Exercise 1: Convolutions of Continuous and Discrete Variables

Random Variables: $Z = X + Y \xrightarrow{\text{PDF}} f_Z(z) = (f_X * f_Y)(z) = \int\limits_{-\infty}^{\infty} f_X(x) f_Y(z-x)\,dx$

$X \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$

To show: $Z = X + Y \sim \mathcal{N}(\mu_Z, \sigma_Z^2)$

$$f_X(x) = \frac{1}{\sqrt{2\pi}\,\sigma_1} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) \quad , f_Y \text{ analogous}$$

$$f_Z(z) = (f_X * f_Y)(z) = \frac{1}{2\pi\sigma_1\sigma_2} \int\limits_{-\infty}^{\infty} dx \, \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2} - \frac{(z-x-\mu_2)^2}{2\sigma_2^2}\right)$$

Consider the exponent:

$$-\frac{(x-\mu_1)^2}{2\sigma_1^2} - \frac{(z-x-\mu_2)^2}{2\sigma^2} = -\frac{x^2 - 2x\mu_1 + \mu_1^2}{2\sigma_1^2} - \frac{x^2 + 2x(\mu_2 - z) + (\mu_2 - z)^2}{2\sigma_2^2}$$

$$= -\frac{1}{2}\left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right) x^2 + x\left(\frac{\mu_1}{\sigma_1^2} + \frac{z-\mu_2}{\sigma_2^2}\right) - \frac{\mu_1^2}{2\sigma_1^2} - \frac{(\mu_2 - z)^2}{2\sigma_2^2}$$

$$= -A(x-B)^2 + C$$

$$A = \frac{1}{2}\left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right), \quad B = \frac{\frac{\mu_1}{\sigma_1^2} + \frac{z-\mu_2}{\sigma_2^2}}{2A}, \quad C = -\frac{\mu_1^2}{2\sigma_1^2} - \frac{(\mu_2 - z)^2}{2\sigma_2^2} + AB^2$$

with $$AB^2 = \frac{\left(\frac{\mu_1}{\sigma_1^2} + \frac{(z-\mu_2)}{\sigma_2^2}\right)^2}{2\left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)} = \frac{\sigma_1^2\sigma_2^2}{2(\sigma_1^2 + \sigma_2^2)}\left(\frac{\mu_1^2}{\sigma_1^4} + \frac{(z-\mu_2)^2}{\sigma_2^4} + \frac{2\mu_1(z-\mu_2)}{\sigma_1^2\sigma_2^2}\right)$$

$$\sigma_Z^2 = \sigma_1^2 + \sigma_2^2 \Rightarrow \frac{1}{2\sigma_Z^2}\left(\mu_1^2 \frac{\sigma_2^2}{\sigma_1^2} + (z-\mu_2)^2 \frac{\sigma_1^2}{\sigma_2^2} + 2\mu_1(z-\mu_2)\right)$$

$$\Rightarrow C = -\left(\frac{\frac{\mu_1^2}{\sigma_1^2}(\sigma_1^2 + \sigma_2^2) + \frac{(z-\mu_2)^2}{\sigma_2^2}(\sigma_1^2 + \sigma_2^2)}{2\sigma_Z^2}\right)$$

$$+ \frac{1}{2\sigma_Z^2}\left(\mu_1^2 \frac{\sigma_2^2}{\sigma_1^2} + (z-\mu_2)^2 \frac{\sigma_1^2}{\sigma_2^2} + 2\mu_1(z-\mu_2)\right)$$

$$= -\left(\frac{\mu_1^2\left(1 + \cancel{\frac{\sigma_2^2}{\sigma_1^2}}\right) + (z-\mu_2)^2\left(1 + \cancel{\frac{\sigma_1^2}{\sigma_2^2}}\right)}{2\sigma_Z^2}\right) + \frac{1}{2\sigma_Z^2}\left(\cancel{\mu_1^2 \frac{\sigma_2^2}{\sigma_1^2}} + \cancel{(z-\mu_2)^2 \frac{\sigma_1^2}{\sigma_2^2}} + 2\mu_1(z-\mu_2)\right)$$

$$= -\frac{\mu_1^2 + (z-\mu_2)^2 - 2\mu_1(z-\mu_2)}{2\sigma_Z^2}$$

$$= -\frac{\mu_1^2 + z^2 + \mu_2^2 - 2\mu_2 z - 2\mu_1 z + 2\mu_1\mu_2}{2\sigma_Z^2}$$

$$= -\frac{\left(z - \overbrace{(\mu_1 + \mu_2)}^{\mu_Z}\right)^2}{2\sigma_Z^2} = -\frac{(z-\mu_Z)^2}{2\sigma_Z^2}$$

$$\Rightarrow \quad f_z(z) = \frac{e^C}{2\pi\sigma_1\sigma_2}\int_{-\infty}^{\infty} dx \ \exp\left(-A(x-B)^2\right) = \frac{e^C}{2\pi\sigma_1\sigma_2}\sqrt{\frac{\pi}{A}}$$

$$= \frac{1}{2\pi\sigma_1\sigma_2}\sqrt{\frac{2\pi}{\frac{1}{\sigma_1^2}+\frac{1}{\sigma_2^2}}} \ \exp\left(-\frac{(z-\mu_z)^2}{2\sigma_z^2}\right)$$

with $\mu_z = \mu_1 + \mu_2$, $\quad \sigma_z^2 = \sigma_1^2 + \sigma_2^2$

2.) $I \in \mathbb{R}^{H\times W}$, $f \in \mathbb{R}^{K_H \times K_W}$ with $K_H < H$, $K_W < W$

and $I_{pad} \in \mathbb{R}^{(H+2p)\times(W+2p)}$, stride $s \in \mathbb{N}$

$$O(i,j) = \sum_{u=0}^{K_H}\sum_{v=0}^{K_W} I_{pad}(i\cdot s+u,\, j\cdot s+v)\cdot f(u,v)$$

$$O \in \mathbb{R}^{H_{out}\times W_{out}}$$

because we can't move closer to the edge than the dimension of the filter

where $H_{out} = \frac{(H+2p)-K_H}{s} + 1$

$W_{out} = \frac{(W+2p)-K_W}{s} + 1$

because we start counting at 0 (ie. $H_{out} = 1$ if $H+2p = K_H$)

Machine Learning Essentials SS25 - Exercise Sheet 6

Instructions

- `TODO` 's indicate where you need to complete the implementations.
- You may use external resources, but **write your own solutions**.
- Provide concise, but comprehensible comments to explain what your code does.
- Code that's unnecessarily extensive and/or not well commented will not be scored.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import torch as tc
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchsummary import summary

np.random.seed(42)
tc.manual_seed(42)

device = tc.device("cuda" if tc.cuda.is_available() else "cpu")
```

Exercise 2 - CNN Classifier

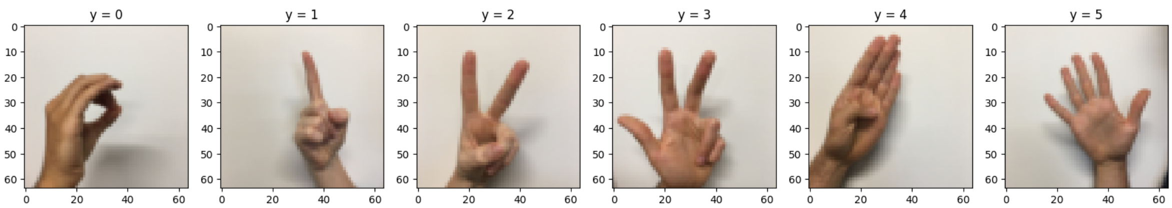
The SIGNS dataset is a collection of 6 signs representing numbers from 0 to 5. We first load the data and have the shapes printed out. The split into train, validation and test set has already been carried out.

In [2]:
```
# Load the dataset
X_train = np.load('sign_data/X_train.npy')
Y_train = np.load('sign_data/Y_train.npy')
X_val = np.load('sign_data/X_val.npy')
Y_val = np.load('sign_data/Y_val.npy')
X_test = np.load('sign_data/X_test.npy')
Y_test = np.load('sign_data/Y_test.npy')

# print the shape of the dataset
print("X_train shape: " + str(X_train.shape))
print("Y_train shape: " + str(Y_train.shape))
print("X_val shape: " + str(X_val.shape))
print("Y_val shape: " + str(Y_val.shape))
print("X_test shape: " + str(X_test.shape))
print("Y_test shape: " + str(Y_test.shape)+"\n")
print("classes: " + str(np.unique(Y_train)))

# check if classes are balanced
print("Counts of classes in Y_train: " + str(np.unique(Y_train, return_co
print("Counts of classes in Y_val: " + str(np.unique(Y_val, return_counts
print("Counts of classes in Y_test: " + str(np.unique(Y_test, return_coun
```

```
X_train shape: (960, 64, 64, 3)
Y_train shape: (960,)
X_val shape: (120, 64, 64, 3)
Y_val shape: (120,)
X_test shape: (120, 64, 64, 3)
Y_test shape: (120,)

classes: [0 1 2 3 4 5]
Counts of classes in Y_train: [160 160 160 160 160 160]
Counts of classes in Y_val: [20 20 20 20 20 20]
Counts of classes in Y_test: [20 20 20 20 20 20]
```

The classes are balanced so that accuracy is an appropriate measure for evaluating a classifier. We next visualize an instance of each class.

In [3]:
```
fig, axs = plt.subplots(1, 6, figsize=(20, 10))
for i in range(6):
    # get indices where the label is i
    idx = np.where(Y_train == i)[0][0]
    axs[i].imshow(X_train[idx])
    axs[i].set_title("y = " + str(i))
```



Pixels in each channel (RGB) of the images take values in the range [0, 255]. However, it is desirable to have absolute values in the range [0, 1] as input for neural network architectures to avoid exploding or vanishing gradient problems. Through the following cell, we apply a simple data scaling procedure: we divide the values of the pixels by 255. As an alternative, you can use the `StandardScaler()` function of the scikit-learn library.

In [4]:
```
X_train = X_train/255
X_val = X_val/255
X_test = X_test/255
```

In [5]:
```
X_train[0].shape
```

Out[5]: (64, 64, 3)

Task 1

Use pytorch to build the model. Take a look at the documentation for an introduction, a detailed tutorial, for example for classifiers, can be found here.

Implement the following architecture:

```
 - Conv2d: 4 output channels, 3 by 3 filter size, stride
1, padding "same"
 - BatchNorm2d: 4 output channels
 - ReLU activation
 - MaxPool2d: 2 by 2 filter size, stride 2, padding 0
 - Conv2d: 8 output channels, 3 by 3 filter size, stride
1, padding "same"
 - BatchNorm2d: 8 output channels
 - ReLU activation
 - MaxPool2d: Use a 2 by 2 filter size, stride 2, padding
0
 - Flatten the previous output
 - Linear: 64 output neurons
 - ReLu activation function
 - Linear: 6 output neurons
 - LogSoftmax
```

We use the LogSoftmax here instead of the Softmax for computational reasons. Accordingly, the loss function is not CrossEntropyLoss but NLLLoss. When flattening, be careful not to do it with the batch dimension but only with the height, width and channel dimension.

In [6]:
```
class CNN_Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.first_conv = nn.Sequential(
            nn.Conv2d(3, 4, (3, 3), 1, "same"),
            nn.BatchNorm2d(4),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),
        )
        # output dim 32 [(64-2)/2 + 1]

        self.second_conv = nn.Sequential(
            nn.Conv2d(4, 8, (3,3), 1, "same"),
            nn.BatchNorm2d(8),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),
```

```
        )
        # output dim = 16 [(32 - 2)/2 + 1]

        self.lin_layer = nn.Sequential(
            nn.Flatten(),
            # output dim = 16 * 16 * 8 = 2048
            nn.Linear(2048, 64),
            nn.ReLU(),
            nn.Linear(64, 6)
        )

    def forward(self, X):
        # TODO: Implement the forward pass
        X = self.first_conv(X)
        X = self.second_conv(X)
        X = self.lin_layer(X)
        return nn.LogSoftmax(dim=1)(X)
```

To test your model you can foward some random numbers. The shape of the output should be (2, 6).

In [7]:
```
cnn_model = CNN_Classifier()
# dummy sample of batch size 2
X_random = tc.randn(2, 3, 64, 64)
output = cnn_model(X_random)

print("Output shape: " + str(output.shape))
```

```
Output shape: torch.Size([2, 6])
```

torchsummary.summary provides a nice overview of the model and the number of learnable parameters:

In [8]:
```
summary(cnn_model, input_size=(3, 64, 64), device="cpu")
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 4, 64, 64]             112
       BatchNorm2d-2            [-1, 4, 64, 64]               8
              ReLU-3            [-1, 4, 64, 64]               0
         MaxPool2d-4            [-1, 4, 32, 32]               0
            Conv2d-5            [-1, 8, 32, 32]             296
       BatchNorm2d-6            [-1, 8, 32, 32]              16
              ReLU-7            [-1, 8, 32, 32]               0
         MaxPool2d-8            [-1, 8, 16, 16]               0
           Flatten-9                 [-1, 2048]               0
           Linear-10                   [-1, 64]         131,136
             ReLU-11                   [-1, 64]               0
           Linear-12                    [-1, 6]             390
================================================================
Total params: 131,958
Trainable params: 131,958
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.63
Params size (MB): 0.50
Estimated Total Size (MB): 1.18
----------------------------------------------------------------
```

Task 2

DataLoaders wrap around Datasets to provide efficient data batching, shuffling, and parallel loading during model training or inference. To define a custom dataset we must implement three functions: **init**, **len** and **get_item**. While **len** defines the length of the dataset and thus the number of batches in the dataloader, **get_item** can be used to get a single sample through the index.

In [9]:
```
class Image_Dataset(Dataset):
    def __init__(self, X, Y):
        self.X = X
        self.Y = Y

    def __len__(self):
        # TODO: Return the length of the dataset
        return len(self.X)

    def __getitem__(self, idx):
        # TODO: Return the image as a float tensor and the label as a lon
        img = self.X[idx]
        label = np.array(self.Y[idx])
        img = tc.from_numpy(img).float()

        #print(label.size())

        # permute the shape to (3, 64, 64)
        img = img.permute(2, 0, 1)
        label = tc.from_numpy(label).long()
        return img, label
```

In [10]:
```
train_batch_size = 64
```

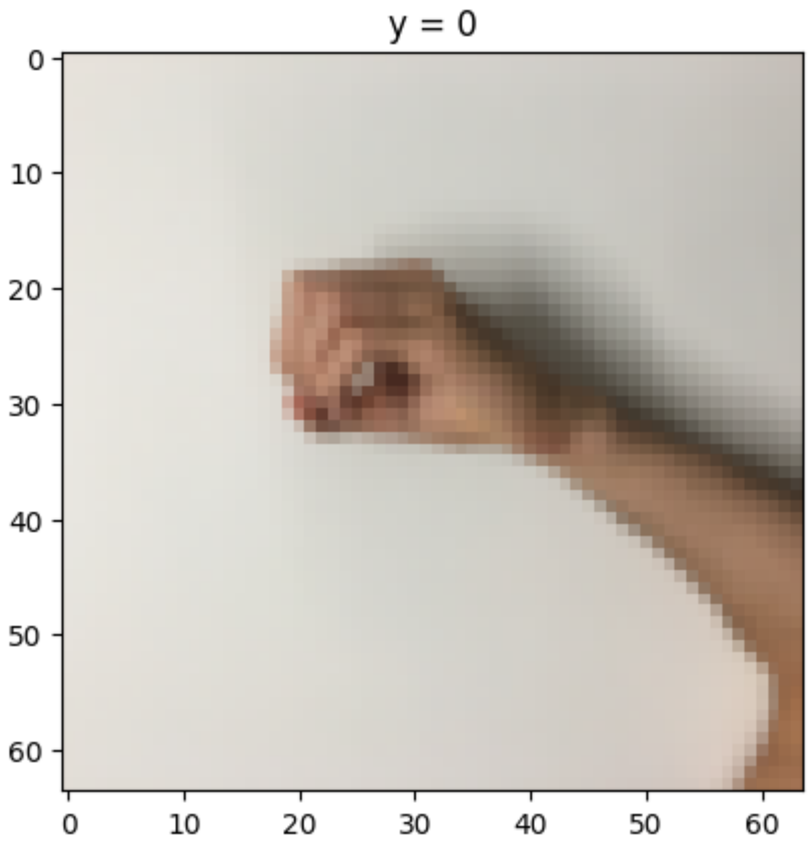
```
val_batch_size = len(Y_val)
test_batch_size = len(Y_test)

# TODO: Create the dataset and dataloader
train_dataset = Image_Dataset(X_train, Y_train)
val_dataset = Image_Dataset(X_val, Y_val)
test_dataset = Image_Dataset(X_test, Y_test)
train_loader = DataLoader(train_dataset, train_batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, val_batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, test_batch_size, shuffle=True)
```

To make sure that everything has worked properly, we take a sample of the data_loader and visualize it.

In [11]:

```
sample_X, sample_Y = next(iter(train_loader))
plt.imshow(sample_X[0].T)
plt.title("y = " + str(int(sample_Y[0].item())))
plt.show()
```

```
/tmp/ipykernel_1184/2461653407.py:2: UserWarning: The use of `x.T` on tens
ors of dimension other than 2 to reverse their shape is deprecated and it
will throw an error in a future release. Consider `x.mT` to transpose batc
hes of matrices or `x.permute(*torch.arange(x.ndim - 1, -1, -1))` to rever
se the dimensions of a tensor. (Triggered internally at /pytorch/aten/src/
ATen/native/TensorShape.cpp:4413.)
  plt.imshow(sample_X[0].T)
```



Task 3

Implement the training loop. Use the negative log-likelihood loss (NLLLoss) and the Adam optimizer. Be sure to zero the gradients after each optimization step to avoid accumulating contributions from previous epochs and batches.

In [12]:
```
def train_cnn(model, train_loader, val_loader, lr, n_epochs, device):
    model = model.to(device)

    # TODO: Initialize the optimizer and loss function
    loss_function = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    train_loss = np.zeros(n_epochs)
    val_loss = np.zeros(n_epochs)
    train_acc = np.zeros(n_epochs)
    val_acc = np.zeros(n_epochs)

    for epoch in range(1, n_epochs + 1):
        model.train()

        epoch_loss = 0

        for X, Y in train_loader:
            X, Y = X.to(device), Y.to(device)
            # TODO: Implement the training loop

            optimizer.zero_grad()

            output = model(X)

            loss = loss_function(output, Y)

            loss.backward()

            optimizer.step()


            epoch_loss += loss.item()/len(train_loader)

        train_loss[epoch - 1] = epoch_loss
        train_acc[epoch - 1] = (output.argmax(dim=1) == Y).float().mean()

        model.eval()

        with tc.no_grad():
            X, Y = next(iter(val_loader))
            X, Y = X.to(device), Y.to(device)
            # TODO: Implement the evaluation step
            output = model(X)
            loss = loss_function(output, Y)

            val_loss[epoch - 1] = loss.item()
            val_acc[epoch - 1] = (output.argmax(dim=1) == Y).float().mean
        print(f"Epoch {epoch}/{n_epochs} - Train Loss: {epoch_loss:.4f},

    return train_loss, val_loss, train_acc, val_acc
```

In [13]:
```
n_epochs = 50
# TODO: Train the model with different learning rates
```

```
learning_rates = [0.00001, 0.0001, 0.001, 0.1]
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

models = [CNN_Classifier() for _ in learning_rates]
for lr in learning_rates:
    model = models[learning_rates.index(lr)]
    train_loss, val_loss, train_acc, val_acc = train_cnn(model, train_loa
    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)
```

```
Epoch 1/50 - Train Loss: 1.7900, Test Loss: 1.7928
Epoch 2/50 - Train Loss: 1.7706, Test Loss: 1.7889
Epoch 3/50 - Train Loss: 1.7515, Test Loss: 1.7732
Epoch 4/50 - Train Loss: 1.7318, Test Loss: 1.7429
Epoch 5/50 - Train Loss: 1.7109, Test Loss: 1.7155
Epoch 6/50 - Train Loss: 1.6911, Test Loss: 1.6952
Epoch 7/50 - Train Loss: 1.6717, Test Loss: 1.6775
Epoch 8/50 - Train Loss: 1.6523, Test Loss: 1.6594
Epoch 9/50 - Train Loss: 1.6329, Test Loss: 1.6422
Epoch 10/50 - Train Loss: 1.6127, Test Loss: 1.6240
Epoch 11/50 - Train Loss: 1.5940, Test Loss: 1.6060
Epoch 12/50 - Train Loss: 1.5749, Test Loss: 1.5880
Epoch 13/50 - Train Loss: 1.5563, Test Loss: 1.5711
Epoch 14/50 - Train Loss: 1.5377, Test Loss: 1.5546
Epoch 15/50 - Train Loss: 1.5185, Test Loss: 1.5382
Epoch 16/50 - Train Loss: 1.5005, Test Loss: 1.5214
Epoch 17/50 - Train Loss: 1.4825, Test Loss: 1.5052
Epoch 18/50 - Train Loss: 1.4637, Test Loss: 1.4895
Epoch 19/50 - Train Loss: 1.4476, Test Loss: 1.4744
Epoch 20/50 - Train Loss: 1.4303, Test Loss: 1.4584
Epoch 21/50 - Train Loss: 1.4126, Test Loss: 1.4437
Epoch 22/50 - Train Loss: 1.3955, Test Loss: 1.4292
Epoch 23/50 - Train Loss: 1.3794, Test Loss: 1.4140
Epoch 24/50 - Train Loss: 1.3633, Test Loss: 1.3998
Epoch 25/50 - Train Loss: 1.3459, Test Loss: 1.3849
Epoch 26/50 - Train Loss: 1.3298, Test Loss: 1.3706
Epoch 27/50 - Train Loss: 1.3131, Test Loss: 1.3565
Epoch 28/50 - Train Loss: 1.2966, Test Loss: 1.3424
Epoch 29/50 - Train Loss: 1.2812, Test Loss: 1.3286
Epoch 30/50 - Train Loss: 1.2654, Test Loss: 1.3156
Epoch 31/50 - Train Loss: 1.2515, Test Loss: 1.3024
Epoch 32/50 - Train Loss: 1.2355, Test Loss: 1.2892
Epoch 33/50 - Train Loss: 1.2210, Test Loss: 1.2759
Epoch 34/50 - Train Loss: 1.2069, Test Loss: 1.2634
Epoch 35/50 - Train Loss: 1.1923, Test Loss: 1.2503
Epoch 36/50 - Train Loss: 1.1774, Test Loss: 1.2388
Epoch 37/50 - Train Loss: 1.1633, Test Loss: 1.2263
Epoch 38/50 - Train Loss: 1.1507, Test Loss: 1.2139
Epoch 39/50 - Train Loss: 1.1362, Test Loss: 1.2013
Epoch 40/50 - Train Loss: 1.1217, Test Loss: 1.1895
Epoch 41/50 - Train Loss: 1.1086, Test Loss: 1.1784
Epoch 42/50 - Train Loss: 1.0950, Test Loss: 1.1654
Epoch 43/50 - Train Loss: 1.0829, Test Loss: 1.1542
Epoch 44/50 - Train Loss: 1.0702, Test Loss: 1.1441
Epoch 45/50 - Train Loss: 1.0581, Test Loss: 1.1335
Epoch 46/50 - Train Loss: 1.0446, Test Loss: 1.1220
Epoch 47/50 - Train Loss: 1.0339, Test Loss: 1.1125
Epoch 48/50 - Train Loss: 1.0229, Test Loss: 1.1024
Epoch 49/50 - Train Loss: 1.0103, Test Loss: 1.0937
Epoch 50/50 - Train Loss: 0.9999, Test Loss: 1.0852
Epoch 1/50 - Train Loss: 1.7328, Test Loss: 1.7763
Epoch 2/50 - Train Loss: 1.5836, Test Loss: 1.6818
Epoch 3/50 - Train Loss: 1.4290, Test Loss: 1.4960
Epoch 4/50 - Train Loss: 1.2821, Test Loss: 1.3244
Epoch 5/50 - Train Loss: 1.1452, Test Loss: 1.1932
Epoch 6/50 - Train Loss: 1.0275, Test Loss: 1.0893
Epoch 7/50 - Train Loss: 0.9219, Test Loss: 1.0050
Epoch 8/50 - Train Loss: 0.8322, Test Loss: 0.9339
Epoch 9/50 - Train Loss: 0.7531, Test Loss: 0.8691
Epoch 10/50 - Train Loss: 0.6791, Test Loss: 0.8124
```

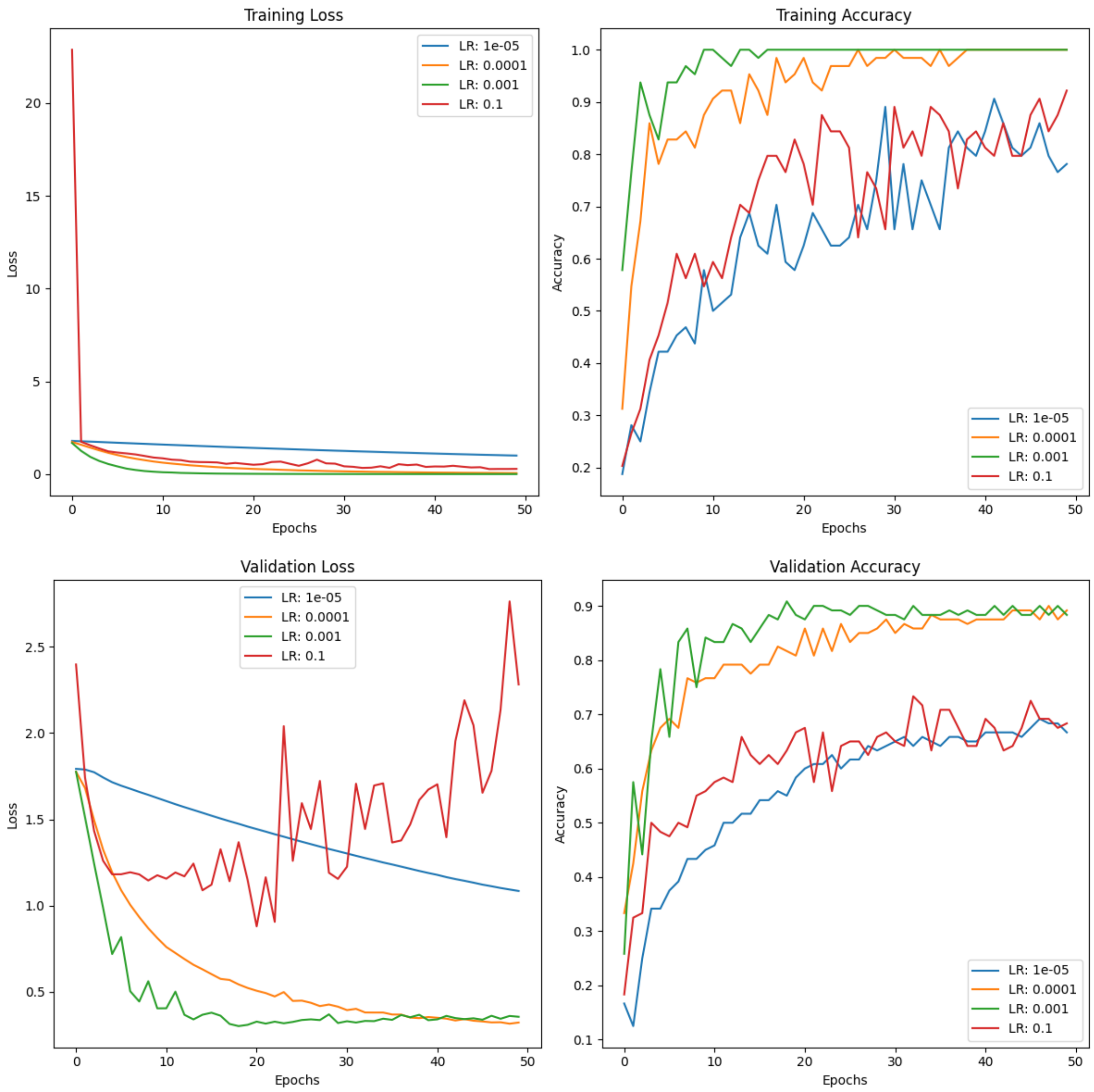
```
Epoch 11/50 - Train Loss: 0.6182, Test Loss: 0.7605
Epoch 12/50 - Train Loss: 0.5684, Test Loss: 0.7257
Epoch 13/50 - Train Loss: 0.5235, Test Loss: 0.6908
Epoch 14/50 - Train Loss: 0.4764, Test Loss: 0.6575
Epoch 15/50 - Train Loss: 0.4430, Test Loss: 0.6312
Epoch 16/50 - Train Loss: 0.4082, Test Loss: 0.6031
Epoch 17/50 - Train Loss: 0.3759, Test Loss: 0.5758
Epoch 18/50 - Train Loss: 0.3474, Test Loss: 0.5696
Epoch 19/50 - Train Loss: 0.3236, Test Loss: 0.5438
Epoch 20/50 - Train Loss: 0.3025, Test Loss: 0.5231
Epoch 21/50 - Train Loss: 0.2816, Test Loss: 0.5069
Epoch 22/50 - Train Loss: 0.2647, Test Loss: 0.4935
Epoch 23/50 - Train Loss: 0.2524, Test Loss: 0.4739
Epoch 24/50 - Train Loss: 0.2338, Test Loss: 0.4992
Epoch 25/50 - Train Loss: 0.2192, Test Loss: 0.4479
Epoch 26/50 - Train Loss: 0.2026, Test Loss: 0.4497
Epoch 27/50 - Train Loss: 0.1934, Test Loss: 0.4371
Epoch 28/50 - Train Loss: 0.1814, Test Loss: 0.4181
Epoch 29/50 - Train Loss: 0.1686, Test Loss: 0.4272
Epoch 30/50 - Train Loss: 0.1601, Test Loss: 0.4147
Epoch 31/50 - Train Loss: 0.1506, Test Loss: 0.3947
Epoch 32/50 - Train Loss: 0.1417, Test Loss: 0.4022
Epoch 33/50 - Train Loss: 0.1352, Test Loss: 0.3811
Epoch 34/50 - Train Loss: 0.1251, Test Loss: 0.3807
Epoch 35/50 - Train Loss: 0.1196, Test Loss: 0.3808
Epoch 36/50 - Train Loss: 0.1154, Test Loss: 0.3693
Epoch 37/50 - Train Loss: 0.1076, Test Loss: 0.3690
Epoch 38/50 - Train Loss: 0.1014, Test Loss: 0.3516
Epoch 39/50 - Train Loss: 0.0973, Test Loss: 0.3483
Epoch 40/50 - Train Loss: 0.0912, Test Loss: 0.3546
Epoch 41/50 - Train Loss: 0.0865, Test Loss: 0.3504
Epoch 42/50 - Train Loss: 0.0824, Test Loss: 0.3455
Epoch 43/50 - Train Loss: 0.0800, Test Loss: 0.3346
Epoch 44/50 - Train Loss: 0.0739, Test Loss: 0.3427
Epoch 45/50 - Train Loss: 0.0710, Test Loss: 0.3334
Epoch 46/50 - Train Loss: 0.0676, Test Loss: 0.3292
Epoch 47/50 - Train Loss: 0.0648, Test Loss: 0.3236
Epoch 48/50 - Train Loss: 0.0621, Test Loss: 0.3245
Epoch 49/50 - Train Loss: 0.0587, Test Loss: 0.3160
Epoch 50/50 - Train Loss: 0.0559, Test Loss: 0.3228
Epoch 1/50 - Train Loss: 1.6778, Test Loss: 1.7759
Epoch 2/50 - Train Loss: 1.2531, Test Loss: 1.5112
Epoch 3/50 - Train Loss: 0.9335, Test Loss: 1.2444
Epoch 4/50 - Train Loss: 0.7094, Test Loss: 0.9866
Epoch 5/50 - Train Loss: 0.5439, Test Loss: 0.7196
Epoch 6/50 - Train Loss: 0.4163, Test Loss: 0.8180
Epoch 7/50 - Train Loss: 0.2978, Test Loss: 0.5042
Epoch 8/50 - Train Loss: 0.2246, Test Loss: 0.4446
Epoch 9/50 - Train Loss: 0.1669, Test Loss: 0.5622
Epoch 10/50 - Train Loss: 0.1306, Test Loss: 0.4050
Epoch 11/50 - Train Loss: 0.1020, Test Loss: 0.4054
Epoch 12/50 - Train Loss: 0.0892, Test Loss: 0.5016
Epoch 13/50 - Train Loss: 0.0654, Test Loss: 0.3675
Epoch 14/50 - Train Loss: 0.0554, Test Loss: 0.3410
Epoch 15/50 - Train Loss: 0.0445, Test Loss: 0.3679
Epoch 16/50 - Train Loss: 0.0369, Test Loss: 0.3798
Epoch 17/50 - Train Loss: 0.0302, Test Loss: 0.3623
Epoch 18/50 - Train Loss: 0.0262, Test Loss: 0.3146
Epoch 19/50 - Train Loss: 0.0234, Test Loss: 0.3022
Epoch 20/50 - Train Loss: 0.0194, Test Loss: 0.3095
```

```
Epoch 21/50 - Train Loss: 0.0171, Test Loss: 0.3281
Epoch 22/50 - Train Loss: 0.0148, Test Loss: 0.3165
Epoch 23/50 - Train Loss: 0.0132, Test Loss: 0.3279
Epoch 24/50 - Train Loss: 0.0113, Test Loss: 0.3183
Epoch 25/50 - Train Loss: 0.0105, Test Loss: 0.3264
Epoch 26/50 - Train Loss: 0.0094, Test Loss: 0.3375
Epoch 27/50 - Train Loss: 0.0085, Test Loss: 0.3413
Epoch 28/50 - Train Loss: 0.0078, Test Loss: 0.3371
Epoch 29/50 - Train Loss: 0.0072, Test Loss: 0.3701
Epoch 30/50 - Train Loss: 0.0067, Test Loss: 0.3199
Epoch 31/50 - Train Loss: 0.0061, Test Loss: 0.3304
Epoch 32/50 - Train Loss: 0.0056, Test Loss: 0.3228
Epoch 33/50 - Train Loss: 0.0052, Test Loss: 0.3322
Epoch 34/50 - Train Loss: 0.0049, Test Loss: 0.3309
Epoch 35/50 - Train Loss: 0.0045, Test Loss: 0.3451
Epoch 36/50 - Train Loss: 0.0042, Test Loss: 0.3385
Epoch 37/50 - Train Loss: 0.0039, Test Loss: 0.3665
Epoch 38/50 - Train Loss: 0.0040, Test Loss: 0.3534
Epoch 39/50 - Train Loss: 0.0036, Test Loss: 0.3676
Epoch 40/50 - Train Loss: 0.0034, Test Loss: 0.3364
Epoch 41/50 - Train Loss: 0.0031, Test Loss: 0.3412
Epoch 42/50 - Train Loss: 0.0031, Test Loss: 0.3612
Epoch 43/50 - Train Loss: 0.0028, Test Loss: 0.3486
Epoch 44/50 - Train Loss: 0.0027, Test Loss: 0.3427
Epoch 45/50 - Train Loss: 0.0026, Test Loss: 0.3474
Epoch 46/50 - Train Loss: 0.0024, Test Loss: 0.3392
Epoch 47/50 - Train Loss: 0.0023, Test Loss: 0.3619
Epoch 48/50 - Train Loss: 0.0022, Test Loss: 0.3447
Epoch 49/50 - Train Loss: 0.0021, Test Loss: 0.3608
Epoch 50/50 - Train Loss: 0.0020, Test Loss: 0.3562
Epoch 1/50 - Train Loss: 22.8699, Test Loss: 2.3977
Epoch 2/50 - Train Loss: 1.7445, Test Loss: 1.7409
Epoch 3/50 - Train Loss: 1.5566, Test Loss: 1.4337
Epoch 4/50 - Train Loss: 1.3847, Test Loss: 1.2602
Epoch 5/50 - Train Loss: 1.2266, Test Loss: 1.1817
Epoch 6/50 - Train Loss: 1.1633, Test Loss: 1.1820
Epoch 7/50 - Train Loss: 1.1160, Test Loss: 1.1936
Epoch 8/50 - Train Loss: 1.0604, Test Loss: 1.1813
Epoch 9/50 - Train Loss: 0.9786, Test Loss: 1.1458
Epoch 10/50 - Train Loss: 0.8941, Test Loss: 1.1762
Epoch 11/50 - Train Loss: 0.8522, Test Loss: 1.1564
Epoch 12/50 - Train Loss: 0.7800, Test Loss: 1.1923
Epoch 13/50 - Train Loss: 0.7536, Test Loss: 1.1700
Epoch 14/50 - Train Loss: 0.6717, Test Loss: 1.2439
Epoch 15/50 - Train Loss: 0.6508, Test Loss: 1.0891
Epoch 16/50 - Train Loss: 0.6444, Test Loss: 1.1217
Epoch 17/50 - Train Loss: 0.6297, Test Loss: 1.3271
Epoch 18/50 - Train Loss: 0.5554, Test Loss: 1.1415
Epoch 19/50 - Train Loss: 0.6024, Test Loss: 1.3680
Epoch 20/50 - Train Loss: 0.5529, Test Loss: 1.1485
Epoch 21/50 - Train Loss: 0.5047, Test Loss: 0.8801
Epoch 22/50 - Train Loss: 0.5346, Test Loss: 1.1646
Epoch 23/50 - Train Loss: 0.6528, Test Loss: 0.9063
Epoch 24/50 - Train Loss: 0.6743, Test Loss: 2.0402
Epoch 25/50 - Train Loss: 0.5574, Test Loss: 1.2598
Epoch 26/50 - Train Loss: 0.4491, Test Loss: 1.5941
Epoch 27/50 - Train Loss: 0.5834, Test Loss: 1.4442
Epoch 28/50 - Train Loss: 0.7787, Test Loss: 1.7228
Epoch 29/50 - Train Loss: 0.5801, Test Loss: 1.1907
Epoch 30/50 - Train Loss: 0.5644, Test Loss: 1.1553
```

```
Epoch 31/50 - Train Loss: 0.4215, Test Loss: 1.2258
Epoch 32/50 - Train Loss: 0.3945, Test Loss: 1.7066
Epoch 33/50 - Train Loss: 0.3358, Test Loss: 1.4442
Epoch 34/50 - Train Loss: 0.3488, Test Loss: 1.6962
Epoch 35/50 - Train Loss: 0.4235, Test Loss: 1.7088
Epoch 36/50 - Train Loss: 0.3402, Test Loss: 1.3661
Epoch 37/50 - Train Loss: 0.5351, Test Loss: 1.3780
Epoch 38/50 - Train Loss: 0.4855, Test Loss: 1.4712
Epoch 39/50 - Train Loss: 0.5154, Test Loss: 1.6123
Epoch 40/50 - Train Loss: 0.3894, Test Loss: 1.6726
Epoch 41/50 - Train Loss: 0.4135, Test Loss: 1.7034
Epoch 42/50 - Train Loss: 0.4078, Test Loss: 1.3964
Epoch 43/50 - Train Loss: 0.4509, Test Loss: 1.9534
Epoch 44/50 - Train Loss: 0.4024, Test Loss: 2.1903
Epoch 45/50 - Train Loss: 0.3604, Test Loss: 2.0453
Epoch 46/50 - Train Loss: 0.3732, Test Loss: 1.6542
Epoch 47/50 - Train Loss: 0.2737, Test Loss: 1.7814
Epoch 48/50 - Train Loss: 0.2783, Test Loss: 2.1356
Epoch 49/50 - Train Loss: 0.2779, Test Loss: 2.7634
Epoch 50/50 - Train Loss: 0.2830, Test Loss: 2.2819
```

In [14]:

```python
# TODO: Visualize the results
# Training loss and accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
for i, lr in enumerate(learning_rates):
    plt.plot(train_losses[i], label=f"LR: {lr}")
plt.legend()
plt.subplot(1, 2, 2)
plt.title("Training Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
for i, lr in enumerate(learning_rates):
    plt.plot(train_accuracies[i], label=f"LR: {lr}")
plt.legend()
plt.tight_layout()
plt.show()


# Validation loss and accuracy
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
for i, lr in enumerate(learning_rates):
    plt.plot(val_losses[i], label=f"LR: {lr}")
plt.legend()
plt.subplot(1, 2, 2)
plt.title("Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
for i, lr in enumerate(learning_rates):
    plt.plot(val_accuracies[i], label=f"LR: {lr}")
plt.legend()
plt.tight_layout()
plt.show()
```



For both extremes in the learning rate (0.1 and 1e-05), one can see, that in the first case the step size is to large to find good parameters in the parameter space. So the training is spiky. For the small learning rate we do not have enough epochs to reach a convergence. The two Learning rates in the middle both converge and reach similar results, while 0.001 converges faster.

Task 4

In [15]:
```
# TODO: apply the best model to the test set
# Take the model with LR = 0.001 as best model because it converges the f

best_model = models[2]

best_model.eval()
test_loss = 0
test_acc = 0
with tc.no_grad():
    for X, Y in test_loader:
        X, Y = X.to(device), Y.to(device)
        output = best_model(X)
        loss = nn.NLLLoss()(output, Y)
        test_loss += loss.item() / len(test_loader)
```

```
        test_acc += (output.argmax(dim=1) == Y).float().mean().item() / l
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_acc:.4f}")
```

```
Test Loss: 0.1991, Test Accuracy: 0.9250
```

The Validation and Test error have around the same accuracy, while the loss is much smaller in the validation loss.

Exercise 3 - CNN Autoencoder

In the next task we want to build an autoencoder. It consists of an encoder, which transforms the data into a low-dimensional code, and a decoder, which reconstructs the original data.

We use the Fashion MNIST dataset, which consists of 28x28 grayscale images. There are 10 classes, each representing different items of clothing. The data can be conveniently downloaded, separated and transformed with torchvision. As we will not be tuning any hyperparameters, we do not need a validation set.

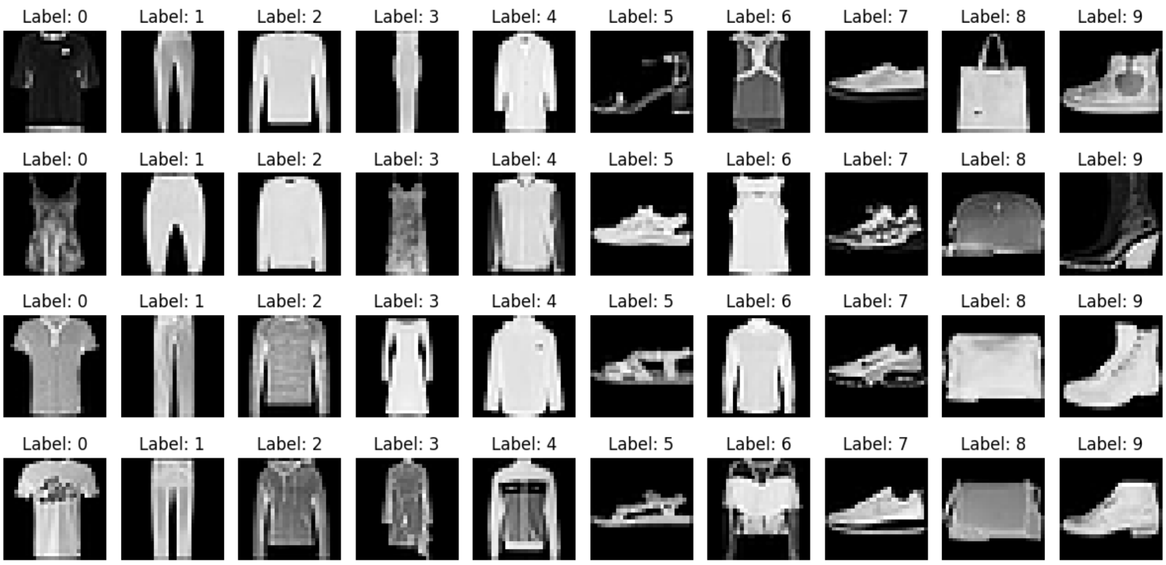
In [96]:

```
from torchvision import datasets, transforms

# Define transformations
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert PIL image to tensor (range [0, 1])
    transforms.Normalize((0.5,), (0.5,))  # Normalize to range [-1, 1]
])

# Load FashionMNIST train and test sets
train_data = datasets.FashionMNIST(
    root='./fashion_mnist',          # Download path
    train=True,           # Load training set
    download=True,        # Download if not already present
    transform=transform   # Apply transformations
)

test_data = datasets.FashionMNIST(
    root='./fashion_mnist',
    train=False,          # Load test set
    download=True,
    transform=transform
)

classes = train_data.targets.unique()

# we only need a subset that consists of 1000 samples of each class for t
# and 10 samples of each class for the test set
indices_train = []
indices_test = []
for i in range(len(classes)):
    indices_train += list(np.where(train_data.targets == classes[i])[0][:
    indices_test += list(np.where(test_data.targets == classes[i])[0][:10

train_data.data = train_data.data[indices_train]
train_data.targets = train_data.targets[indices_train]
test_data.data = test_data.data[indices_test]
test_data.targets = test_data.targets[indices_test]
```

```
# Create DataLoaders
train_batch_size = 256
test_batch_size = len(test_data)
train_loader = DataLoader(train_data, batch_size=train_batch_size, shuffl
test_loader = DataLoader(test_data, batch_size=test_batch_size, shuffle=F

# Print the shape of the dataset
print("train images shape: " + str(train_data.data.shape))
print("train labels shape: " + str(train_data.targets.shape))
print("test images shape: " + str(test_data.data.shape))
print("test labels shape: " + str(test_data.targets.shape))
print("classes: " + str(classes))

# check if classes are balanced
print("Counts of classes in train set: " + str(train_data.targets.unique(
print("Counts of classes in test set: " + str(test_data.targets.unique(re
```

```
train images shape: torch.Size([10000, 28, 28])
train labels shape: torch.Size([10000])
test images shape: torch.Size([100, 28, 28])
test labels shape: torch.Size([100])
classes: tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Counts of classes in train set: tensor([1000, 1000, 1000, 1000, 1000, 1000
, 1000, 1000, 1000, 1000])
Counts of classes in test set: tensor([10, 10, 10, 10, 10, 10, 10, 10, 10,
10])
```

In [30]:
```python
# Get a batch of data
data_iter = iter(train_loader)
images, labels = next(data_iter)

# Visualize a batch of images
fig, axes = plt.subplots(4, 10, figsize=(12, 6))
for i in range(4):
    for j in range(10):
        # find the ith image of class j
        idx = np.where(labels == j)[0][i]
        axes[i, j].imshow(images[idx].squeeze(), cmap='gray')
        axes[i, j].set_title(f"Label: {j}")
        axes[i, j].axis('off')
plt.tight_layout()
plt.show()
```



Task 1

We compare two architectures: a linear autoencoder and a CNN autoencoder. The latter typically consists of convolutional layers for the encoder and transposed convolutional layers for the decoder. In addition, fully connected layers can bring the feature to the desired code dimension (also called latent dimension). Implement the following architecture:

Encoder:

- Conv2d: 16 output channels, 3 by 3 filter size, stride 1, padding "same"
- ReLU activation
- MaxPool2d: 2 by 2 filter, stride 1, padding 0
- Conv2d: 32 output channels, 3 by 3, stride 1, padding "same"
- ReLU activation
- MaxPool2d: 2 by 2 filter, stride 1, padding 0
- Conv2d: 64 output channel, 3 by 3 filter size, stride 1, padding "same"
- ReLU activation
- MaxPool2d: 2 by 2 filter, stride 1, padding 0
- Flatten the previous output

- Linear: <*latent dimension*> output neurons

Decoder:

- Linear: 64x3x3 = 576 output neurons
- Unflatten the previous output to shape (64, 3, 3)
- ConvTranspose2d: 32 output channels, 3 by 3 filter size, stride 2, padding 0, output padding 1
- ReLU activation
- ConvTranspose2d: 16 output channels, 3 by 3 filter size, stride 2, padding 1, output padding 1
- ReLU activation
- ConvTranspose2d: 1 output channel, 3 by 3 filter size, stride 2, padding 1, output padding 1

You might need to infer the input dimension of the linear layer in the encoder.

In [59]:
```
class Conv_AE(nn.Module):
    def __init__(self, latent_dim):
        super(Conv_AE, self).__init__()

        # TODO: Initialize the layers of the encoder
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, (3,3), 1, 'same'),
            nn.ReLU(),
            nn.MaxPool2d((2,2), 1),
            # output dim= 27
            nn.Conv2d(16, 32, (3,3), 1, 'same'),
            nn.ReLU(),
            nn.MaxPool2d((2,2), 1, 0),
            # output dim = 26
            nn.Conv2d(32, 64, (3,3), 1, 'same'),
            nn.ReLU(),
            nn.MaxPool2d((2,2), 1, 0),
            # output dim = 25
            nn.Flatten(),
            # output dim = 25*25*64
            nn.Linear(40000, latent_dim)
        )

        # TODO: Initialize the layers of the decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 576),
            nn.Unflatten(1, (64, 3, 3)),
            nn.ConvTranspose2d(64, 32, (3,3), 2, 0, 0),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, (3,3), 2, 1, 1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, (3,3), 2, 1, 1)
        )

    def forward(self, X):
        # TODO: Implement the forward pass
        latent = self.encoder(X)

        return self.decoder(latent), latent
```

We check again whether the model is working properly.

In [60]:
```
conv_ae_model = Conv_AE(2)
X_random = tc.randn(2, 1, 28, 28)
reconstructed, latent = conv_ae_model(X_random)

print("Reconstructed shape: " + str(reconstructed.shape), "\n", "Latent s

summary(conv_ae_model, input_size=(1, 28, 28), device="cpu")
```

```
Reconstructed shape: torch.Size([2, 1, 28, 28])
 Latent shape: torch.Size([2, 2])
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]             160
              ReLU-2           [-1, 16, 28, 28]               0
         MaxPool2d-3           [-1, 16, 27, 27]               0
            Conv2d-4           [-1, 32, 27, 27]           4,640
              ReLU-5           [-1, 32, 27, 27]               0
         MaxPool2d-6           [-1, 32, 26, 26]               0
            Conv2d-7           [-1, 64, 26, 26]          18,496
              ReLU-8           [-1, 64, 26, 26]               0
         MaxPool2d-9           [-1, 64, 25, 25]               0
          Flatten-10                [-1, 40000]               0
           Linear-11                    [-1, 2]          80,002
           Linear-12                  [-1, 576]           1,728
        Unflatten-13             [-1, 64, 3, 3]               0
  ConvTranspose2d-14             [-1, 32, 7, 7]          18,464
             ReLU-15             [-1, 32, 7, 7]               0
  ConvTranspose2d-16           [-1, 16, 14, 14]           4,624
             ReLU-17           [-1, 16, 14, 14]               0
  ConvTranspose2d-18            [-1, 1, 28, 28]             145
================================================================
Total params: 128,259
Trainable params: 128,259
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 2.16
Params size (MB): 0.49
Estimated Total Size (MB): 2.65
----------------------------------------------------------------
```

Task 2

The training of an autoencoder compares the original input to the reconstruction, usually by means of the mean squared error.

In [61]:
```python
def train_ae(model, train_loader, test_loader, lr, n_epochs, device):
    model = model.to(device)

    # TODO: Initialize the loss function
    loss_function = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr)

    train_loss = np.zeros(n_epochs)
    test_loss = np.zeros(n_epochs)

    for epoch in range(1, n_epochs + 1):
        model.train()

        epoch_loss = 0

        for X, _ in train_loader:
            X = X.to(device)
            # TODO: Implement the training loop
            optimizer.zero_grad()
```

```
            output, _ = model(X)

            loss = loss_function(X, output)

            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()/len(train_loader)

        train_loss[epoch - 1] = epoch_loss

        model.eval()

        with tc.no_grad():
            X, _ = next(iter(test_loader))
            X = X.to(device)
            # TODO: Implement the evaluation step
            output, _ = model(X)
            loss = loss_function(X, output)

            test_loss[epoch - 1] = loss.item()
        print(f"Epoch {epoch}/{n_epochs} - Train Loss: {epoch_loss:.4f},

    return train_loss, test_loss
```

In [62]:

```
n_epochs = 60
lr = 1e-3
# TODO: Train the convolutional autoencoder model with different latent d
latent_dims = [3, 10, 50, 100, 250]

models = [Conv_AE(i) for i in latent_dims]
train_losses = []
test_losses = []
for latent_dim in latent_dims:
    model = models[latent_dims.index(latent_dim)]
    train_loss, test_loss = train_ae(model, train_loader, test_loader, lr
    train_losses.append(train_loss)
    test_losses.append(test_loss)
```

```
Epoch 1/60 - Train Loss: 0.3350, Test Loss: 0.1932
Epoch 2/60 - Train Loss: 0.1757, Test Loss: 0.1613
Epoch 3/60 - Train Loss: 0.1577, Test Loss: 0.1504
Epoch 4/60 - Train Loss: 0.1460, Test Loss: 0.1429
Epoch 5/60 - Train Loss: 0.1386, Test Loss: 0.1356
Epoch 6/60 - Train Loss: 0.1336, Test Loss: 0.1280
Epoch 7/60 - Train Loss: 0.1284, Test Loss: 0.1234
Epoch 8/60 - Train Loss: 0.1240, Test Loss: 0.1175
Epoch 9/60 - Train Loss: 0.1213, Test Loss: 0.1146
Epoch 10/60 - Train Loss: 0.1186, Test Loss: 0.1134
Epoch 11/60 - Train Loss: 0.1167, Test Loss: 0.1133
Epoch 12/60 - Train Loss: 0.1150, Test Loss: 0.1120
Epoch 13/60 - Train Loss: 0.1144, Test Loss: 0.1090
Epoch 14/60 - Train Loss: 0.1131, Test Loss: 0.1082
Epoch 15/60 - Train Loss: 0.1121, Test Loss: 0.1074
Epoch 16/60 - Train Loss: 0.1108, Test Loss: 0.1060
Epoch 17/60 - Train Loss: 0.1102, Test Loss: 0.1069
Epoch 18/60 - Train Loss: 0.1097, Test Loss: 0.1049
Epoch 19/60 - Train Loss: 0.1097, Test Loss: 0.1054
Epoch 20/60 - Train Loss: 0.1091, Test Loss: 0.1047
Epoch 21/60 - Train Loss: 0.1092, Test Loss: 0.1070
Epoch 22/60 - Train Loss: 0.1086, Test Loss: 0.1034
Epoch 23/60 - Train Loss: 0.1076, Test Loss: 0.1037
Epoch 24/60 - Train Loss: 0.1075, Test Loss: 0.1031
Epoch 25/60 - Train Loss: 0.1064, Test Loss: 0.1026
Epoch 26/60 - Train Loss: 0.1065, Test Loss: 0.1047
Epoch 27/60 - Train Loss: 0.1072, Test Loss: 0.1021
Epoch 28/60 - Train Loss: 0.1052, Test Loss: 0.1027
Epoch 29/60 - Train Loss: 0.1055, Test Loss: 0.1023
Epoch 30/60 - Train Loss: 0.1062, Test Loss: 0.1039
Epoch 31/60 - Train Loss: 0.1063, Test Loss: 0.1010
Epoch 32/60 - Train Loss: 0.1041, Test Loss: 0.1036
Epoch 33/60 - Train Loss: 0.1043, Test Loss: 0.1002
Epoch 34/60 - Train Loss: 0.1043, Test Loss: 0.1025
Epoch 35/60 - Train Loss: 0.1044, Test Loss: 0.1017
Epoch 36/60 - Train Loss: 0.1045, Test Loss: 0.1017
Epoch 37/60 - Train Loss: 0.1035, Test Loss: 0.1007
Epoch 38/60 - Train Loss: 0.1029, Test Loss: 0.0993
Epoch 39/60 - Train Loss: 0.1025, Test Loss: 0.0993
Epoch 40/60 - Train Loss: 0.1027, Test Loss: 0.0994
Epoch 41/60 - Train Loss: 0.1023, Test Loss: 0.0999
Epoch 42/60 - Train Loss: 0.1024, Test Loss: 0.1011
Epoch 43/60 - Train Loss: 0.1022, Test Loss: 0.0996
Epoch 44/60 - Train Loss: 0.1022, Test Loss: 0.1005
Epoch 45/60 - Train Loss: 0.1017, Test Loss: 0.1001
Epoch 46/60 - Train Loss: 0.1015, Test Loss: 0.0983
Epoch 47/60 - Train Loss: 0.1017, Test Loss: 0.1012
Epoch 48/60 - Train Loss: 0.1012, Test Loss: 0.0987
Epoch 49/60 - Train Loss: 0.1000, Test Loss: 0.0983
Epoch 50/60 - Train Loss: 0.1002, Test Loss: 0.0991
Epoch 51/60 - Train Loss: 0.1005, Test Loss: 0.0978
Epoch 52/60 - Train Loss: 0.1004, Test Loss: 0.0977
Epoch 53/60 - Train Loss: 0.1007, Test Loss: 0.0997
Epoch 54/60 - Train Loss: 0.1005, Test Loss: 0.0999
Epoch 55/60 - Train Loss: 0.1005, Test Loss: 0.0985
Epoch 56/60 - Train Loss: 0.0996, Test Loss: 0.0994
Epoch 57/60 - Train Loss: 0.0993, Test Loss: 0.0977
Epoch 58/60 - Train Loss: 0.0991, Test Loss: 0.0972
Epoch 59/60 - Train Loss: 0.0995, Test Loss: 0.0973
Epoch 60/60 - Train Loss: 0.0991, Test Loss: 0.0968
```

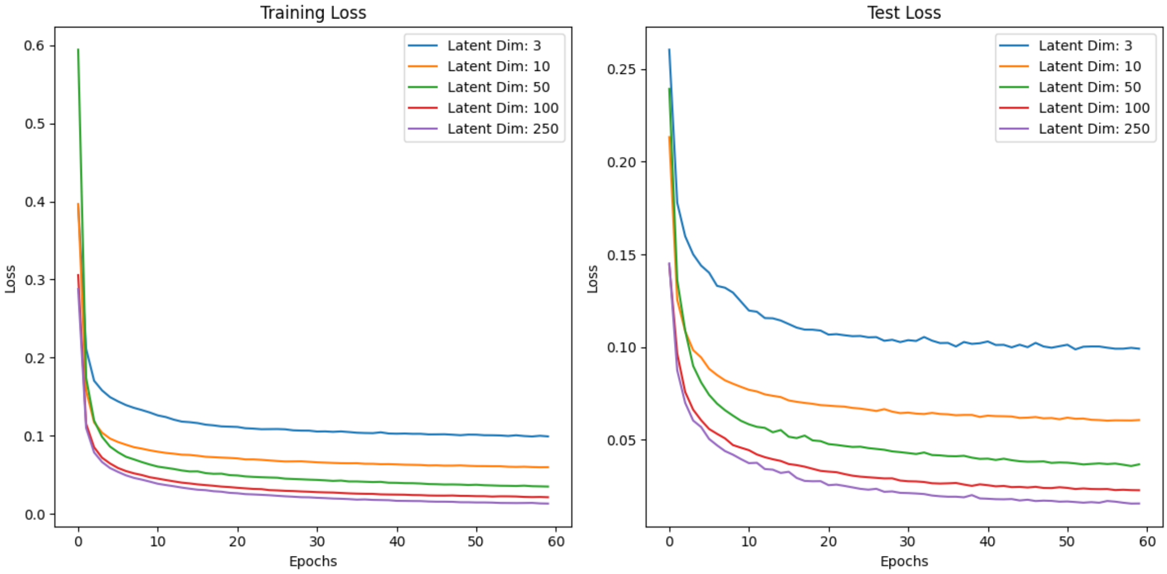
```
Epoch 1/60 - Train Loss: 0.4024, Test Loss: 0.2187
Epoch 2/60 - Train Loss: 0.1651, Test Loss: 0.1288
Epoch 3/60 - Train Loss: 0.1202, Test Loss: 0.1101
Epoch 4/60 - Train Loss: 0.1063, Test Loss: 0.0995
Epoch 5/60 - Train Loss: 0.0988, Test Loss: 0.0945
Epoch 6/60 - Train Loss: 0.0939, Test Loss: 0.0894
Epoch 7/60 - Train Loss: 0.0911, Test Loss: 0.0861
Epoch 8/60 - Train Loss: 0.0879, Test Loss: 0.0839
Epoch 9/60 - Train Loss: 0.0857, Test Loss: 0.0821
Epoch 10/60 - Train Loss: 0.0837, Test Loss: 0.0804
Epoch 11/60 - Train Loss: 0.0821, Test Loss: 0.0799
Epoch 12/60 - Train Loss: 0.0812, Test Loss: 0.0779
Epoch 13/60 - Train Loss: 0.0803, Test Loss: 0.0775
Epoch 14/60 - Train Loss: 0.0792, Test Loss: 0.0768
Epoch 15/60 - Train Loss: 0.0781, Test Loss: 0.0752
Epoch 16/60 - Train Loss: 0.0767, Test Loss: 0.0742
Epoch 17/60 - Train Loss: 0.0765, Test Loss: 0.0741
Epoch 18/60 - Train Loss: 0.0757, Test Loss: 0.0736
Epoch 19/60 - Train Loss: 0.0749, Test Loss: 0.0731
Epoch 20/60 - Train Loss: 0.0742, Test Loss: 0.0720
Epoch 21/60 - Train Loss: 0.0736, Test Loss: 0.0719
Epoch 22/60 - Train Loss: 0.0730, Test Loss: 0.0707
Epoch 23/60 - Train Loss: 0.0727, Test Loss: 0.0711
Epoch 24/60 - Train Loss: 0.0727, Test Loss: 0.0701
Epoch 25/60 - Train Loss: 0.0718, Test Loss: 0.0702
Epoch 26/60 - Train Loss: 0.0715, Test Loss: 0.0696
Epoch 27/60 - Train Loss: 0.0706, Test Loss: 0.0687
Epoch 28/60 - Train Loss: 0.0704, Test Loss: 0.0682
Epoch 29/60 - Train Loss: 0.0704, Test Loss: 0.0690
Epoch 30/60 - Train Loss: 0.0698, Test Loss: 0.0678
Epoch 31/60 - Train Loss: 0.0691, Test Loss: 0.0681
Epoch 32/60 - Train Loss: 0.0693, Test Loss: 0.0675
Epoch 33/60 - Train Loss: 0.0690, Test Loss: 0.0669
Epoch 34/60 - Train Loss: 0.0681, Test Loss: 0.0668
Epoch 35/60 - Train Loss: 0.0676, Test Loss: 0.0664
Epoch 36/60 - Train Loss: 0.0681, Test Loss: 0.0665
Epoch 37/60 - Train Loss: 0.0675, Test Loss: 0.0664
Epoch 38/60 - Train Loss: 0.0681, Test Loss: 0.0663
Epoch 39/60 - Train Loss: 0.0673, Test Loss: 0.0658
Epoch 40/60 - Train Loss: 0.0668, Test Loss: 0.0657
Epoch 41/60 - Train Loss: 0.0665, Test Loss: 0.0650
Epoch 42/60 - Train Loss: 0.0659, Test Loss: 0.0649
Epoch 43/60 - Train Loss: 0.0658, Test Loss: 0.0646
Epoch 44/60 - Train Loss: 0.0660, Test Loss: 0.0653
Epoch 45/60 - Train Loss: 0.0656, Test Loss: 0.0649
Epoch 46/60 - Train Loss: 0.0653, Test Loss: 0.0644
Epoch 47/60 - Train Loss: 0.0652, Test Loss: 0.0644
Epoch 48/60 - Train Loss: 0.0654, Test Loss: 0.0643
Epoch 49/60 - Train Loss: 0.0654, Test Loss: 0.0641
Epoch 50/60 - Train Loss: 0.0648, Test Loss: 0.0638
Epoch 51/60 - Train Loss: 0.0648, Test Loss: 0.0638
Epoch 52/60 - Train Loss: 0.0644, Test Loss: 0.0638
Epoch 53/60 - Train Loss: 0.0644, Test Loss: 0.0633
Epoch 54/60 - Train Loss: 0.0641, Test Loss: 0.0633
Epoch 55/60 - Train Loss: 0.0643, Test Loss: 0.0632
Epoch 56/60 - Train Loss: 0.0641, Test Loss: 0.0630
Epoch 57/60 - Train Loss: 0.0633, Test Loss: 0.0628
Epoch 58/60 - Train Loss: 0.0631, Test Loss: 0.0626
Epoch 59/60 - Train Loss: 0.0632, Test Loss: 0.0625
Epoch 60/60 - Train Loss: 0.0630, Test Loss: 0.0631
```

```
Epoch 1/60 - Train Loss: 0.3719, Test Loss: 0.2227
Epoch 2/60 - Train Loss: 0.1554, Test Loss: 0.1114
Epoch 3/60 - Train Loss: 0.0994, Test Loss: 0.0903
Epoch 4/60 - Train Loss: 0.0834, Test Loss: 0.0772
Epoch 5/60 - Train Loss: 0.0745, Test Loss: 0.0696
Epoch 6/60 - Train Loss: 0.0689, Test Loss: 0.0651
Epoch 7/60 - Train Loss: 0.0647, Test Loss: 0.0631
Epoch 8/60 - Train Loss: 0.0614, Test Loss: 0.0590
Epoch 9/60 - Train Loss: 0.0585, Test Loss: 0.0562
Epoch 10/60 - Train Loss: 0.0563, Test Loss: 0.0555
Epoch 11/60 - Train Loss: 0.0540, Test Loss: 0.0546
Epoch 12/60 - Train Loss: 0.0536, Test Loss: 0.0514
Epoch 13/60 - Train Loss: 0.0513, Test Loss: 0.0500
Epoch 14/60 - Train Loss: 0.0499, Test Loss: 0.0489
Epoch 15/60 - Train Loss: 0.0490, Test Loss: 0.0482
Epoch 16/60 - Train Loss: 0.0479, Test Loss: 0.0467
Epoch 17/60 - Train Loss: 0.0467, Test Loss: 0.0458
Epoch 18/60 - Train Loss: 0.0461, Test Loss: 0.0452
Epoch 19/60 - Train Loss: 0.0452, Test Loss: 0.0441
Epoch 20/60 - Train Loss: 0.0440, Test Loss: 0.0434
Epoch 21/60 - Train Loss: 0.0438, Test Loss: 0.0434
Epoch 22/60 - Train Loss: 0.0432, Test Loss: 0.0420
Epoch 23/60 - Train Loss: 0.0420, Test Loss: 0.0418
Epoch 24/60 - Train Loss: 0.0415, Test Loss: 0.0412
Epoch 25/60 - Train Loss: 0.0410, Test Loss: 0.0403
Epoch 26/60 - Train Loss: 0.0407, Test Loss: 0.0396
Epoch 27/60 - Train Loss: 0.0398, Test Loss: 0.0389
Epoch 28/60 - Train Loss: 0.0394, Test Loss: 0.0388
Epoch 29/60 - Train Loss: 0.0390, Test Loss: 0.0384
Epoch 30/60 - Train Loss: 0.0385, Test Loss: 0.0388
Epoch 31/60 - Train Loss: 0.0384, Test Loss: 0.0374
Epoch 32/60 - Train Loss: 0.0374, Test Loss: 0.0369
Epoch 33/60 - Train Loss: 0.0369, Test Loss: 0.0366
Epoch 34/60 - Train Loss: 0.0367, Test Loss: 0.0367
Epoch 35/60 - Train Loss: 0.0363, Test Loss: 0.0359
Epoch 36/60 - Train Loss: 0.0358, Test Loss: 0.0358
Epoch 37/60 - Train Loss: 0.0358, Test Loss: 0.0359
Epoch 38/60 - Train Loss: 0.0354, Test Loss: 0.0358
Epoch 39/60 - Train Loss: 0.0349, Test Loss: 0.0350
Epoch 40/60 - Train Loss: 0.0347, Test Loss: 0.0350
Epoch 41/60 - Train Loss: 0.0345, Test Loss: 0.0347
Epoch 42/60 - Train Loss: 0.0342, Test Loss: 0.0340
Epoch 43/60 - Train Loss: 0.0342, Test Loss: 0.0347
Epoch 44/60 - Train Loss: 0.0338, Test Loss: 0.0336
Epoch 45/60 - Train Loss: 0.0333, Test Loss: 0.0342
Epoch 46/60 - Train Loss: 0.0333, Test Loss: 0.0338
Epoch 47/60 - Train Loss: 0.0329, Test Loss: 0.0336
Epoch 48/60 - Train Loss: 0.0325, Test Loss: 0.0329
Epoch 49/60 - Train Loss: 0.0325, Test Loss: 0.0332
Epoch 50/60 - Train Loss: 0.0323, Test Loss: 0.0327
Epoch 51/60 - Train Loss: 0.0320, Test Loss: 0.0331
Epoch 52/60 - Train Loss: 0.0321, Test Loss: 0.0333
Epoch 53/60 - Train Loss: 0.0317, Test Loss: 0.0325
Epoch 54/60 - Train Loss: 0.0320, Test Loss: 0.0328
Epoch 55/60 - Train Loss: 0.0317, Test Loss: 0.0326
Epoch 56/60 - Train Loss: 0.0314, Test Loss: 0.0324
Epoch 57/60 - Train Loss: 0.0309, Test Loss: 0.0321
Epoch 58/60 - Train Loss: 0.0309, Test Loss: 0.0320
Epoch 59/60 - Train Loss: 0.0307, Test Loss: 0.0320
Epoch 60/60 - Train Loss: 0.0305, Test Loss: 0.0320
```

```
Epoch 1/60 - Train Loss: 0.3333, Test Loss: 0.1486
Epoch 2/60 - Train Loss: 0.1203, Test Loss: 0.0985
Epoch 3/60 - Train Loss: 0.0900, Test Loss: 0.0802
Epoch 4/60 - Train Loss: 0.0766, Test Loss: 0.0708
Epoch 5/60 - Train Loss: 0.0686, Test Loss: 0.0635
Epoch 6/60 - Train Loss: 0.0629, Test Loss: 0.0605
Epoch 7/60 - Train Loss: 0.0583, Test Loss: 0.0554
Epoch 8/60 - Train Loss: 0.0547, Test Loss: 0.0517
Epoch 9/60 - Train Loss: 0.0552, Test Loss: 0.0506
Epoch 10/60 - Train Loss: 0.0495, Test Loss: 0.0473
Epoch 11/60 - Train Loss: 0.0474, Test Loss: 0.0455
Epoch 12/60 - Train Loss: 0.0457, Test Loss: 0.0443
Epoch 13/60 - Train Loss: 0.0442, Test Loss: 0.0427
Epoch 14/60 - Train Loss: 0.0423, Test Loss: 0.0421
Epoch 15/60 - Train Loss: 0.0415, Test Loss: 0.0399
Epoch 16/60 - Train Loss: 0.0398, Test Loss: 0.0395
Epoch 17/60 - Train Loss: 0.0391, Test Loss: 0.0378
Epoch 18/60 - Train Loss: 0.0378, Test Loss: 0.0371
Epoch 19/60 - Train Loss: 0.0370, Test Loss: 0.0369
Epoch 20/60 - Train Loss: 0.0366, Test Loss: 0.0358
Epoch 21/60 - Train Loss: 0.0354, Test Loss: 0.0347
Epoch 22/60 - Train Loss: 0.0345, Test Loss: 0.0340
Epoch 23/60 - Train Loss: 0.0337, Test Loss: 0.0331
Epoch 24/60 - Train Loss: 0.0332, Test Loss: 0.0324
Epoch 25/60 - Train Loss: 0.0328, Test Loss: 0.0320
Epoch 26/60 - Train Loss: 0.0323, Test Loss: 0.0315
Epoch 27/60 - Train Loss: 0.0313, Test Loss: 0.0311
Epoch 28/60 - Train Loss: 0.0312, Test Loss: 0.0310
Epoch 29/60 - Train Loss: 0.0305, Test Loss: 0.0302
Epoch 30/60 - Train Loss: 0.0301, Test Loss: 0.0296
Epoch 31/60 - Train Loss: 0.0296, Test Loss: 0.0293
Epoch 32/60 - Train Loss: 0.0292, Test Loss: 0.0296
Epoch 33/60 - Train Loss: 0.0294, Test Loss: 0.0285
Epoch 34/60 - Train Loss: 0.0283, Test Loss: 0.0281
Epoch 35/60 - Train Loss: 0.0279, Test Loss: 0.0279
Epoch 36/60 - Train Loss: 0.0279, Test Loss: 0.0275
Epoch 37/60 - Train Loss: 0.0274, Test Loss: 0.0267
Epoch 38/60 - Train Loss: 0.0271, Test Loss: 0.0271
Epoch 39/60 - Train Loss: 0.0265, Test Loss: 0.0266
Epoch 40/60 - Train Loss: 0.0266, Test Loss: 0.0276
Epoch 41/60 - Train Loss: 0.0264, Test Loss: 0.0260
Epoch 42/60 - Train Loss: 0.0255, Test Loss: 0.0257
Epoch 43/60 - Train Loss: 0.0254, Test Loss: 0.0257
Epoch 44/60 - Train Loss: 0.0251, Test Loss: 0.0253
Epoch 45/60 - Train Loss: 0.0248, Test Loss: 0.0251
Epoch 46/60 - Train Loss: 0.0247, Test Loss: 0.0248
Epoch 47/60 - Train Loss: 0.0244, Test Loss: 0.0249
Epoch 48/60 - Train Loss: 0.0243, Test Loss: 0.0246
Epoch 49/60 - Train Loss: 0.0240, Test Loss: 0.0245
Epoch 50/60 - Train Loss: 0.0238, Test Loss: 0.0245
Epoch 51/60 - Train Loss: 0.0237, Test Loss: 0.0240
Epoch 52/60 - Train Loss: 0.0234, Test Loss: 0.0243
Epoch 53/60 - Train Loss: 0.0232, Test Loss: 0.0240
Epoch 54/60 - Train Loss: 0.0230, Test Loss: 0.0241
Epoch 55/60 - Train Loss: 0.0229, Test Loss: 0.0238
Epoch 56/60 - Train Loss: 0.0229, Test Loss: 0.0238
Epoch 57/60 - Train Loss: 0.0227, Test Loss: 0.0236
Epoch 58/60 - Train Loss: 0.0227, Test Loss: 0.0235
Epoch 59/60 - Train Loss: 0.0222, Test Loss: 0.0233
Epoch 60/60 - Train Loss: 0.0223, Test Loss: 0.0232
```

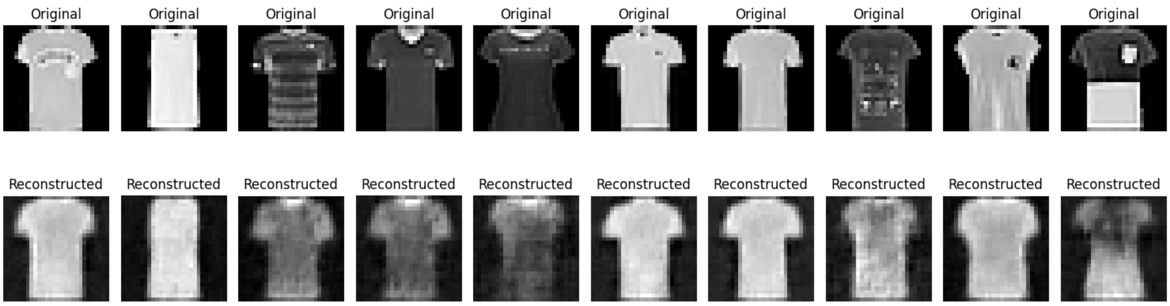
```
Epoch 1/60 - Train Loss: 0.3049, Test Loss: 0.1403
Epoch 2/60 - Train Loss: 0.1108, Test Loss: 0.0917
Epoch 3/60 - Train Loss: 0.0819, Test Loss: 0.0735
Epoch 4/60 - Train Loss: 0.0693, Test Loss: 0.0662
Epoch 5/60 - Train Loss: 0.0614, Test Loss: 0.0567
Epoch 6/60 - Train Loss: 0.0549, Test Loss: 0.0526
Epoch 7/60 - Train Loss: 0.0503, Test Loss: 0.0483
Epoch 8/60 - Train Loss: 0.0471, Test Loss: 0.0450
Epoch 9/60 - Train Loss: 0.0448, Test Loss: 0.0437
Epoch 10/60 - Train Loss: 0.0419, Test Loss: 0.0405
Epoch 11/60 - Train Loss: 0.0398, Test Loss: 0.0383
Epoch 12/60 - Train Loss: 0.0378, Test Loss: 0.0373
Epoch 13/60 - Train Loss: 0.0366, Test Loss: 0.0351
Epoch 14/60 - Train Loss: 0.0350, Test Loss: 0.0336
Epoch 15/60 - Train Loss: 0.0338, Test Loss: 0.0336
Epoch 16/60 - Train Loss: 0.0328, Test Loss: 0.0316
Epoch 17/60 - Train Loss: 0.0313, Test Loss: 0.0308
Epoch 18/60 - Train Loss: 0.0304, Test Loss: 0.0303
Epoch 19/60 - Train Loss: 0.0295, Test Loss: 0.0304
Epoch 20/60 - Train Loss: 0.0289, Test Loss: 0.0279
Epoch 21/60 - Train Loss: 0.0277, Test Loss: 0.0271
Epoch 22/60 - Train Loss: 0.0269, Test Loss: 0.0267
Epoch 23/60 - Train Loss: 0.0265, Test Loss: 0.0279
Epoch 24/60 - Train Loss: 0.0264, Test Loss: 0.0256
Epoch 25/60 - Train Loss: 0.0251, Test Loss: 0.0248
Epoch 26/60 - Train Loss: 0.0246, Test Loss: 0.0247
Epoch 27/60 - Train Loss: 0.0241, Test Loss: 0.0242
Epoch 28/60 - Train Loss: 0.0234, Test Loss: 0.0240
Epoch 29/60 - Train Loss: 0.0232, Test Loss: 0.0263
Epoch 30/60 - Train Loss: 0.0231, Test Loss: 0.0229
Epoch 31/60 - Train Loss: 0.0222, Test Loss: 0.0227
Epoch 32/60 - Train Loss: 0.0217, Test Loss: 0.0222
Epoch 33/60 - Train Loss: 0.0213, Test Loss: 0.0219
Epoch 34/60 - Train Loss: 0.0215, Test Loss: 0.0221
Epoch 35/60 - Train Loss: 0.0208, Test Loss: 0.0215
Epoch 36/60 - Train Loss: 0.0203, Test Loss: 0.0211
Epoch 37/60 - Train Loss: 0.0202, Test Loss: 0.0209
Epoch 38/60 - Train Loss: 0.0199, Test Loss: 0.0222
Epoch 39/60 - Train Loss: 0.0200, Test Loss: 0.0204
Epoch 40/60 - Train Loss: 0.0190, Test Loss: 0.0200
Epoch 41/60 - Train Loss: 0.0189, Test Loss: 0.0204
Epoch 42/60 - Train Loss: 0.0187, Test Loss: 0.0196
Epoch 43/60 - Train Loss: 0.0182, Test Loss: 0.0195
Epoch 44/60 - Train Loss: 0.0182, Test Loss: 0.0191
Epoch 45/60 - Train Loss: 0.0180, Test Loss: 0.0193
Epoch 46/60 - Train Loss: 0.0176, Test Loss: 0.0188
Epoch 47/60 - Train Loss: 0.0173, Test Loss: 0.0190
Epoch 48/60 - Train Loss: 0.0172, Test Loss: 0.0185
Epoch 49/60 - Train Loss: 0.0171, Test Loss: 0.0192
Epoch 50/60 - Train Loss: 0.0168, Test Loss: 0.0183
Epoch 51/60 - Train Loss: 0.0166, Test Loss: 0.0186
Epoch 52/60 - Train Loss: 0.0164, Test Loss: 0.0180
Epoch 53/60 - Train Loss: 0.0160, Test Loss: 0.0177
Epoch 54/60 - Train Loss: 0.0160, Test Loss: 0.0186
Epoch 55/60 - Train Loss: 0.0161, Test Loss: 0.0180
Epoch 56/60 - Train Loss: 0.0156, Test Loss: 0.0175
Epoch 57/60 - Train Loss: 0.0152, Test Loss: 0.0174
Epoch 58/60 - Train Loss: 0.0153, Test Loss: 0.0175
Epoch 59/60 - Train Loss: 0.0150, Test Loss: 0.0174
Epoch 60/60 - Train Loss: 0.0148, Test Loss: 0.0173
```

In [52]:
```
# TODO: Visualize the results
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
for i, latent_dim in enumerate(latent_dims):
    plt.plot(train_losses[i], label=f"Latent Dim: {latent_dim}")
plt.legend()
plt.subplot(1, 2, 2)
plt.title("Test Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
for i, latent_dim in enumerate(latent_dims):
    plt.plot(test_losses[i], label=f"Latent Dim: {latent_dim}")
plt.legend()
plt.tight_layout()
plt.show()
```



In [69]:
```
#TODO: Visualize the reconstructed images and the original images
def visualize_reconstruction(model, data_loader, device):
    model.eval()
    with tc.no_grad():
        X, _ = next(iter(data_loader))
        X = X.to(device)
        reconstructed, _ = model(X)

    fig, axs = plt.subplots(2, 10, figsize=(15, 5))
    for i in range(10):
        axs[0, i].imshow(X[i].cpu().squeeze(), cmap='gray')
        axs[0, i].set_title("Original")
        axs[0, i].axis('off')

        axs[1, i].imshow(reconstructed[i].cpu().squeeze(), cmap='gray')
        axs[1, i].set_title("Reconstructed")
        axs[1, i].axis('off')

    plt.tight_layout()
    plt.show()

# Visualize the reconstruction for the model with latent dimension 10
```

```
visualize_reconstruction(models[1], test_loader, device)
```



The shapes of the different fashion pieces is visible but the reconstructed image looses a lot of details.

Task 3

A linear autoencoder aims to represent the data $X \in \mathbb{R}^{n \times m}$ in a new basis using only $d < m$ directions. The objective is to minimize the squared error between X and $D(E(X))$ where $E : \mathbb{R}^m \to \mathbb{R}^d$ is the encoder and $D : \mathbb{R}^d \to \mathbb{R}^m$ is the decoder:

$$\|D(E(X)) - X\|_2^2$$

In geometric terms, we want to find d axes along which most of the variance occurs which is exactly what Principal Component Analysis does. The optimal weights of a linear autoencoder with code dimension d thus span the same space as the first d principal components.

The PCA autoencoder just consists of 1 linear layer for the encoder and 1 linear layer for the decoder. The bias is necessary to subtract the mean value. Since we are dealing with three-dimensional images, we also have to flatten (when encoding) or unflatten (when decoding) the input.

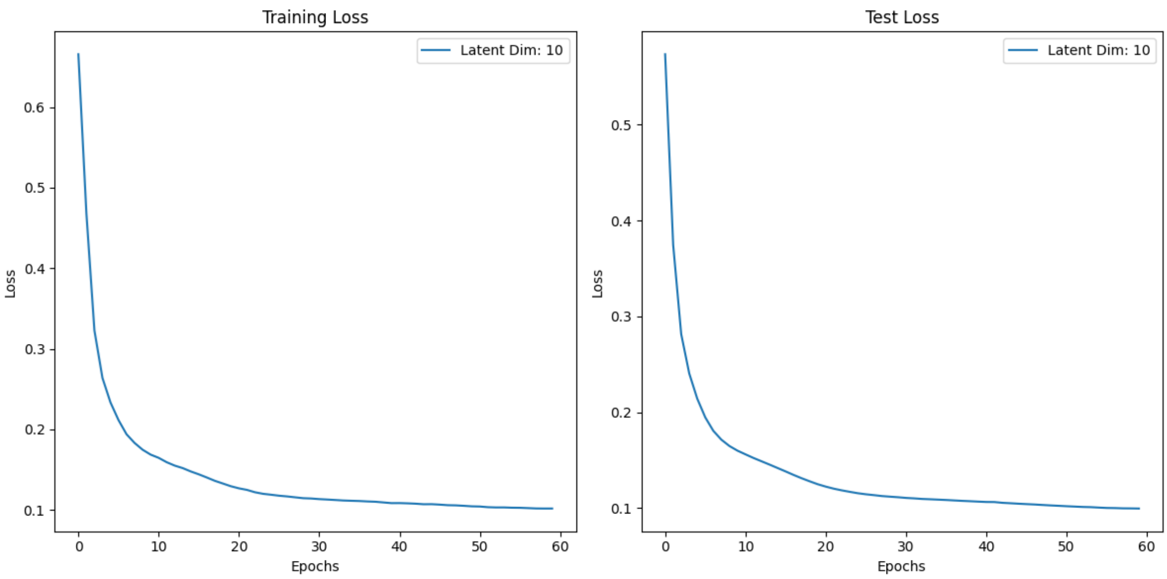
In [65]:
```
class PCA_AE(nn.Module):
    def __init__(self, input_dim, latent_dim):
        super(PCA_AE, self).__init__()

        # TODO: Initialize the layers of the autoencoder
        self.encoder = nn.Sequential(
            nn.Flatten(),
            nn.Linear(input_dim, latent_dim)
        )

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, input_dim),
            nn.Unflatten(1, (1, 28, 28))
        )

    def forward(self, X):
        # TODO: Implement the forward call
        latent = self.encoder(X)
        return self.decoder(latent), latent
```

For the training we can use the same function as for the convolutional autoencoder.

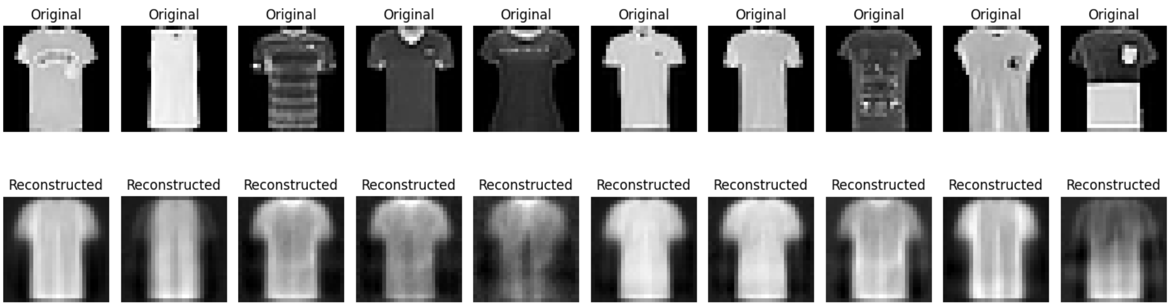
In [66]:
```
n_epochs = 60
lr = 1e-3
# TODO: Train the PCA autoencoder model with a latent dimension of 10

latent_dim = 10
pca_ae_model = PCA_AE(28*28, latent_dim)
train_loss, test_loss = train_ae(pca_ae_model, train_loader, test_loader,
```

```
Epoch 1/60 - Train Loss: 0.6656, Test Loss: 0.5735
Epoch 2/60 - Train Loss: 0.4680, Test Loss: 0.3745
Epoch 3/60 - Train Loss: 0.3228, Test Loss: 0.2815
Epoch 4/60 - Train Loss: 0.2639, Test Loss: 0.2404
Epoch 5/60 - Train Loss: 0.2334, Test Loss: 0.2141
Epoch 6/60 - Train Loss: 0.2115, Test Loss: 0.1945
Epoch 7/60 - Train Loss: 0.1939, Test Loss: 0.1805
Epoch 8/60 - Train Loss: 0.1832, Test Loss: 0.1713
Epoch 9/60 - Train Loss: 0.1748, Test Loss: 0.1647
Epoch 10/60 - Train Loss: 0.1687, Test Loss: 0.1599
Epoch 11/60 - Train Loss: 0.1647, Test Loss: 0.1560
Epoch 12/60 - Train Loss: 0.1593, Test Loss: 0.1522
Epoch 13/60 - Train Loss: 0.1551, Test Loss: 0.1488
Epoch 14/60 - Train Loss: 0.1520, Test Loss: 0.1453
Epoch 15/60 - Train Loss: 0.1478, Test Loss: 0.1418
Epoch 16/60 - Train Loss: 0.1442, Test Loss: 0.1382
Epoch 17/60 - Train Loss: 0.1403, Test Loss: 0.1345
Epoch 18/60 - Train Loss: 0.1362, Test Loss: 0.1311
Epoch 19/60 - Train Loss: 0.1328, Test Loss: 0.1279
Epoch 20/60 - Train Loss: 0.1294, Test Loss: 0.1248
Epoch 21/60 - Train Loss: 0.1268, Test Loss: 0.1223
Epoch 22/60 - Train Loss: 0.1249, Test Loss: 0.1202
Epoch 23/60 - Train Loss: 0.1220, Test Loss: 0.1184
Epoch 24/60 - Train Loss: 0.1200, Test Loss: 0.1169
Epoch 25/60 - Train Loss: 0.1189, Test Loss: 0.1154
Epoch 26/60 - Train Loss: 0.1177, Test Loss: 0.1143
Epoch 27/60 - Train Loss: 0.1168, Test Loss: 0.1134
Epoch 28/60 - Train Loss: 0.1157, Test Loss: 0.1125
Epoch 29/60 - Train Loss: 0.1146, Test Loss: 0.1118
Epoch 30/60 - Train Loss: 0.1142, Test Loss: 0.1112
Epoch 31/60 - Train Loss: 0.1135, Test Loss: 0.1105
Epoch 32/60 - Train Loss: 0.1129, Test Loss: 0.1100
Epoch 33/60 - Train Loss: 0.1124, Test Loss: 0.1095
Epoch 34/60 - Train Loss: 0.1117, Test Loss: 0.1091
Epoch 35/60 - Train Loss: 0.1114, Test Loss: 0.1087
Epoch 36/60 - Train Loss: 0.1111, Test Loss: 0.1084
Epoch 37/60 - Train Loss: 0.1106, Test Loss: 0.1079
Epoch 38/60 - Train Loss: 0.1102, Test Loss: 0.1075
Epoch 39/60 - Train Loss: 0.1093, Test Loss: 0.1071
Epoch 40/60 - Train Loss: 0.1085, Test Loss: 0.1067
Epoch 41/60 - Train Loss: 0.1085, Test Loss: 0.1063
Epoch 42/60 - Train Loss: 0.1082, Test Loss: 0.1062
Epoch 43/60 - Train Loss: 0.1078, Test Loss: 0.1054
Epoch 44/60 - Train Loss: 0.1071, Test Loss: 0.1050
Epoch 45/60 - Train Loss: 0.1072, Test Loss: 0.1045
Epoch 46/60 - Train Loss: 0.1066, Test Loss: 0.1041
Epoch 47/60 - Train Loss: 0.1059, Test Loss: 0.1037
Epoch 48/60 - Train Loss: 0.1057, Test Loss: 0.1032
Epoch 49/60 - Train Loss: 0.1052, Test Loss: 0.1027
Epoch 50/60 - Train Loss: 0.1045, Test Loss: 0.1023
Epoch 51/60 - Train Loss: 0.1043, Test Loss: 0.1019
Epoch 52/60 - Train Loss: 0.1035, Test Loss: 0.1015
Epoch 53/60 - Train Loss: 0.1031, Test Loss: 0.1011
Epoch 54/60 - Train Loss: 0.1032, Test Loss: 0.1009
Epoch 55/60 - Train Loss: 0.1028, Test Loss: 0.1005
Epoch 56/60 - Train Loss: 0.1027, Test Loss: 0.1001
Epoch 57/60 - Train Loss: 0.1023, Test Loss: 0.0999
Epoch 58/60 - Train Loss: 0.1019, Test Loss: 0.0997
Epoch 59/60 - Train Loss: 0.1018, Test Loss: 0.0996
Epoch 60/60 - Train Loss: 0.1018, Test Loss: 0.0995
```

In [67]:
```
# TODO: Visualize the training and test loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(train_loss, label=f"Latent Dim: {latent_dim}")
plt.legend()
plt.subplot(1, 2, 2)
plt.title("Test Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.plot(test_loss, label=f"Latent Dim: {latent_dim}")
plt.legend()
plt.tight_layout()
plt.show()
```



In [70]:
```
# TODO: Visualize the reconstructed images and the original images
visualize_reconstruction(pca_ae_model, test_loader, device)
```



The results for the PCA reconstruction are worse compared to the CNN and the differences for the shape of the different classes are barely visible.

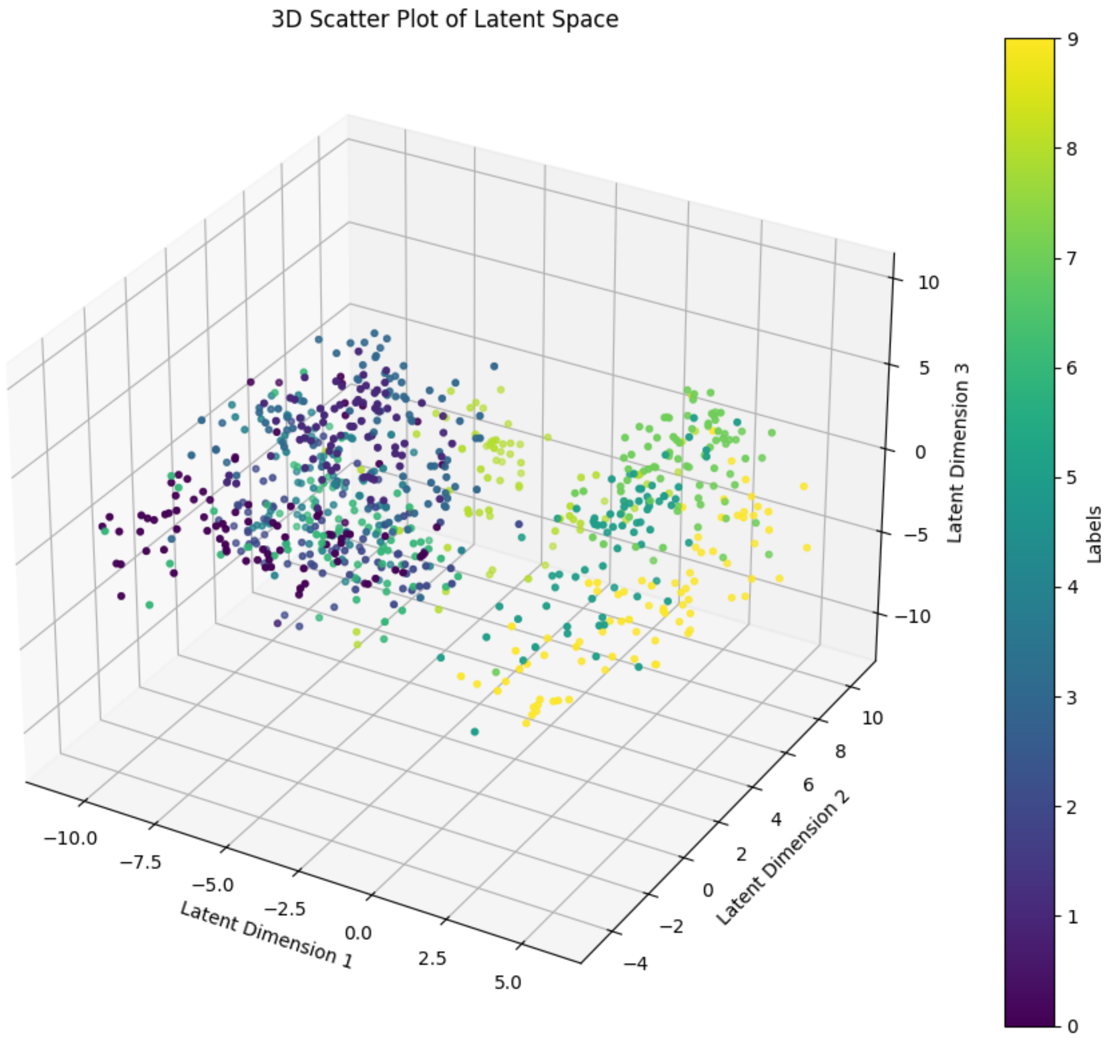
Task 4

In [107…
```
# TODO: Choose the convolutional autoencoder with latent dimension 3 and
conv_ae_model = models[latent_dims.index(3)]
conv_ae_model.eval()

latent_samples = []
```

```
latent_labels = []

with tc.no_grad():
    for X, Y in train_loader:
        X = X.to(device)
        output, latent = conv_ae_model(X)
        latent_samples.append(latent.cpu().numpy())
        latent_labels.append(Y.cpu().numpy())

        if len(latent_samples) * train_batch_size >= 800:
            break

latent_samples = np.concatenate(latent_samples)[:800]
latent_labels = np.concatenate(latent_labels)[:800]
```

In [110…

```
# TODO: Make a 3D scatter plot of the code colored by the labels
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(latent_samples[:, 0], latent_samples[:, 1], latent_samples[:,
ax.set_xlabel('Latent Dimension 1')
ax.set_ylabel('Latent Dimension 2')
ax.set_zlabel('Latent Dimension 3')
plt.title('3D Scatter Plot of Latent Space')
plt.colorbar(ax.scatter(latent_samples[:, 0], latent_samples[:, 1], laten
plt.tight_layout()
plt.show()
```



As one can see, there happens a clustering into two clusters but the points are still mixed together. But the clustering agrees with the underlying images, because classes 0 to 4 are all clothes and the shoes and bags (classes 5 to 9) are in the other cluster.