Autoencoders

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.metrics import accuracy_score, precision_score, recall_score

from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.models import Model
```

```
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
```

```
print (x_train.shape)
```

```
(60000, 28, 28)
```

```
print(x_test.shape)
```

```
(10000, 28, 28)
```

```
## normalizing
x_train =x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

```
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

```
latent_dim = 64

class Autoencoder(Model):
  def __init__(self, latent_dim) :
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([layers.Flatten(),
                                        layers.Dense(latent_dim,activation='relu'),])

    self.decoder = tf.keras.Sequential([layers.Dense(784, activation = 'sigmoid'),
                                        layers.Reshape((28,28))
                        ])
  def call(self, x):
    encod
```

```
## adam optimizer
autoencoder.fit(x_train, x_train,
                epochs=20,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 sequential (Sequential)     (None, 64)                50240

 sequential_1 (Sequential)   (None, 28, 28)            50960

=================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
_________________________________________________________________
```

```
encoded_imgs = autoencoder.encoder(x_test).numpy()
encoded_imgs
```

```
array([[0.        , 0.        , 0.        , ..., 0.62209547, 0.        ,
        0.6131118 ],
       [0.        , 0.        , 0.        , ..., 0.6675215 , 0.        ,
        0.6515752 ],
       [0.        , 0.        , 0.        , ..., 0.5867858 , 0.        ,
        0.57612276],
       ...,
       [0.        , 0.        , 0.        , ..., 0.6840693 , 0.        ,
        0.6743623 ],
       [0.        , 0.        , 0.        , ..., 0.6630395 , 0.        ,
        0.65383035],
       [0.        , 0.        , 0.        , ..., 0.72586584, 0.        ,
        0.7154327 ]], dtype=float32)
```

```
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

```
decoded_imgs
```

```
array([[[1.4959258e-04, 1.5144574e-04, 1.5347135e-04, ...,
         1.5121271e-04, 1.5217328e-04, 1.5511039e-04],
        [1.5353062e-04, 1.4961739e-04, 1.4935156e-04, ...,
         1.5367796e-04, 1.5126519e-04, 1.4799263e-04],
        [1.5269041e-04, 1.5259493e-04, 1.5048472e-04, ...,
         1.5277008e-04, 1.5342979e-04, 1.5180140e-04],
        ...,
        [1.5130731e-04, 1.5199925e-04, 1.5312112e-04, ...,
         1.5068891e-04, 1.5349008e-04, 1.5187524e-04],
        [1.5486006e-04, 1.5152953e-04, 1.5327918e-04, ...,
         1.5007290e-04, 1.5056840e-04, 1.5359023e-04],
        [1.5523765e-04, 1.5315703e-04, 1.5064022e-04, ...,
         1.4986470e-04, 1.5262127e-04, 1.5021321e-04]],

       [[8.5934327e-05, 8.7177432e-05, 8.8341076e-05, ...,
         8.6877095e-05, 8.7550994e-05, 8.9272944e-05],
        [8.8326327e-05, 8.5918517e-05, 8.5693420e-05, ...,
         8.8451416e-05, 8.7052576e-05, 8.4984204e-05],
        [8.7901259e-05, 8.7767155e-05, 8.6545770e-05, ...,
         8.8018009e-05, 8.8427711e-05, 8.7273256e-05],
        ...,
        [8.6954424e-05, 8.7327950e-05, 8.8039829e-05, ...,
         8.6581349e-05, 8.8310073e-05, 8.7335604e-05],
        [8.9220601e-05, 8.7158813e-05, 8.8265369e-05, ...,
         8.6044791e-05, 8.6530585e-05, 8.8322449e-05],
        [8.9380272e-05, 8.8221197e-05, 8.6547421e-05, ...,
         8.6162443e-05, 8.7943430e-05, 8.6303226e-05]],

       [[2.4555204e-04, 2.4853274e-04, 2.5154292e-04, ...,
         2.4805454e-04, 2.4954759e-04, 2.5401782e-04],
        [2.5181935e-04, 2.4581814e-04, 2.4542000e-04, ...,
         2.5184074e-04, 2.4811769e-04, 2.4323563e-04],
        [2.5031393e-04, 2.5019248e-04, 2.4699938e-04, ...,
         2.5063846e-04, 2.5146882e-04, 2.4892500e-04],
        ...,
        [2.4832785e-04, 2.4914509e-04, 2.5120907e-04, ...,
         2.4736562e-04, 2.5154292e-04, 2.4935801e-04],
        [2.5367920e-04, 2.4866904e-04, 2.5122511e-04, ...,
         2.4637821e-04, 2.4716239e-04, 2.5165832e-04],
        [2.5412100e-04, 2.5100267e-04, 2.4733614e-04, ...,
         2.4628825e-04, 2.5014573e-04, 2.4667836e-04]],

       ...,

       [[6.5140506e-05, 6.6073189e-05, 6.7094741e-05, ...,
         6.5907050e-05, 6.6455039e-05, 6.7849403e-05],
        [6.7014174e-05, 6.5166045e-05, 6.4927845e-05, ...,
         6.7163812e-05, 6.5973647e-05, 6.4407373e-05],
        [6.6655346e-05, 6.6563945e-05, 6.5557782e-05, ...,
         6.6684021e-05, 6.7096080e-05, 6.6204375e-05],
```

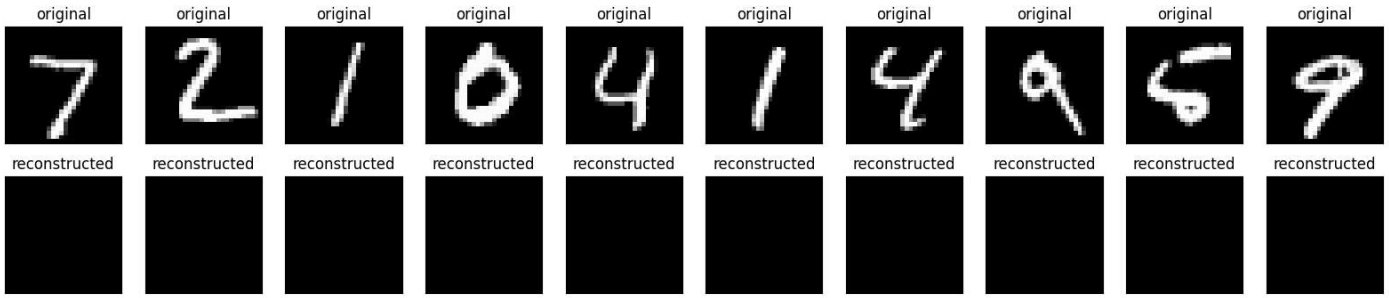
```
              ...,
              [6.5950495e-05, 6.6226479e-05, 6.6751716e-05, ...,
               6.5587301e-05, 6.7033150e-05, 6.6186323e-05],
              [6.7708817e-05, 6.6046792e-05, 6.6995002e-05, ...,
               6.5294371e-05, 6.5577791e-05, 6.7090070e-05],
              [6.7918343e-05, 6.6924316e-05, 6.5601249e-05, ...,
               6.5229520e-05, 6.6706474e-05, 6.5407417e-05]],
```

```
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.title("original")
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    #Display recontructed

    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i])
    plt.title("reconstructed")
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()
```




```
###    Adding a new column for the random noise

x_train = x_train[..., tf.newaxis]

x_test = x_test[..., tf.newaxis]

print(x_train.shape, x_test.shape)
```

```
     (60000, 28, 28, 1) (10000, 28, 28, 1)
```

```
#Adding random noise to the images

noise_factor = 0.2

x_train_noisy = x_train + noise_factor * tf.random.normal(shape=x_train.shape)

x_test_noisy = x_test + noise_factor * tf.random.normal(shape=x_test.shape)

x_train_noisy = tf.clip_by_value(x_train_noisy, clip_value_min=0., clip_value_max=1.)

x_test_noisy = tf.clip_by_value(x_test_noisy, clip_value_min=0., clip_value_max=1.)


#plotting the noisy image

n =10

plt.figure(figsize=(20,2))

for i in range(n):
  ax = plt.subplot(1, n, i + 1)
  plt.title("original + noise")
  plt.imshow(tf.squeeze(x_test_noisy[i]))
  plt.gray()

plt.show()
```