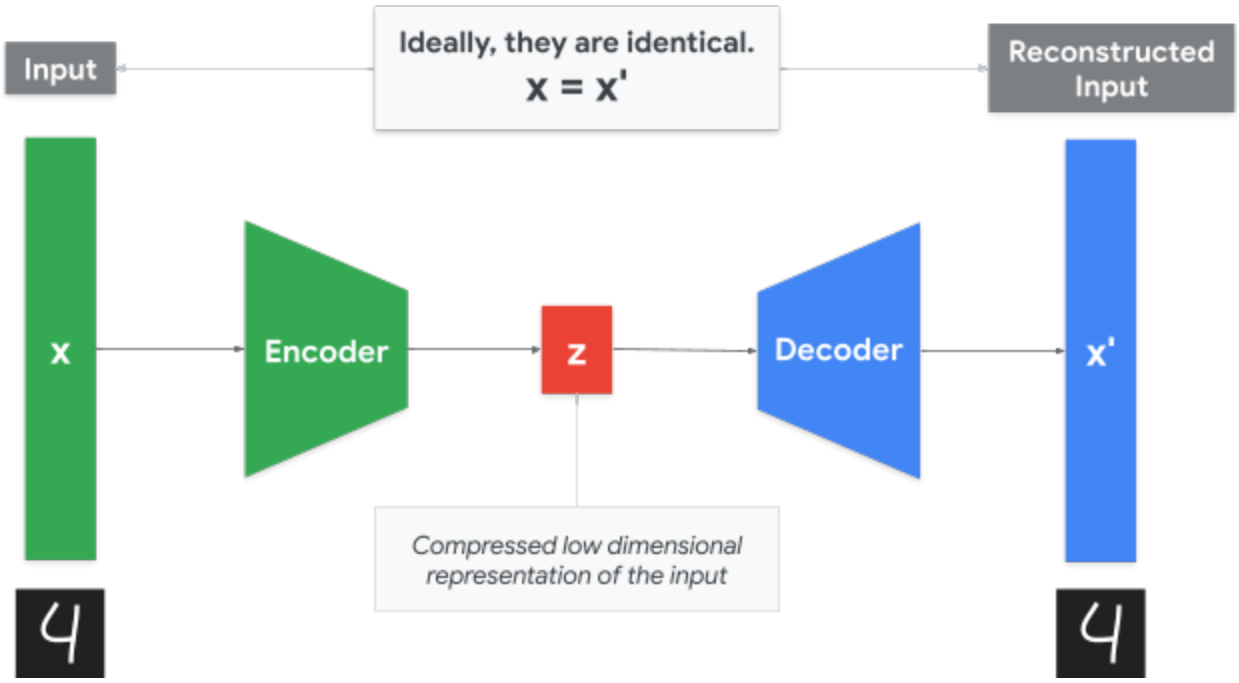
Autoencoder Model Architecture



You've previously been introduced to the basic concept of an autoencoder model. At its core, an autoencoder is trained to reconstruct data after compressing it into an embedding. You've learned how autoencoders can be applied to anomaly detection by training only on normal data and then using the reconstruction error to determine if a new input is similar to the normal training data. In this reading, we will examine autoencoders in more detail through a high level overview describing the strengths of different types of autoencoders. While autoencoders have many uses, including denoising, compression, and data generation, in this reading we will continue to focus on time-series anomaly detection. If you'd like to examine more applications of autoencoders check out this article.

Types of Autoencoders

It turns out that there are many kinds of autoencoders that are used in the wild. In this rest of this article we will survey some of the most popular types. To see how to implement all of these different autoencoders in keras check out this article.

Fully Connected Autoencoder

A deep fully-connected autoencoder has multiple fully connected layers with some small embedding in the middle. You've already seen an example of these in the previous video. They are the most basic form of an autoencoder and are simple to implement and deploy.

Convolutional Autoencoder

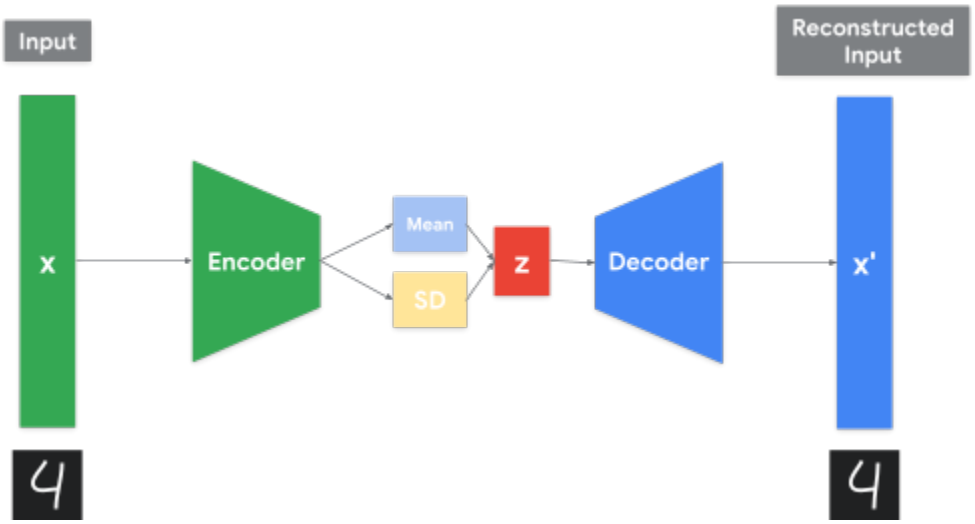
A convolutional autoencoder operates under the same principle as the deep fully-connected autoencoder but uses convolutions for the encoder layers and transpose convolutions for the decoder layers. Due to the spatial nature of convolutions, these autoencoders are particularly useful for images and spectrograms (which are essentially images). Unfortunately, at this time, transpose convolutions are less commonly supported by tinyML frameworks, so deploying convolutional autoencoders to microcontrollers is more challenging. That said, hopefully they will become more accessible soon. In the meantime, if you would like to try using a convolutional autoencoder for anomaly detection in Colab, check out this keras example.

LSTM Autoencoders

Long short term memory networks are a type of recurrent neural network. They are useful for processing longer sequences while capturing the temporal structure of the data like the time-series data we have explored thus far. This can lead to a more detailed analysis of longer sequences which leads to higher accuracy in some cases. However, these RNN layers generally use more working memory as they maintain and modify a state over long sequences. This means they often consume more SRAM, which is a precious resource in tinyML. Additionally, it is less common for tinyML frameworks to support LSTMs or RNNs in general. For more detail on LSTM autoencoders check out this article.

Variational Autoencoders

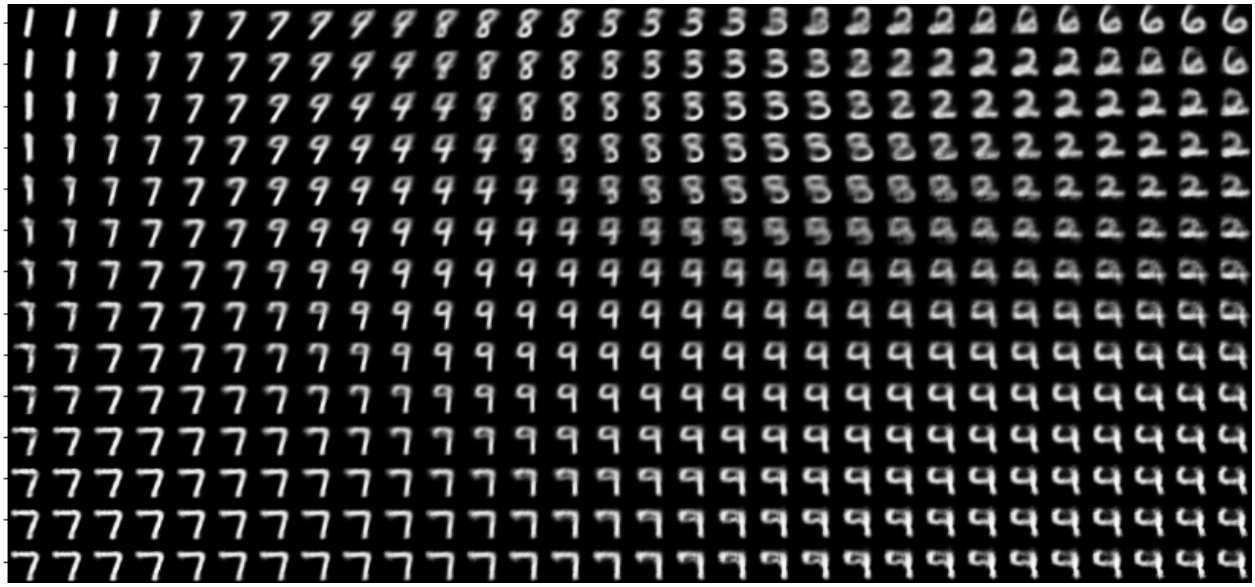
Unlike the previous examples, the unique aspect of variational autoencoders is not that they use a different type of layer. Instead, variational autoencoders impose an additional constraint during training, which forces the model to map the input onto a distribution, which means the normal latent vector of an autoencoder is replaced by a mean vector and a standard deviation vector from which a latent vector is sampled. For anomaly detection, we can use the reconstruction probability that the model produces instead of the reconstruction error used in normal autoencoders, which can at times lead to better predictions.



Visualizing the Latent Space of Variational Autoencoders

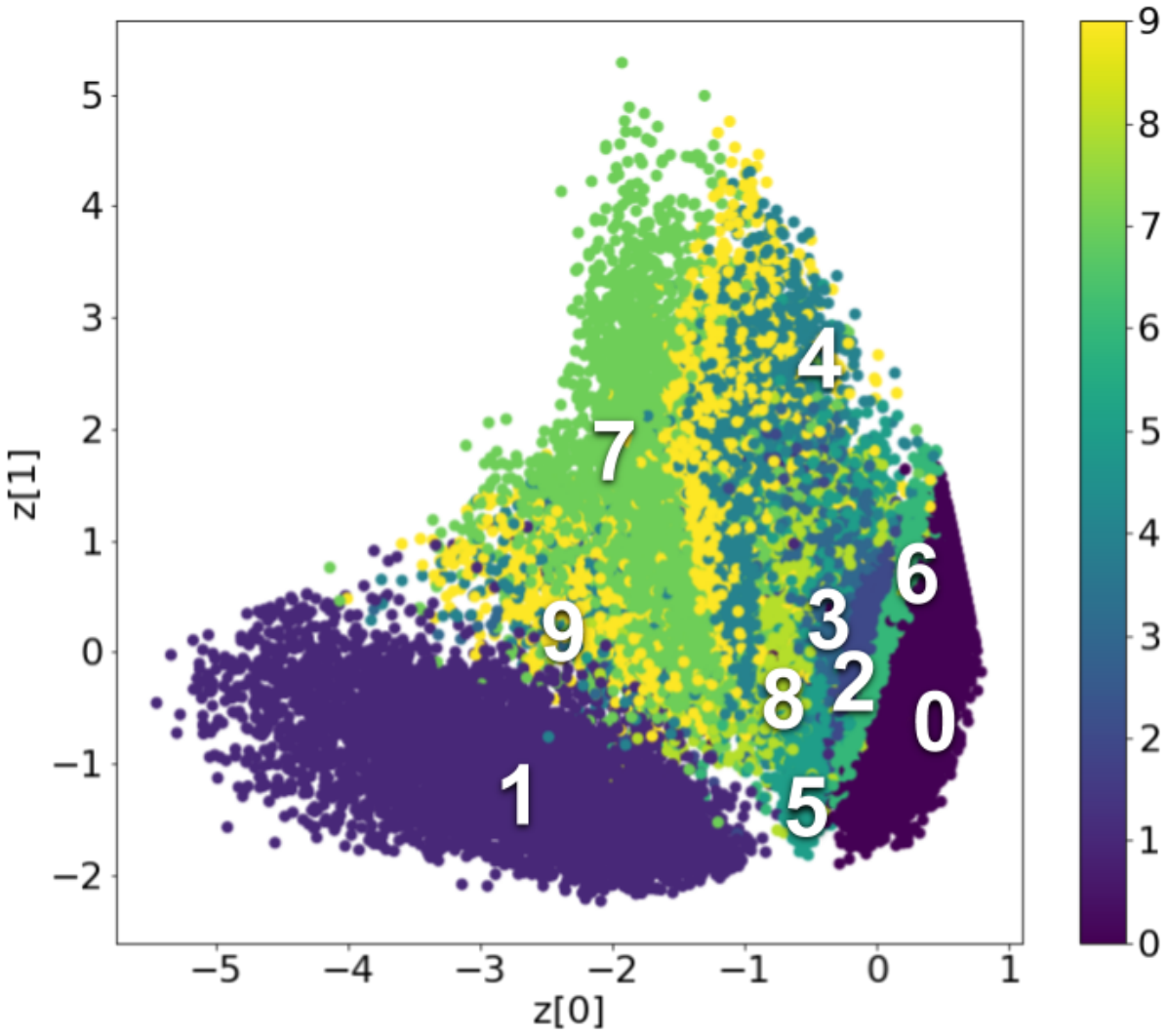
There are multiple ways of visualizing the learned low-dimensional representation, or latent space, in an autoencoder. Due to the additional constraints imposed on the latent representation in variational autoencoders, visualizing the latent space is more straightforward than with traditional autoencoders.

When variational autoencoders are trained on images, a common technique to visualize the latent space is to randomly draw values from the learned distributions and use these values as inputs to the decoder (ignoring the encoder entirely). This produces a novel image sampled from the latent space. For example, after training on MNIST digits, the resultant images will usually resemble specific digits. Note that these generated images are not present in the training set - they are representations of what the network has learned about encoding digits. In fact, by running linearly-spaced sample values through the decoder, you will notice that the output images appear to interpolate between digits. We are thus visualizing samples along a continuous latent space (i.e., a manifold):



<alt text: Visualizing the latent space of the MNIST digit model. We see how the model learns each of the digits as points in the latent space and how areas in between the digits become combinations of the neighboring digits.>

If your training images belong to specific classes, as MNIST digits do, you may also observe potential clustering within the latent space. You might expect each digit to be spatially separated from other digits, and indeed this is the case. The variational autoencoder in this article (which we used for inspiration for this article) encodes MNIST digits in a two-dimensional latent space. In the below figure, each test image from the MNIST dataset is fed through the encoder (the decoder is ignored). The two-dimensional mean vector of the encoder's output is plotted on the x-y plane, and each digit (indicated by the legend on the right using different colors) is fairly well-clustered:



For variational autoencoder networks with larger latent spaces (more than 2 dimensions), common dimensionality reduction algorithms like PCA, t-SNE, or UMAP may help with visualization, though these are outside the scope of our discussion.

Unfortunately, variational autoencoders are more complicated to train and deploy, therefore in the remainder of this course we will focus on standard autoencoders. However, since you may run into these new and powerful techniques in the future we wanted to make sure to introduce you to them now. Next, you will train a fully connected autoencoder to identify abnormal heart rhythms from ECG data and learn how to evaluate an anomaly detection model.