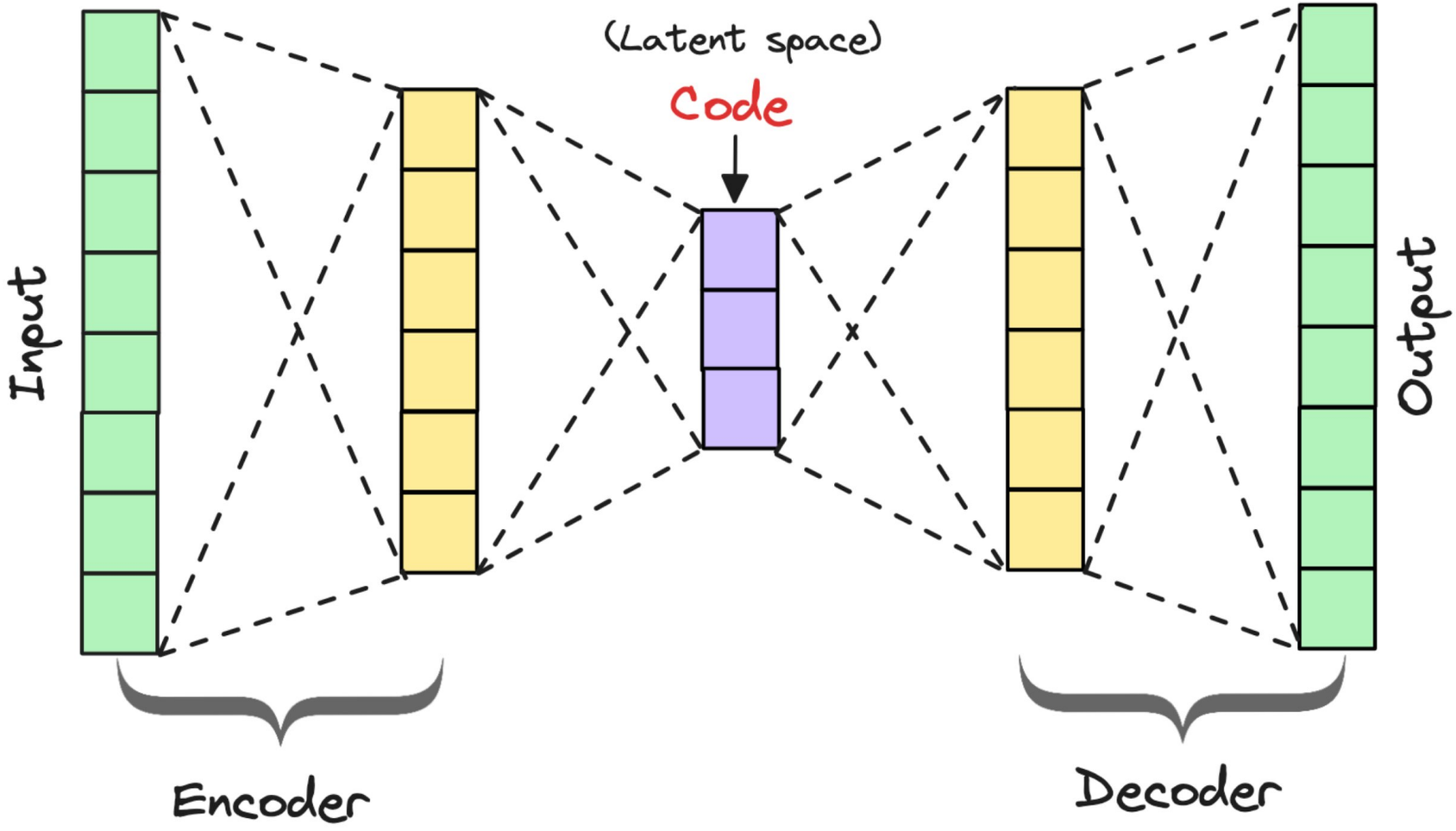
Understanding Autoencoders!🚀



Implementation from scratch using PyTorch Lightning!⚡

@akshay_pachaar

Autoencoders have two main parts:

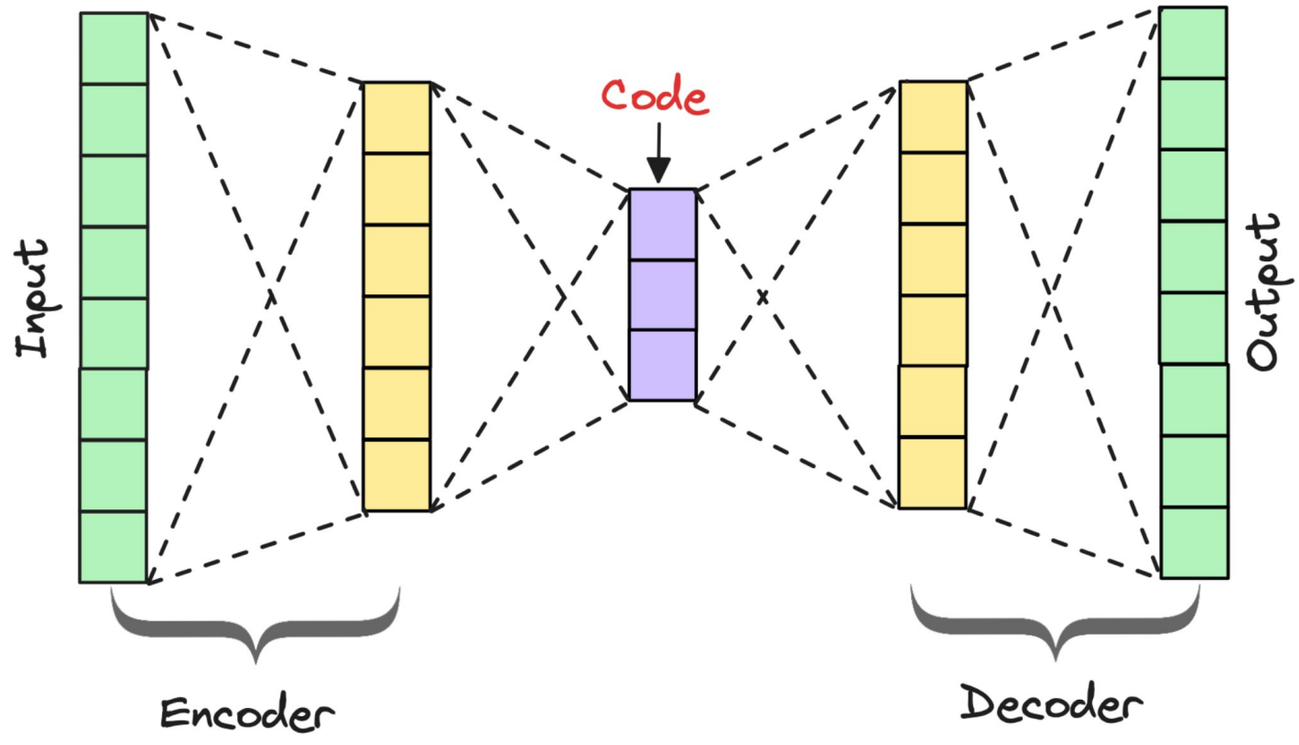
1️⃣ Encoder: Compresses the input into a dense representation (latent space)

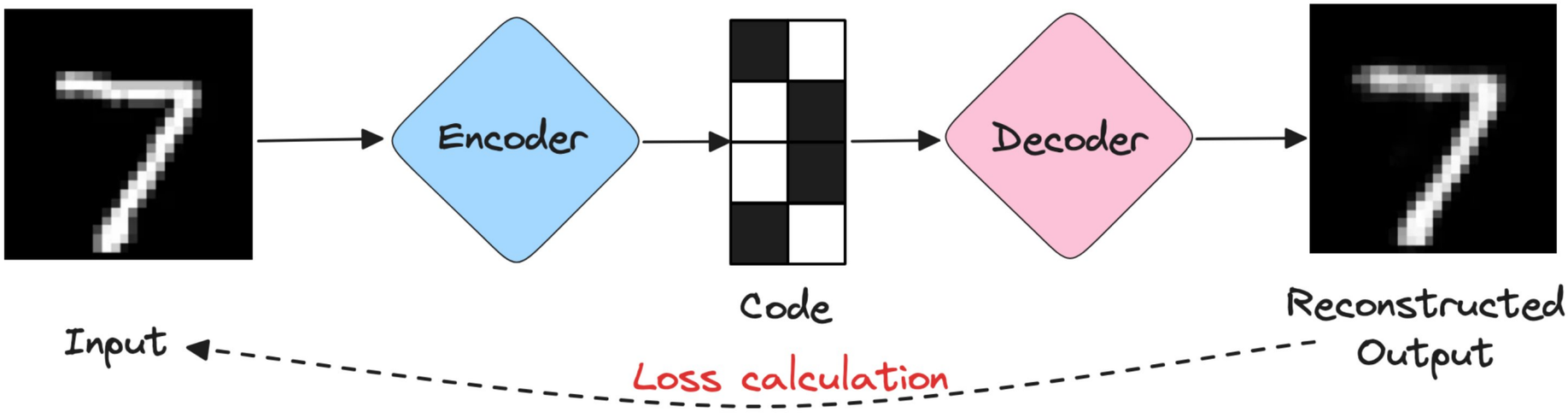
2️⃣ Decoder: Reconstructs the input from this dense representation.

The idea is to make the reconstructed output as close to the original input as possible

Swipe 👉

@akshay_pachaar





Loss is the difference between input and reconstructed output.

We train them using backpropagation, adjusting weights based on the reconstruction loss!

@akshay_pachaar

Applications of Autoencoders:

- Dimensionality Reduction: Like PCA but cooler. 😎
- Anomaly Detection: If reconstruction error is high, something's fishy!
- Data Denoising: Clean noisy data by training on noise.

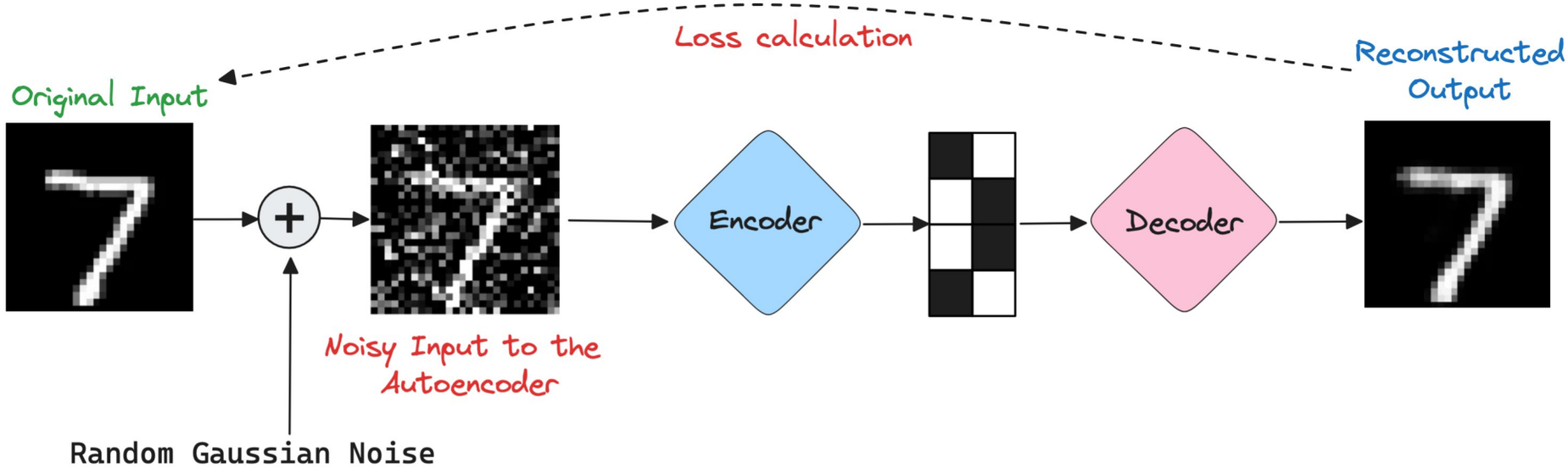
A glimpse on how a denoising autoencoder is trained

Swipe 👉

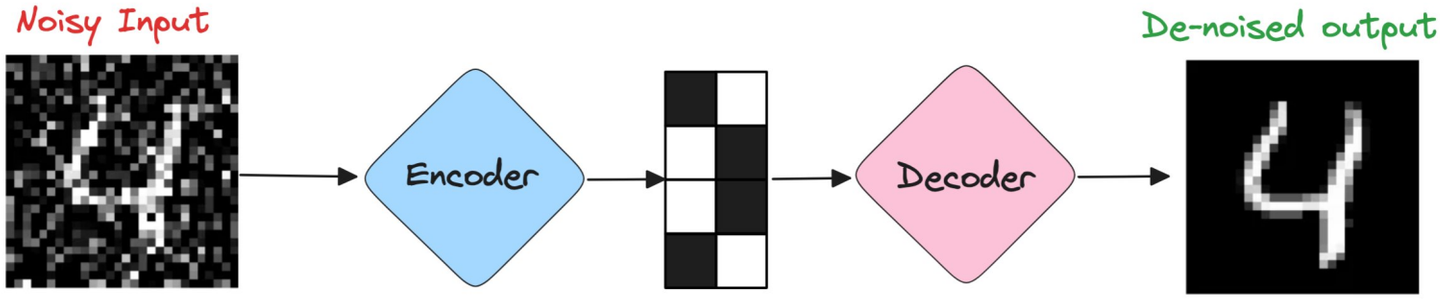
@akshay_pachaar

A denoising Autoencoder!🚀

Training process



Inference



The encoder and decoder learns to remove noise!

@akshay_pachaar

Enough talk! Let's Code!

We'll use PyTorchLightning ⚡ for this.

- First, let's define our autoencoder!

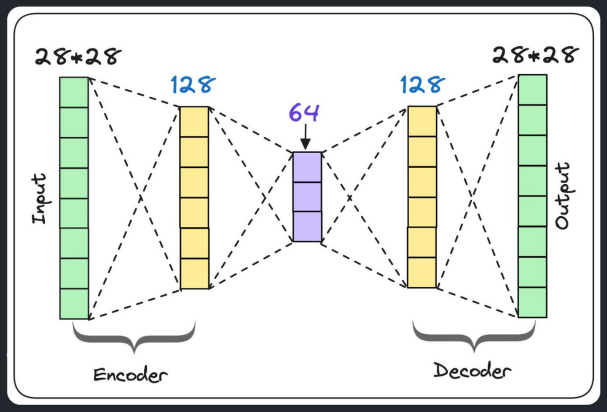
A LightningModule enables your PyTorch nn.Module to play together in complex ways inside the training_step.

- I'll provide link to all the code at the end.

Swipe 👉

@akshay_pachaar

```python
import os
from torch import optim, nn, utils, Tensor
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor
import lightning.pytorch as pl

# define any number of nn.Modules (or use your current ones)
encoder = nn.Sequential(nn.Linear(28 * 28, 128), nn.ReLU(), nn.Linear(128, 64))
decoder = nn.Sequential(nn.Linear(64, 128), nn.ReLU(), nn.Linear(128, 28 * 28))


# define the LightningModule
class LitAutoEncoder(pl.LightningModule):
    def __init__(self, encoder, decoder):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder

    def training_step(self, batch, batch_idx):
        # training_step defines the train loop.
        # it is independent of forward
        x, y = batch
        x = x.view(x.size(0), -1)
        z = self.encoder(x)
        x_hat = self.decoder(z)
        loss = nn.functional.mse_loss(x_hat, x)
        # Logging to TensorBoard (if installed) by default
        self.log("train_loss", loss)
        return loss

    def configure_optimizers(self):
        optimizer = optim.Adam(self.parameters(), lr=1e-3)
        return optimizer


# init the autoencoder
autoencoder = LitAutoEncoder(encoder, decoder)
```

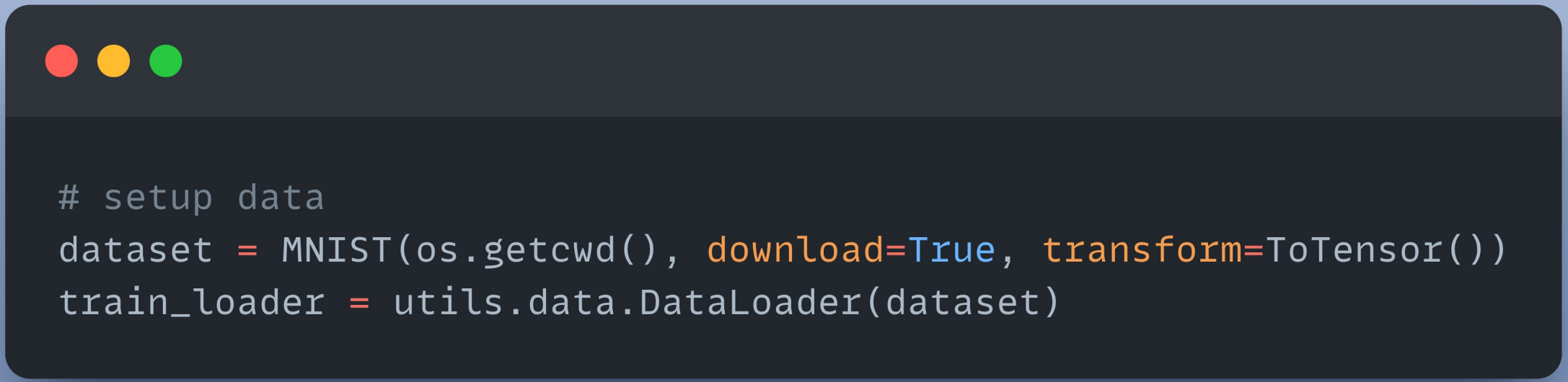


- Define a dataset

Lightning supports ANY iterable (DataLoader, numpy, etc…) for the train/val/test/predict splits.

Swipe 👉

@akshay_pachaar


# setup data
dataset = MNIST(os.getcwd(), download=True, transform=ToTensor())
train_loader = utils.data.DataLoader(dataset)

🔹 Train the model

The Lightning Trainer automates 40+ tricks including:

- Epoch and batch iteration
- optimizer.step(), loss.backward() etc.
- Calling of model.eval()
- enabling/disabling grads during evaluation
- Ckpt Saving and Loading
- Multi-GPU
- 16-bit precision AMP

Swipe 👉

@akshay_pachaar

```
# train the model
# (hint: here are some helpful Trainer arguments for rapid idea iteration)
trainer = pl.Trainer(limit_train_batches=100, max_epochs=1)
trainer.fit(model=autoencoder, train_dataloaders=train_loader)
```

Use the model

Once you've trained the model you can export to onnx, torchscript and put it into production or simply load the weights and run predictions.

Swipe 👉

@akshay_pachaar

```python
# load checkpoint
checkpoint = "./lightning_logs/version_0/checkpoints/epoch=0-step=100.ckpt"
autoencoder = LitAutoEncoder.load_from_checkpoint(checkpoint, encoder=encoder, decoder=decoder)

# choose your trained nn.Module
encoder = autoencoder.encoder
encoder.eval()

# embed 4 fake images!
fake_image_batch = torch.rand(1, 28 * 28, device=autoencoder.device)
embeddings = encoder(fake_image_batch)
print("⚡" * 5, "\nPredictions (Embedding for an image):\n", embedding, "\n", "⚡" * 20)
```

⚡⚡⚡⚡⚡ Predictions (Embedding for an image):

tensor([[-0.0851, 0.1755, 0.3341, -0.3455, 0.1206, 0.3917, ...

-0.2160, 0.1877, 0.4324, -0.0497, -0.1032, 0.0408, -0.1427, -0.0320, 0.6609, - ...]]

Visualize results

Let's encode/decode an image using out trained model!

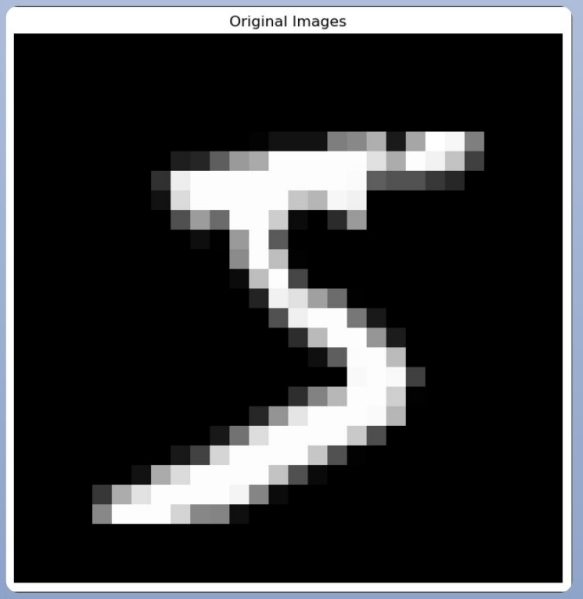
Visualization show an original image which we encode & a reconstructed image!

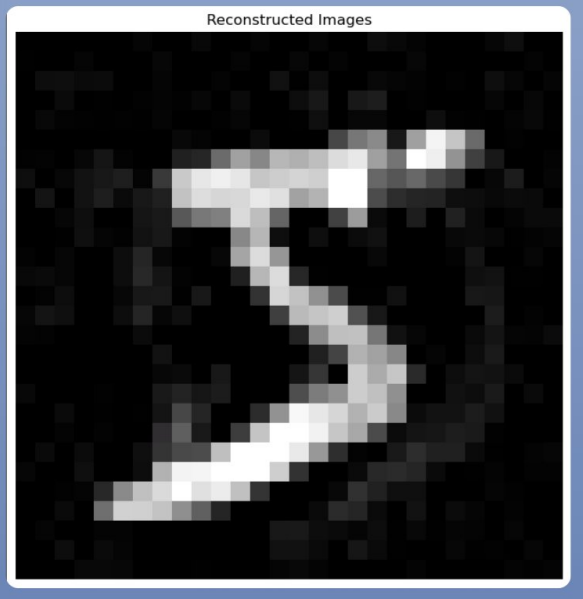
Swipe 👉

@akshay_pachaar

@akshay_pachaar

```python
import matplotlib.pyplot as plt
from torchvision.utils import make_grid

def show_images(images, title="Images"):
    """Utility function to display a batch of images."""
    grid_img = make_grid(images, nrow=4)
    plt.figure(figsize=(8, 8))
    plt.imshow(grid_img.permute(1, 2, 0))
    plt.title(title)
    plt.axis('off')
    plt.show()

# Load a batch of images from the dataset
images, _ = next(iter(train_loader))
show_images(images, title="Original Images")

# Preprocess the images
images = images.view(images.size(0), -1)

# Generate image from embeddings
embeddings = autoencoder.encoder(images)
reconstructed_images =autoencoder.decoder(embeddings).view(-1, 1, 28, 28)
show_images(reconstructed_images, title="Reconstructed Images")
```





That's a wrap!

If you interested in:

- Python 🐍
- Data Science 📈
- Machine Learning 🤖
- MLOps 🛠️
- NLP 🗣️
- Computer Vision 📽️
- LLMs 🧠

Follow me on LinkedIn ✔️
Everyday, I share tutorials on above topics!

Cheers!! 🙂

@akshay_pachaar