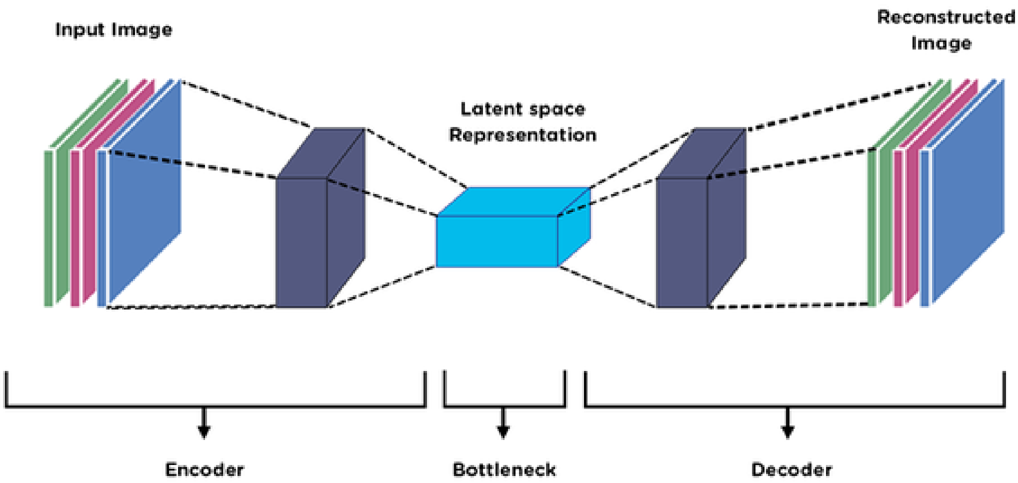
ASSIGNMENT 4: AUTOENCODER TO IMPLEMENT ANOMALY DETECTION.

An autoencoder is a type of **neural network** used to learn efficient representations of input data, typically for tasks like **dimensionality reduction or feature learning**. Autoencoders are also used in **anomaly detection by learning the normal patterns** in data and then identifying inputs that deviate significantly from these learned patterns.



1. Autoencoder Architecture

- An autoencoder consists of two main parts:
  - **Encoder**: Compresses the input data into a lower-dimensional representation called the **latent space** or **bottleneck**.
  - **Decoder**: Attempts to reconstruct the original data from this compressed representation.
- **Loss Function**: Measures the difference between the original input and its reconstructed version. Commonly used loss functions are **Mean Squared Error (MSE)** or **Binary Cross-Entropy**.

2. Latent Space / Bottleneck Layer

- **Purpose**: The latent space (or bottleneck layer) is where the encoder compresses the data into a lower-dimensional form.
- **Anomaly Detection Insight**: This bottleneck captures essential patterns in the data. If a data point deviates from these patterns (as an anomaly), the autoencoder will struggle to reconstruct it well.

3. Reconstruction Loss

- **Definition**: The difference between the input and reconstructed output.

- **Anomaly Detection**: Normal data typically has low reconstruction loss, while anomalies (unusual patterns) show a higher loss, making them easier to identify.
- **Threshold Setting**: In practice, a threshold is set for reconstruction loss; if the loss exceeds this threshold, the data point is flagged as an anomaly.

8. Common Challenges

- **Setting the Threshold**: Selecting an appropriate threshold for reconstruction error is crucial. Setting it too low may cause false positives, while too high can miss true anomalies.
- **Data Imbalance**: Anomaly detection often deals with highly imbalanced datasets, where normal data vastly outnumbers anomalous data. This requires specific tuning and robust metrics.

```
[1] #Import TensorFlow and other libraries
    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.datasets import fashion_mnist
    from tensorflow.keras.models import Model
```

```
[2] (x_train, _), (x_test, _) = fashion_mnist.load_data()

    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.

    print (x_train.shape)  # Output: (60000, 28, 28)
    print (x_test.shape)   # Output: (10000, 28, 28)
```

- The data is loaded and normalized to a range of [0, 1] by dividing by 255.
- Shapes (60000, 28, 28) and (10000, 28, 28) mean there are 60,000 training images and 10,000 testing images, each 28x28 pixels.

```
[3] 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):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

    autoencoder = Autoencoder(latent_dim)
```

```
[4] autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
[5] autoencoder.fit(x_train, x_train, epochs=10, shuffle=True, validation_data=(x_test, x_test))
```

Autoencoder Class:

- **latent_dim** is set to 64, indicating the dimensionality of the compressed feature space (latent space).
- encoder is defined with:
  - A **Flatten** layer to convert the 28x28 images into a 784-dimensional vector.
  - A **Dense** layer to reduce this 784-dimensional vector to a 64-dimensional latent representation.
- decoder is defined with:
  - A **Dense** layer to reconstruct the flattened 784-dimensional image from the latent space.
  - A **Reshape** layer to transform it back to a 28x28 image.
- call method defines the forward pass, calling the encoder and then the decoder.

```python
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # Display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # Display reconstruction
  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()
```

1.
    - **Display original images** in the top row and **reconstructed images** in the bottom row.
    - Differences between original and reconstructed images reveal the quality of the autoencoder's ability to reconstruct images from latent representations.

Important Terms

- **Autoencoder**: A neural network trained to reconstruct its input. Used here to compress images to a lower-dimensional latent space and reconstruct them.
- **Encoder**: The part of the autoencoder that compresses the input.
- **Decoder**: The part of the autoencoder that reconstructs the input from its compressed form.
- **Latent Space/Representation**: The compressed feature space where only essential information is stored.
- **Reconstruction Loss**: Measures how well the autoencoder reconstructs the input; used in this code as mean squared error.
- **Activation Functions**:
    - ReLU in the encoder layer: Adds non-linearity, helps learn better representations.
    - Sigmoid in the output layer of the decoder: Squashes values between 0 and 1, which aligns with the input data normalization.