Module 3

Autoencoders

Autoencoders have emerged as one of the technologies and techniques that enable computer systems to solve data compression problems more efficiently.

They became a popular solution for reducing noisy data.

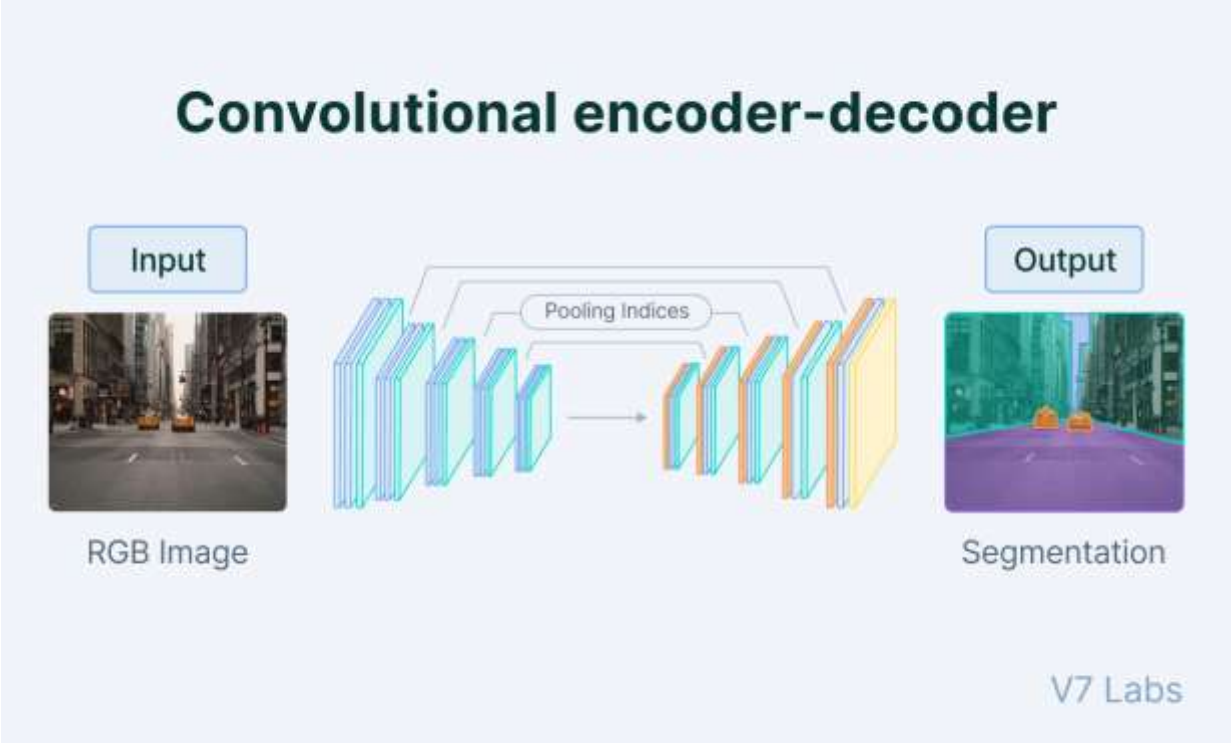
Simple autoencoders provide outputs that are the same or similar to the input data—only compressed. In the case of variational autoencoders, often discussed in the context of large language models, the output is newly generated content.

What is an autoencoder?

An autoencoder is a type of artificial neural network used to learn data encodings in an unsupervised manner.

The aim of an autoencoder is to learn a lower-dimensional representation (encoding) for a higher-dimensional data, typically for dimensionality reduction, by training the network to capture the most important parts of the input image.

Module 3



The architecture of autoencoders

Let's start with a quick overview of autoencoders' architecture.

Autoencoders consist of 3 parts:

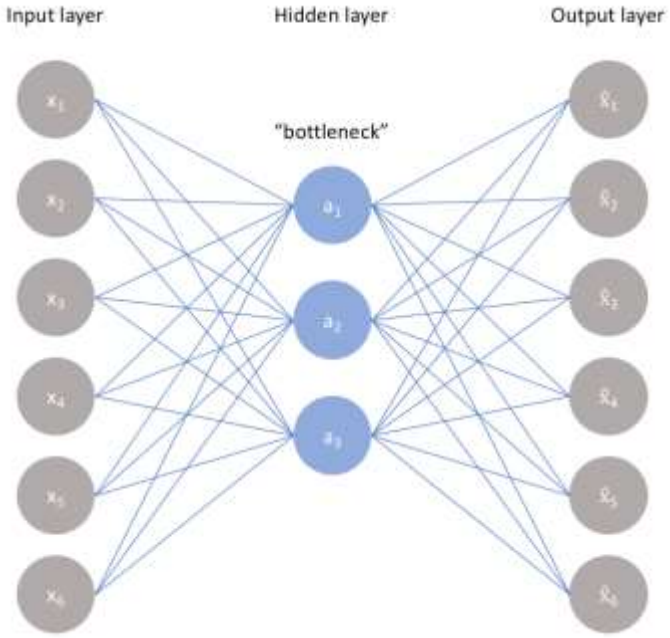
1. **Encoder**: A module that compresses the train-validate-test set input data into an encoded representation that is typically several orders of magnitude smaller than the input data.

2. **Bottleneck:** A module that contains the compressed knowledge representations and is therefore the most important part of the network.

3. **Decoder:** A module that helps the network"decompress" the knowledge representations and reconstructs the data back from its encoded form. The output is then compared with a ground truth.

Module 3

The architecture as a whole looks something like this:



The relationship between the Encoder, Bottleneck, and Decoder

Encoder

The encoder is a set of convolutional blocks followed by pooling modules that compress the input to the model into a compact section called the bottleneck.

The bottleneck is followed by the decoder that consists of a series of upsampling modules to bring the compressed feature back into the form of an image. In case of simple autoencoders, the output is expected to be the same as the input data with reduced noise.

However, for variational autoencoders it is a completely new image, formed with information the model has been provided as input.

Bottleneck

Module 3

The most important part of the neural network, and ironically the smallest one, is the bottleneck. The bottleneck exists to restrict the flow of information to the decoder from the encoder, thus,allowing only the most vital information to pass through.

Since the bottleneck is designed in such a way that the maximum information possessed by an image is captured in it, we can say that the bottleneck helps us form a *knowledge-representation* of the input.

Thus, the encoder-decoder structure helps us extract the most from an image in the form of data and establish useful correlations between various inputs within the network.

A bottleneck as a compressed representation of the input further prevents the neural network from memorising the input and overfitting on the data.

As a rule of thumb, remember this: The smaller the bottleneck, the lower the risk of overfitting.

However—

Very small bottlenecks would restrict the amount of information storable, which increases the chances of important information slipping out through the pooling layers of the encoder.

Decoder

Finally, the decoder is a set of upsampling and convolutional blocks that reconstructs the bottleneck's output.

Since the input to the decoder is a compressed knowledge representation, the decoder serves as a “decompressor” and builds back the image from its latent attributes.

How to train autoencoders?

You need to set 4 hyperparameters before *training* an autoencoder:

1. **Code size:** The code size or the size of the bottleneck is the most important hyperparameter used to tune the autoencoder. The bottleneck size decides how much the data has to be compressed. This can also act as a regularisation term.
2. **Number of layers:** Like all neural networks, an important hyperparameter to tune autoencoders is the depth of the encoder and the decoder. While a higher depth increases model complexity, a lower depth is faster to process.
3. **Number of nodes per layer:** The number of nodes per layer defines the weights we use per layer. Typically, the number of nodes decreases with each subsequent layer in the autoencoder as the input to each of these layers becomes smaller across the layers.
4. **Reconstruction Loss:** The loss function we use to train the autoencoder is highly dependent on the type of input and output we want the autoencoder to adapt to. If we are working with image data, the most popular loss functions for reconstruction are MSE Loss and L1 Loss. In case the inputs and outputs are within the range [0,1], as in MNIST, we can also make use of Binary Cross Entropy as the reconstruction loss.