experiment-13

October 30, 2025

Name:	Tufan Kundu
Registration no:	24MDT0184
Course Name:	Deep Learning Lab
Course Code:	PMDS603P
Experiment:	13
Date:	30 October,2025

0.1 Question 1: Today, we will look at how we can implement autoencoders in some specific scenarios. First, we will look at the denoising autoencoder that can be used to perform some denoising tasks with respect to images. We will work with MNIST Images for the task.

0.1.1 Importing the necessary libraries

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.datasets import mnist
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
2025-10-30 07:28:50.609534: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
E0000 00:00:1761809330.818599      37 cuda_dnn.cc:8310] Unable to register cuDNN
factory: Attempting to register factory for plugin cuDNN when one has already
been registered
E0000 00:00:1761809330.894862      37 cuda_blas.cc:1418] Unable to register
cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has
already been registered
```

[2]:
```
(train_images,_),(test_images, _) = mnist.load_data()
train_images = train_images.astype('float32')/255.0
test_images = test_images.astype('float32')/255.0
train_images = np.reshape(train_images, (len(train_images), 28,28,1))
test_images = np.reshape(test_images, (len(test_images), 28,28,1))
print("Train Shape:", train_images.shape,"Test Shape:", test_images.shape)
```

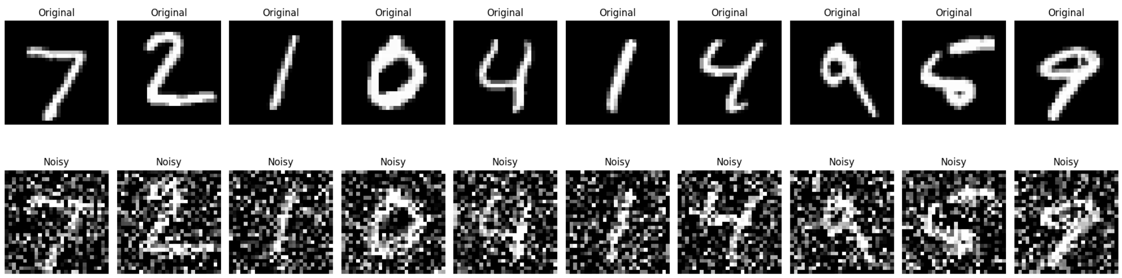
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434            0s
0us/step
Train Shape: (60000, 28, 28, 1) Test Shape: (10000, 28, 28, 1)
```

[3]:
```
train_images, val_images = train_test_split(train_images, test_size = 0.2,␣
 ↪random_state=42)
noise_factor = 0.5

## Adding noise to the data
train_noisy = train_images+noise_factor*np.random.normal(loc = 0.0, scale = 1.
 ↪0, size = train_images.shape)
val_noisy = val_images+noise_factor*np.random.normal(loc = 0.0, scale = 1.0,␣
 ↪size = val_images.shape)
test_noisy = test_images+noise_factor*np.random.normal(loc = 0.0, scale = 1.0,␣
 ↪size = test_images.shape)
train_noisy = np.clip(train_noisy, 0.,1.)
val_noisy = np.clip(val_noisy,0.,1.)
test_noisy = np.clip(test_noisy, 0.,1.)
print("Train noisy:", train_noisy.shape, "val noisy:", val_noisy.shape)
```

```
Train noisy: (48000, 28, 28, 1) val noisy: (12000, 28, 28, 1)
```

0.1.2 Plotting the original ans noisy images

[4]:
```
n = 10
plt.figure(figsize = (20,6))
for i in range(n):
    ax = plt.subplot(2,n,i+1)
    plt.imshow(test_images[i].reshape(28,28), cmap = 'gray')
    plt.title("Original")
    ax.axis("off")
    ax = plt.subplot(2,n,i+1+n)
    plt.imshow(test_noisy[i].reshape(28,28), cmap = "gray")
    plt.title("Noisy")
    ax.axis("off")

plt.tight_layout()
plt.show()
```



0.1.3 Building the model

[5]:
```python
model = Sequential()

## Encoder

model.add(Conv2D(32,(3,3), activation='relu', padding='same', input_shape =(28,28,1)))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))

## Decoder

model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(32,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(1,(3,3), activation = 'sigmoid', padding = 'same'))
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
I0000 00:00:1761809348.112068      37 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
I0000 00:00:1761809348.112753      37 gpu_device.cc:2022] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
```

[6]:
```python
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	36,928
up_sampling2d (UpSampling2D)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 14, 14, 32)	18,464
up_sampling2d_1 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_4 (Conv2D)	(None, 28, 28, 1)	289

Total params: 74,497 (291.00 KB)

Trainable params: 74,497 (291.00 KB)

Non-trainable params: 0 (0.00 B)

[7]:
```
model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics =␣
 ↪['accuracy'])
early_stopping = EarlyStopping(monitor='val_loss', patience = 3,␣
 ↪restore_best_weights=True, verbose = 1)
history = model.fit(train_noisy, train_images, epochs = 50, batch_size = 128,␣
 ↪shuffle = True, validation_data = (val_noisy,val_images), verbose = 1,␣
 ↪callbacks = [early_stopping])
```

```
Epoch 1/50
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1761809353.014337      99 service.cc:148] XLA service 0x7e3484004ec0
initialized for platform CUDA (this does not guarantee that XLA will be used).
Devices:
I0000 00:00:1761809353.015112      99 service.cc:156]   StreamExecutor device
(0): Tesla T4, Compute Capability 7.5
I0000 00:00:1761809353.015128      99 service.cc:156]   StreamExecutor device
```

```
(1): Tesla T4, Compute Capability 7.5
I0000 00:00:1761809353.356553      99 cuda_dnn.cc:529] Loaded cuDNN version
90300

 27/375              2s 6ms/step -
accuracy: 0.7101 - loss: 0.5446

I0000 00:00:1761809356.012641      99 device_compiler.h:188] Compiled cluster
using XLA!  This line is logged at most once for the lifetime of the process.

375/375              10s 11ms/step -
accuracy: 0.7943 - loss: 0.2430 - val_accuracy: 0.8122 - val_loss: 0.1186
Epoch 2/50
375/375              3s 7ms/step -
accuracy: 0.8108 - loss: 0.1144 - val_accuracy: 0.8123 - val_loss: 0.1082
Epoch 3/50
375/375              3s 7ms/step -
accuracy: 0.8115 - loss: 0.1077 - val_accuracy: 0.8129 - val_loss: 0.1048
Epoch 4/50
375/375              3s 7ms/step -
accuracy: 0.8123 - loss: 0.1045 - val_accuracy: 0.8121 - val_loss: 0.1035
Epoch 5/50
375/375              3s 7ms/step -
accuracy: 0.8126 - loss: 0.1023 - val_accuracy: 0.8129 - val_loss: 0.1013
Epoch 6/50
375/375              3s 7ms/step -
accuracy: 0.8123 - loss: 0.1009 - val_accuracy: 0.8127 - val_loss: 0.1006
Epoch 7/50
375/375              3s 7ms/step -
accuracy: 0.8132 - loss: 0.0997 - val_accuracy: 0.8136 - val_loss: 0.0995
Epoch 8/50
375/375              3s 7ms/step -
accuracy: 0.8130 - loss: 0.0988 - val_accuracy: 0.8132 - val_loss: 0.0985
Epoch 9/50
375/375              3s 7ms/step -
accuracy: 0.8134 - loss: 0.0979 - val_accuracy: 0.8132 - val_loss: 0.0980
Epoch 10/50
375/375              3s 7ms/step -
accuracy: 0.8135 - loss: 0.0973 - val_accuracy: 0.8136 - val_loss: 0.0973
Epoch 11/50
375/375              3s 7ms/step -
accuracy: 0.8134 - loss: 0.0967 - val_accuracy: 0.8135 - val_loss: 0.0970
Epoch 12/50
375/375              3s 7ms/step -
accuracy: 0.8137 - loss: 0.0963 - val_accuracy: 0.8135 - val_loss: 0.0964
Epoch 13/50
375/375              3s 7ms/step -
accuracy: 0.8131 - loss: 0.0960 - val_accuracy: 0.8136 - val_loss: 0.0961
Epoch 14/50
```

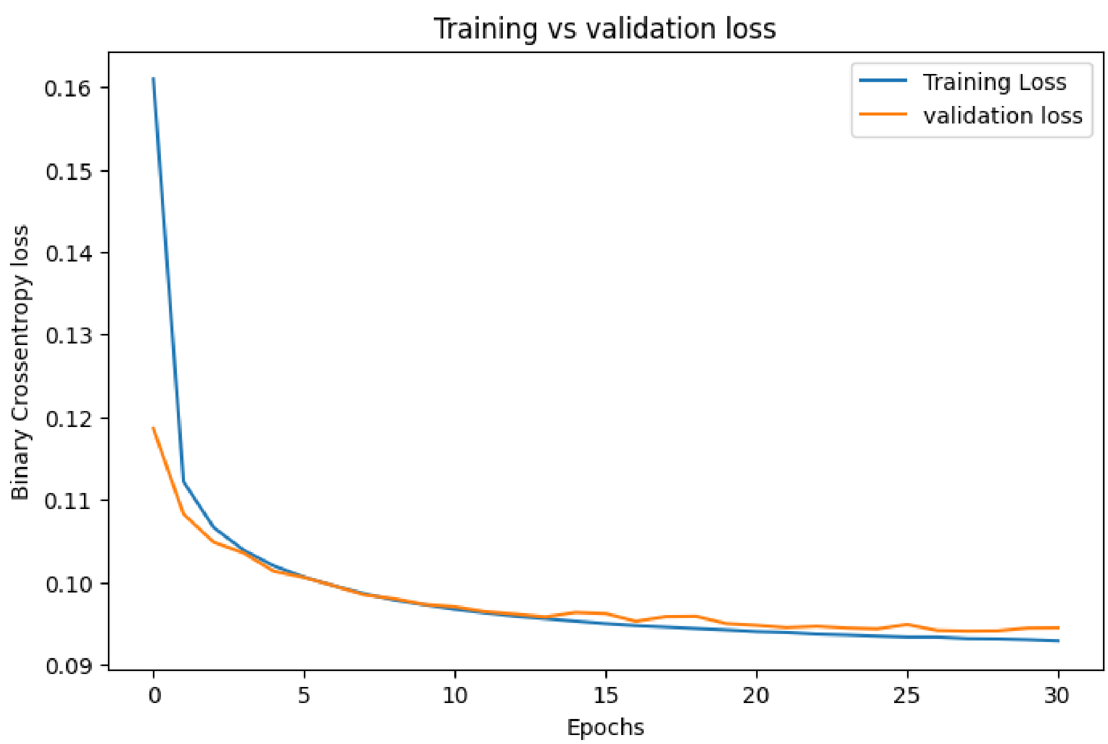
```
375/375              3s 7ms/step -
accuracy: 0.8133 - loss: 0.0957 - val_accuracy: 0.8136 - val_loss: 0.0958
Epoch 15/50
375/375              3s 7ms/step -
accuracy: 0.8135 - loss: 0.0952 - val_accuracy: 0.8131 - val_loss: 0.0963
Epoch 16/50
375/375              3s 7ms/step -
accuracy: 0.8132 - loss: 0.0950 - val_accuracy: 0.8131 - val_loss: 0.0962
Epoch 17/50
375/375              3s 7ms/step -
accuracy: 0.8134 - loss: 0.0948 - val_accuracy: 0.8137 - val_loss: 0.0953
Epoch 18/50
375/375              3s 7ms/step -
accuracy: 0.8133 - loss: 0.0946 - val_accuracy: 0.8132 - val_loss: 0.0958
Epoch 19/50
375/375              3s 7ms/step -
accuracy: 0.8131 - loss: 0.0944 - val_accuracy: 0.8142 - val_loss: 0.0958
Epoch 20/50
375/375              3s 7ms/step -
accuracy: 0.8135 - loss: 0.0942 - val_accuracy: 0.8139 - val_loss: 0.0950
Epoch 21/50
375/375              3s 7ms/step -
accuracy: 0.8134 - loss: 0.0941 - val_accuracy: 0.8139 - val_loss: 0.0948
Epoch 22/50
375/375              3s 7ms/step -
accuracy: 0.8133 - loss: 0.0940 - val_accuracy: 0.8138 - val_loss: 0.0945
Epoch 23/50
375/375              3s 7ms/step -
accuracy: 0.8139 - loss: 0.0937 - val_accuracy: 0.8140 - val_loss: 0.0946
Epoch 24/50
375/375              3s 7ms/step -
accuracy: 0.8136 - loss: 0.0936 - val_accuracy: 0.8138 - val_loss: 0.0944
Epoch 25/50
375/375              3s 7ms/step -
accuracy: 0.8134 - loss: 0.0936 - val_accuracy: 0.8139 - val_loss: 0.0943
Epoch 26/50
375/375              3s 7ms/step -
accuracy: 0.8138 - loss: 0.0933 - val_accuracy: 0.8141 - val_loss: 0.0949
Epoch 27/50
375/375              3s 7ms/step -
accuracy: 0.8139 - loss: 0.0932 - val_accuracy: 0.8138 - val_loss: 0.0941
Epoch 28/50
375/375              3s 7ms/step -
accuracy: 0.8140 - loss: 0.0930 - val_accuracy: 0.8138 - val_loss: 0.0941
Epoch 29/50
375/375              3s 7ms/step -
accuracy: 0.8137 - loss: 0.0930 - val_accuracy: 0.8137 - val_loss: 0.0941
Epoch 30/50
```

```
375/375                3s 7ms/step -
accuracy: 0.8134 - loss: 0.0930 - val_accuracy: 0.8134 - val_loss: 0.0944
Epoch 31/50
375/375                3s 7ms/step -
accuracy: 0.8133 - loss: 0.0929 - val_accuracy: 0.8141 - val_loss: 0.0945
Epoch 31: early stopping
Restoring model weights from the end of the best epoch: 28.
```

[8]:
```python
res = model.evaluate(test_noisy, test_images, verbose=1)
print(f"Test loss: {res[0]:.4f}, Test accuracy: {res[1]*100:.2f}%")
```

```
313/313                2s 3ms/step -
accuracy: 0.8199 - loss: 0.0928
Test loss: 0.0934, Test accuracy: 81.27%
```

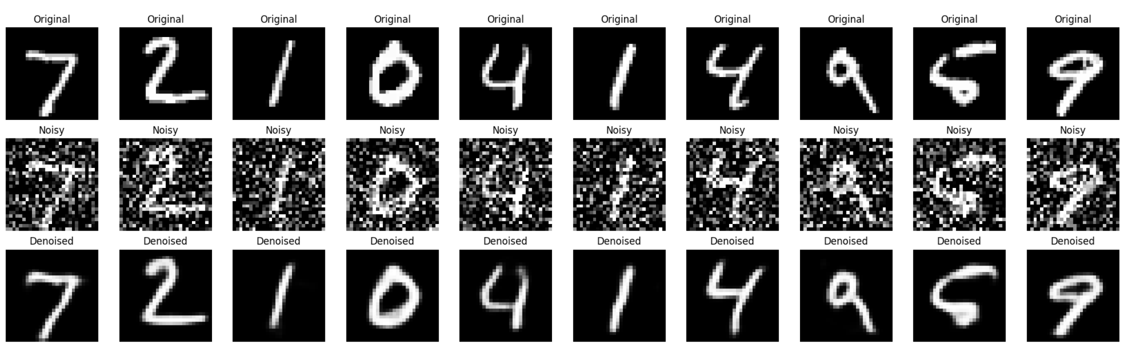
[9]:
```python
plt.figure(figsize = (8,5))
plt.plot(history.history['loss'], label = 'Training Loss')
plt.plot(history.history['val_loss'], label = 'validation loss')
plt.title("Training vs validation loss")
plt.xlabel("Epochs")
plt.ylabel("Binary Crossentropy loss")
plt.legend()
plt.show()
```



```
[10]: denoised_images = model.predict(test_noisy)
```

```
313/313                1s 3ms/step
```

```
[11]: n = 10
plt.figure(figsize = (20,6))
for i in range(n):
    ax = plt.subplot(3,n,i+1)
    plt.imshow(test_images[i].reshape(28,28), cmap = 'gray')
    plt.title("Original")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+n)
    plt.imshow(test_noisy[i].reshape(28,28), cmap = "gray")
    plt.title("Noisy")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+2*n)
    plt.imshow(denoised_images[i].reshape(28,28), cmap = "gray")
    plt.title("Denoised")
    ax.axis("off")
plt.tight_layout()
plt.show()
```



0.2 Q2- Model fitting with mse

```
[12]: model = Sequential()

## Encoder
model.add(Conv2D(32,(3,3), activation='relu', padding='same', input_shape = (28,28,1)))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))
```

```python
## Decoder
model.add(Conv2D(64,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(32,(3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(1,(3,3), activation = 'linear', padding = 'same'))

model.compile(optimizer='adam', loss = 'mean_squared_error')
history = model.fit(train_noisy, train_images, epochs = 50, batch_size = 128,␣
 ↪shuffle = True, validation_data = (val_noisy,val_images), verbose = 1,␣
 ↪callbacks = [early_stopping])
```

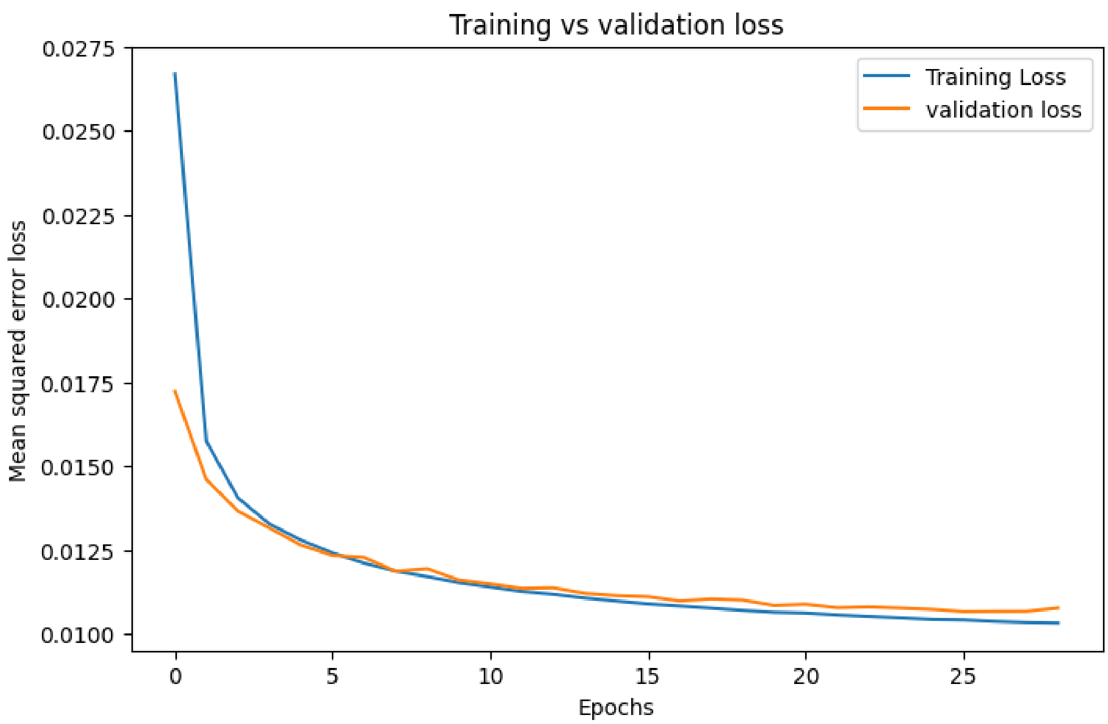
```
Epoch 1/50
375/375              7s 9ms/step -
loss: 0.0412 - val_loss: 0.0172
Epoch 2/50
375/375              2s 7ms/step -
loss: 0.0164 - val_loss: 0.0146
Epoch 3/50
375/375              3s 7ms/step -
loss: 0.0143 - val_loss: 0.0137
Epoch 4/50
375/375              3s 7ms/step -
loss: 0.0134 - val_loss: 0.0132
Epoch 5/50
375/375              3s 7ms/step -
loss: 0.0129 - val_loss: 0.0126
Epoch 6/50
375/375              3s 7ms/step -
loss: 0.0125 - val_loss: 0.0123
Epoch 7/50
375/375              3s 7ms/step -
loss: 0.0122 - val_loss: 0.0123
Epoch 8/50
375/375              3s 7ms/step -
loss: 0.0119 - val_loss: 0.0119
Epoch 9/50
375/375              3s 7ms/step -
loss: 0.0117 - val_loss: 0.0119
Epoch 10/50
375/375              3s 7ms/step -
loss: 0.0116 - val_loss: 0.0116
Epoch 11/50
375/375              3s 7ms/step -
loss: 0.0114 - val_loss: 0.0115
Epoch 12/50
375/375              3s 7ms/step -
loss: 0.0113 - val_loss: 0.0114
```

```
Epoch 13/50
375/375              2s 7ms/step -
loss: 0.0112 - val_loss: 0.0114
Epoch 14/50
375/375              3s 7ms/step -
loss: 0.0111 - val_loss: 0.0112
Epoch 15/50
375/375              3s 7ms/step -
loss: 0.0110 - val_loss: 0.0111
Epoch 16/50
375/375              3s 7ms/step -
loss: 0.0109 - val_loss: 0.0111
Epoch 17/50
375/375              3s 7ms/step -
loss: 0.0108 - val_loss: 0.0110
Epoch 18/50
375/375              3s 7ms/step -
loss: 0.0108 - val_loss: 0.0110
Epoch 19/50
375/375              3s 7ms/step -
loss: 0.0107 - val_loss: 0.0110
Epoch 20/50
375/375              3s 7ms/step -
loss: 0.0106 - val_loss: 0.0108
Epoch 21/50
375/375              2s 7ms/step -
loss: 0.0106 - val_loss: 0.0109
Epoch 22/50
375/375              2s 7ms/step -
loss: 0.0106 - val_loss: 0.0108
Epoch 23/50
375/375              2s 7ms/step -
loss: 0.0105 - val_loss: 0.0108
Epoch 24/50
375/375              2s 7ms/step -
loss: 0.0105 - val_loss: 0.0108
Epoch 25/50
375/375              2s 7ms/step -
loss: 0.0104 - val_loss: 0.0107
Epoch 26/50
375/375              3s 7ms/step -
loss: 0.0104 - val_loss: 0.0107
Epoch 27/50
375/375              3s 7ms/step -
loss: 0.0104 - val_loss: 0.0107
Epoch 28/50
375/375              2s 7ms/step -
loss: 0.0103 - val_loss: 0.0107
```

```
Epoch 29/50
375/375              2s 7ms/step -
loss: 0.0103 - val_loss: 0.0108
Epoch 29: early stopping
Restoring model weights from the end of the best epoch: 26.
```

[13]:

```
plt.figure(figsize = (8,5))
plt.plot(history.history['loss'], label = 'Training Loss')
plt.plot(history.history['val_loss'], label = 'validation loss')
plt.title("Training vs validation loss")
plt.xlabel("Epochs")
plt.ylabel("Mean squared error loss")
plt.legend()
plt.show()
```

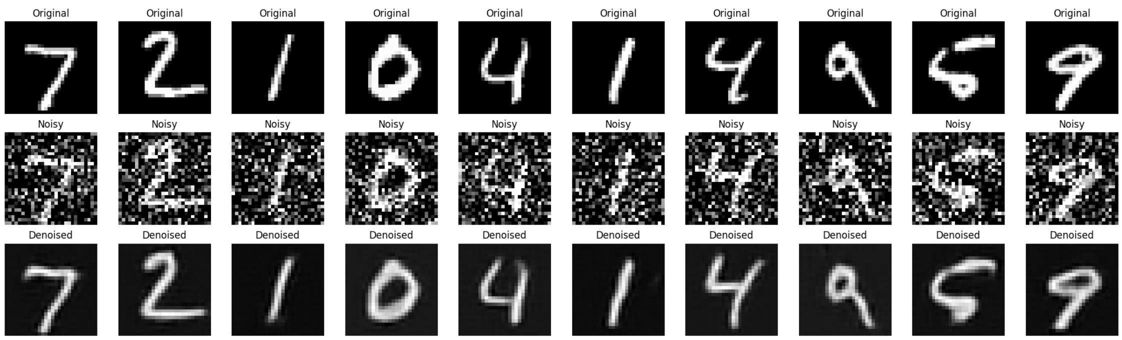


[14]:

```
denoised_images = model.predict(test_noisy)

n = 10
plt.figure(figsize = (20,6))
for i in range(n):
    ax = plt.subplot(3,n,i+1)
    plt.imshow(test_images[i].reshape(28,28), cmap = 'gray')
    plt.title("Original")
    ax.axis("off")
```

```
    ax = plt.subplot(3,n,i+1+n)
    plt.imshow(test_noisy[i].reshape(28,28), cmap = "gray")
    plt.title("Noisy")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+2*n)
    plt.imshow(denoised_images[i].reshape(28,28), cmap = "gray")
    plt.title("Denoised")
    ax.axis("off")

plt.tight_layout()
plt.show()
```

```
313/313          1s 3ms/step
```



Question 3: Try to implement a denoising autoencoder for CIFAR10 dataset and come up with your findings.

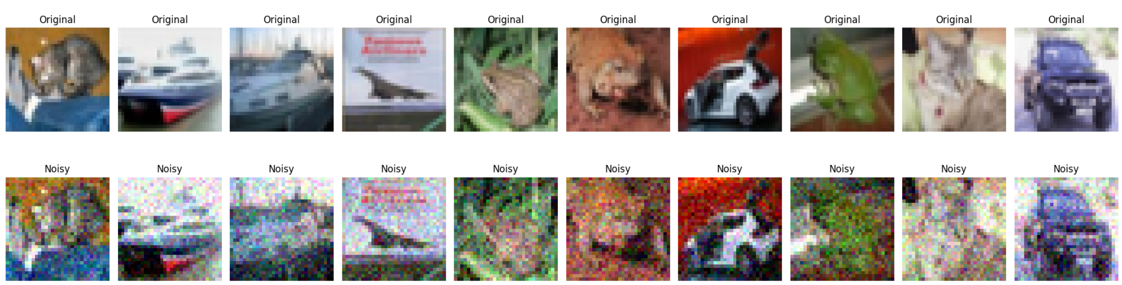
[15]:
```
from tensorflow.keras.datasets import cifar10

(train_images,_),(test_images, _) = cifar10.load_data()
train_images = train_images.astype('float32')/255.0
test_images = test_images.astype('float32')/255.0

train_images = np.reshape(train_images, (len(train_images), 32,32,3))
test_images = np.reshape(test_images, (len(test_images), 32,32,3))

print("Train Shape:", train_images.shape,"Test Shape:", test_images.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071          4s
0us/step
Train Shape: (50000, 32, 32, 3) Test Shape: (10000, 32, 32, 3)
```

[16]:
```python
train_images, val_images = train_test_split(train_images, test_size = 0.2, random_state=42)

noise_factor = 0.1
train_noisy = train_images+noise_factor*np.random.normal(loc = 0.0, scale = 1.0, size = train_images.shape)
val_noisy = val_images+noise_factor*np.random.normal(loc = 0.0, scale = 1.0, size = val_images.shape)
test_noisy = test_images+noise_factor*np.random.normal(loc = 0.0, scale = 1.0, size = test_images.shape)
train_noisy = np.clip(train_noisy, 0.,1.)
val_noisy = np.clip(val_noisy,0.,1.)
test_noisy = np.clip(test_noisy, 0.,1.)
print("Train noisy:", train_noisy.shape, "val noisy:", val_noisy.shape)
```

```
Train noisy: (40000, 32, 32, 3) val noisy: (10000, 32, 32, 3)
```

[17]:
```python
n = 10

plt.figure(figsize = (20,6))
for i in range(n):
    ax = plt.subplot(2,n,i+1)
    plt.imshow(test_images[i].reshape(32,32,3))
    plt.title("Original")
    ax.axis("off")
    ax = plt.subplot(2,n,i+1+n)
    plt.imshow(test_noisy[i].reshape(32,32,3))
    plt.title("Noisy")
    ax.axis("off")

plt.tight_layout()
plt.show()
```



0.2.1 Building the model

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D,␣
↪BatchNormalization, Dropout

model = Sequential()

##Encoder
model.add(Conv2D(64, (3,3), activation='relu', padding='same',␣
↪input_shape=(32,32,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D((2,2), padding='same'))

model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2,2), padding='same'))

model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Dropout(0.3))

##Decoder
model.add(Conv2D(256, (3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(UpSampling2D((2,2)))

model.add(Conv2D(128, (3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(UpSampling2D((2,2)))

model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(UpSampling2D((2,2)))

model.add(Conv2D(3, (3,3), activation='sigmoid', padding='same'))

model.summary()
```

```
Model: "sequential_2"


 Layer (type)                    Output Shape                  Param #

 conv2d_10 (Conv2D)              (None, 32, 32, 64)              1,792
```

batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_11 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_12 (Conv2D)	(None, 8, 8, 256)	295,168
batch_normalization_2 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_6 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout (Dropout)	(None, 4, 4, 256)	0
conv2d_13 (Conv2D)	(None, 4, 4, 256)	590,080
batch_normalization_3 (BatchNormalization)	(None, 4, 4, 256)	1,024
up_sampling2d_4 (UpSampling2D)	(None, 8, 8, 256)	0
conv2d_14 (Conv2D)	(None, 8, 8, 128)	295,040
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
up_sampling2d_5 (UpSampling2D)	(None, 16, 16, 128)	0
conv2d_15 (Conv2D)	(None, 16, 16, 64)	73,792
batch_normalization_5 (BatchNormalization)	(None, 16, 16, 64)	256
up_sampling2d_6 (UpSampling2D)	(None, 32, 32, 64)	0
conv2d_16 (Conv2D)	(None, 32, 32, 3)	1,731

[19]:
```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

model.compile(optimizer='adam', loss = 'binary_crossentropy', metrics =
 ↪['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
 ↪restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.5,
 ↪verbose=1)

history = model.fit(train_noisy, train_images,
                    epochs=50,
                    batch_size=128,
                    shuffle=True,
                    validation_data=(val_noisy, val_images),
                    callbacks=[early_stopping, reduce_lr],
                    verbose=1)
```

```
Epoch 1/50
313/313              31s 55ms/step -
accuracy: 0.6022 - loss: 0.6042 - val_accuracy: 0.5496 - val_loss: 0.5980 -
learning_rate: 0.0010
Epoch 2/50
313/313              8s 27ms/step -
accuracy: 0.7110 - loss: 0.5714 - val_accuracy: 0.7085 - val_loss: 0.5799 -
learning_rate: 0.0010
Epoch 3/50
313/313              9s 27ms/step -
accuracy: 0.7251 - loss: 0.5667 - val_accuracy: 0.7439 - val_loss: 0.5641 -
learning_rate: 0.0010
Epoch 4/50
313/313              9s 28ms/step -
accuracy: 0.7374 - loss: 0.5650 - val_accuracy: 0.7216 - val_loss: 0.5602 -
learning_rate: 0.0010
Epoch 5/50
313/313              9s 27ms/step -
accuracy: 0.7422 - loss: 0.5636 - val_accuracy: 0.7534 - val_loss: 0.5597 -
learning_rate: 0.0010
Epoch 6/50
313/313              8s 27ms/step -
accuracy: 0.7486 - loss: 0.5631 - val_accuracy: 0.7732 - val_loss: 0.5589 -
learning_rate: 0.0010
Epoch 7/50
313/313              8s 26ms/step -
accuracy: 0.7516 - loss: 0.5624 - val_accuracy: 0.7606 - val_loss: 0.5580 -
learning_rate: 0.0010
Epoch 8/50
```

```
313/313            8s 26ms/step -
accuracy: 0.7540 - loss: 0.5617 - val_accuracy: 0.7645 - val_loss: 0.5573 -
learning_rate: 0.0010
Epoch 9/50
313/313            8s 26ms/step -
accuracy: 0.7554 - loss: 0.5606 - val_accuracy: 0.7706 - val_loss: 0.5573 -
learning_rate: 0.0010
Epoch 10/50
313/313            8s 26ms/step -
accuracy: 0.7578 - loss: 0.5607 - val_accuracy: 0.7770 - val_loss: 0.5567 -
learning_rate: 0.0010
Epoch 11/50
313/313            8s 26ms/step -
accuracy: 0.7588 - loss: 0.5603 - val_accuracy: 0.7692 - val_loss: 0.5562 -
learning_rate: 0.0010
Epoch 12/50
313/313            8s 26ms/step -
accuracy: 0.7612 - loss: 0.5600 - val_accuracy: 0.7823 - val_loss: 0.5560 -
learning_rate: 0.0010
Epoch 13/50
313/313            8s 26ms/step -
accuracy: 0.7619 - loss: 0.5596 - val_accuracy: 0.7485 - val_loss: 0.5562 -
learning_rate: 0.0010
Epoch 14/50
313/313            8s 26ms/step -
accuracy: 0.7620 - loss: 0.5594 - val_accuracy: 0.7793 - val_loss: 0.5551 -
learning_rate: 0.0010
Epoch 15/50
313/313            8s 27ms/step -
accuracy: 0.7644 - loss: 0.5593 - val_accuracy: 0.7634 - val_loss: 0.5608 -
learning_rate: 0.0010
Epoch 16/50
313/313            8s 27ms/step -
accuracy: 0.7671 - loss: 0.5586 - val_accuracy: 0.7824 - val_loss: 0.5551 -
learning_rate: 0.0010
Epoch 17/50
313/313            8s 27ms/step -
accuracy: 0.7667 - loss: 0.5596 - val_accuracy: 0.7839 - val_loss: 0.5549 -
learning_rate: 0.0010
Epoch 18/50
313/313            8s 27ms/step -
accuracy: 0.7682 - loss: 0.5586 - val_accuracy: 0.7865 - val_loss: 0.5547 -
learning_rate: 0.0010
Epoch 19/50
313/313            8s 26ms/step -
accuracy: 0.7670 - loss: 0.5587 - val_accuracy: 0.7885 - val_loss: 0.5550 -
learning_rate: 0.0010
Epoch 20/50
```

```
313/313            8s 26ms/step -
accuracy: 0.7680 - loss: 0.5583 - val_accuracy: 0.7735 - val_loss: 0.5550 -
learning_rate: 0.0010
Epoch 21/50
312/313            0s 24ms/step -
accuracy: 0.7690 - loss: 0.5580
Epoch 21: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
313/313            8s 26ms/step -
accuracy: 0.7690 - loss: 0.5580 - val_accuracy: 0.7770 - val_loss: 0.5551 -
learning_rate: 0.0010
Epoch 22/50
313/313            8s 26ms/step -
accuracy: 0.7743 - loss: 0.5575 - val_accuracy: 0.7939 - val_loss: 0.5538 -
learning_rate: 5.0000e-04
Epoch 23/50
313/313            8s 26ms/step -
accuracy: 0.7740 - loss: 0.5577 - val_accuracy: 0.7815 - val_loss: 0.5542 -
learning_rate: 5.0000e-04
Epoch 24/50
313/313            8s 26ms/step -
accuracy: 0.7729 - loss: 0.5568 - val_accuracy: 0.7794 - val_loss: 0.5538 -
learning_rate: 5.0000e-04
Epoch 25/50
313/313            0s 24ms/step -
accuracy: 0.7740 - loss: 0.5579
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
313/313            8s 26ms/step -
accuracy: 0.7740 - loss: 0.5579 - val_accuracy: 0.7954 - val_loss: 0.5537 -
learning_rate: 5.0000e-04
Epoch 26/50
313/313            8s 26ms/step -
accuracy: 0.7772 - loss: 0.5574 - val_accuracy: 0.7851 - val_loss: 0.5536 -
learning_rate: 2.5000e-04
Epoch 27/50
313/313            8s 26ms/step -
accuracy: 0.7751 - loss: 0.5576 - val_accuracy: 0.7946 - val_loss: 0.5535 -
learning_rate: 2.5000e-04
Epoch 28/50
313/313            8s 27ms/step -
accuracy: 0.7746 - loss: 0.5568 - val_accuracy: 0.7879 - val_loss: 0.5538 -
learning_rate: 2.5000e-04
Epoch 29/50
313/313            8s 27ms/step -
accuracy: 0.7751 - loss: 0.5567 - val_accuracy: 0.7874 - val_loss: 0.5534 -
learning_rate: 2.5000e-04
Epoch 30/50
313/313            8s 27ms/step -
accuracy: 0.7776 - loss: 0.5576 - val_accuracy: 0.7874 - val_loss: 0.5534 -
```

```
learning_rate: 2.5000e-04
Epoch 31/50
313/313             8s 26ms/step -
accuracy: 0.7787 - loss: 0.5581 - val_accuracy: 0.7932 - val_loss: 0.5536 -
learning_rate: 2.5000e-04
Epoch 32/50
311/313             0s 24ms/step -
accuracy: 0.7767 - loss: 0.5569
Epoch 32: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
313/313             8s 26ms/step -
accuracy: 0.7767 - loss: 0.5569 - val_accuracy: 0.7902 - val_loss: 0.5536 -
learning_rate: 2.5000e-04
Epoch 33/50
313/313             8s 26ms/step -
accuracy: 0.7782 - loss: 0.5573 - val_accuracy: 0.7922 - val_loss: 0.5532 -
learning_rate: 1.2500e-04
Epoch 34/50
313/313             8s 26ms/step -
accuracy: 0.7784 - loss: 0.5568 - val_accuracy: 0.7989 - val_loss: 0.5532 -
learning_rate: 1.2500e-04
Epoch 35/50
313/313             8s 27ms/step -
accuracy: 0.7780 - loss: 0.5567 - val_accuracy: 0.7919 - val_loss: 0.5532 -
learning_rate: 1.2500e-04
Epoch 36/50
311/313             0s 24ms/step -
accuracy: 0.7784 - loss: 0.5570
Epoch 36: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
313/313             8s 26ms/step -
accuracy: 0.7784 - loss: 0.5570 - val_accuracy: 0.7946 - val_loss: 0.5534 -
learning_rate: 1.2500e-04
Epoch 37/50
313/313             8s 26ms/step -
accuracy: 0.7781 - loss: 0.5574 - val_accuracy: 0.7944 - val_loss: 0.5531 -
learning_rate: 6.2500e-05
Epoch 38/50
313/313             8s 26ms/step -
accuracy: 0.7778 - loss: 0.5578 - val_accuracy: 0.7922 - val_loss: 0.5531 -
learning_rate: 6.2500e-05
Epoch 39/50
313/313             8s 26ms/step -
accuracy: 0.7809 - loss: 0.5573 - val_accuracy: 0.7938 - val_loss: 0.5531 -
learning_rate: 6.2500e-05
Epoch 40/50
312/313             0s 24ms/step -
accuracy: 0.7791 - loss: 0.5570
Epoch 40: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
313/313             8s 26ms/step -
```

```
accuracy: 0.7791 - loss: 0.5570 - val_accuracy: 0.7980 - val_loss: 0.5533 -
learning_rate: 6.2500e-05
Epoch 41/50
313/313            8s 26ms/step -
accuracy: 0.7802 - loss: 0.5568 - val_accuracy: 0.7970 - val_loss: 0.5531 -
learning_rate: 3.1250e-05
Epoch 42/50
313/313            8s 26ms/step -
accuracy: 0.7784 - loss: 0.5568 - val_accuracy: 0.7981 - val_loss: 0.5531 -
learning_rate: 3.1250e-05
Epoch 43/50
312/313            0s 24ms/step -
accuracy: 0.7793 - loss: 0.5569
Epoch 43: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
313/313            8s 26ms/step -
accuracy: 0.7793 - loss: 0.5569 - val_accuracy: 0.7947 - val_loss: 0.5531 -
learning_rate: 3.1250e-05
Epoch 44/50
313/313            8s 26ms/step -
accuracy: 0.7800 - loss: 0.5570 - val_accuracy: 0.7960 - val_loss: 0.5530 -
learning_rate: 1.5625e-05
Epoch 45/50
313/313            8s 26ms/step -
accuracy: 0.7789 - loss: 0.5564 - val_accuracy: 0.7957 - val_loss: 0.5531 -
learning_rate: 1.5625e-05
Epoch 46/50
313/313            0s 24ms/step -
accuracy: 0.7805 - loss: 0.5574
Epoch 46: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
313/313            8s 26ms/step -
accuracy: 0.7805 - loss: 0.5574 - val_accuracy: 0.7956 - val_loss: 0.5531 -
learning_rate: 1.5625e-05
Epoch 47/50
313/313            8s 26ms/step -
accuracy: 0.7780 - loss: 0.5569 - val_accuracy: 0.7967 - val_loss: 0.5530 -
learning_rate: 7.8125e-06
Epoch 48/50
313/313            8s 26ms/step -
accuracy: 0.7788 - loss: 0.5560 - val_accuracy: 0.7959 - val_loss: 0.5530 -
learning_rate: 7.8125e-06
Epoch 49/50
311/313            0s 24ms/step -
accuracy: 0.7796 - loss: 0.5579
Epoch 49: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.
313/313            8s 26ms/step -
accuracy: 0.7796 - loss: 0.5579 - val_accuracy: 0.7939 - val_loss: 0.5530 -
learning_rate: 7.8125e-06
Epoch 50/50
```

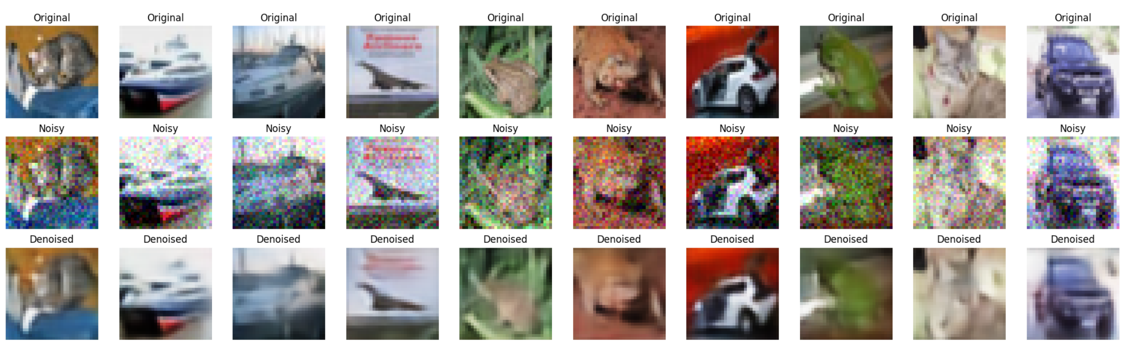
```
313/313                 8s 26ms/step -
accuracy: 0.7788 - loss: 0.5571 - val_accuracy: 0.7956 - val_loss: 0.5530 -
learning_rate: 3.9063e-06
Restoring model weights from the end of the best epoch: 50.
```

[20]:
```python
res = model.evaluate(test_noisy, test_images, verbose = 0)
print(f"Test Accuracy: {res[1]*100:.2f}%")
```

```
Test Accuracy: 79.75%
```

[21]:
```python
denoised_images = model.predict(test_noisy)

n = 10
plt.figure(figsize = (20,6))
for i in range(n):
    ax = plt.subplot(3,n,i+1)
    plt.imshow(test_images[i].reshape(32,32,3), cmap = 'gray')
    plt.title("Original")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+n)
    plt.imshow(test_noisy[i].reshape(32,32,3), cmap = "gray")
    plt.title("Noisy")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+2*n)
    plt.imshow(denoised_images[i].reshape(32,32,3), cmap = "gray")
    plt.title("Denoised")
    ax.axis("off")

plt.tight_layout()
plt.show()
```

```
313/313                 3s 6ms/step
```



0.3 Question 4: Next, we will try to implement a sparse autoencoder using the MNIST dataset. As mentioned in the class, the sparse encoder works like a regularized autoencoder, which learn useful feature representations by forcing the network to activate only a small number of neurons in the latent (hidden) layer at any given time. Now the first question model we have used is a overcomplete autoencoder. Let's try to implement the same problem by including the sparsity constraint in the bottleneck layer.

0.3.1 Loading the MNIST dataset

[30]:
```
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test  = x_test.astype('float32') / 255.


x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test  = np.reshape(x_test, (len(x_test), 28, 28, 1))

print("Train:", x_train.shape, "Test:", x_test.shape)
```

```
Train: (60000, 28, 28, 1) Test: (10000, 28, 28, 1)
```

[31]:
```
train_images, val_images = train_test_split(x_train, test_size=0.2,␣
 ↪random_state=42)

noise_factor = 0.5
train_noisy = train_images + noise_factor * np.random.normal(loc=0.0, scale=1.
 ↪0, size=train_images.shape)
val_noisy   = val_images   + noise_factor * np.random.normal(loc=0.0, scale=1.
 ↪0, size=val_images.shape)
test_noisy  = x_test       + noise_factor * np.random.normal(loc=0.0, scale=1.
 ↪0, size=x_test.shape)


train_noisy = np.clip(train_noisy, 0., 1.)
val_noisy   = np.clip(val_noisy, 0., 1.)
test_noisy  = np.clip(test_noisy, 0., 1.)

print("Train noisy:", train_noisy.shape, "Val noisy:", val_noisy.shape)
```

```
Train noisy: (48000, 28, 28, 1) Val noisy: (12000, 28, 28, 1)
```

[32]:
```
from tensorflow.keras import regularizers

model = Sequential()

##␣Encoder
```

```
model.add(Conv2D(32, (3,3), activation='relu', padding='same',␣
 ↪input_shape=(28,28,1)))
model.add(MaxPooling2D((2,2), padding='same'))

model.add(Conv2D(64, (3,3), activation='relu', padding='same',
                 activity_regularizer=regularizers.l1(1e-5)))
model.add(MaxPooling2D((2,2), padding='same'))

## Decoder
model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))

model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))

model.add(Conv2D(1, (3,3), activation='sigmoid', padding='same'))

model.summary()
```

```
Model: "sequential_5"


 Layer (type)                    Output Shape              Param #

 conv2d_27 (Conv2D)              (None, 28, 28, 32)            320

 max_pooling2d_11 (MaxPooling2D) (None, 14, 14, 32)              0

 conv2d_28 (Conv2D)              (None, 14, 14, 64)         18,496

 max_pooling2d_12 (MaxPooling2D) (None, 7, 7, 64)                0

 conv2d_29 (Conv2D)              (None, 7, 7, 64)           36,928

 up_sampling2d_11 (UpSampling2D) (None, 14, 14, 64)              0

 conv2d_30 (Conv2D)              (None, 14, 14, 32)         18,464

 up_sampling2d_12 (UpSampling2D) (None, 28, 28, 32)              0

 conv2d_31 (Conv2D)              (None, 28, 28, 1)             289


 Total params: 74,497 (291.00 KB)

 Trainable params: 74,497 (291.00 KB)
```

Non-trainable params: 0 (0.00 B)

[33]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
↪restore_best_weights=True, verbose=1)

history = model.fit(train_noisy, train_images,
                    epochs = 50,
                    batch_size = 128,
                    shuffle = True,
                    validation_data = (val_noisy,val_images),
                    verbose = 1,
                    callbacks = [early_stopping])
```

```
Epoch 1/50
375/375              8s 11ms/step -
accuracy: 0.8027 - loss: 0.3734 - val_accuracy: 0.8053 - val_loss: 0.1593
Epoch 2/50
375/375              3s 7ms/step -
accuracy: 0.8059 - loss: 0.1513 - val_accuracy: 0.8081 - val_loss: 0.1363
Epoch 3/50
375/375              3s 7ms/step -
accuracy: 0.8089 - loss: 0.1336 - val_accuracy: 0.8101 - val_loss: 0.1282
Epoch 4/50
375/375              3s 7ms/step -
accuracy: 0.8090 - loss: 0.1273 - val_accuracy: 0.8104 - val_loss: 0.1234
Epoch 5/50
375/375              3s 7ms/step -
accuracy: 0.8102 - loss: 0.1226 - val_accuracy: 0.8109 - val_loss: 0.1201
Epoch 6/50
375/375              3s 7ms/step -
accuracy: 0.8107 - loss: 0.1193 - val_accuracy: 0.8120 - val_loss: 0.1181
Epoch 7/50
375/375              3s 7ms/step -
accuracy: 0.8106 - loss: 0.1172 - val_accuracy: 0.8119 - val_loss: 0.1154
Epoch 8/50
375/375              3s 7ms/step -
accuracy: 0.8117 - loss: 0.1146 - val_accuracy: 0.8128 - val_loss: 0.1154
Epoch 9/50
375/375              3s 7ms/step -
accuracy: 0.8111 - loss: 0.1131 - val_accuracy: 0.8120 - val_loss: 0.1120
Epoch 10/50
375/375              3s 7ms/step -
```

```
accuracy: 0.8118 - loss: 0.1112 - val_accuracy: 0.8120 - val_loss: 0.1107
Epoch 11/50
375/375              3s 7ms/step -
accuracy: 0.8117 - loss: 0.1104 - val_accuracy: 0.8125 - val_loss: 0.1098
Epoch 12/50
375/375              3s 7ms/step -
accuracy: 0.8122 - loss: 0.1096 - val_accuracy: 0.8129 - val_loss: 0.1098
Epoch 13/50
375/375              3s 7ms/step -
accuracy: 0.8121 - loss: 0.1087 - val_accuracy: 0.8126 - val_loss: 0.1083
Epoch 14/50
375/375              3s 7ms/step -
accuracy: 0.8124 - loss: 0.1081 - val_accuracy: 0.8123 - val_loss: 0.1077
Epoch 15/50
375/375              3s 7ms/step -
accuracy: 0.8121 - loss: 0.1075 - val_accuracy: 0.8129 - val_loss: 0.1074
Epoch 16/50
375/375              3s 7ms/step -
accuracy: 0.8120 - loss: 0.1072 - val_accuracy: 0.8132 - val_loss: 0.1077
Epoch 17/50
375/375              3s 7ms/step -
accuracy: 0.8127 - loss: 0.1066 - val_accuracy: 0.8122 - val_loss: 0.1066
Epoch 18/50
375/375              3s 7ms/step -
accuracy: 0.8123 - loss: 0.1061 - val_accuracy: 0.8129 - val_loss: 0.1066
Epoch 19/50
375/375              3s 7ms/step -
accuracy: 0.8123 - loss: 0.1059 - val_accuracy: 0.8137 - val_loss: 0.1091
Epoch 20/50
375/375              3s 7ms/step -
accuracy: 0.8128 - loss: 0.1056 - val_accuracy: 0.8124 - val_loss: 0.1053
Epoch 21/50
375/375              3s 7ms/step -
accuracy: 0.8125 - loss: 0.1050 - val_accuracy: 0.8128 - val_loss: 0.1052
Epoch 22/50
375/375              3s 7ms/step -
accuracy: 0.8127 - loss: 0.1047 - val_accuracy: 0.8118 - val_loss: 0.1059
Epoch 23/50
375/375              3s 7ms/step -
accuracy: 0.8125 - loss: 0.1045 - val_accuracy: 0.8132 - val_loss: 0.1047
Epoch 24/50
375/375              3s 7ms/step -
accuracy: 0.8128 - loss: 0.1044 - val_accuracy: 0.8134 - val_loss: 0.1051
Epoch 25/50
375/375              3s 7ms/step -
accuracy: 0.8127 - loss: 0.1042 - val_accuracy: 0.8128 - val_loss: 0.1044
Epoch 26/50
375/375              3s 7ms/step -
```

```
accuracy: 0.8126 - loss: 0.1041 - val_accuracy: 0.8134 - val_loss: 0.1044
Epoch 27/50
375/375              3s 7ms/step -
accuracy: 0.8132 - loss: 0.1037 - val_accuracy: 0.8132 - val_loss: 0.1048
Epoch 28/50
375/375              3s 7ms/step -
accuracy: 0.8127 - loss: 0.1036 - val_accuracy: 0.8130 - val_loss: 0.1037
Epoch 29/50
375/375              3s 7ms/step -
accuracy: 0.8127 - loss: 0.1035 - val_accuracy: 0.8128 - val_loss: 0.1035
Epoch 30/50
375/375              3s 7ms/step -
accuracy: 0.8128 - loss: 0.1033 - val_accuracy: 0.8130 - val_loss: 0.1032
Epoch 31/50
375/375              3s 7ms/step -
accuracy: 0.8125 - loss: 0.1030 - val_accuracy: 0.8131 - val_loss: 0.1035
Epoch 32/50
375/375              3s 7ms/step -
accuracy: 0.8128 - loss: 0.1030 - val_accuracy: 0.8136 - val_loss: 0.1040
Epoch 33/50
375/375              3s 7ms/step -
accuracy: 0.8123 - loss: 0.1030 - val_accuracy: 0.8133 - val_loss: 0.1031
Epoch 34/50
375/375              3s 7ms/step -
accuracy: 0.8133 - loss: 0.1028 - val_accuracy: 0.8131 - val_loss: 0.1029
Epoch 35/50
375/375              3s 7ms/step -
accuracy: 0.8130 - loss: 0.1027 - val_accuracy: 0.8123 - val_loss: 0.1036
Epoch 36/50
375/375              3s 7ms/step -
accuracy: 0.8122 - loss: 0.1025 - val_accuracy: 0.8132 - val_loss: 0.1029
Epoch 37/50
375/375              3s 7ms/step -
accuracy: 0.8131 - loss: 0.1025 - val_accuracy: 0.8133 - val_loss: 0.1026
Epoch 38/50
375/375              3s 7ms/step -
accuracy: 0.8127 - loss: 0.1023 - val_accuracy: 0.8124 - val_loss: 0.1031
Epoch 39/50
375/375              3s 7ms/step -
accuracy: 0.8130 - loss: 0.1022 - val_accuracy: 0.8125 - val_loss: 0.1030
Epoch 40/50
375/375              3s 7ms/step -
accuracy: 0.8129 - loss: 0.1023 - val_accuracy: 0.8135 - val_loss: 0.1031
Epoch 40: early stopping
Restoring model weights from the end of the best epoch: 37.
```

[37]:

```
res = model.evaluate(test_noisy, x_test, verbose=1)
print(f"Test loss: {res[0]:.4f}, Test accuracy: {res[1]*100:.2f}%")
```

```
313/313            1s 2ms/step -
accuracy: 0.8195 - loss: 0.0993
Test loss: 0.0999, Test accuracy: 81.23%
```

[40]:

```
denoised_images = model.predict(test_noisy)
n = 10
plt.figure(figsize = (20,6))
for i in range(n):
    ax = plt.subplot(3,n,i+1)
    plt.imshow(x_test[i].reshape(28,28), cmap = 'gray')
    plt.title("Original")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+n)
    plt.imshow(test_noisy[i].reshape(28,28), cmap = "gray")
    plt.title("Noisy")
    ax.axis("off")

    ax = plt.subplot(3,n,i+1+2*n)
    plt.imshow(denoised_images[i].reshape(28,28), cmap = "gray")
    plt.title("Denoised")
    ax.axis("off")
plt.tight_layout()
plt.show()
```

```
313/313            1s 2ms/step
```