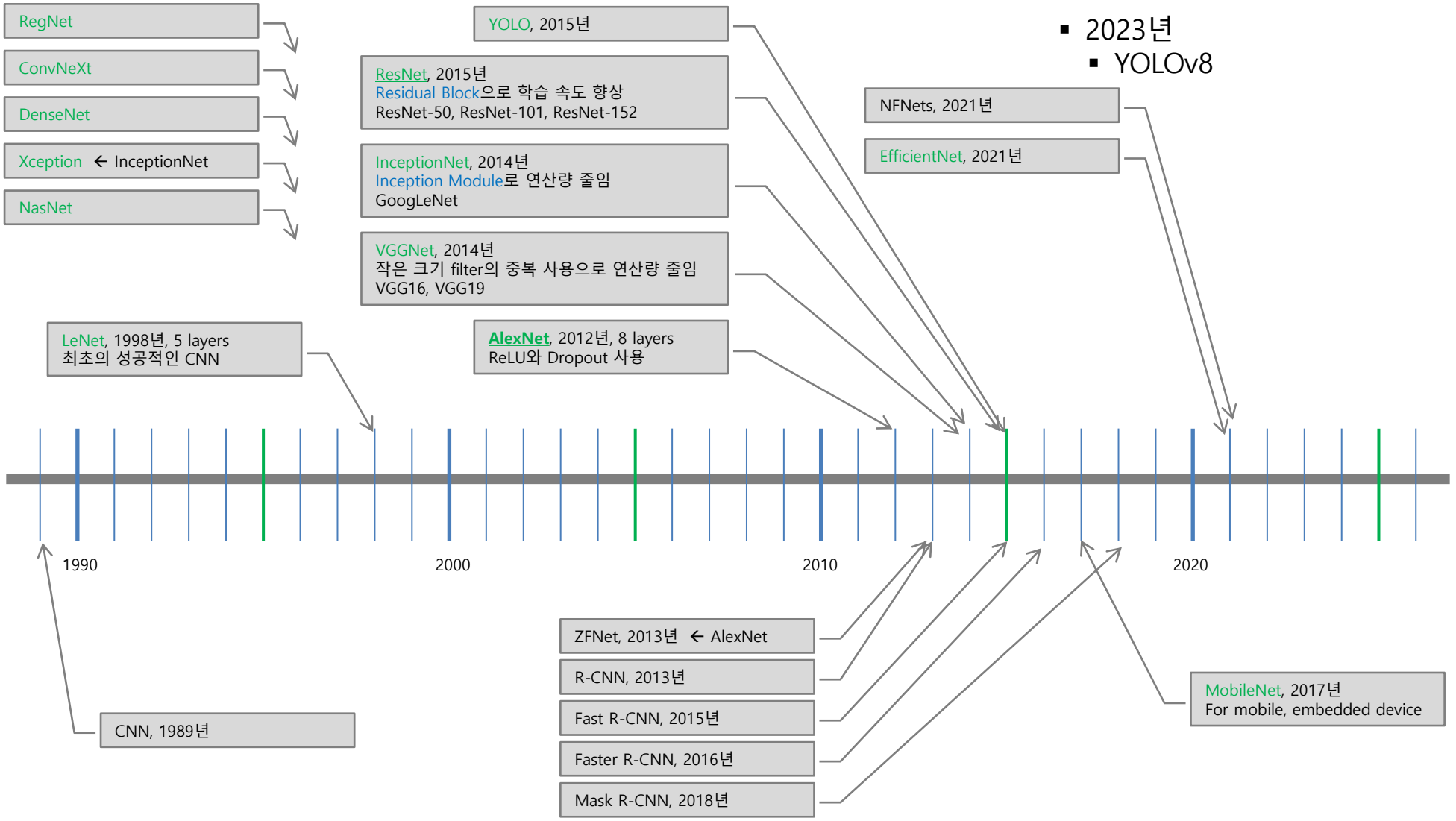
- CNN Application



출처: https://velog.io/@kbm970709/%EC%8B%A0%EC%9E%85%EC%83%9D-%EC%84%B8%EB%AF%B8%EB%82%98-Lecture-8-CNN-Architectures-DL-for-CV

작성중

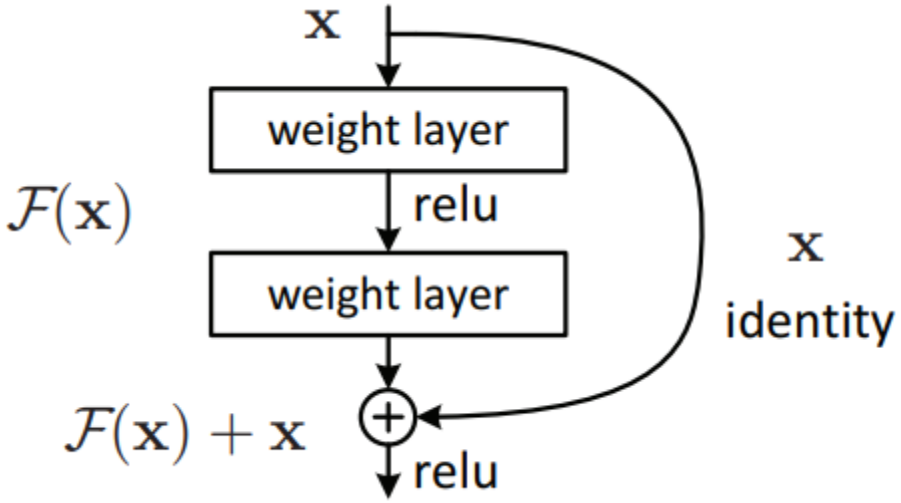
CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)



작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- ResNet (레지듀얼 망)
  - Residual block
    - Identity를 이후 계층에서 사용함으로 학습 속도를 향상 시킨다.

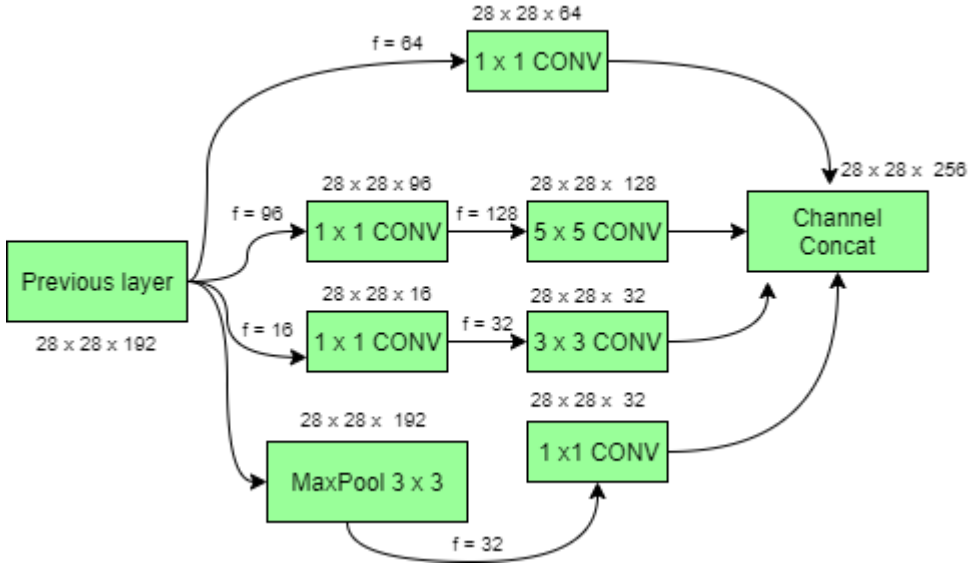


출처: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- InceptionNet
  - Inception Module
    - 1*1 Conv를 사용하여 연산량을 획기적으로 감소

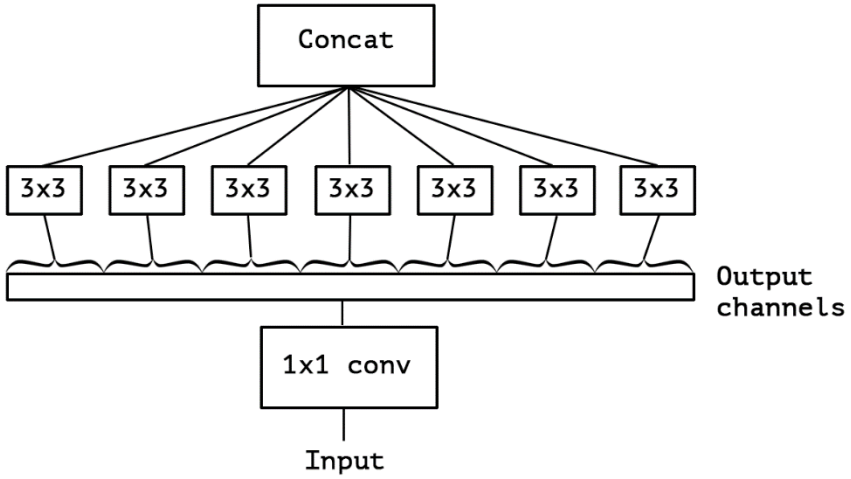


출처: https://www.geeksforgeeks.org/ml-inception-network-v1/

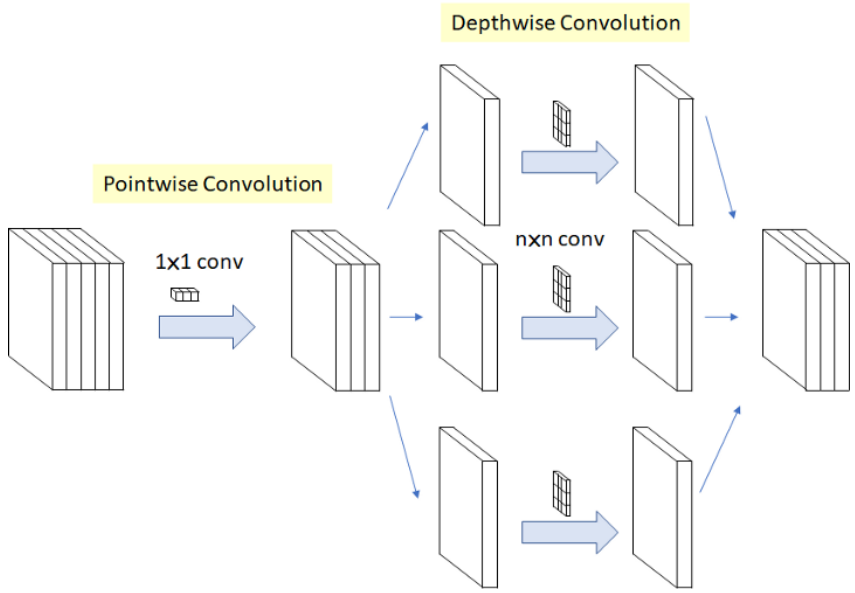
작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- Xception



출처: https://sotudy.tistory.com/14

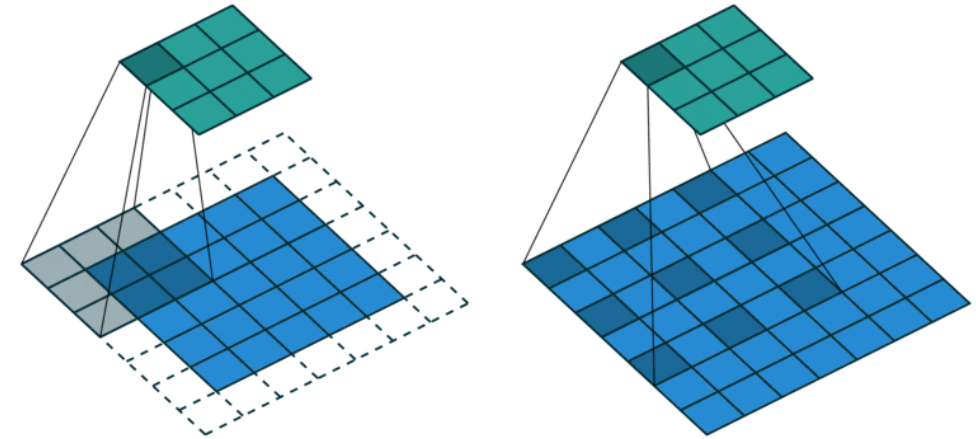


출처: https://www.linkedin.com/pulse/xception-depthwise-separable-convolution-ayoub-kirouane

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- 컨볼루션 연산 요약
  - 확장 컨볼루션 : Atrous Convolution, Dilated Convolution, Hole Convolution



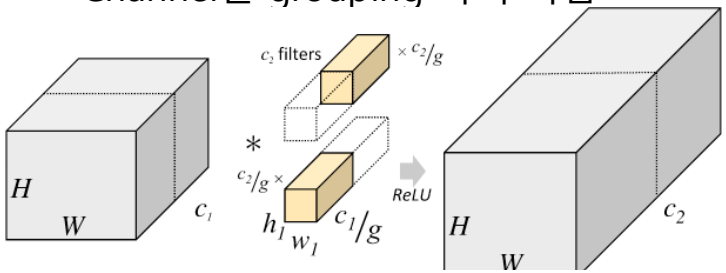
출처: https://gaussian37.github.io/dl-concept-dilated_residual_network/

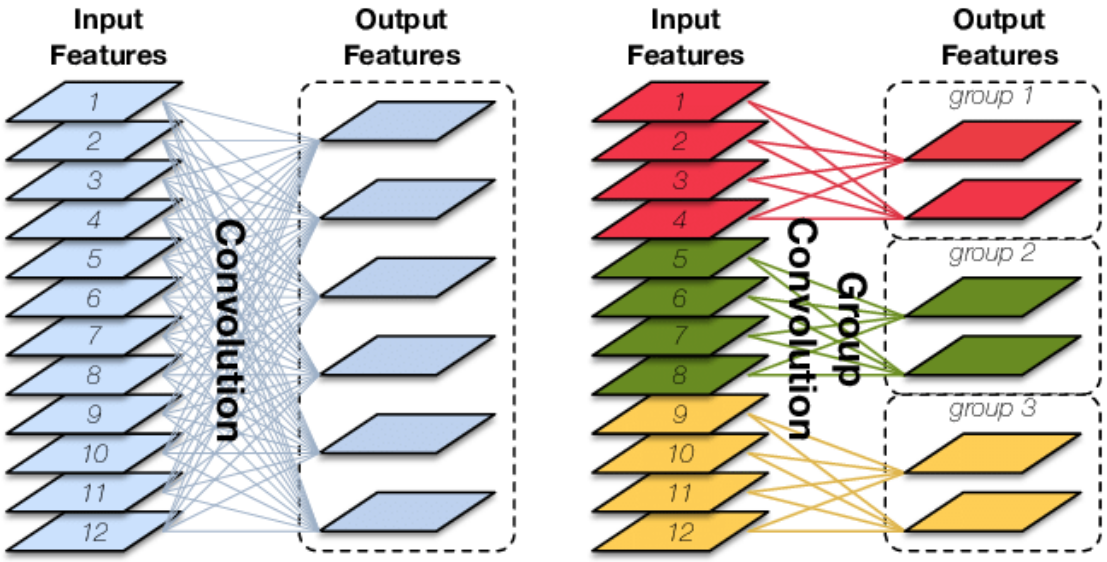
  - 전치 컨볼루션
    - Conv2D ←→ Conv@dTranspose

작성중

CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- 컨볼루션 연산 요약
  - Grouped Convoltion
    - Channel을 grouping 하여 학습





출처: https://velog.io/@juyeon048/%EB%94%A5-%EB%9F%AC%EB%8B%9D-Deep-Neural-Network-%EA%B2%BD%EB%9F%89%ED%99%94

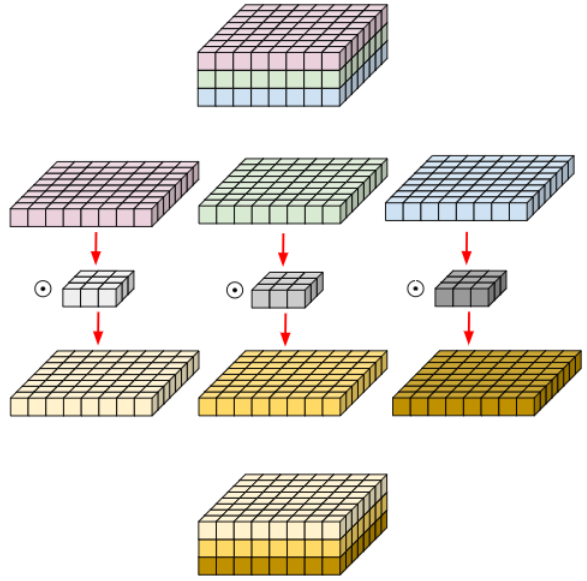
CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- 컨볼루션 연산 요약
  - Separable Convoltion (분리 가능 컨볼루션)
    - 1개의 filter를 여러 개의 filter로 분리하여 적용하여도 동일한 결과를 가진다.

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} 2*3 & 2*4 \\ 3*3 & 3*4 \end{bmatrix}$$

출처: https://velog.io/@sunbei00/Separable-Convolution

  - Depthwise Convolution (깊이별 컨볼루션)
    - Channel마다 따로 필터를 학습

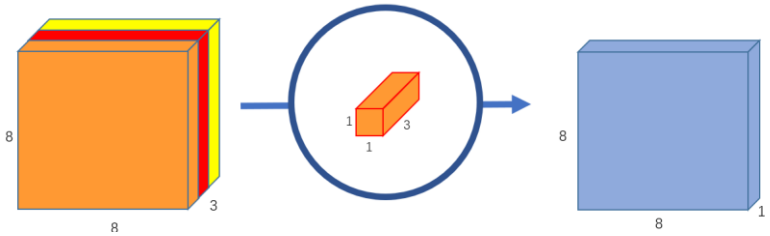


출처: https://eehoeskrap.tistory.com/431

작성중

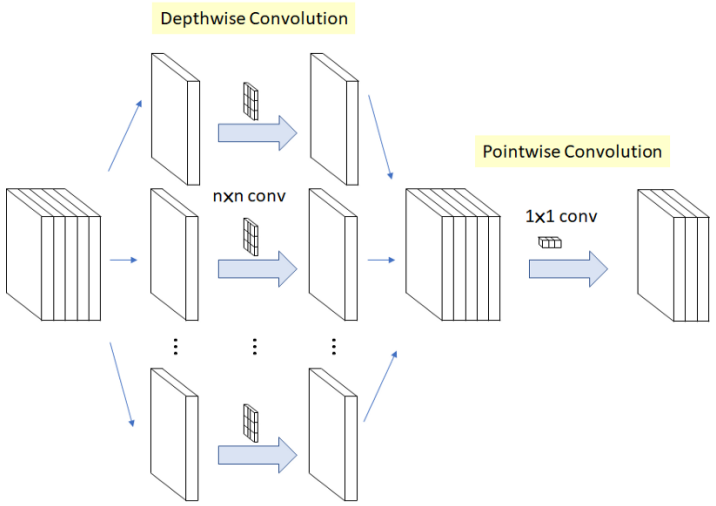
CNN (Convolutional Neural Networks, 합성곱 신경망, 컨볼루션 신경망)

- 컨볼루션 연산 요약
  - Point-wise Convolution



출처: https://gaussian37.github.io/dl-concept-dwsconv/

  - Depthwise Separable Convolution (깊이별 분리가능 컨볼루션)



출처: https://gaussian37.github.io/dl-concept-dwsconv/

LSTM (Long Short Term Memory Network)

작성중

RNN

- RNN에서 확장

RNN (Recurrent Neural Networks)

RNN

작성중

RNN (Recurrent Neural Networks, 순환 신경망)

- SimpleRNN
  - BPTT (Back Propagation Through Time, 시간에 따른 역전파)
- LSTM (Long Short-Term Memory)
  - 장기 종속성을 학습
- GRU (Gated Recurrent Unit)
- **RNN 위상**
  - 인코더-디코더 아키텍처
  - 변환기 아키텍처
- 활용
  - 음성 인식, 언어 모델링, 기계 번역
  - 감정 분석, 이미지 캡션

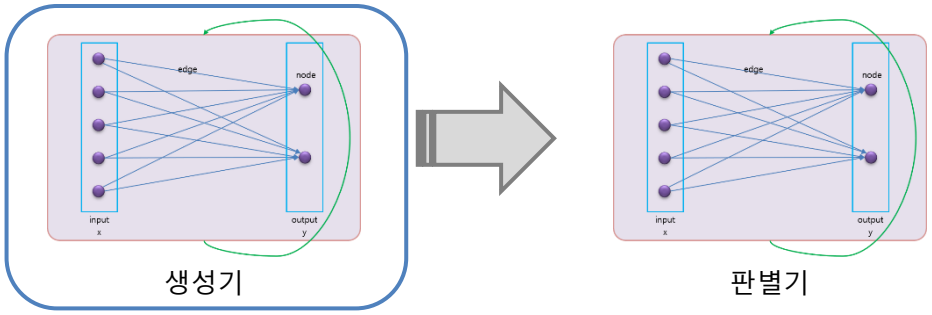
GAN (Generative Adversarial Networks)

GAN

작성중

GAN (Generative Adversarial Networks, 생성적 적대 신경망)

- Generator (생성기)
  - 예) 노이즈 입력을 받아 원하는 이미지 생성
  - 예) **전치 컨볼루션**을 사용하여 이미지 생성
  - 예) **특정 이미지를 받아 다른 이미지 생성**
- Discriminator (판별기)
- DCGAN (Deep Convolution GAN)
- 활용
  - GAN의 생성기로 산술 연산 가능
  - CycleGAN : 이미지 변환



- y = f(x) 에서 y에 대한 판단을 판별기로 대체한다.

RBFN (Radial Basis Function Network)

MLP (Multilayer Perceptron)

SOM (Self Organizing Map)

DBN (Deep Belief Network)

RBM (Restricted Boltzmann Machine)

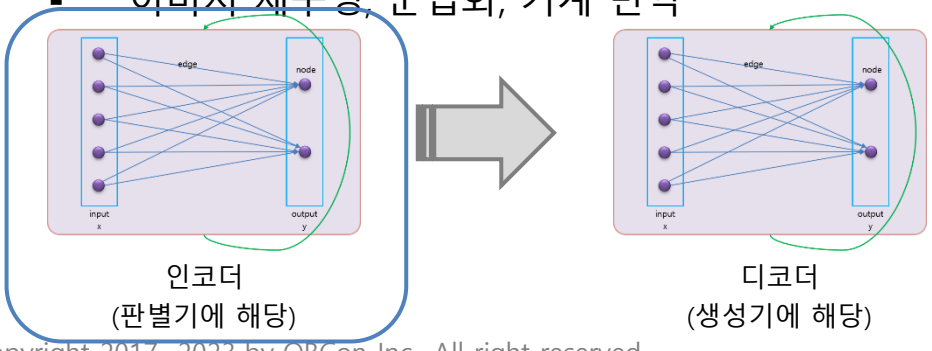
AutoEncoder

AutoEncoder

작성중

AutoEncoder

- Vanilla AutoEncoders (바닐라 오토인코더)
- VAE (Variation AutoEncoders, 가변 오토인코더)
- Sparse AutoEncoders (희소 오토인코더)
- Denosing AutoEncoders (디노이징 오토인코더)
  - 손상된 이미지를 재구성
- Convolution AutoEncoders
- 활용
  - 이미지 재구성, 군집화, 기계 번역



인코더
(판별기에 해당)

디코더
(생성기에 해당)

- 역전파 활용

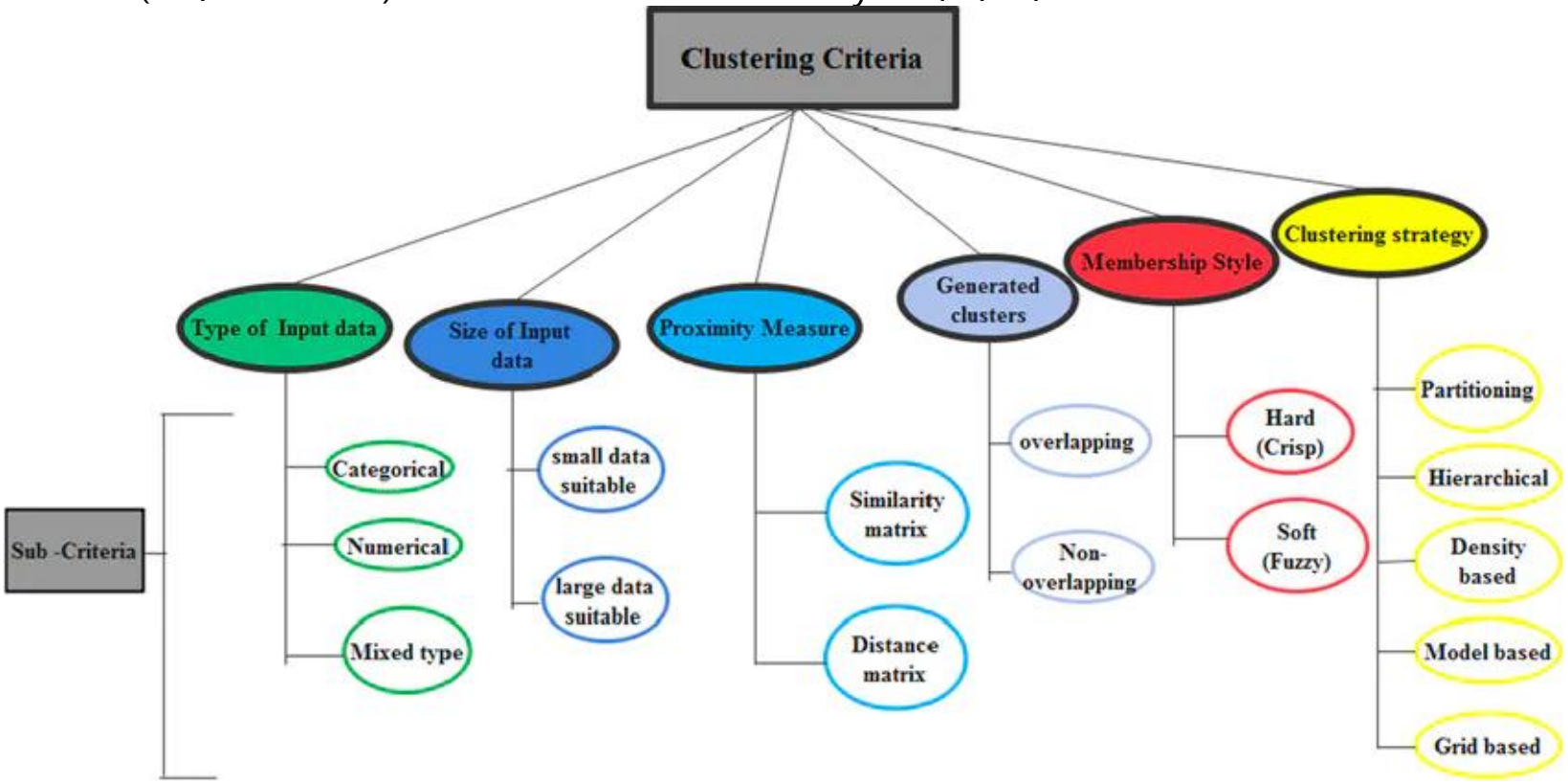
Word Embedding

Embedding

Image Embedding

- Image Embedding
  - 이미지를 저차원 공간에 수학적으로 표현한 것

- Vector (1차원 Dense) : Dense layer에서 사용

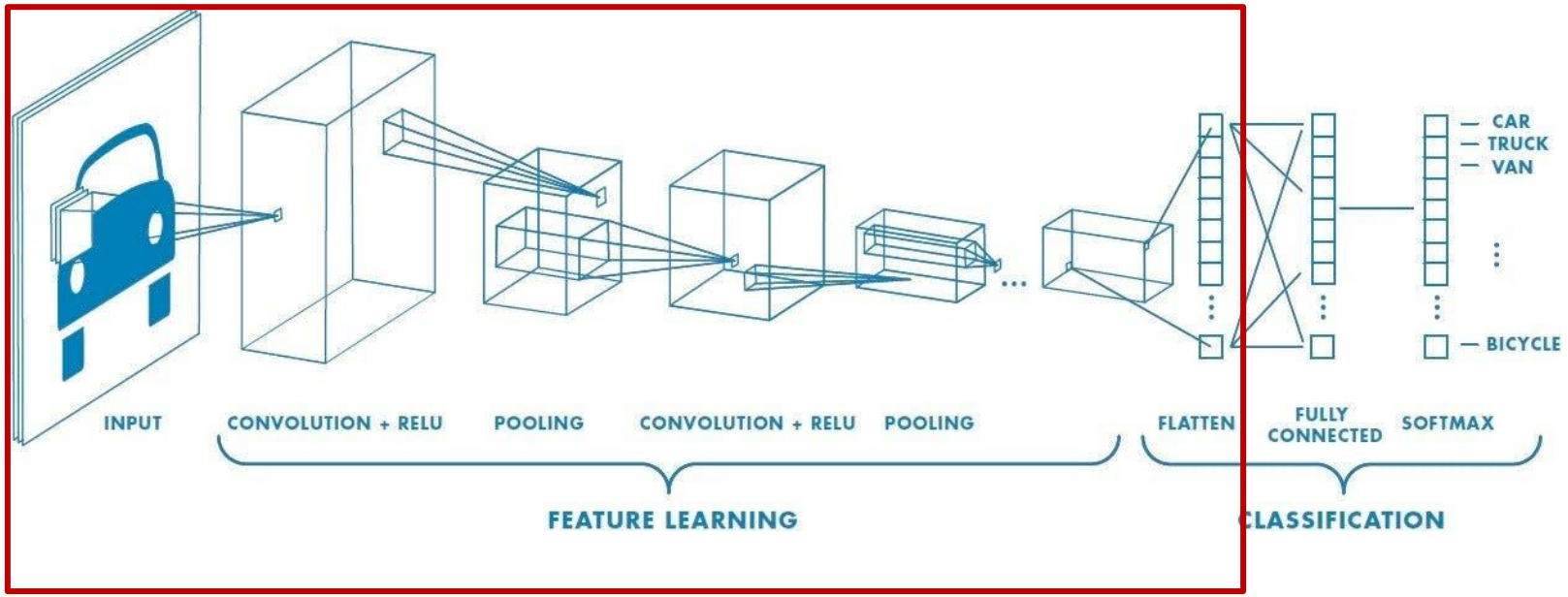


출처: https://blog-ko.superb-ai.com/what-are-image-embeddings-for-computer-vision-data-curation/

Embedding

Image Embedding

- Tensor : Conv layer에서 사용
- Feature

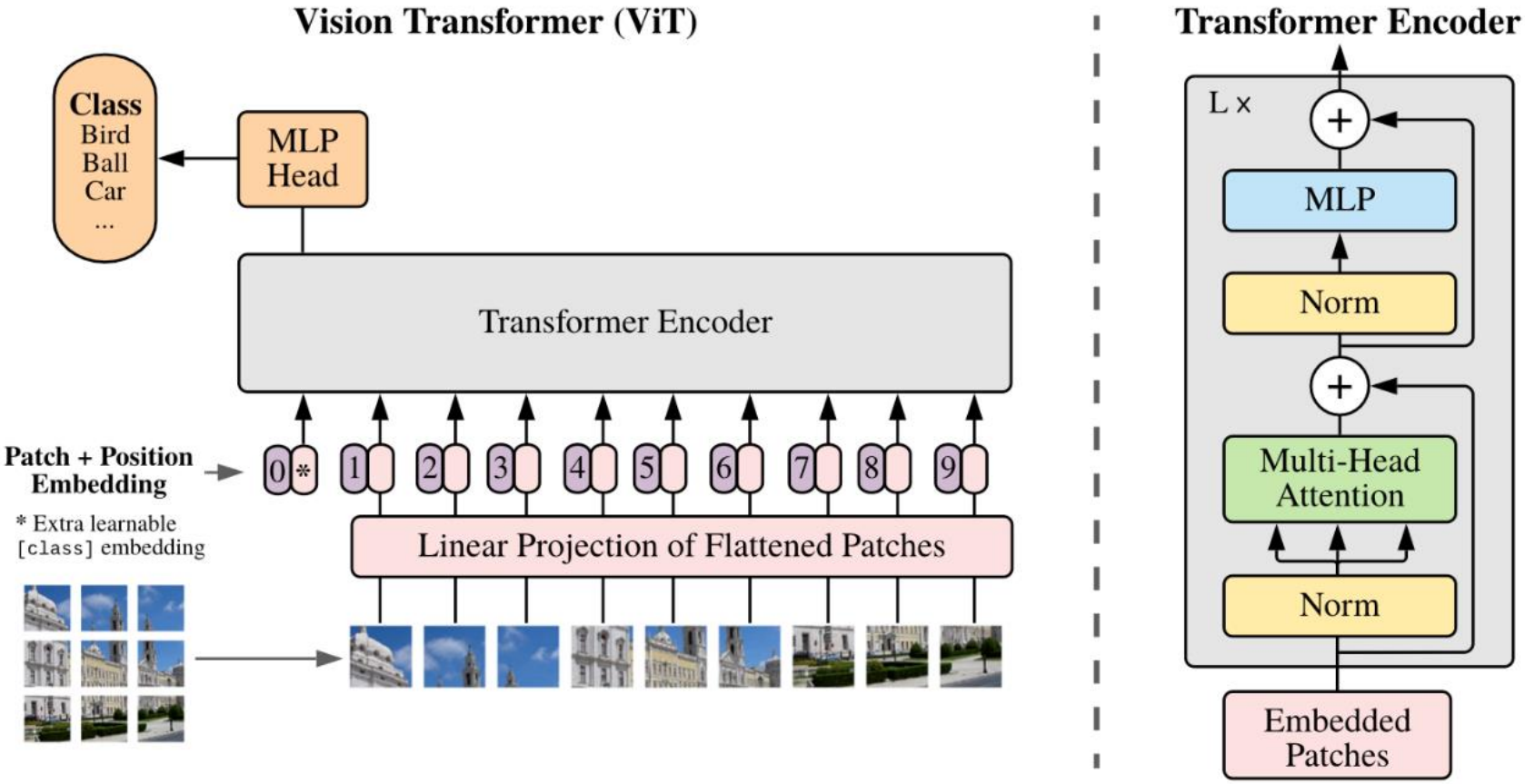


출처: https://encord.com/blog/image-embeddings-to-improve-model-performance/

Embedding

Image Embedding

- n개의 이미지로 분할 : ViT (Vision Transformer)



출처: https://mishuni.tistory.com/137

Embedding

Video Embedding

- 각각의 frame을 Image Embedding하여 처리 한다.
  - 결과를 합산하여 처리 한다
- 각각의 frame을 Image Embedding하여 RNN의 입력으로 사용 한다.

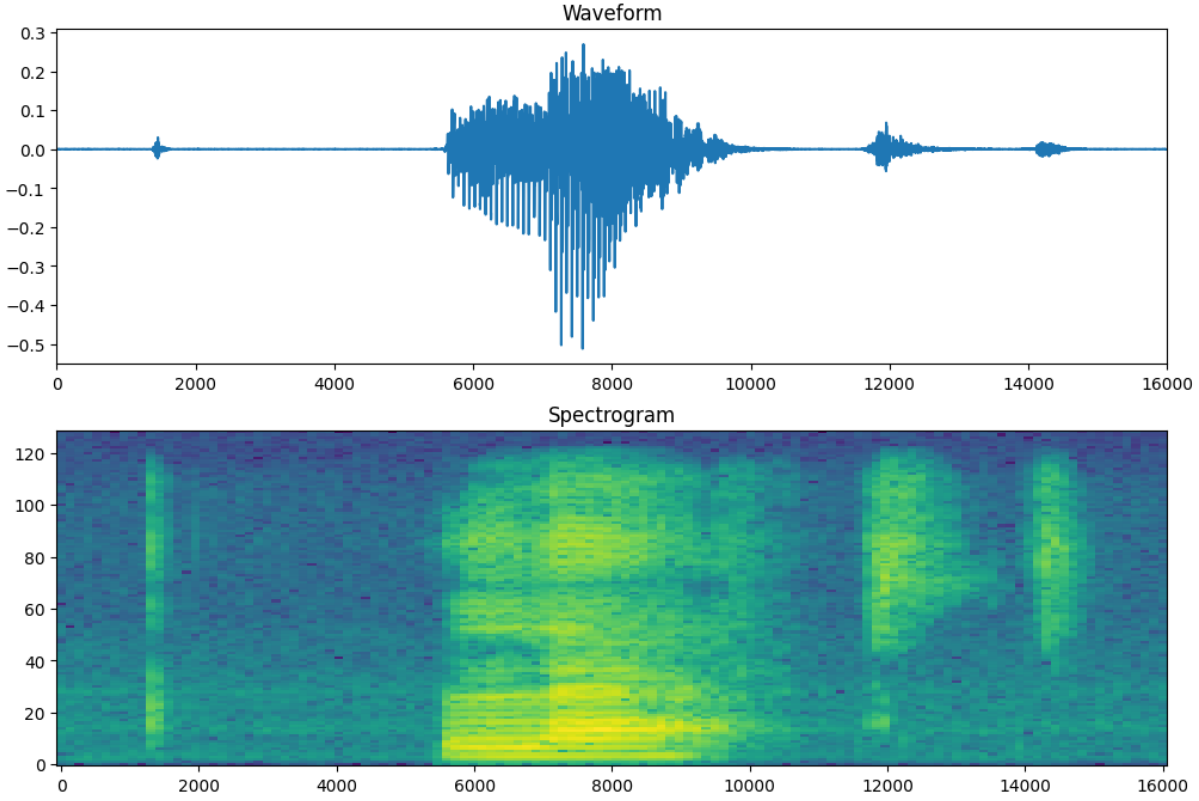
- 3D ConvNet을 사용 한다.
- 각각의 frame을 ConvNet을 사용하여 Feature로 변환한 후 사용 한다.
  - RNN이나 다층 퍼셉트론으로 전달 한다.
- 3D ConvNet을 사용하여 Feature로 변환한 후 사용 한다.
  - RNN이나 다층 퍼셉트론으로 전달 한다.

Embedding

Audio Embedding

- Digital sound : 초당 16,000샘플

- Audio의 파형을 Spectrogram(2D 이미지)으로 변환하여 사용 한다.
  - https://huggingface.co/docs/transformers/model_doc/audio-spectrogram-transformer



출처: https://www.tensorflow.org/tutorials/audio/simple_audio?hl=ko

Embedding

Audio 생성

- WaveNet, 2016년 (Conv 사용)
  - https://www.deepmind.com/blog/high-fidelity-speech-synthesis-with-wavenet
  - 원시 오디오 파형을 생성하는 심층 생성 모델
  - 초당 24,000개의 샘플로 파형을 생성
  - TTS (Text To Speech) 등에서 사용

출처: https://medium.com/@sthanikamsanthosh1994/speech-to-text-generation-wavenet-wavernn-f4434647dc27

- WaveRNN, 2018년

Word Embedding

Word Embedding

- 어휘의 word나 phrase가 실수의 벡터로 매핑되는 NLP에서의 언어 모델링과 특징 학습 기술의 집합

- NLP
  - Natural Language Processing
  - 자연어 처리

- Corpus: 말뭉치
- Word
- Phrase

작성중

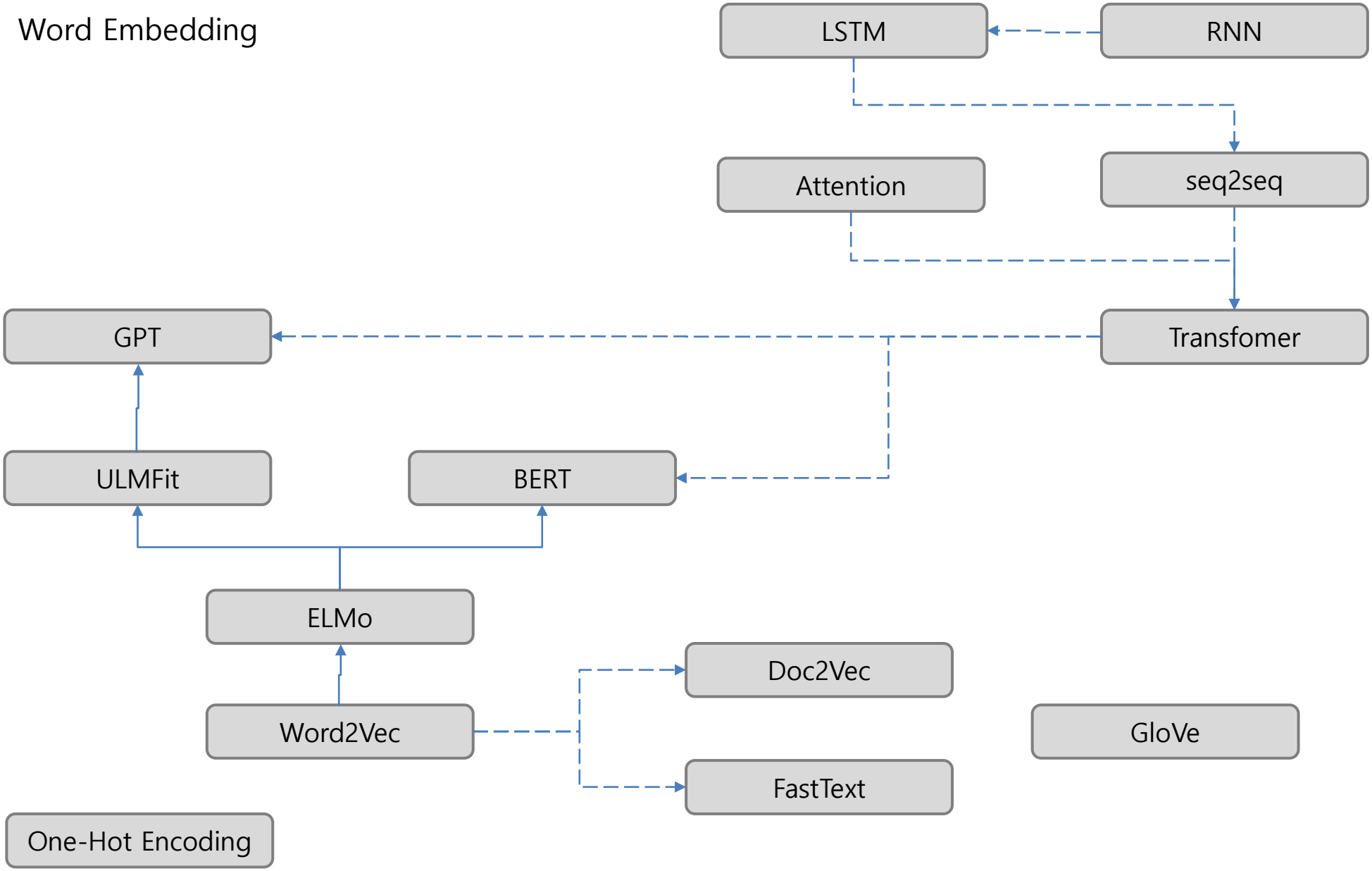
Word Embedding

- Static Embeddings (정적 임베딩)
  - IT (Information Retrieval, 정보 검색) 기술을 사용하여 변환
  - OHE (One-Hot Encoding)
  - TF-IDF (Term Frequency-Inverse Document Frequency, 용어 빈도- 역 문서 빈도)
  - LSA (Latent Semantic Analysis, 잠재 문맥 분석)
  - 주제 모델링
  - 단점
    - Polysemi(중의어) 문제를 해결할 수 없다.
- 동적 임베딩
  - 문맥에 따라 단어 임베딩
  - seq2seq

- 분산 가설
  - 유사한 맥락에서 등장하는 단어들은 비슷한 의미를 갖는 경향이 있다.
- Distributed Representations (분산 표현)
  - 한 단어의 의미를 문맥상에서 다른 단어와의 관계를 고려하여 포착

Word Embedding

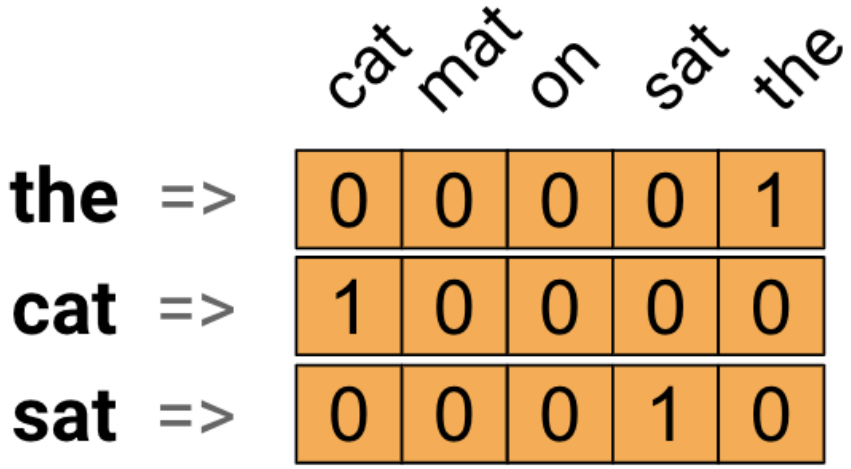
Word Embedding



Word Embedding

단어 임베딩

- OHE (One-Hot Encoding)
  - 범주형 변수를 숫자형 변수로 변환
    - Sparse Representation(희소 표현)의 Sparse vector로 변환 ←→ Dense Representation (밀집 표현)
  - 예) [0-9]의 값을 가지는 범주형 변수를 [0, 0, 0, 0, 0, 1, 0, 0, 0] vector로 변환 한다.
    - vector에서 해당되는 범주가 일치하는 항목만 1의 값을 가진다.
  - 단점
    - 단어간의 유사도를 계산할 수 없다.



출처: https://www.tensorflow.org/text/guide/word_embeddings

Word Embedding

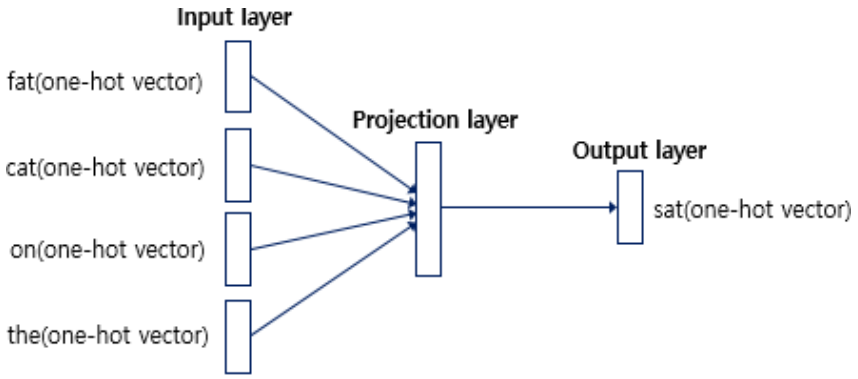
단어 임베딩

- Word2Vec : 예측 기반 예측 모델
  - CBOW (Continuous Bag of Words) : 주변 단어로 현재 단어 예측
  - Skip-Gram : 현재 단어로 주변 단어 예측
    - SGNS (Skip-Gram with Negative Sampling)

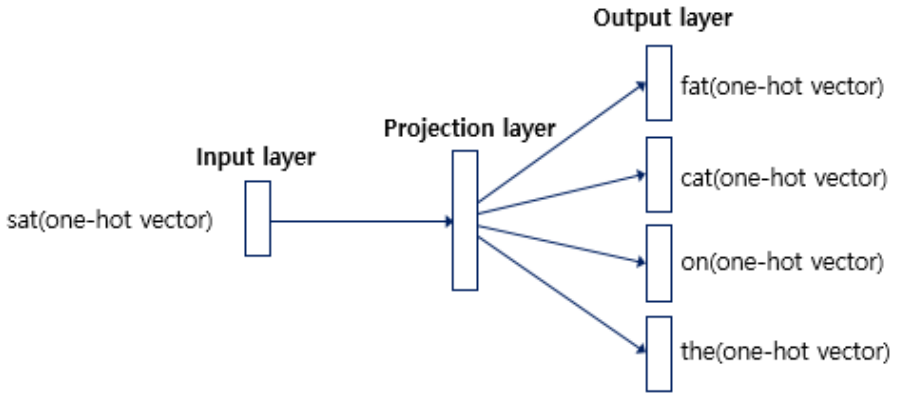
  - 구글 뉴스 30억개 토큰 기반으로 하는 300만 단어 벡터
  - https://drive.google.com/file/d/0B7XkCwpI5KDYNINUTTISS21pQmM/edit

The fat cat sat on the mat

CBOW



Skip-Gram



출처: https://wikidocs.net/22660

작성중

단어 임베딩

- GloVe (Golbal Vectors for word representation, 단어 표현 전역 벡터)
  - Gigawords 60억개 토큰 기반 40만개 단어 벡터
  - 카운트와 예측 기반 모델
  - https://nlp.standford.edu/projects/glove/
  - OOV 문제가 있음
  - Window based Co-occurrence Matrix (윈도우 기반 동시 등장 행렬)
    - 2개의 단어가 문장에서 동시해 등장하는 횟수

- https://wikidocs.net/22885

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

출처: https://iq.opengenus.org/glove-and-word2vec/

작성중

문자와 부분 단어 임베딩

- 문자 임베딩 : n-gram을 단어로 취급해 Embedding

작성중

문자와 부분 단어 임베딩

- FastText ← Word2Vec
  - https://wikidocs.net/22883
  - 단어 임베딩 + n-gram 임베딩 (n = 3 ~ 6)

  - 구문의 유사성 추가
  - GloVe의 OOP 문제 해결

  - 단어를 n-gram으로 나누어 학습

Word Embedding

작성중

문장과 문단 임베딩

- 문장 또는 문단을 단어로 취급하여 처리 한다.
  - https://wikidocs.net/103496
  - 구성 단어의 단어 벡터를 평균화 한다.

  - Google Universal Sentence Encoder
  - Doc2Vec
  - Paragraph2Vec

작성중

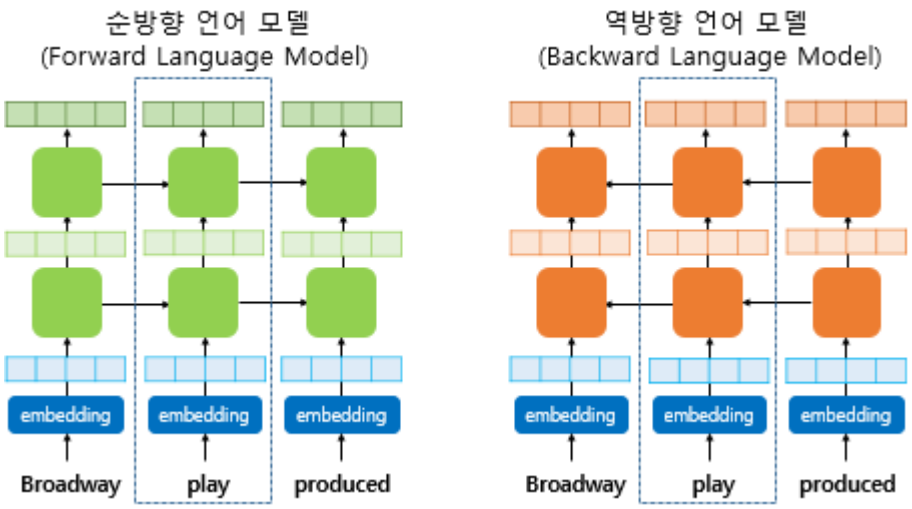
신경망 임베딩

- 신경망 임베딩
  - Item2Vec
  - node2vec
  - ELMo

작성중

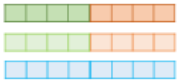
신경망 임베딩

- ELMo (Embeddings from Language Models)
  - 사전 훈련된 언어 모델(biLM, Bidirectinal Language Model)을 사용 한다.

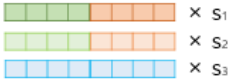


출처: https://wikidocs.net/33930

1) 각 층의 출력값을 연결(concatenate)한다.

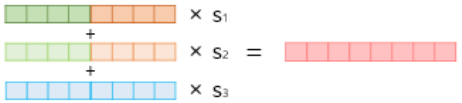


2) 각 층의 출력값 별로 가중치를 준다.



이 가중치를 여기서는 s_1, s_2, s_3라고 합시다.

3) 각 층의 출력값을 모두 더한다.



2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.

출처: https://wikidocs.net/33930

작성중

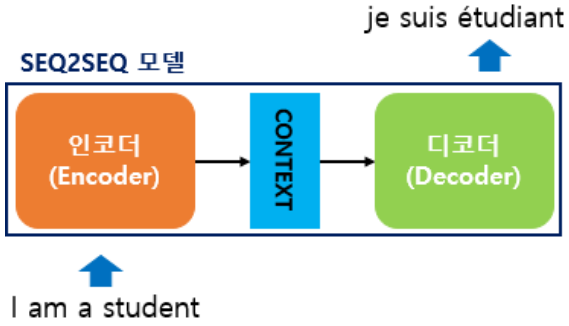
언어 모델 기반 임베딩

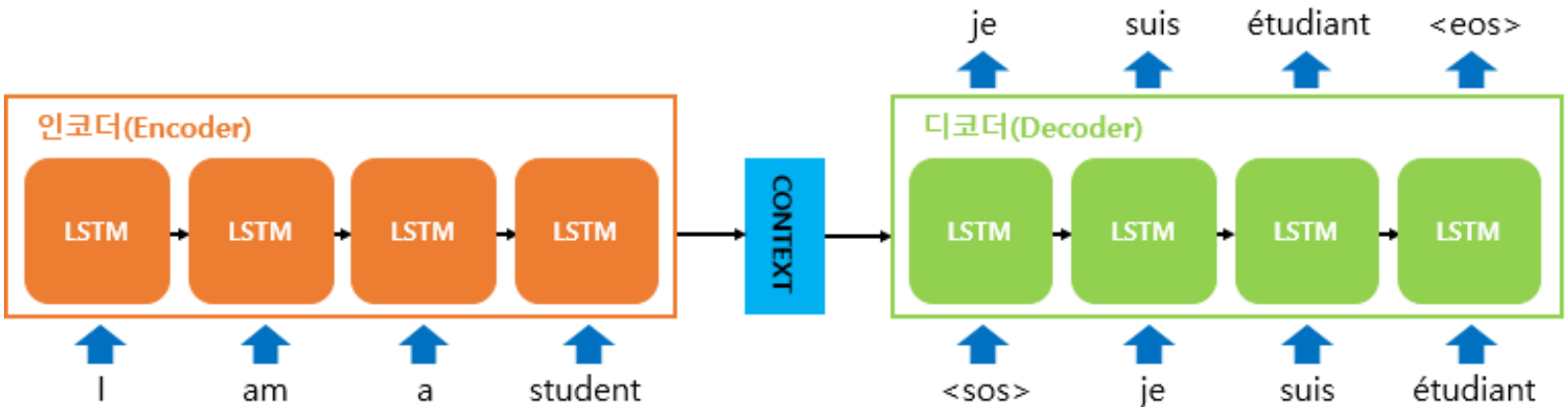
- 언어 모델 기반 임베딩 ← ELMo
  - 언어 모델은 단어 시퀀스에 대한 확률 분포 이다.
  - ULMFit
    - GPT (Generative Pre-trained Transformer) : ULMFit의 LSTM 대신 표준 Transfomer의 디코더 스택을 사용
      - 단방향만 지원
  - BERT (Bidirectional Encoder Representations for Transformers)
    - Transfomer의 인코더 스텍을 사용
    - 입력의 최대 15%를 마스킹해 양방향을 지원
    - 단일, 다중-문장 분류, 질의 응답, 태깅 등 여러가지 지도 학습에 사용
    - BERT-base
      - 11개의 인코더 계층, 768개의 은닉 유닛
      - 12개의 attention 헤드
      - 1억 1천만개의 매개 변수
    - BERT-large
      - 24개의 인코더 계층, 1024개의 은닉 유닛
      - 16개의 attention 헤드
      - 3억 4천만개의 매개 변수

작성중

언어 모델 기반 임베딩

- seq2seq ← RNN
  - https://wikidocs.net/24996





작성중

언어 모델 기반 임베딩

- Transfomer
  - seq2seq에서 LSTM을 Attention으로 대체
  - Attention: https://wikidocs.net/22893
    - 디코더에서 출력 단어를 예측하는 매 시점(time step)마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고한다
    - 해당 시점에서 예측해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중(attention)해서 본다.

  - http://www.incodom.kr/Transformer_%EA%B8%B0%EB%B0%98_%EB%AA%A8%EB%8D%B8

작성중

Word Embedding

- 기타 임베딩
  - ConceptNet Numberbatch
  - InferSent : 문장 인코딩
  - SkipThoughts : 문장 인코딩
  - CoherenceModel
  - HdpModel
  - LdaModel
  - LsiModel
  - TfidfModel

작성중

Word Embedding

- 기타 임베딩
  - OkapiBM25Model, LuceneBM25Model, AtireBM25Model
  - RpModel
  - LogEntropyModel

  - KeyedVectors
  - LdaMulticore
  - Phrases
  - NormModel
  - AuthorTopicModel
  - LdaSeqModel
  - TranslationMatrix, BackMappingTranslationMatrix
  - EnsembleLda
  - Nmf

비지도학습

작성중

비지도학습

- PCA (Prinicipal Component Analysis, 주성분분석)
- SOM (Self-Organized Maps, 자체 구성 맵)
- RBM
  - 이미지 재구성
- VAE (Variation AutoEncoders, 가변 오토인코더)

RL (Reinforcement Learning)

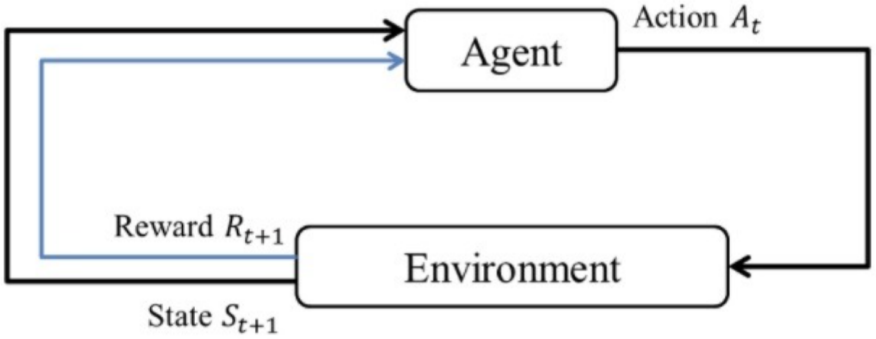
RL

RL (Reinforcement Learning, 강화학습)

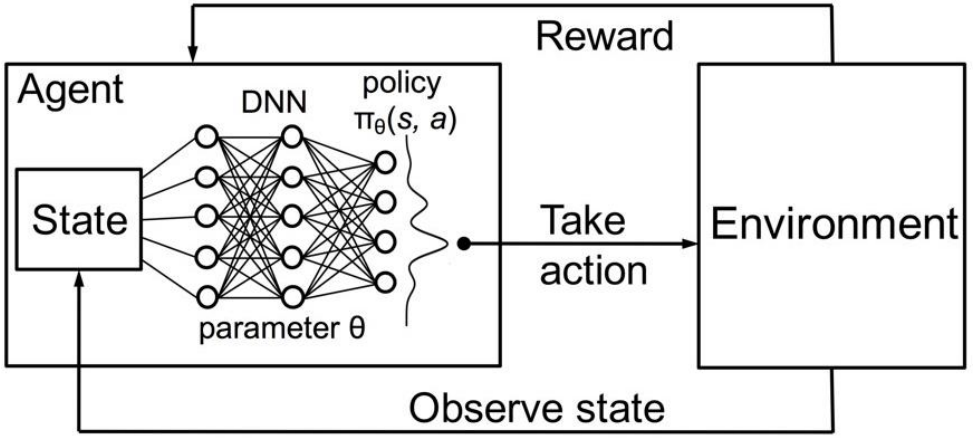
- Environment Model (환경 모델)
- Agent

- AGI (Artificial General Intelligence, 인공 일반 지능)

- State (상태) : S_t
- Action (행동) : A_t
- Reward (보상) : $R(S_t, A_t, S_{t+1})$



출처: https://brunch.co.kr/@namujini/22



출처: https://rnd.gmdsa.org/chatgpt-series-2/

RL (Reinforcement Learning, 강화학습)

- Policy (정책) : π(S) → A

- Return : G_t
  - 향후 가능한 모든 보상의 할인 총합 (미래 보상을 평가)
  - Discont factor (할인 인수, γ) : 0 ~ 1 사이의 값

  - Value function (가치 함수) : V(S)
    - State-Value (상태 가치) 함수 : V^{π}(S) → Return
      - S 상태에서 π 정책을 따랐을 때 기대되는 return
    - Action-Value (행동 가치) 함수 : Q^{π}(S, **A**) → Return
      - S 상태에서 A 행동을 한 후, π 정책을 따랐을 때 기대되는 return

RL

DRL (Deep Reinforcement Learning, 심층 강화학습)

- 심층 신경망을 사용하여 학습
  - Policy-based 기법 : 정책 함수 근사화
  - Value-based 기법 : 가치 함수 근사화

- Value-based 기법
  - Action-Value (행동 가치) 함수 : $Q^{\pi}(S, \mathbf{A})$ → Return
  - 가치 함수를 최대화하는 행동을 취한다.
    - S, A, Return을 알고 있으면 학습이 가능 하다.

- Policy-based 기법
  - Policy (정책) : $\pi(S)$ → **A**
  - State-Value (상태 가치) 함수 : $V^{\pi}(S)$ → Return
  - 최적의 정책 (기대 리턴을 최대화 하는 것)을 예측 한다.
    - S, A와 $V^{\pi}(S)$ 함수를 알고 있으면 학습이 가능 하다.

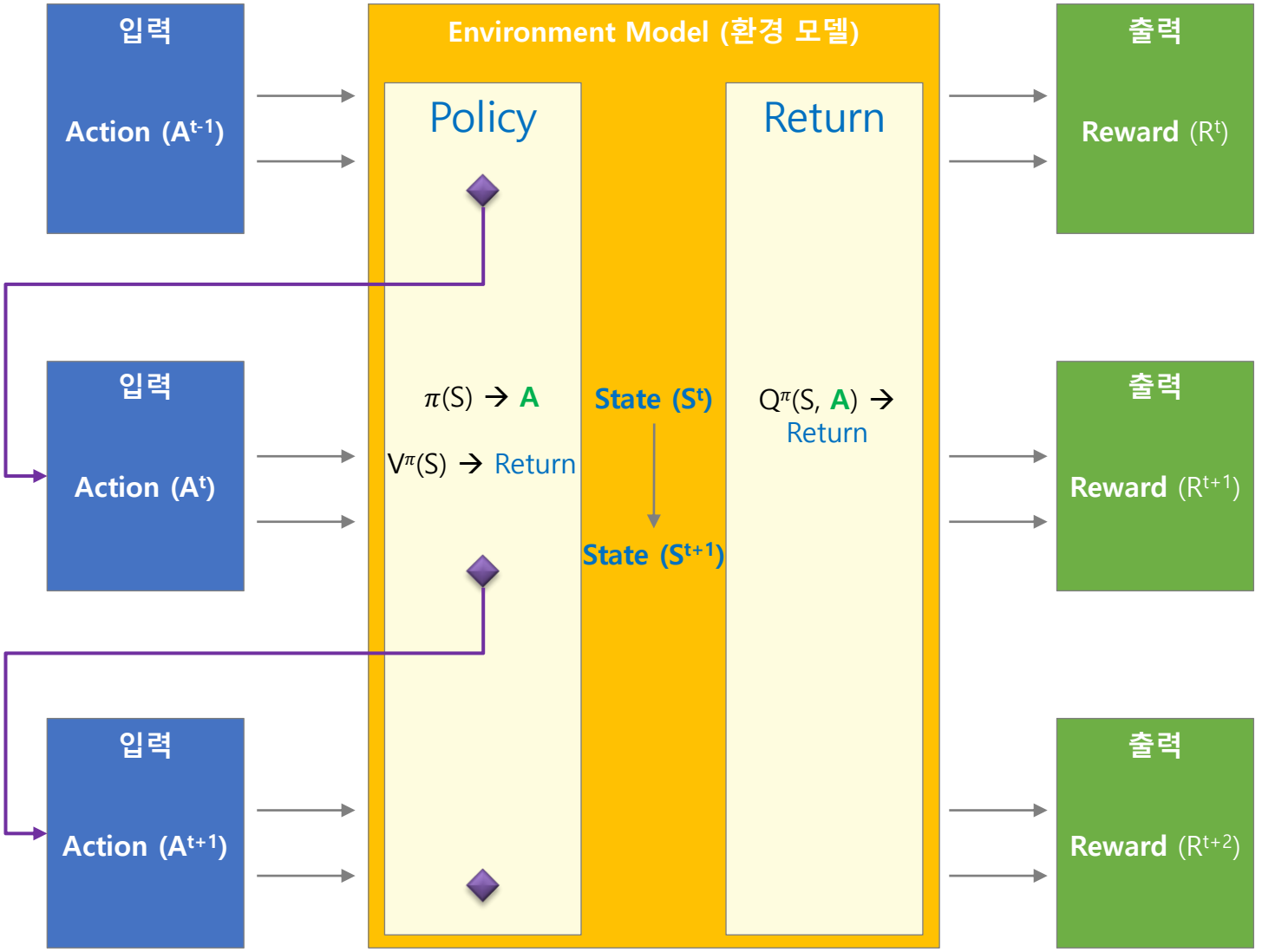
RL

DRL (Deep Reinforcement Learning, 심층 강화학습)

- Exploration vs Exploitation (탐색 대 활용)
  - Agent는 탐색을 사용해 정보를 수집하고, 이후 수집된 정보를 활용해 최선의 결정을 내린다.
  - Epsilon-Greedy (∈-greedy) policy
    - ∈ 확률: 랜덤 행동, 1 - ∈ 확률 : 가치 함수를 최대화 하는 행동
    - ∈ 는 점근적으로 감소

- Training 방법
  - Experience Replay (경험 재생)
    - (S^t, A^t, R^t, S^{t+1})로 학습
  - PER (Prioritized Experience Replay, 우선 경험 재생)
    - 경험에 가중치 부여
  - 움직이는 목표 문제 처리
    - 로컬 신경망 (단기 고정 목표를 가짐), 목표 신경망
    - n step 학습후 로컬 신경망의 가중치를 목표 신경망으로 복사

DRL (Deep Reinforcement Learning, 심층 강화학습)



- RL (Reinforcement Learning, 강화학습)
- Environment Model (환경 모델)
  - State (상태)
  - Action (행동)
  - Reward (보상)
    - R (S_n, A_n, S_{n+1})
- Policy (정책)
  - Policy-based 기법
  - $\pi(S)$ → A
- Return : G_t
  - Value-based 기법
  - 향후 가능한 모든 보상의 할인 총합
  - $V^{\pi}(S)$ → Return
  - $Q^{\pi}(S, A)$ → Return

작성중

DRL (Deep Reinforcement Learning, 심층 강화학습)

- 성공 사례
  - AlphaGo Zero
    - 하나의 신경망으로 바둑돌의 움직임(Action)과 가치(Return)를 학습함
    - 40 Block Residual CNN (ResNet)을 사용하여 40일간 학습 (2,900만번 바둑을 둠)
    - 64 GPU, 19 CPU 장비에서 TensorFlow를 사용하여 학습
  - AI 제어 sailplanes
    - 자동 조종 항법을 위한 컨트롤러 시스템
  - Locomotion behavior (보행 능력)
    - 복잡 다양한 환경에서 보행 능력

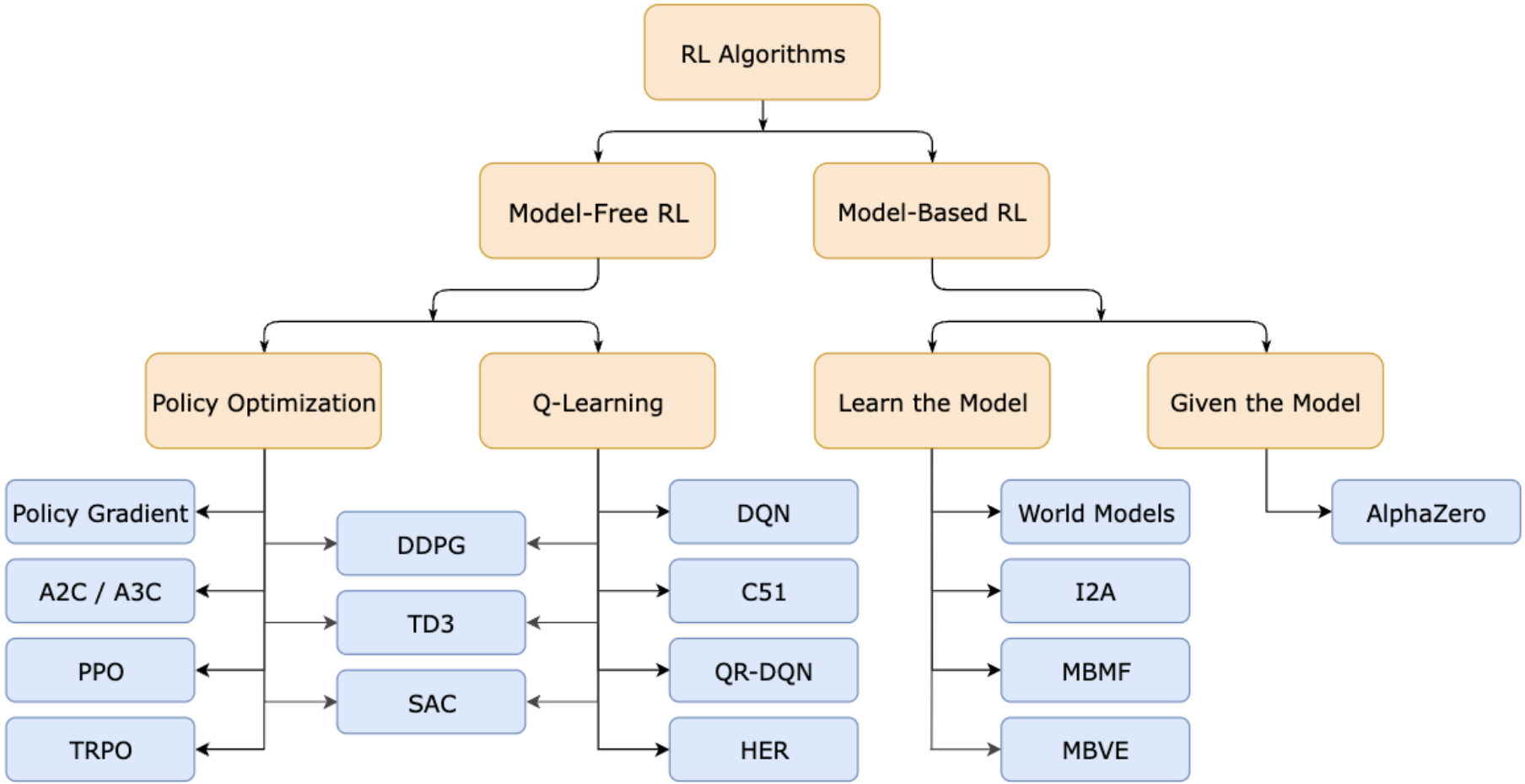
- DQN (Deep Q-Network, 심층 Q 신경망)
  - Action-Value (행동 가치) 함수 : $Q^{\pi}(S, \mathbf{A})$ → Return
  - Double DQN
  - Dueling DQN
  - Rainbow

- DDPG (Deep Deterministic Policy Gradient, 심층 확정적 정책 그래디언트)

작성중

DRL (Deep Reinforcement Learning, 심층 강화학습)

- 강화학습 알고리즘 분류

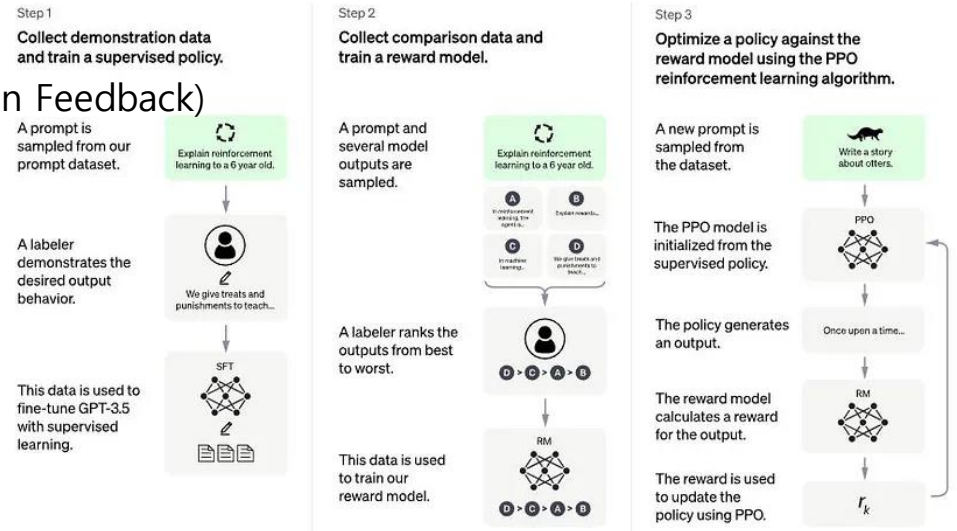


출처: https://www.thedatahunt.com/trend-insight/reinforcement-learning

작성중

DRL (Deep Reinforcement Learning, 심층 강화학습)

- 적용 분야
  - 로보틱스
    - 제품 조립과 결함 검사
    - 재고 관리
  - 게임
  - 자율 주행
  - NLP
    - Text generation (텍스트 생성)
    - Dialogue systems (대화 시스템)
    - Sentiment analysis (감정 분석)
    - Text summarization (텍스트 요약)
    - RLHF (Reinforcement Learning from Human Feedback)



출처: https://www.thedatahunt.com/trend-insight/reinforcement-learning

AutoML

AutoML

- 도메인 전문가가 머신러닝 기술을 사용할 수 있도록 한다.

출처: https://fastcampus.co.kr/media_data_automl?gclid=CjwKCAjwivemBhBhEiwAJxNWNxQdm51PI5iX1NpCN7TLabK_0kDcp-swrlk4v2Lgn75di3HWximGuRoCMogQAvD_BwE

AutoML

AutoML

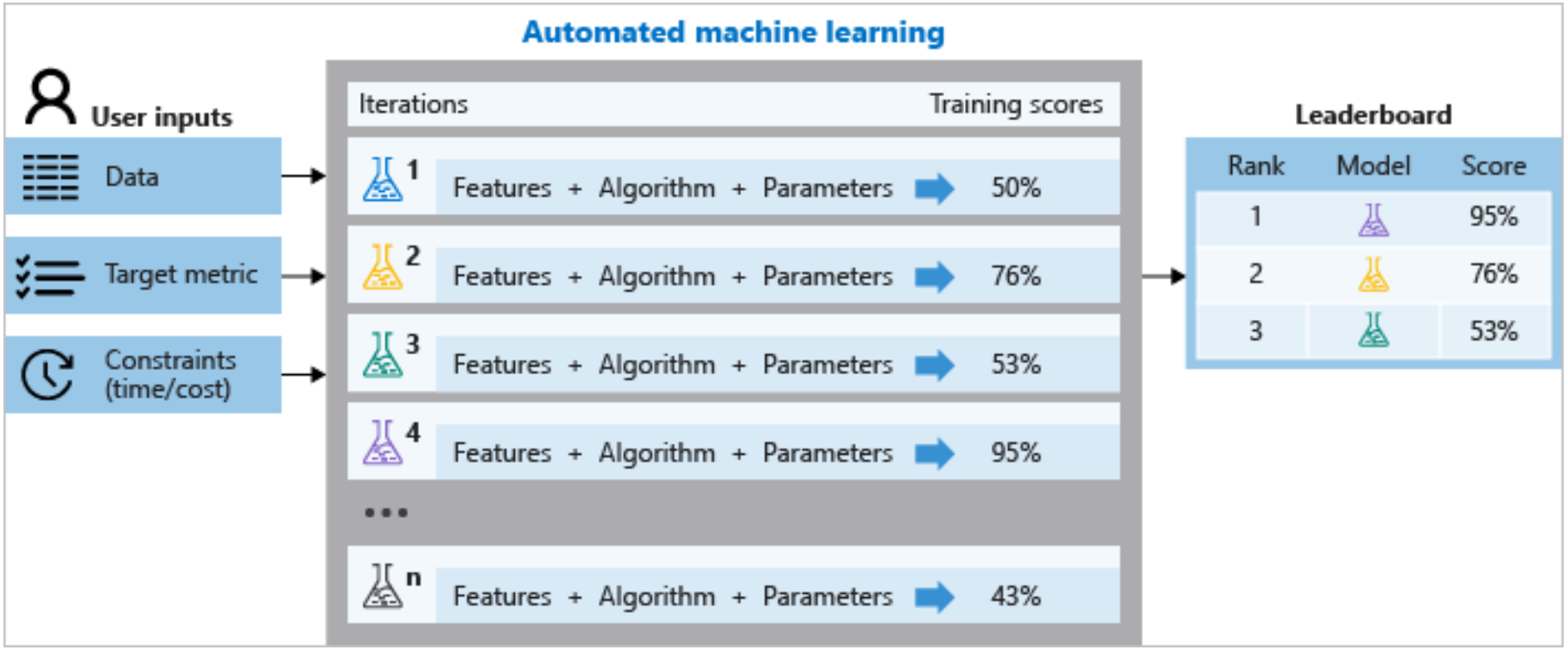
- 데이터 준비
  - Cleansing (데이터 정리) : 데이터 품질 향상
  - Synthesis (데이터 합성) : 합성 데이터 생성

- Feature Engineering
  - Feature selection (특징 선택) : 의미 있는 특징의 부분 집합을 선택
  - Feature construction (특징 구성) : 새로운 파생 특징을 구축
  - Feature extraction (특징 추출) : 매핑 기능을 사용해 원래 특징 공간을 변경

- 모델 생성
  - Model Generation (모델 생성)
    - NAS (Neural Architecture Search, 신경 구조 검색)
      - 거의 모든 딥러닝 모델을 포함한 일련의 기본 작업을 결합해 모델을 생성
      - 신경망을 사용하여 신경망을 설계하는 것
    - ENAS (Efficient Neural Architecture Search) : NAS보다 1,000배 빠름
    - 전이 학습
    - GP (Genetic Programming, 유전자 프로그래밍) + EA (Evolutionary Algorithms, 진화 알고리즘)
  - 초매개 변수 튜닝
    - Batch size, Learning rate, …

AutoML

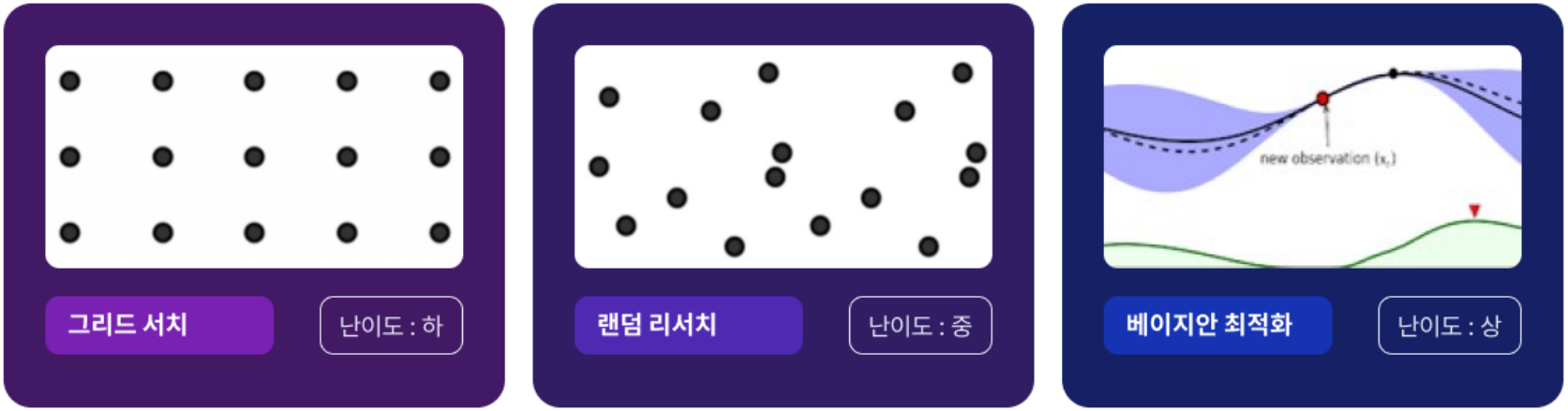
- AutoML



출처: https://learn.microsoft.com/en-us/azure/machine-learning/concept-automated-ml?view=azureml-api-2

AutoML

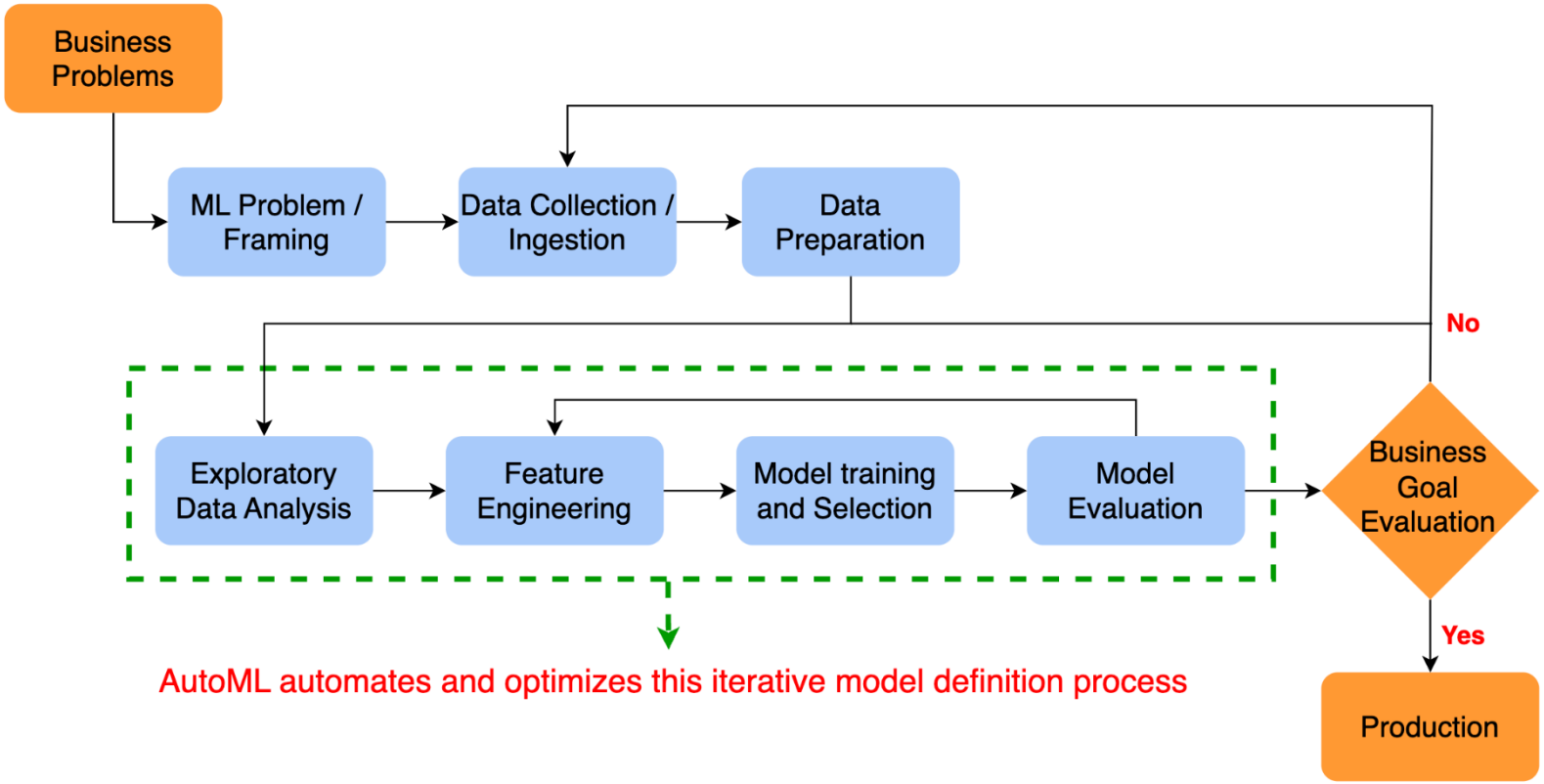
- 초매개 변수 튜닝



출처: https://fastcampus.co.kr/media_data_automl?gclid=CjwKCAjwivemBhBhEiwAJxNWNxQdm51PI5iX1NpCN7TLabK_0kDcp-swrlk4v2Lgn75di3HWximGuRoCMogQAvD_BwE

AutoML

- AutoML



출처: https://docs.treasuredata.com/display/public/PD/AutoML

AI 반도체

작성중

AI 반도체

- 업체
  - 퀄컴, 인텔, 엔비디아
  - SKT, 구글, 아마존, 애플, 테슬라

- 종류
  - 엔비디아
    - GPU
  - AMD
    - CDNA-2
    - CDNA-3
  - Google
    - TPU (Tensor Process Unit)
  - 삼성전자
    - HBM3-PIM
  - SK 하이닉스
    - GDDR6-AiM
  - SKT
    - SAPEON X220

Deep Learning 활용

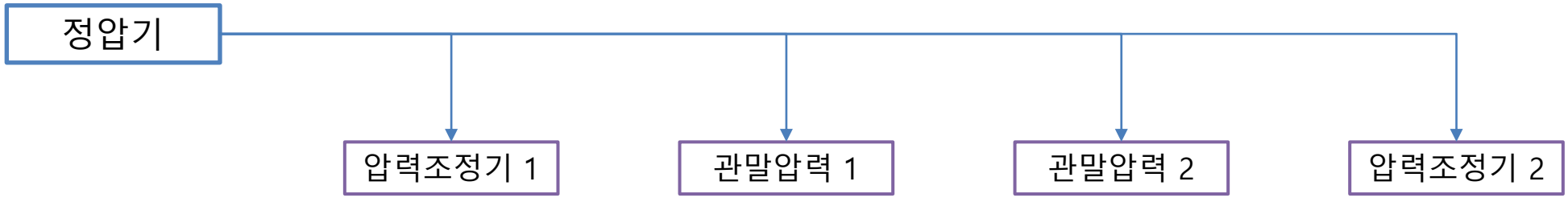
1. IoT
2. Stock

Deep Learning 활용

작성중

IoT

- RL (Reinforcement Learning, 강화학습)



Deep Learning 활용

작성중

Stock

- ppp

감사 합니다

Q & A

오픈소스 비즈니스 컨설팅