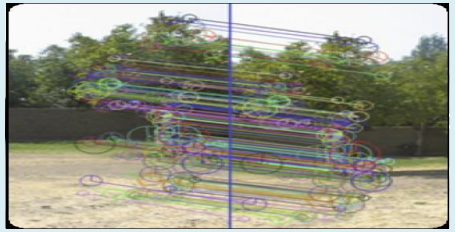
Visual Intelligence Theory
(#13: Autoencoder Concept and Applications)

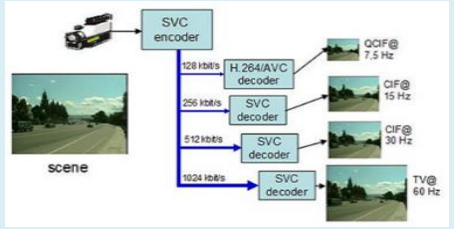
2020 Spring

Prof. Byung-Gyu Kim
Intelligent Vision Processing Lab. (IVPL)
http://ivpl.sookmyung.ac.kr
Dept. of IT Engineering, Sookmyung Women's University
E-mail: bg.kim@ivpl.sookmyung.ac.kr

Goal of this lecture

- What is the Autoencoder?
  - Concept
  - Model Structure
  - Actual Practices and Applications





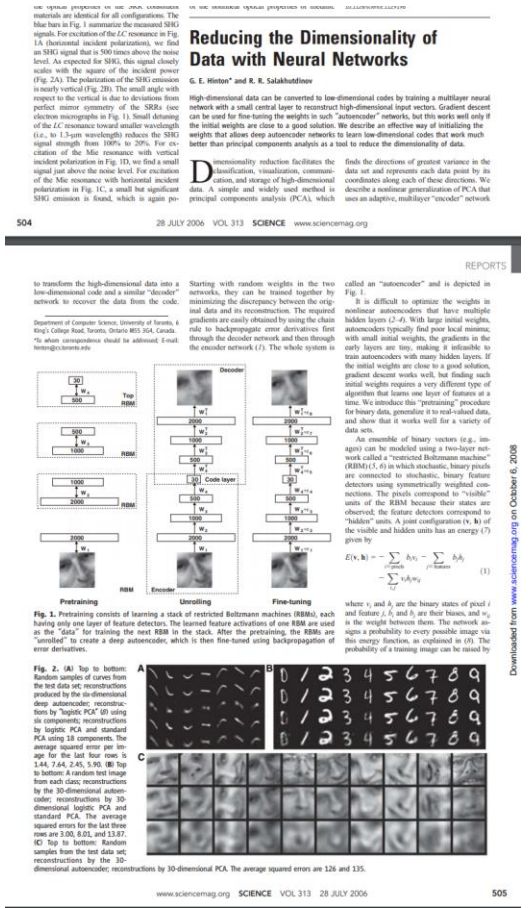
Contents

- **The Concept of Autoencoder**

The Concept of Autoencoders (0)

❖ **Autoencoder** is:

- **a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input** while **compressing the data** in the process so as to discover a more efficient and compressed representation.
- The idea was originated in the 1980s, and later promoted by the seminal paper by [Hinton & Salakhutdinov, 2006][1].



Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

504 28 JULY 2006 VOL 313 SCIENCE www.sciencemag.org

REPORTS

Pretraining Unrolling Fine-tuning

Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Fig. 2. (A) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (8) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (B) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (C) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

www.sciencemag.org SCIENCE VOL 313 28 JULY 2006 505

Downloaded from www.sciencemag.org on October 6, 2008

[1] https://pdfs.semanticscholar.org/c50d/ca78e97e335d362d6b991ae0e1448914e9a3.pdf

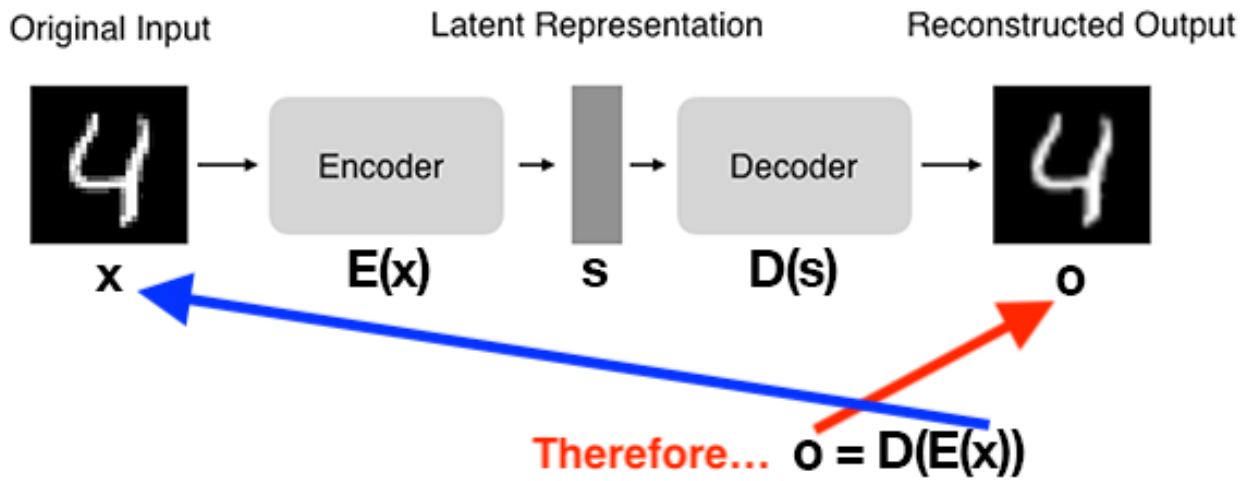
The Concept of Autoencoders (1)

- A type of **unsupervised neural network** (i.e., no class labels or labeled data) that seek to:
  1) Accept an input set of data (i.e., the *input*).
  2) Internally *compress* the input data into a **latent-space representation** (i.e., a single vector that *compresses* and *quantifies* the input).
  3) **Reconstruct the input data** from this latent representation (i.e., the output).
- Two components/subnetworks:
  - **Encoder:** Accepts the input data and compresses it into the latent-space. If we denote our input data as x and the encoder as E, then the output **latent-space representation**, s, would be $s = E(x)$.
  - **Decoder:** The decoder is responsible for accepting the latent-space representation s and then reconstructing the original input. If we denote the decoder function as D and the output of the detector as o, then we can represent the decoder as $o = D(s) = D(E(x))$.

The Concept of Autoencoders (2)

❖ Basic Structure

- The autoencoder as a whole can thus be described by the function **D**(**E**(***x***)) = ***o*** where you want ***o*** as close as the original input ***x***.



Why just copy data????

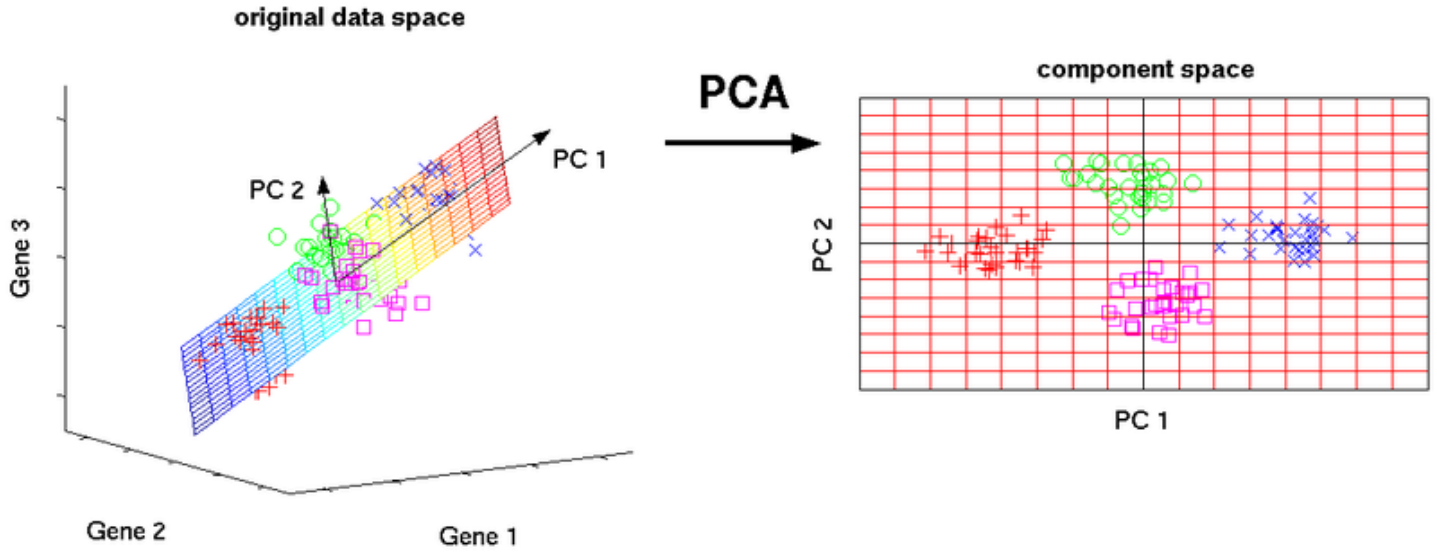
The Concept of Autoencoders (3)

❖ Latent (space) representation $\mathbf{s}$

- Can be defined as "**Feature space to explain observational data well**".
- Widely used in dimensionality reduction technique.

Original observation space

Latent space



Principle Component Analysis (PCA)

The Concept of Autoencoders (4)

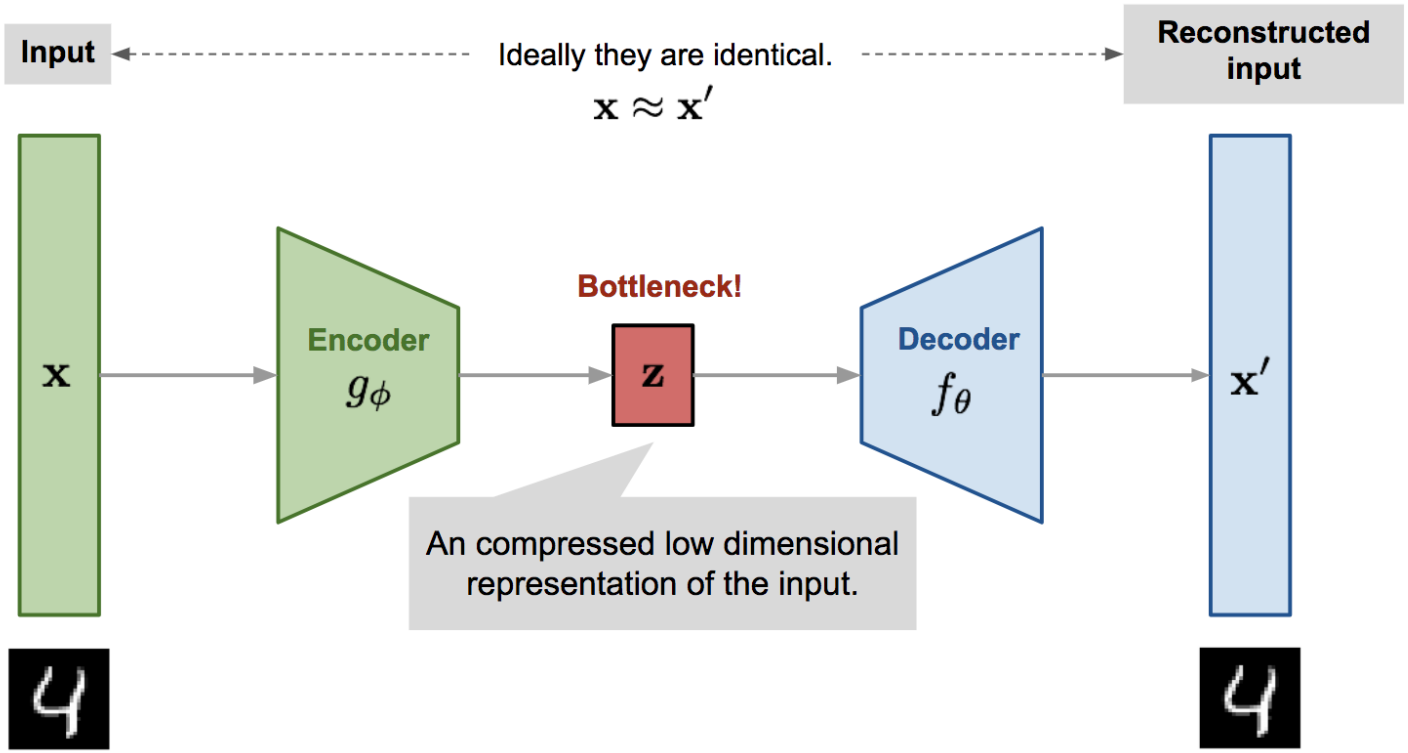
❖ Constraints for the latent space representation $\mathbf{s}$

- Undercomplete:
  - One way to obtain useful features from the autoencoder is to constrain $\mathbf{s}$ to have smaller dimensions than $\boldsymbol{x}$

- Overcomplete:
  - the dimension of the latent representation is greater than the input $\boldsymbol{x}$.

Model Structure (1)

❖ Autoencoder model architecture:



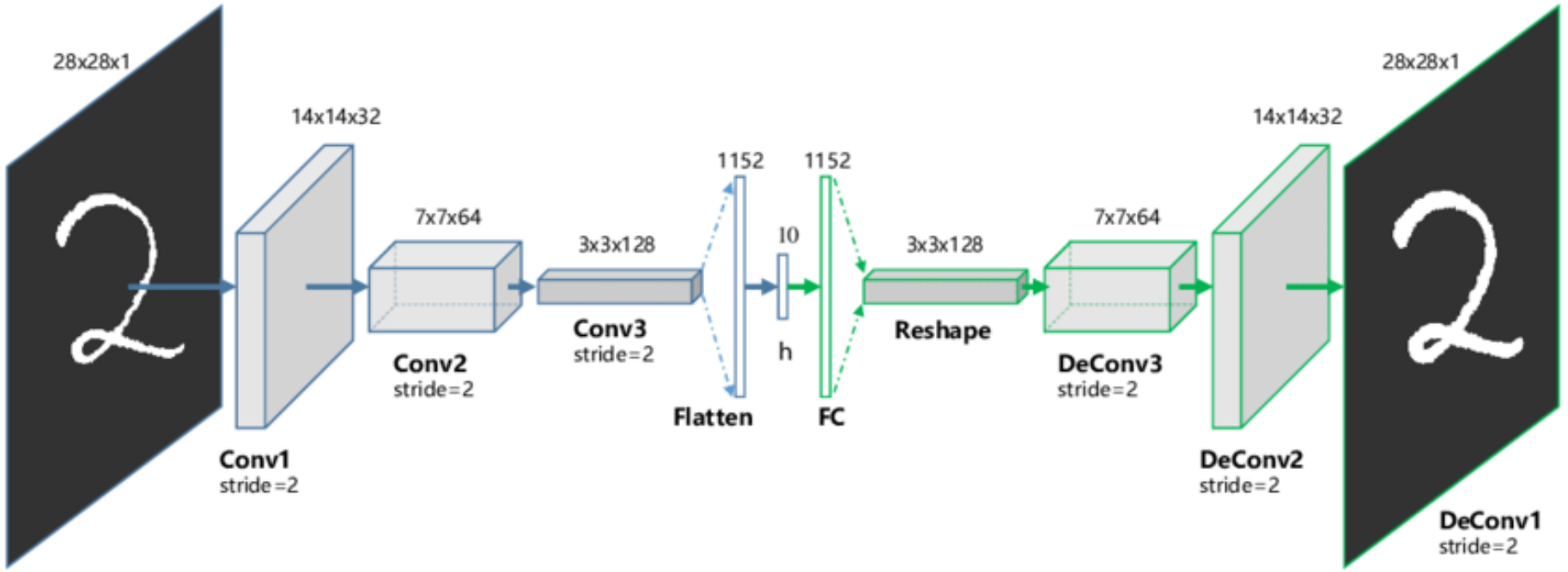
Model Structure (2)

- Types of Autoencoder model:
  - Vanilla autoencoder
  - Multilayer autoencoder
  - **Convolutional autoencoder**
  - Regularized autoencoder
  - **Denoising autoencoder**

```
x = Input(shape=(28, 28,1))


# Encoder
conv1_1 = Conv2D(16, (3, 3), activation='relu', padding='same')(x)

pool1 = MaxPooling2D((2, 2), padding='same')(conv1_1)
conv1_2 = Conv2D(8, (3, 3), activation='relu', padding='same')(pool1)

pool2 = MaxPooling2D((2, 2), padding='same')(conv1_2)
conv1_3 = Conv2D(8, (3, 3), activation='relu', padding='same')(pool2)

h = MaxPooling2D((2, 2), padding='same')(conv1_3)


# Decoder
conv2_1 = Conv2D(8, (3, 3), activation='relu', padding='same')(h)

up1 = UpSampling2D((2, 2))(conv2_1)
conv2_2 = Conv2D(8, (3, 3), activation='relu', padding='same')(up1)

up2 = UpSampling2D((2, 2))(conv2_2)
conv2_3 = Conv2D(16, (3, 3), activation='relu')(up2)
up3 = UpSampling2D((2, 2))(conv2_3)
r = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up3)


autoencoder = Model(input=x, output=r)
autoencoder.compile(optimizer='adam', loss='mse')
```

Model Structure (2-1)

❖ Convolutional autoencoder



Actual Practices and Applications (1)

❖ Applications of Autoencoders

- **Dimensionality reduction**
  - i.e., you can compare with PCA but more powerful/intelligent, **data compression especially for video and image data compression**.
- **Denoising**
  - removing noise and preprocessing images to improve OCR accuracy.
- **Anomaly/outlier detection**
  - detecting mislabeled data points in a dataset or detecting when an input data point falls well outside our typical data distribution.

❖ Default AutoEncoder

- Project structure

```
(BGKim) C:₩Users₩vicl₩practices₩cnn₩Autoencoders_Keras-tf2.0>tree /f
폴더 PATH의 목록입니다.
볼륨 일련 번호는 5417-ADDA입니다.
C:.
    output.png
    plot.png
    train_conv_autoencoder.py

└─pyimagesearch
        convautoencoder.py
        __init__.py


(BGKim) C:₩Users₩vicl₩practices₩cnn₩Autoencoders_Keras-tf2.0>
```

Actual Practices and Applications (3)

- Insight Program Sources: convautoencoder.py

```
# import the necessary packages
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Conv2DTranspose
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Reshape
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
import numpy as np

class ConvAutoencoder:
	@staticmethod
	def build(width, height, depth, filters=(32, 64),
	latentDim=16):
		# initialize the input shape to be "channels
		last" along with
		# the channels dimension itself
		# channels dimension itself
		inputShape = (height, width, depth)
		chanDim = -1

		# define the input to the encoder
		inputs = Input(shape=inputShape)
		x = inputs
		~~~~~~~
```

```
		# loop over the number of filters
		for f in filters:
			# apply a CONV => RELU => BN operation
			x = Conv2D(f, (3, 3), strides=2, padding="same")(x)
			x = LeakyReLU(alpha=0.2)(x)
			x = BatchNormalization(axis=chanDim)(x)

		# flatten the network and then construct our latent vector
		volumeSize = K.int_shape(x)
		x = Flatten()(x)
		latent = Dense(latentDim)(x)

		# build the encoder model
		encoder = Model(inputs, latent, name="encoder")

		# start building the decoder model which will accept the
		# output of the encoder as its inputs
		latentInputs = Input(shape=(latentDim,))
		x = Dense(np.prod(volumeSize[1:]))(latentInputs)
		x = Reshape((volumeSize[1], volumeSize[2], volumeSize[3]))(x)

		# loop over our number of filters again, but this time in
		# reverse order
		for f in filters[::-1]:
			# apply a CONV_TRANSPOSE => RELU => BN operation
			x = Conv2DTranspose(f, (3, 3), strides=2,
			padding="same")(x)
			x = LeakyReLU(alpha=0.2)(x)
			x = BatchNormalization(axis=chanDim)(x)

		# apply a single CONV_TRANSPOSE layer used to recover the
		# original depth of the image
		x = Conv2DTranspose(depth, (3, 3), padding="same")(x)
		outputs = Activation("sigmoid")(x)

		# build the decoder model
		decoder = Model(latentInputs, outputs, name="decoder")

		# our autoencoder is the encoder + decoder
		autoencoder = Model(inputs, decoder(encoder(inputs)),
				name="autoencoder")

		# return a 3-tuple of the encoder, decoder, and autoencoder
		return (encoder, decoder, autoencoder)
```

Actual Practices and Applications (4)

train_conv_autoencoder.py

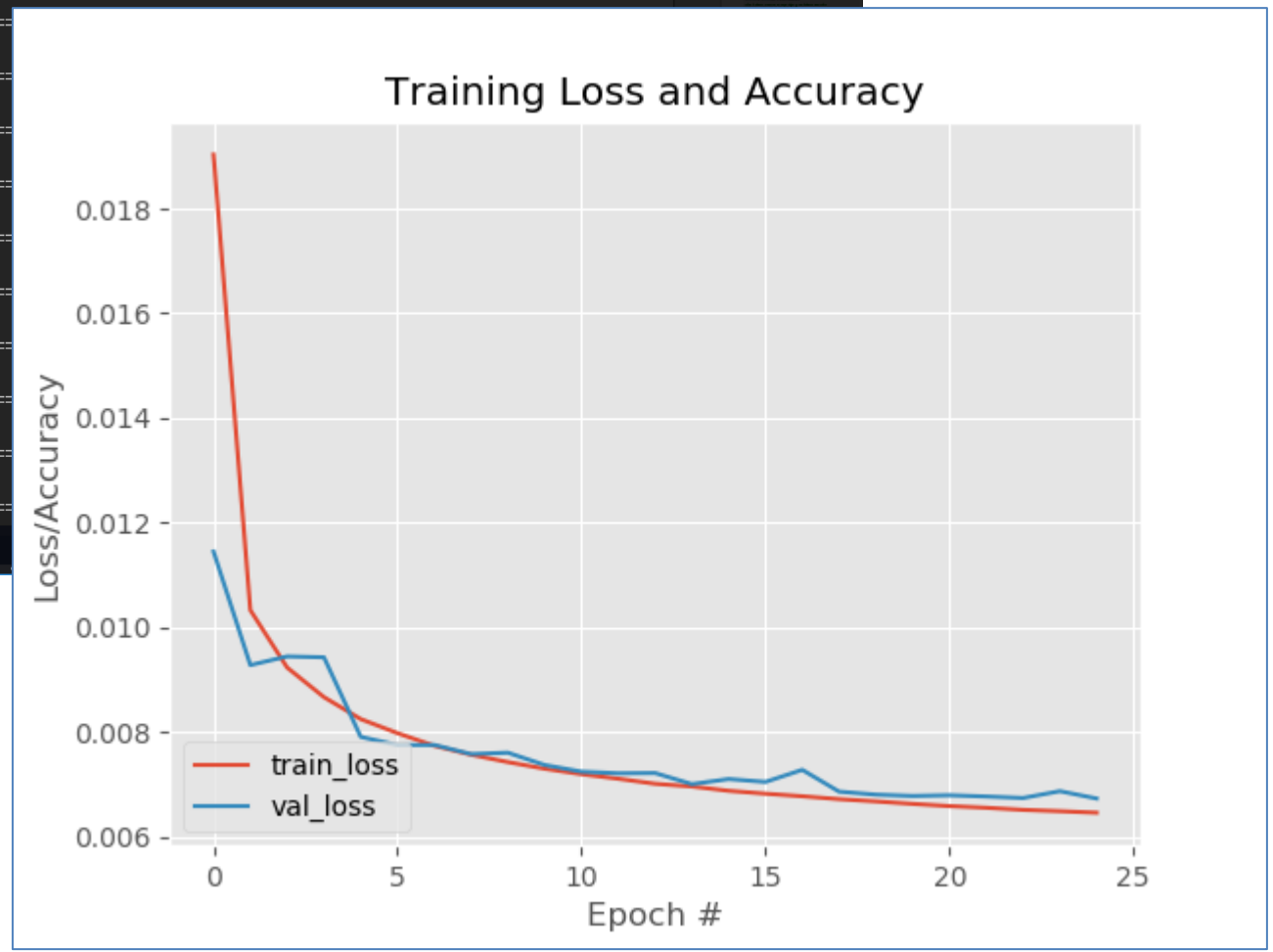
```
# set the matplotlib backend so figures can be saved in the background
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import matplotlib
matplotlib.use("Agg")

# import the necessary packages
from pyimagesearch.convautoencoder import ConvAutoencoder
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-s", "--samples", type=int, default=8,
help="# number of samples to visualize when decoding")
ap.add_argument("-o", "--output", type=str, default="output1.png",
help="path to output visualization file")
ap.add_argument("-p", "--plot", type=str, default="plot1.png",
help="path to output plot file")
args = vars(ap.parse_args())

# initialize the number of epochs to train for and batch size
EPOCHS = 25
BS = 32

# load the MNIST dataset
print("[INFO] loading MNIST dataset...")
((trainX, _), (testX, _)) = mnist.load_data()

# add a channel dimension to every image in the dataset, then scale
# the pixel intensities to the range [0, 1]
trainX = np.expand_dims(trainX, axis=-1)
testX = np.expand_dims(testX, axis=-1)
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0
```

```
# construct our convolutional autoencoder
print("[INFO] building autoencoder...")
(encoder, decoder, autoencoder) = ConvAutoencoder.build(28, 28, 1)
opt = Adam(lr=1e-3)
autoencoder.compile(loss="mse", optimizer=opt)

# train the convolutional autoencoder
H = autoencoder.fit(
trainX, trainX,
validation_data=(testX, testX),
epochs=EPOCHS,
batch_size=BS)

# construct a plot that plots and saves the training history
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])

# use the convolutional autoencoder to make predictions on the
# testing images, then initialize our list of output images
print("[INFO] making predictions...")
decoded = autoencoder.predict(testX)
outputs = None

# loop over our number of output samples
for i in range(0, args["samples"]):
        # grab the original image and reconstructed image
        original = (testX[i] * 255).astype("uint8")
        recon = (decoded[i] * 255).astype("uint8")

        # stack the original and reconstructed image side-by-side
        output = np.hstack([original, recon])

        # if the outputs array is empty, initialize it as the current
        # side-by-side image display
        if outputs is None:
                outputs = output

        # otherwise, vertically stack the outputs
        else:
                outputs = np.vstack([outputs, output])

# save the outputs image to disk
```

Actual Practices and Applications (4)

- Check on your results...!!!

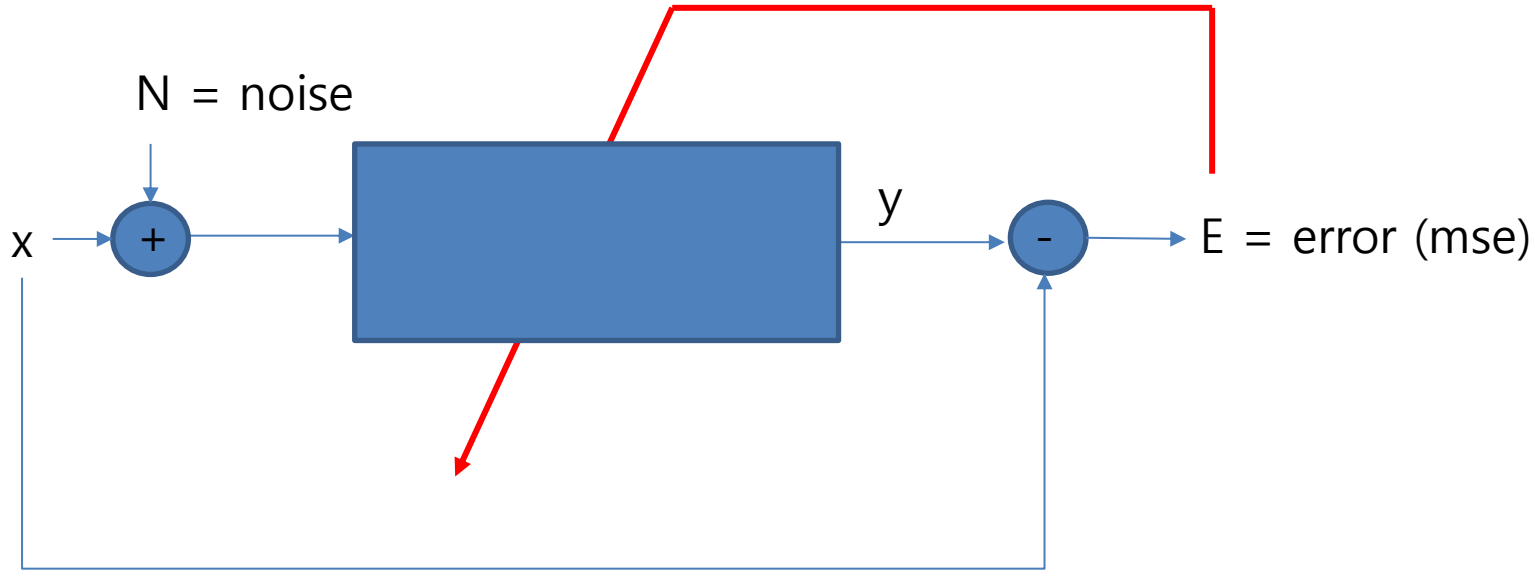
```
[INFO] loading MNIST dataset...
[INFO] building autoencoder...
Train on 60000 samples, validate on 10000 samples
Epoch 1/25
60000/60000 [==============================] - 21s 346us/sample - loss: 0.0190 - val_loss: 0.0114
Epoch 2/25
60000/60000 [==============================] - 19s 323us/sample - loss: 0.0103 - val_loss: 0.0093
Epoch 3/25
60000/60000 [==============================
Epoch 4/25
60000/60000 [==============================
Epoch 5/25
60000/60000 [==============================
Epoch 6/25
60000/60000 [==============================
Epoch 7/25
60000/60000 [==============================
Epoch 8/25
60000/60000 [==============================
Epoch 9/25
60000/60000 [==============================
Epoch 10/25
60000/60000 [==============================
Epoch 11/25
60000/60000 [==============================
Epoch 12/25
60000/60000 [==============================
Epoch 13/25
 4512/60000 [=>............................
```



Actual Practices and Applications (5)

❖ Denoising Autoencoder

- Training:
  - What kind of noise???



- Testing:
  - Input is the noised (corrupted) image, then we want to get an almost original image.

Actual Practices and Applications (6)

- Example of Source Code (1)

```
#Denoising Autoencoder : TF 2.0
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

# MNIST 로딩(라벨은 필요없기 때문에 버림)
(x_train, _), (x_test, _) = mnist.load_data()

# 데이터 정규화 및 Reshape
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 784))
x_test = np.reshape(x_test, (len(x_test), 784))

# 원본데이터에 Noise 추가
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc = 0.0, scale = 1.0, size = x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc = 0.0, scale = 1.0, size = x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Actual Practices and Applications (7)

- Example of Source Code (2)

```
# Noise가 추가된 데이터 확인
n = 10
plt.figure(figsize=(20, 2))
for i in range(1,n):
    ax = plt.subplot(1, n, i)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

# 모형 구성
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=784))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(784, activation='sigmoid'))
model.compile(optimizer='adam',
loss='binary_crossentropy')
```

```
# 모형 학습
H=model.fit(x_train_noisy, x_train,
          nb_epoch=100,
          batch_size=256,
          shuffle=True,
          validation_data=(x_test_noisy, x_test))

#학습 결과 plot1
# construct a plot that plots and saves the training
history
EPOCHS = 100
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure()
plt.plot(N, H.history["loss"], label="train_loss")
plt.plot(N, H.history["val_loss"], label="val_loss")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("training_loss_graph.png")
```

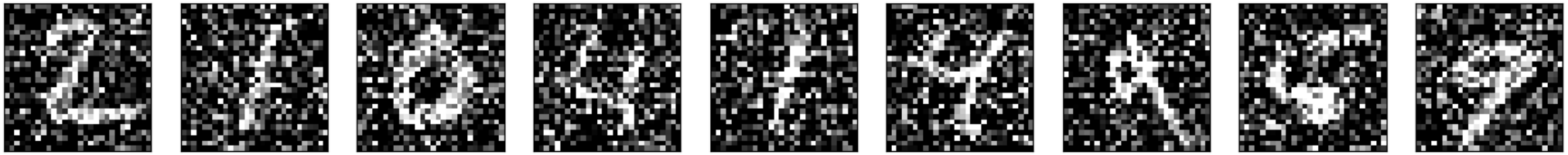
Actual Practices and Applications (8)

- Example of Source Code (3)

```
# 결과 확인
decoded_imgs = model.predict(x_test)
n = 10
plt.figure(figsize=(20, 6))
for i in range(1, n):
    # display original
    ax = plt.subplot(3, n, i)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display noisy
    ax = plt.subplot(3, n, i + n)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(3, n, i + 2*n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

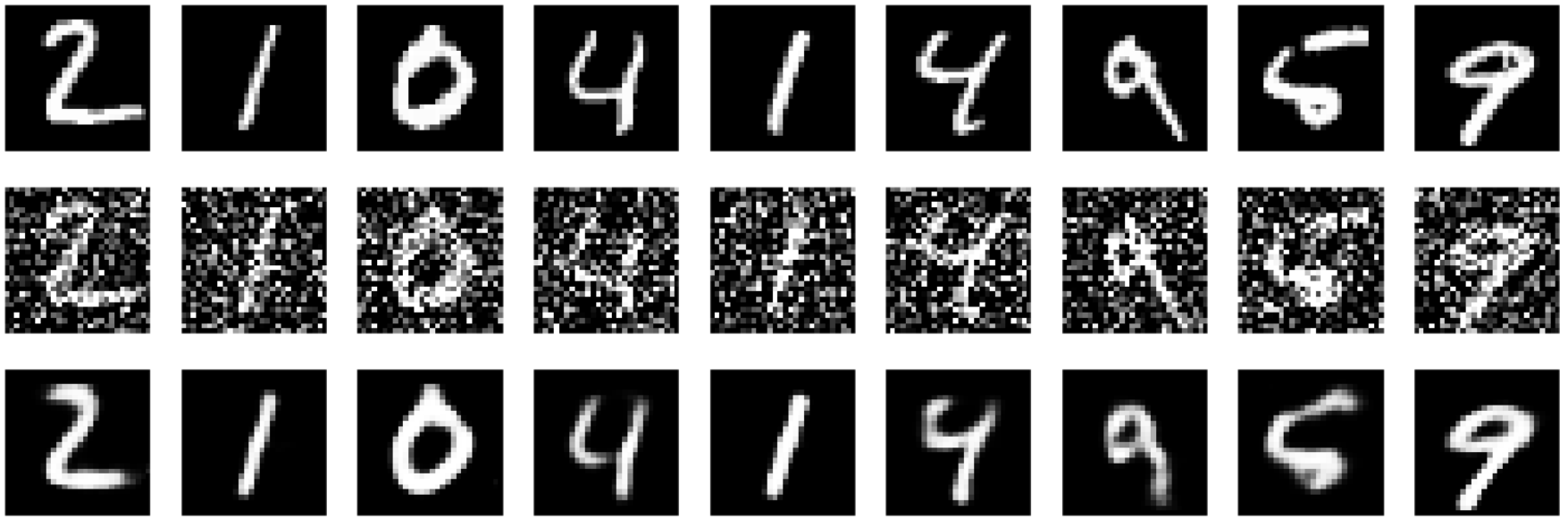
Actual Practices and Applications (9)

- Run the program…!!!
  1) Noise images



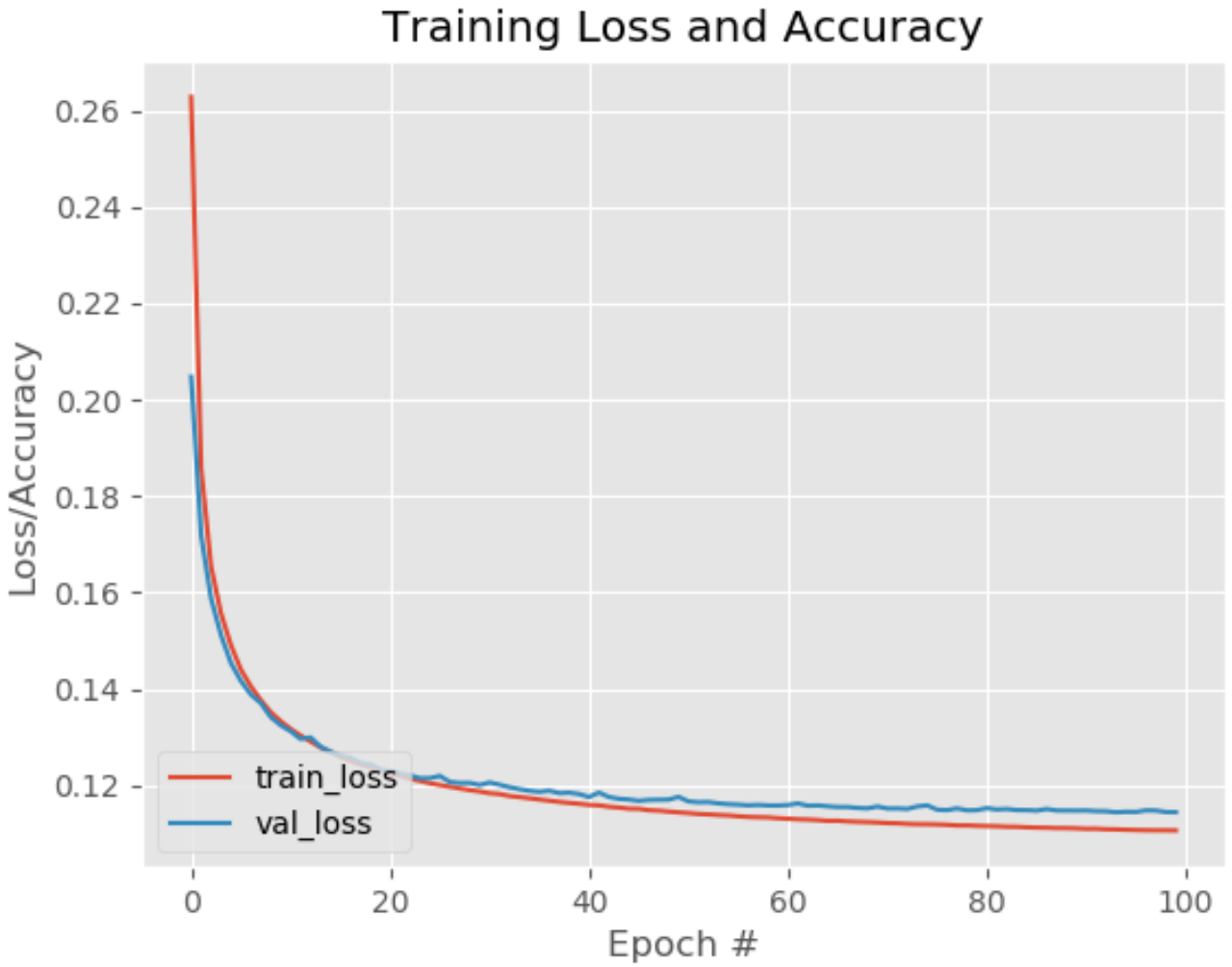
  2) Training process

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/100
60000/60000 [==============================] - 1s 25us/sample - loss: 0.2628 - val_loss: 0.2048
Epoch 2/100
60000/60000 [==============================
Epoch 3/100
60000/60000 [==============================
Epoch 4/100
60000/60000 [==============================
Epoch 5/100
60000/60000 [==============================
Epoch 6/100
60000/60000 [==============================
Epoch 7/100
60000/60000 [==============================
Epoch 8/100
60000/60000 [==============================
Epoch 9/100
29184/60000 [==========>
```



Actual Practices and Applications (10)

- Training Graphs



[Check Points]

❖ So many noise filtering approaches. For example,

- Average filtering
- Gaussian filtering
- Median filtering
- ……

You need to compare the performance with autoencoder in terms of the image quality (PSNR) !!!!

과제#3 - 학기 과제 발표 및 최종보고서 제출

❖ 학기과제 결과발표회
- 일시: 7월 1일 (수) 오후 4시 (조금 당길까요??)
- 형식: 온라인 발표

❖ 최종보고서 제출
- 형식: 영문논문 형식 (양식은 강의자료실에 업로드해 놓았음)
- 보고서 == 논문 (저널 논문 형식임)
- 제출물: 최종보고서(논문)+최종발표자료
- 기　한: 7월 6일 오전11시 59분까지
- 제출 방식: 이메일로 압축하여 제출할 것 (bg.kim@sookmyung.ac.kr)
- 평가 가점
  - **영어 논문(보고서) 작성 시 가점** (한국어로 작성도 가능함)
  - "**서론->문제 제기->solution->결과 분석**" 의 내용이 논리적으로 잘 구성되었는지 판단함

Thank you for your attention.!!!
QnA

http://ivpl.sookmyung.ac.kr