




10주차 비/대면- autoencoder

모두의 딥러닝 개정 3판: 4 이미지의 특징을 추출하는 오토인코더
 딥러닝을 이용해 가상의 이미지를 만드는 또 하나의 유명한 알고리즘이 있습니다. 바로 오토인코더 (Auto-Encoder, AE)입니다. 지금까지 설명한 GAN을 이해했다면 오토인코더의 핵심적인 부분은 이미 거의 이해한 셈입니다

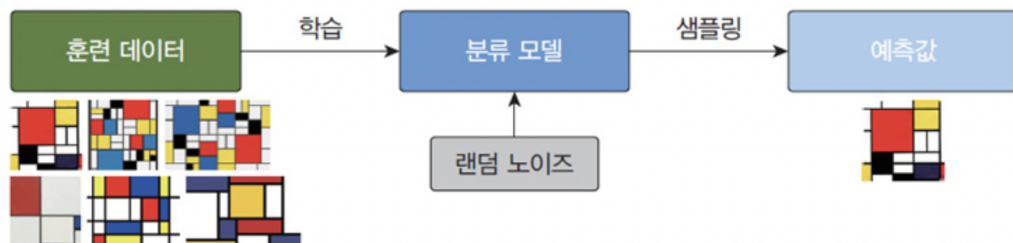
 <https://thebook.io/080324/part05/ch19/04/>



오토인코더는 GAN과 비슷한 결과를 만들지만, 다른 성질을 지니고 있습니다. GAN이 세상에 존재하지 않는 완전한 가상의 것을 만들어 내는 반면에, 오토인코더는 입력 데이터의 특징을 효율적으로 담아낸 이미지

생성모델 : autoencoder , GAN

- 생성모델 : 주어진 학습 데이터를 학습하여 학습 데이터의 분포를 따르는 유사한 데이터 생성
- 훈련 데이터를 생성하는 규칙을 파악함
 - 훈련데이터 → 학습 → 분류모델 (랜덤 노이즈) → 샘플링 → 예측값



- 분류 모델 : 조건부확률 $p(y|x)$ → x 가 주어진 상태에서 레이블 y 는 무엇인가?
- 생성 모델 : 입력 데이터의 확률 분포 $p(x)$

Autoencoder

- autoencoder : 입력 = 출력 “목표” 신경망
- 특징 학습, 차원 축소, 표현 학습 →
 - 차원축소 dimensionality reduction : PCA
 - 많은 피처로 구성된 다차원 데이터의 세트의 차원을 축소하여 새로운 차원의 데이터 세트를 생성함
 - 1. 피처 선택 : 불필요한 피처는 제거
 - 2. 피처 추출 : 기존 피처를 저차원의 중요 피처로 압축해서 추출
- Encoder : 인코더 - 입력을 잠재표현으로 압축
- Decoder 디코더 - 잠재 표현을 풀어서 입력을 복원
- 손실 함수 : 입력 이미지와 출력 이미지의 MSE 사용 : 픽셀간의 차이를 계산
- Mnist 데이터베이스 사용

```
(X_train,y_train), (X_test,y_test_) = mnist.load_data()
원래 X_train, X_test shape (60000,28,28)
X_train= X_train.reshape(60000,784) : 28 X 28 => Sequential => array형태로 input을 넣기
위함
X_test=X_test.reshape(60000,784)

X_train,X_test=X_train/255, X_test/255 #그림이미지 정규화

model=Sequential()
model.add(Dense(784,input_dim=28*28,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(784,activation='sigmoid')) #binary -> sigmoid인 이유 다중 분류 X : 색칠유무
( 0 or 1 )

model.compile(optimizer='adam', loss='mse')
history=model.fit(X_train,X_train,epochs=15,validation_data=(X_test,X_test))
즉, output이 X 가 나오도록 학습

그림이 나오도록

model.compile(optimizer='adam', loss='mse')
history=model.fit(X_train,X_train,epochs=15,validation_data=(X_test,X_test))
```

```

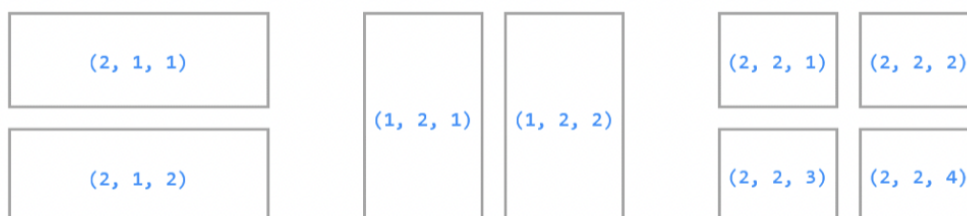
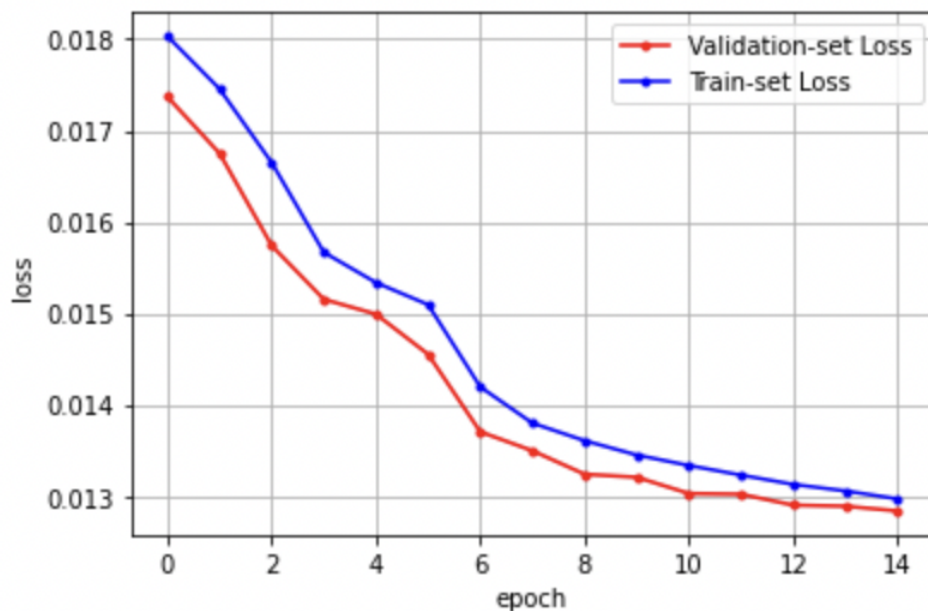
y_vloss=history.history['val_loss']
y_loss=history.history['loss']

x_len=np.arange(len(y_loss)) -> epoch

plt.plot(x_len,y_vloss,marker='.',c='red',label="Validation-set Loss")
plt.plot(x_len,y_loss,marker='.',c='blue',label="Train-set Loss")

plt.legend(loc="upper right")
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()

```



`plt.subplot(row, column, index)`

plt.subplot (row, column, index)

```

autoencoder_imgs=model.predict(X_test)

n=10
plt.figure(figsize=(20,4)) #그래프 크기 조절하기
for i in range(n):
    ax=plt.subplot(2,n,i+1)
    plt.imshow(X_test[i].reshape(28,28)) #flatten했던 데이터 되돌리고 X_test 출력
    plt.gray() #흑백으로 출력

    ax=plt.subplot(2,n,i+1+n)
    plt.imshow(autoencoder_imgs[i].reshape(28,28))
    #flatten했던 데이터 되돌리고 X_test 출력
    plt.gray()

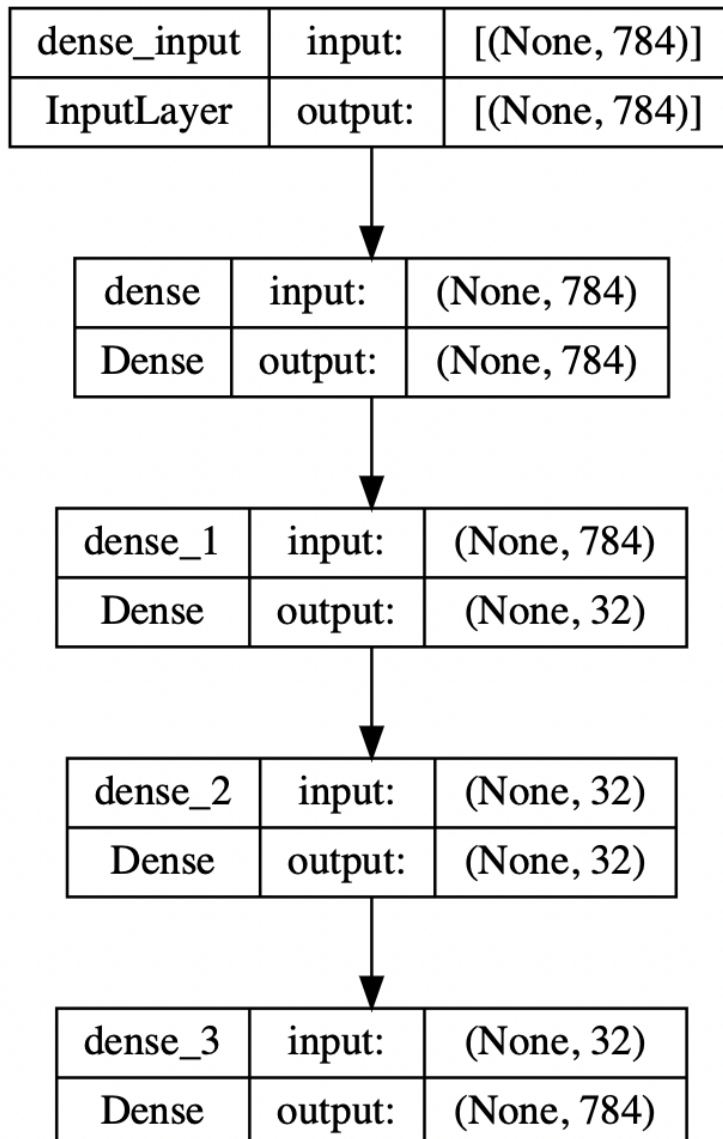
plt.show()

!pip install graphviz
!pip install pydot

from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(model, show_shape=True, dpi=70).create(prog
'dot', format='svg'))

```



02.auto fashion mnist

```
(x_train, _), (x_test, y_test) = fashion_mnist.load_data()

x_train = x_train.reshape(60000, 784).astype('float32') / 255
x_test = x_test.reshape(10000, 784).astype('float32') / 255 #정규화 0~1 /127.5 -1~1

scaling시 최댓값인 255사용함
```

Normalization vs Standardization

두 feature scaling 방식의 차이점을 정리해보았다.

Normalization	Standardization
스케일링 시 최대, 최소값이 사용된다	스케일링 시 평균과 표준편차가 사용된다
피쳐의 크기가 다를 때 사용한다	평균이 0, 표준편차가 1인 것을 확인하고 싶을 때 (그렇게 만들고 싶을 때) 사용한다
[0,1] (또는 [-1,1]) 사이의 값으로 스케일링	특정 범위로 제한되지 않는다
분포에 대해 모를 때 유용하다	피쳐가 정규분포(가우시안 분포)인 경우 유용하다
MinMaxScaler, Normalizer	StandardScaler, RobustScaler

```
autoencoder=Sequential([
    Dense(784,input_shape=(784,)),
    Activation('relu'),
    Dense(32),
    Activation('relu'),
    Dense(32),
    Activation('relu'),
    Dense(784),
    Activation('sigmoid')
])

# autoencoder = Sequential([
#     Dense(784, input_shape=(784,), activation = 'relu'),
#     Dense(32, activation = 'relu'),
#     Dense(32, activation = 'relu'),
#     Dense(784, activation = 'sigmoid')
# ])

#compile => autoencoder - mse사용
autoencoder.compile(compile(optimizer='adam', loss='mse') -? #옵티마이저, 손실함수

#train
history=autoencoder.fit(x_train, x_train,
epochs=50,batch_size=256,shuffle=True,validation_data=(x_test,x_test))

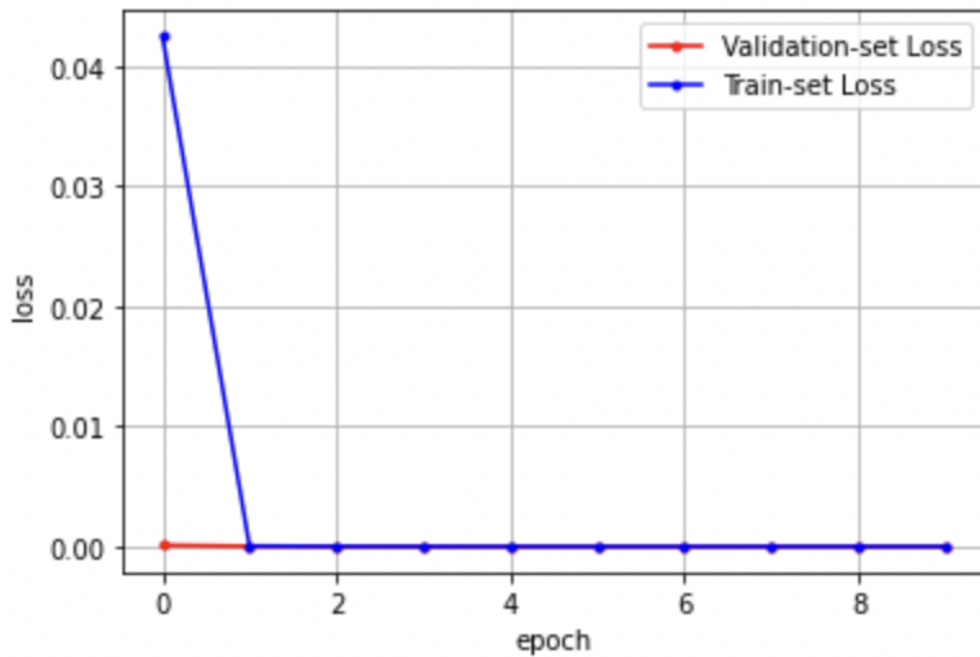
#shuffle : model.fit 의 인수 shuffle 을 (기본값인) True 로 설정하면, 훈련 데이터는 매 epoch마다
무작위로 섞이게 됨
```

```
y_vloss=history.history['val_loss']
y_loss=history.history['loss']

x_len=np.arange(len(y_loss))
```

```
plt.plot(x_len,y_vloss,marker='.',c='red',label="Validation_set Loss")
plt.plot(x_len,y_loss, marker='.',c='blue',label="Train set Loss")

plt.legend(loc="upper right")
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



```
import matplotlib.pyplot as plt

decoded_imgs=autoencoder.predict(x_test)

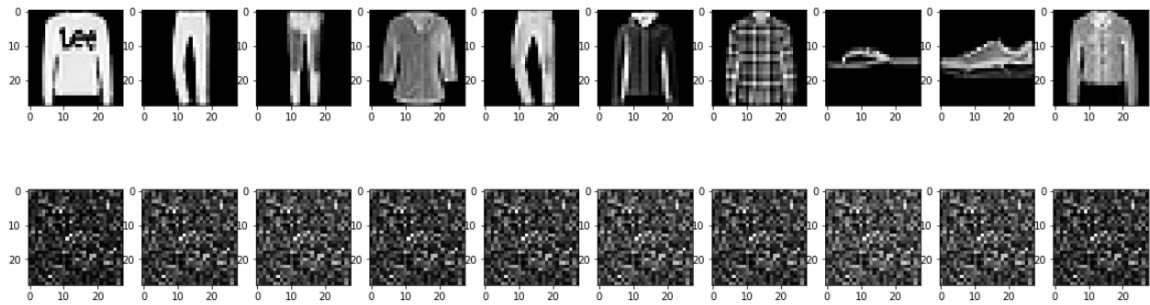
plt.figure(figsize=(20,6))

for i in range(1,11):
    ax=plt.subplot(2,10,i) #순서대로 채워지는 쪽으로 넣는 것임 -> 10개씩 2개 -> 20개 넣겠다
    plt.imshow(x_test[i].reshape(28,28), cmap='gray') #cmap='gray' 흑백 이미지

    ax=plt.subplot(2,10,i+10)
    plt.imshow(decoded_imgs[i].reshape(28,28), cmap='gray') #학습이 잘 안된상태 -> 어떻게 극복해
    야하는가

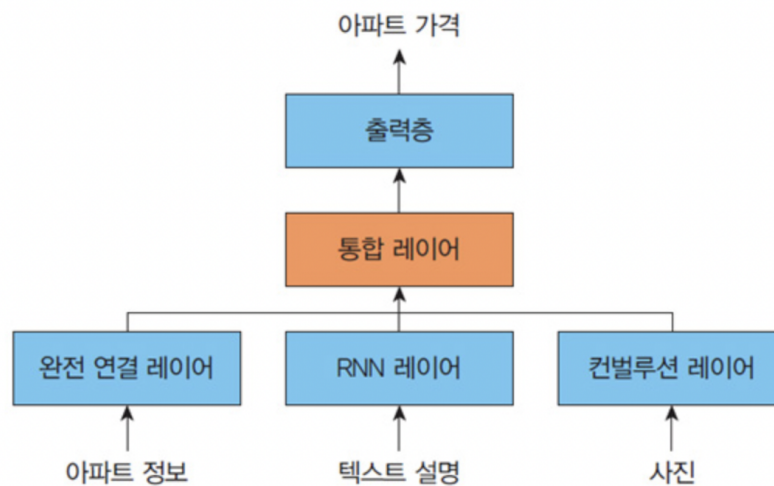
plt.show()
```

313/313 [=====] - 1s 3ms/step



→ project → 복원하여 조합 <https://silvercoding.tistory.com/9>

03. autoencoder 함수형 API 모델 사용



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras import layers
import numpy as np

(X_train,y_train),(X_test,y_test)=mnist.load_data()

#정규화
X_train=X_train.reshape(60000,784).astype('float32')/255
X_test=X_test.reshape(60000,784).astype('float32')/255

#latent - 중간 점검;;
```


어떤 차이가 있는가

```
autoencoder=Sequential([
    Dense(784,input_shape=(784,)),
    Actiavtion('relu'),
    Dense(32),
    Activation('relu'),
    Dense(32),
    Activation('relu'),
    Dense(784),
    Activation('sigmoid')
])

inputs=keras.Input(shape=(784,))
encoded=layers.Dense(32,activation='relu')(inputs)
latent_space=layers.Dense(36,activation='relu')(encoded)
decoded=layers.Dense(784,activation='sigmoid')(latent_space)
#중간점검할 수 있는 모델 중간에서 뺄 수 있음
latent_model=Model(inputs=inputs,outputs=latent_space)

autoencoder=Model(inputs=inputs,outputs=decoded)

autoencoder.compile(optimizer=keras.optimizers.Adam(), loss=keras.losses.MeanSquaredError()) #아래와 동일한 형태임
autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.fit(X_train,X_train,batch_size=256,epochs=10,verbose=2,shuffle=True)
```

Generative Images → predict

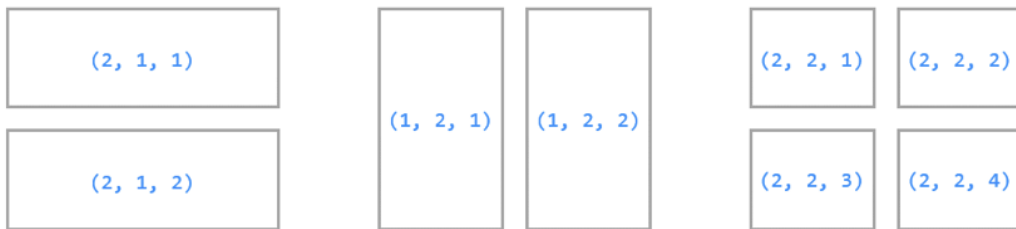
```
latent_imgs=latent_model.predict(X_test)
decoded_imgs=autoencoder.predict(X_test)

n=10
plt.figure(figsize=(20,6))
for i in range(n+1):
    ax=plt.subplot(3,n,i)
    plt.imshow(X_test[i].reshape(28,28),cmap='gray')

    ax=plt.subplot(3,n,i+1)
    plt.imshow(latent_imgs[i].reshape(28,28),cmap='gray')

    ax=plt.subplot(3,n,i+2)
    plt.imshow(decoded_imgs[i].reshape(28,28),cmap='gray')

plt.show()
```



`plt.subplot(row, column, index)`

subplot: 축 공유하기

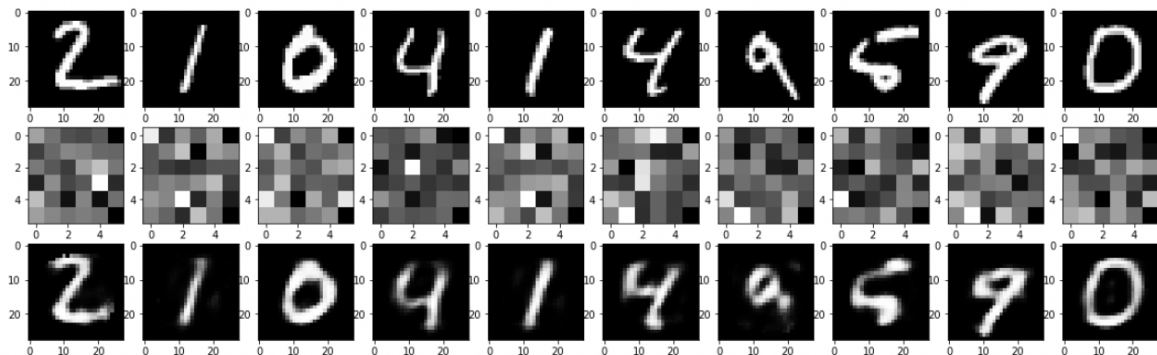
`plt.subplot(1, 3, 1)` row. column. index

`plt.subplot(1, 3, 2)`



313/313 [=====] - 0s 791us/step

313/313 [=====] - 0s 870us/step



04 노이즈 제거

오토인코더는 노이즈(noise)가 있는 이미지에서 노이즈를 제거하는 용도로도 사용할 수 있다.



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras import layers
import numpy as np

(X_train,y_train),(X_test,y_test)= mnist.load_data()

X_train =X_train.reshape(60000,784).astype('float32')/255
X_test=X_test.reshape(10000,784).astype('float32')/255
```

- ***np.random.normal(loc=0.0, scale=1.0, size=None)****
 - Mean of the distribution
 - Standard deviation of the distribution

```
noise_factor=0.55
original_train=X_train
original_test=X_test
noise_train=np.random.normal(0,1,original_train.shape) #train data shape만큼의 데이터 생성
noise_test=np.random.normal(0,1, original_test.shape)

noise_train=original_train+noist_rain
noisy_train=orginal_train+noise_fator *noise_train
noisy_test=original_test+noise_factor*noisy_test

import matplotlib.pyplot as plt

plt.imshow(noisy_train[1].reshape(28,28),cmap='gray')
```

```
inputs=keras.Input(shape=(784,))
encoded=layers.Dense(32,activation='relu')(inputs)
decoded=layers.Dense(784,activation='sigmoid')(encoded)
autoencoder=Model(inputs=inputs,outputs=decoded)
```

```
autoencoder.compile(optimizer=keras.optimizers.Adam(), loss='mse')
autoencoder.fit(noisy_train,original_train,epochs=50,batch_size=256,suffle=True)
```

```
import matplotlib.pyplot as plt

denoised_imgs=autoencoder.predict(noisy_test)

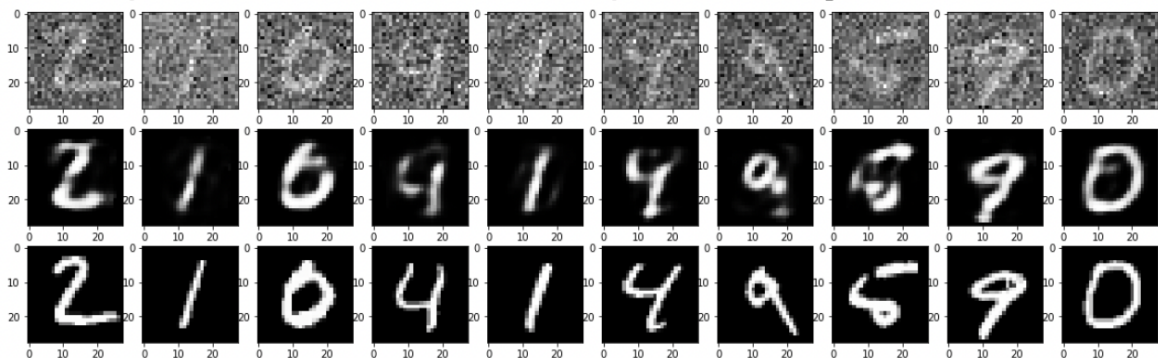
n=10
plt.figure(figsize=(20,6))
for i in range(1, n+1):
    ax=plt.subplot(3,n,i)
    plt.imshow(denoised_imgs[i].reshape(28,28),cmap='gray')

    ax=plt.subplot(3,n,i+n)
    plt.imshow(denoised_imgs[i].reshape(28,28),cmap='gray')

    ax=plt.subplot(3,n,i+n*2)
    plt.imshow(original_test[i].reshape(28,28),cmap='gray')

plt.show()
```

313/313 [=====] - 1s 2ms/step



05 대면수업

```

from tensorflow.keras.datasets import cifar10

(X_train,_),(X_test,_)=cifar10.load_data()
plt.imshow(X_train[1])
X_train.shape
#width=32,height=32,rgb=3
#결과값 (50000, 32, 32, 3) 50000개의 이미지

L,W,H,C =X_test.shape

X_train.reshape(-1,W*H*C).dtype - uint8 & flatten &정규화 필요 왜?CNN이 아님
CNN은 Width , height, channel 바로 사용할 수 있다 = 시세포 역할
( inputs data - flatten 필요기존 방식(autoencoder 참고)과의 차이점)

X_train=X_train.reshape(-1,W*H*C).astype('float32')/255
X_test=X_test.reshape(-1,W*H*C).astype('float32')/255

X_test.shape
(10000,3072)

```

```

inputs=keras.Input(shape=(3072,))
encoded=layers.Dense(32,activation='relu')(inputs)
latent=layers.Dense(32,activation='relu')(encoded)
decoded=layers.Dense(3072,activation='sigmoid')(latent)

autoencoder=Model(inputs=inputs,outputs=decoded)

```

```

autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(X_train,X_train,epochs=10,shuffle=True,validation_data=(X_test,X_test))

decoded_img=autoencoder.predict(X_test)
n=10
plt.figure(figsize=(20,6))
for i in range(1,n+1):
    ax=plt.subplot(2,n,i)
    plt.imshow(X_test[i].reshape(32,32,3))
    ax=plt.subplot(2,n,i+n)
    plt.imshow(decoded_img[i].reshape(32,32,3))
color images -> channel 3개 필요함

plt.show()

```

#데이터가 많아져야함 => 복잡한 이미지 데이터 수가 몇배 더더 많아야해

313/313 [=====] - 1s 4ms/step

