# PMDS603P Deep Learning Lab Experiment 13

October 2025

## 1 Work to do today

Note: Make a single PDF file of the work you are doing in a Jupyter notebook. Upload with the proper format. Please mention your name and roll no properly with the Experiment number on the first page of your submission.

**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.

First, let us import basic modules in our work

```
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
```

Now next part we will import the dataset and split into train, test and validation.

```
(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)
```

Next we will try to induce some noise (Gaussian noise) to the images and try to print them and check.

```
train_images, val_images = train_test_split(train_images, 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  = 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)
```

Here, the noise we are adding follows a normal distribution with mean 0 and variance 1. each time a value is sampled from this distribution and is added to a pixel in the image. And the same is done for all pixels in an image.

Now we can print and check the images

```
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()
```

Next, we can move on to define the architecture of our autoencoder model.

Here we have gone for a undercomplete autoencoder. You have to make sure that the dimensions of the input and output layers are same, which is required for an autoencoder.

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

    # Decoder
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    UpSampling2D((2, 2)),
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    UpSampling2D((2, 2)),
    Conv2D(1, (3, 3), activation='sigmoid', padding='same')
])
```

Next, we can compile and fit the model by including early stopping also. Now we can

```
model.summary()

model.compile(optimizer='adam', loss='binary_crossentropy')

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

history = model.fit(
    train_noisy, train_images,
    epochs=10,
    batch_size=128,
    shuffle=True,
    validation_data=(val_noisy, val_images),
    callbacks=[early_stopping]
)
```

predict the output images of test cases and check for the accuracy and other metrics.

```
model.summary()

model.compile(optimizer='adam', loss='binary_crossentropy')

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

history = model.fit(
    train_noisy, train_images,
    epochs=10,
    batch_size=128,
    shuffle=True,
    validation_data=(val_noisy, val_images),
    callbacks=[early_stopping]
)
```

In addition, try printing the test examples and see the predictions, how your model is performing in n.

```
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()
```

Very well, you can print the loss and visualize it

```
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()
```

Question 2: Try to fit the model with loss='mse' which is mean squared error by changing the output layer activation as linear.

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

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.

Now you can do the necessary import

**from tensorflow.keras import regularizers**

To include the sparsity constraint, you can include this activity regularizer in your bottleneck layer.

```
# 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"))
```

The hyperparameter value $\lambda = 1e - 5$ is just a choice. The best one should be found by tuning. Check whether your model performance also increases in this case, whether there is a significant difference in the loss.