Ex7.VariationalAutoEncoder

January 16, 2024

7th exercise: Work with Variational Autoencoders (Generative Model)

- Course: AML
- Lecturer: Gernot Heisenberg
- Author of notebook: Finn Heydemann
- Date: 07.11.2023

GENERAL NOTE 1: Please make sure you are reading the entire notebook, since it contains a lot of information on your tasks (e.g. regarding the set of certain paramaters or a specific computational trick), and the written mark downs as well as comments contain a lot of information on how things work together as a whole.
GENERAL NOTE 2: * Please, when commenting source code, just use English language only. * When describing an observation please use English language, too. * This applies to all exercises throughout this course.

0.0.1 DESCRIPTION:

A Variational Autoencoder (VAE), instead of compressing its input image into a fixed code in the latent space (as the classic autoencoder does), turns the input image into the parameters of a statistical distribution: a mean and a variance.

This implies / imputes that the input image has been generated by a statistical process and that the randomness of this process should be taken into accounting during encoding and decoding.

The VAE then uses the mean and variance parameters to randomly sample one element of that distribution, and decodes that element back to the original input.

The stochasticity of this process improves robustness and forces the latent space to encode meaningful representations everywhere: every point sampled in the latent space is decoded to a valid output.

0.0.2 TASKS:

The tasks that you need to work on within this notebook are always indicated below as bullet points. If a task is more challenging and consists of several steps, this is indicated as well. Make sure you have worked down the task list and commented your doings. This should be done by using markdown. Make sure you don't forget to specify your name and your matriculation number in the notebook.

YOUR TASKS in this exercise are as follows: 1. import the notebook to Google Colab or use your local machine. 2. make sure you specified you name and your matriculation number in the header below my name and date. * set the date too and remove mine. 3. read the entire notebook carefully * add comments whereever you feel it necessary for better understanding * run the notebook for the first time. * try to understand each single step. 4. the notebooks code, especially keras is sometimes utilized a bit cumbersome. Try to optimize the code where you feel necessary. 5. experiment with different hyperparameters (search for the keyword 'task') 6. describe the three different loss curve plots. What do they show? Is this what you expected? 7. the main task is to visualize the latent space, the encoder has created. If you set high dimensions for the latent dim you can use T_SNE (plot 4). 8. describe the latent space with respect to its structure. Is this what you expected from a VAE? ——————————————

0.0.3 VAEs

This code demonstrates a VAE using the MNIST dataset. Just like a regular autoencoder a VAE returns an array (image) of same dimensions as the input but variation can be introduced by tweaking the so-called latent vector.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from   tensorflow.keras import layers, models, losses, metrics, optimizers
from   tensorflow.keras.datasets import mnist
```

```
2023-11-12 18:37:25.797225: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA, in
other operations, rebuild TensorFlow with the appropriate compiler flags.
```

0.0.4 Model: "Encoder"

Create an encoder model with the following properties:

[2]:
```
# ____________________________________________________________________________
#  Layer (type)                    Output Shape          Activation     Input
# ============================================================================
#  encoder_input (InputLayer)        [(None, 28, 28, 1)]  None
#  encoder_flatten (Flatten)         (None, 784)          None               ␣
↪enc_input
```

```
#  encoder_dense_1 (Dense)          (None, 2000)        ReLU            ␣
↪enc_flatten
#  encoder_dense_2 (Dense)          (None, 256)         ReLU            ␣
↪enc_dense_1
#  z_mean (Dense)                   (None, 2)           None            ␣
↪enc_dense_2
#  z_log_var (Dense)                (None, 2)           None            ␣
↪enc_dense_2

def make_encoder(latent_dim:int = 2):
    encoder_input = layers.Input(shape=(28,28,1), dtype='float32')
    encoder_flatten = layers.Flatten(name = 'flat')(encoder_input)
    encoder_dense_1 = layers.Dense(units = 2000, activation = 'relu', name =␣
↪'dense_1')(encoder_flatten)
    encoder_dense_2 = layers.Dense(units = 256, activation = 'relu', name =␣
↪'dense_2')(encoder_dense_1)
    z_mean = layers.Dense(units = latent_dim, name =␣
↪'latent_mean')(encoder_dense_2)
    z_log_var  = layers.Dense(units = latent_dim, name =␣
↪'latent_log_var')(encoder_dense_2)

    encoder = models.Model(inputs = encoder_input, outputs = (z_mean,␣
↪z_log_var), name = 'encoder')
    return encoder

make_encoder().summary()
```

```
Model: "encoder"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 input_1 (InputLayer)           [(None, 28, 28, 1)]  0           []

 flat (Flatten)                 (None, 784)          0
['input_1[0][0]']

 dense_1 (Dense)                (None, 2000)         1570000     ['flat[0][0]']

 dense_2 (Dense)                (None, 256)          512256
['dense_1[0][0]']

 latent_mean (Dense)            (None, 2)            514
['dense_2[0][0]']

 latent_log_var (Dense)         (None, 2)            514
```

```
['dense_2[0][0]']

=================================================================================
=================
Total params: 2,083,284
Trainable params: 2,083,284
Non-trainable params: 0
---------------------------------------------------------------------------------
-----------------
2023-11-12 18:37:26.947239: I
tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool
with default inter op setting: 2. Tune using inter_op_parallelism_threads for
best performance.
```

0.0.5 Model: "decoder"

Create a decoder model with the following properties:

[3]:
```python
# _______________________________________________________________________________
#  Layer (type)                 Output Shape          Activation     Input
# ===============================================================================
#  decoder_input (InputLayer)      [(None, 2)]           None
#  decoder_dense_1 (Dense)         (None, 256)           ReLU              ␣
 ↪dec_input
#  decoder_dense_2 (Dense)         (None, 2000)          ReLU              ␣
 ↪dec_dense_1
#  decoder_dense_3 (Dense)         (None, 784)           Sigmoid           ␣
 ↪dec_dense_2
#  img_out (Reshape)               (None, 28, 28, 1)     None              ␣
 ↪des_dense_3

def make_decoder(latent_dim:int = 2):
    decoder_input = layers.Input(shape=(latent_dim,), dtype='float32')
    decoder_dense_1 = layers.Dense(units = 256, activation = 'relu', name =␣
 ↪'dec_dense_1')(decoder_input)
    decoder_dense_2 = layers.Dense(units = 2000, activation = 'relu', name =␣
 ↪'dec_dense_2')(decoder_dense_1)
    decoder_dense_3 = layers.Dense(units = 784, activation = 'sigmoid', name =␣
 ↪'dec_dense_3')(decoder_dense_2)
    img_out = layers.Reshape((28,28,1), name = 'img_out')(decoder_dense_3)

    decoder = models.Model(inputs = decoder_input, outputs = img_out, name =␣
 ↪'decoder')
    return decoder

make_decoder().summary()
```

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 2)]               0

 dec_dense_1 (Dense)         (None, 256)               768

 dec_dense_2 (Dense)         (None, 2000)              514000

 dec_dense_3 (Dense)         (None, 784)               1568784

 img_out (Reshape)           (None, 28, 28, 1)         0

=================================================================
Total params: 2,083,552
Trainable params: 2,083,552
Non-trainable params: 0
_________________________________________________________________
```

[4]:
```
# class 'latent_sampling', which subclasses layers.Layer.
# The class should perform the reparameterisation trick in its .call() method.

'''___________________________________________________________________________'''
# Reparameterization Trick: z = mean + epsilon * exp(ln(variance) * 0.5)
# epsilon = N(0,1), a unit normal with same dims as mean and variance
'''___________________________________________________________________________'''
class latent_sampling(layers.Layer):

    def call(self, z_mean, z_log_var):
        tf.keras.layers.Layer(trainable = True)
        self.batch = tf.shape(z_mean)[0]
        self.dim = tf.shape(z_mean)[1]
        self.epsilon = tf.keras.backend.random_normal(shape=(self.batch, self.
↪dim))
        self.z =  z_mean + self.epsilon * tf.exp(z_log_var * 0.5)

        return self.z
```

[5]:
```
# Loss functions

# Modified loss function for the model. The standard binary cross entropy
# takes a mean over all pixels in all images, but the VAE needs the
# reconstruction loss to be the sum of the pixel-wise losses, averaged over
# samples in the batch. Otherwise the reconstruction loss is becoming too small.

def recon_loss(y_true, y_pred):
```

```
    loss = tf.reduce_sum(losses.binary_crossentropy(y_true, y_pred),axis=(1, 2))
    return loss

# Method that calculates the Kullback-Liebler divergence between the
# posterier distribution, N(mean, variance), and the prior, N(0,1).
# Can be added to the model as a loss or metric, using self.add_loss and
# self.add_metric


def kl_loss(mean, logvar):
    # Calculate the element wise KL divergence
    kl = -0.5 * (1 + logvar - tf.square(mean) - tf.exp(logvar))
    # Sum up the divergence of all the variables in each data sample
    kl = tf.reduce_sum(kl, axis=1)
    # Average the divergence across all samples in the batch
    kl = tf.reduce_mean(kl)
    return kl
```

[6]:
```
### The 'VAE' Class.
### The __init__ method-which will set up the layers and submodels-and the␣
↪call() method.

class VAE(tf.keras.Model):
    """
    A Keras Model that implements a Variational Autoencoder. Model properties
    should include the encoder and decoder models, a sampling layer, and the
    number of latent variables in the encoded space.
    """
    def __init__(self, latent_dim):
        super(VAE, self).__init__()
        """
        Take in model properties and assign them to self.
        """
        # self.latent_dim = latent_dim
        self.encoder = make_encoder(latent_dim)
        self.sampling = latent_sampling()
        self.decoder = make_decoder(latent_dim)

    def encode(self, x):
        """
        Method that applies the encoder model to input data. Returns the mean
        and ln(variance) of the encoded variables.
        """
        mean, logvar = self.encoder(x)
        return mean, logvar

    def decode(self, z):
```

```
        """
        Method that applies the decoder model to a set of encoded variables.
        Returns the generated images from the encoded data.
        """
        x_hat = self.decoder(z)
        return x_hat


    def call(self, inputs):
        """
        Apply the encoder, sampling layer and decoder to the input data. Add
        the kl divergence to the model losses and metrics. Return the generated
        image.
        """
        z_mean, z_log_var = self.encoder(inputs)
        sampled_output = self.sampling(z_mean, z_log_var)
        output = self.decoder(sampled_output)
        kll = kl_loss(z_mean, z_log_var)
        self.add_loss(kll)
        self.add_metric(kll, name = 'kl_loss_metric')
        return output
```

[7]:
```
# Create the VAE model, using your encoder and decoder models.
# Compile the model with appropriate optimizer settings, losses and metrics.
'''
(TASK: don't be afraid to experiment with different settings here (e.g.␣
 ↪latent_dim))
'''
autoencoder = VAE(latent_dim = 2)

# Default learning rate, optimizer = nAdam.
autoencoder.compile(tf.keras.optimizers.Nadam(),loss = recon_loss,
                    metrics = [recon_loss, 'accuracy'])
```

[7]:
```
# Load the MNIST data set
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Function to preprocess the data
def preprocessing(image):
    image =  tf.expand_dims(image, -1)
    image =  tf.image.random_flip_left_right(image)
    image  = tf.image.convert_image_dtype(image, 'float32')

    return image, image

# Slice off the training data
dataset = tf.data.Dataset.from_tensor_slices(x_train)
```

```
# Preparing the data for training
final_dataset = dataset.shuffle(1000).batch(64, drop_remainder=True).
↪map(preprocessing)
```

0.0.6 Train the model

Train the model on the images from the training set until the losses converge. "history = model.fit" allows for storing the training and validation losses in a dictionary so they can be visualized later.

[9]:
```
history = autoencoder.fit(final_dataset, batch_size = 256, epochs =30)
history.history.keys()
```

```
Epoch 1/30

2023-11-12 16:08:10.032667: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'Placeholder/_0' with dtype uint8 and shape [60000,28,28]
         [[{{node Placeholder/_0}}]]
2023-11-12 16:08:10.032900: I tensorflow/core/common_runtime/executor.cc:1197]
[/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an
error and you can ignore this message): INVALID_ARGUMENT: You must feed a value
for placeholder tensor 'Placeholder/_0' with dtype uint8 and shape [60000,28,28]
         [[{{node Placeholder/_0}}]]

937/937 [==============================] - 60s 62ms/step - loss: 178.7526 -
recon_loss: 173.8159 - accuracy: 0.7934 - kl_loss_metric: 4.9365
Epoch 2/30
937/937 [==============================] - 69s 74ms/step - loss: 164.7248 -
recon_loss: 159.3978 - accuracy: 0.7940 - kl_loss_metric: 5.3269
Epoch 3/30
937/937 [==============================] - 69s 74ms/step - loss: 161.5014 -
recon_loss: 155.8864 - accuracy: 0.7942 - kl_loss_metric: 5.6150
Epoch 4/30
937/937 [==============================] - 75s 80ms/step - loss: 159.8295 -
recon_loss: 154.0589 - accuracy: 0.7945 - kl_loss_metric: 5.7706
Epoch 5/30
937/937 [==============================] - 77s 82ms/step - loss: 158.5398 -
recon_loss: 152.6684 - accuracy: 0.7946 - kl_loss_metric: 5.8713
Epoch 6/30
937/937 [==============================] - 75s 80ms/step - loss: 157.0656 -
recon_loss: 151.0471 - accuracy: 0.7950 - kl_loss_metric: 6.0185
Epoch 7/30
937/937 [==============================] - 72s 77ms/step - loss: 155.8707 -
recon_loss: 149.7180 - accuracy: 0.7955 - kl_loss_metric: 6.1526
Epoch 8/30
937/937 [==============================] - 81s 86ms/step - loss: 154.5947 -
recon_loss: 148.3158 - accuracy: 0.7960 - kl_loss_metric: 6.2789
```

```
Epoch 9/30
937/937 [==============================] - 78s 84ms/step - loss: 153.7154 -
recon_loss: 147.3602 - accuracy: 0.7963 - kl_loss_metric: 6.3555
Epoch 10/30
937/937 [==============================] - 76s 81ms/step - loss: 152.7589 -
recon_loss: 146.2833 - accuracy: 0.7966 - kl_loss_metric: 6.4756
Epoch 11/30
937/937 [==============================] - 77s 82ms/step - loss: 152.1922 -
recon_loss: 145.6443 - accuracy: 0.7969 - kl_loss_metric: 6.5479
Epoch 12/30
937/937 [==============================] - 77s 82ms/step - loss: 151.5773 -
recon_loss: 144.9607 - accuracy: 0.7970 - kl_loss_metric: 6.6166
Epoch 13/30
937/937 [==============================] - 78s 83ms/step - loss: 151.1019 -
recon_loss: 144.4544 - accuracy: 0.7972 - kl_loss_metric: 6.6475
Epoch 14/30
937/937 [==============================] - 70s 75ms/step - loss: 150.3717 -
recon_loss: 143.6648 - accuracy: 0.7974 - kl_loss_metric: 6.7068
Epoch 15/30
937/937 [==============================] - 66s 71ms/step - loss: 149.8681 -
recon_loss: 143.0974 - accuracy: 0.7975 - kl_loss_metric: 6.7708
Epoch 16/30
937/937 [==============================] - 69s 74ms/step - loss: 149.7432 -
recon_loss: 142.9622 - accuracy: 0.7976 - kl_loss_metric: 6.7809
Epoch 17/30
937/937 [==============================] - 71s 75ms/step - loss: 149.1606 -
recon_loss: 142.3198 - accuracy: 0.7978 - kl_loss_metric: 6.8409
Epoch 18/30
937/937 [==============================] - 76s 82ms/step - loss: 149.1718 -
recon_loss: 142.3278 - accuracy: 0.7978 - kl_loss_metric: 6.8439
Epoch 19/30
937/937 [==============================] - 63s 67ms/step - loss: 148.8766 -
recon_loss: 141.9970 - accuracy: 0.7979 - kl_loss_metric: 6.8797
Epoch 20/30
937/937 [==============================] - 66s 71ms/step - loss: 148.5491 -
recon_loss: 141.6313 - accuracy: 0.7980 - kl_loss_metric: 6.9178
Epoch 21/30
937/937 [==============================] - 64s 68ms/step - loss: 148.3071 -
recon_loss: 141.3691 - accuracy: 0.7981 - kl_loss_metric: 6.9381
Epoch 22/30
937/937 [==============================] - 70s 74ms/step - loss: 148.0677 -
recon_loss: 141.0969 - accuracy: 0.7981 - kl_loss_metric: 6.9708
Epoch 23/30
937/937 [==============================] - 67s 72ms/step - loss: 147.8079 -
recon_loss: 140.8108 - accuracy: 0.7982 - kl_loss_metric: 6.9972
Epoch 24/30
937/937 [==============================] - 66s 71ms/step - loss: 147.6178 -
recon_loss: 140.5965 - accuracy: 0.7983 - kl_loss_metric: 7.0214
```

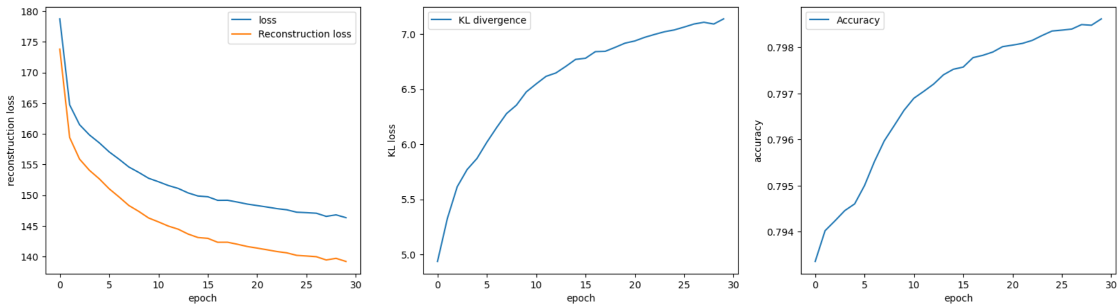
```
Epoch 25/30
937/937 [==============================] - 76s 81ms/step - loss: 147.2259 -
recon_loss: 140.1879 - accuracy: 0.7984 - kl_loss_metric: 7.0381
Epoch 26/30
937/937 [==============================] - 69s 74ms/step - loss: 147.1502 -
recon_loss: 140.0860 - accuracy: 0.7984 - kl_loss_metric: 7.0642
Epoch 27/30
937/937 [==============================] - 57s 61ms/step - loss: 147.0564 -
recon_loss: 139.9646 - accuracy: 0.7984 - kl_loss_metric: 7.0918
Epoch 28/30
937/937 [==============================] - 71s 76ms/step - loss: 146.5436 -
recon_loss: 139.4360 - accuracy: 0.7985 - kl_loss_metric: 7.1076
Epoch 29/30
937/937 [==============================] - 69s 73ms/step - loss: 146.7914 -
recon_loss: 139.6999 - accuracy: 0.7985 - kl_loss_metric: 7.0916
Epoch 30/30
937/937 [==============================] - 71s 75ms/step - loss: 146.3331 -
recon_loss: 139.1948 - accuracy: 0.7986 - kl_loss_metric: 7.1384
```

[9]:
```
dict_keys(['loss', 'recon_loss', 'accuracy', 'kl_loss_metric'])
```

0.0.7 Visualize the results (plot 1)

Create plots that show the losses and metrics, the reconstruction quality of the trained network, and the generative quality of the network.

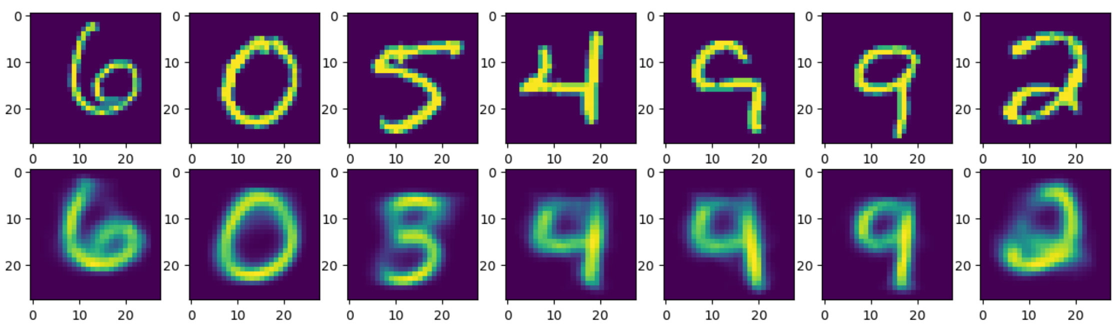
[10]:
```python
f, ax = plt.subplots(1,3, figsize = (20,5))

ax[0].plot(history.history['loss'], label = 'loss')
ax[0].plot(history.history['recon_loss'], label = 'Reconstruction loss')
ax[0].set_ylabel('reconstruction loss')
ax[0].set_xlabel('epoch')
ax[0].legend()

ax[1].plot(history.history['kl_loss_metric'], label = 'KL divergence')
ax[1].set_ylabel('KL loss')
ax[1].set_xlabel('epoch')
ax[1].legend()

ax[2].plot(history.history['accuracy'], label = 'Accuracy')
ax[2].set_ylabel('accuracy')
ax[2].set_xlabel('epoch')
ax[2].legend()

plt.show()
```



0.0.8 Prediction of test data (plot 2)

[11]:
```
predict = autoencoder.predict(x_test/255.)
f, ax = plt.subplots(2, 7, figsize = (15,4))

# Testing the reconstruction quality of the network using the Test Images
for i in range(7):
    ax[0,i].imshow(x_test[i+100])
    ax[1,i].imshow(predict[i+100,:,:,0])
```

```
313/313 [==============================] - 2s 6ms/step
```



0.0.9 Testing the generative quality of the network (plot 3)

[12]:
```
x_axis = np.linspace(-1, 1, 10)
y_axis = np.linspace(-1, 1, 10)
figure = np.zeros((28 * 10, 28 * 10))

# loop through each number for decoding
for i_x, x in enumerate(x_axis):
    for i_y, y in enumerate(y_axis):
        latent = np.array([[x, y]])
        print(latent.shape)
```

```
        generated_image = autoencoder.decoder.predict(latent)[0] # decode the␣
↪numbers
        figure[i_x*28:(i_x+1)*28, i_y*28:(i_y+1)*28,] = generated_image[:,:,-1]
```

```
(1, 2)
1/1 [==============================] - 0s 34ms/step
(1, 2)
1/1 [==============================] - 0s 44ms/step
(1, 2)
1/1 [==============================] - 0s 11ms/step
(1, 2)
1/1 [==============================] - 0s 15ms/step
(1, 2)
1/1 [==============================] - 0s 17ms/step
(1, 2)
1/1 [==============================] - 0s 22ms/step
(1, 2)
1/1 [==============================] - 0s 35ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 56ms/step
(1, 2)
1/1 [==============================] - 0s 54ms/step
(1, 2)
1/1 [==============================] - 0s 50ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 17ms/step
(1, 2)
1/1 [==============================] - 0s 19ms/step
(1, 2)
1/1 [==============================] - 0s 17ms/step
(1, 2)
1/1 [==============================] - 0s 18ms/step
(1, 2)
1/1 [==============================] - 0s 18ms/step
(1, 2)
1/1 [==============================] - 0s 19ms/step
(1, 2)
1/1 [==============================] - 0s 18ms/step
(1, 2)
1/1 [==============================] - 0s 18ms/step
(1, 2)
1/1 [==============================] - 0s 19ms/step
(1, 2)
```

```
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 28ms/step
(1, 2)
1/1 [==============================] - 0s 19ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 23ms/step
(1, 2)
1/1 [==============================] - 0s 15ms/step
(1, 2)
1/1 [==============================] - 0s 17ms/step
(1, 2)
1/1 [==============================] - 0s 28ms/step
(1, 2)
1/1 [==============================] - 0s 36ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 22ms/step
(1, 2)
1/1 [==============================] - 0s 23ms/step
(1, 2)
1/1 [==============================] - 0s 22ms/step
(1, 2)
1/1 [==============================] - 0s 23ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
```

```
1/1 [==============================] - 0s 22ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 20ms/step
(1, 2)
1/1 [==============================] - 0s 23ms/step
(1, 2)
1/1 [==============================] - 0s 52ms/step
(1, 2)
1/1 [==============================] - 0s 48ms/step
(1, 2)
1/1 [==============================] - 0s 30ms/step
(1, 2)
1/1 [==============================] - 0s 19ms/step
(1, 2)
1/1 [==============================] - 0s 73ms/step
(1, 2)
1/1 [==============================] - 0s 36ms/step
(1, 2)
1/1 [==============================] - 0s 36ms/step
(1, 2)
1/1 [==============================] - 0s 53ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 33ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 30ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 27ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
```

```
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 27ms/step
(1, 2)
1/1 [==============================] - 0s 34ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 34ms/step
(1, 2)
1/1 [==============================] - 0s 27ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 33ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 31ms/step
(1, 2)
1/1 [==============================] - 0s 28ms/step
(1, 2)
1/1 [==============================] - 0s 37ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 29ms/step
(1, 2)
1/1 [==============================] - 0s 33ms/step
(1, 2)
1/1 [==============================] - 0s 35ms/step
(1, 2)
1/1 [==============================] - 0s 31ms/step
(1, 2)
1/1 [==============================] - 0s 27ms/step
(1, 2)
1/1 [==============================] - 0s 26ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 25ms/step
(1, 2)
```

```
1/1 [==============================] - 0s 34ms/step
(1, 2)
1/1 [==============================] - 0s 21ms/step
(1, 2)
1/1 [==============================] - 0s 38ms/step
(1, 2)
1/1 [==============================] - 0s 33ms/step
(1, 2)
1/1 [==============================] - 0s 58ms/step
(1, 2)
1/1 [==============================] - 0s 43ms/step
(1, 2)
1/1 [==============================] - 0s 75ms/step
```

[13]:

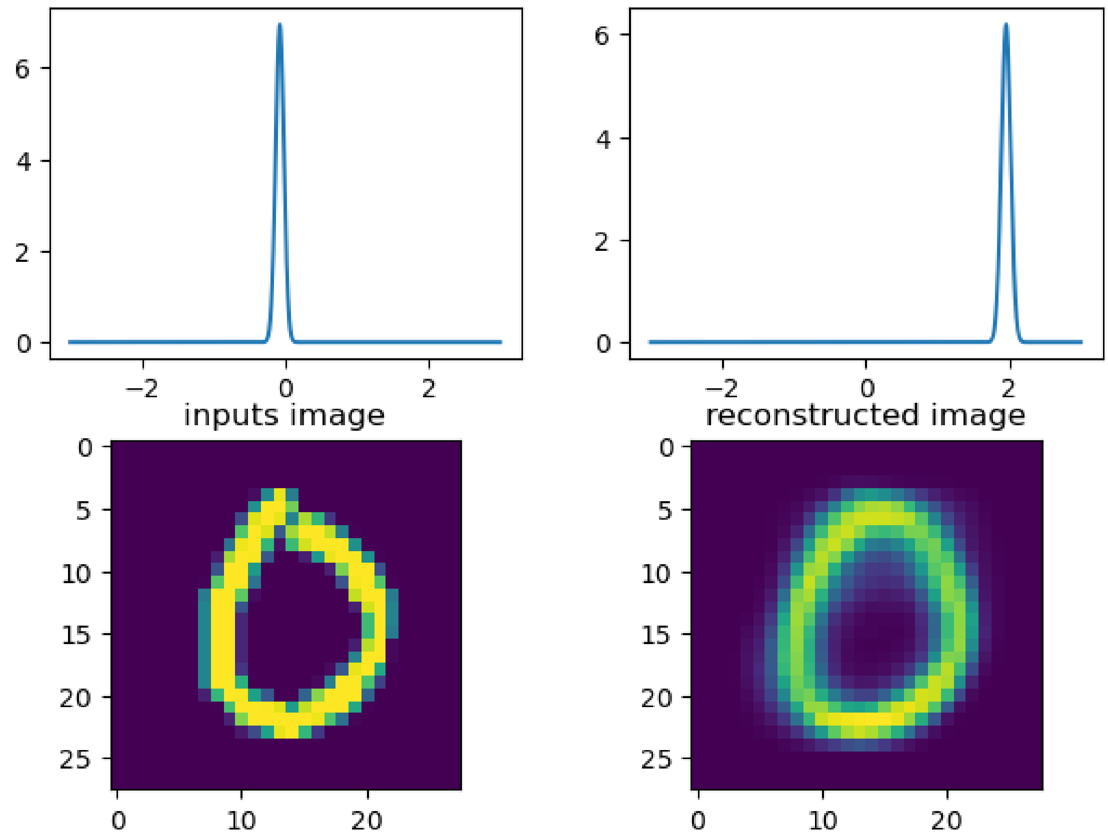
```
plt.figure(figsize=(15, 15))
plt.imshow(figure, extent=[1,-1,1,-1])
plt.show()
```

0.0.10 Task: Visualize the latent space (if latent_dim > 2 then by using T_SNE) (plot 4)

Describe the latent space with respect to its structure. Remember: t-SNE is stochastic and therefore the results may appear slightly different every time it is re-run. So don't worry.

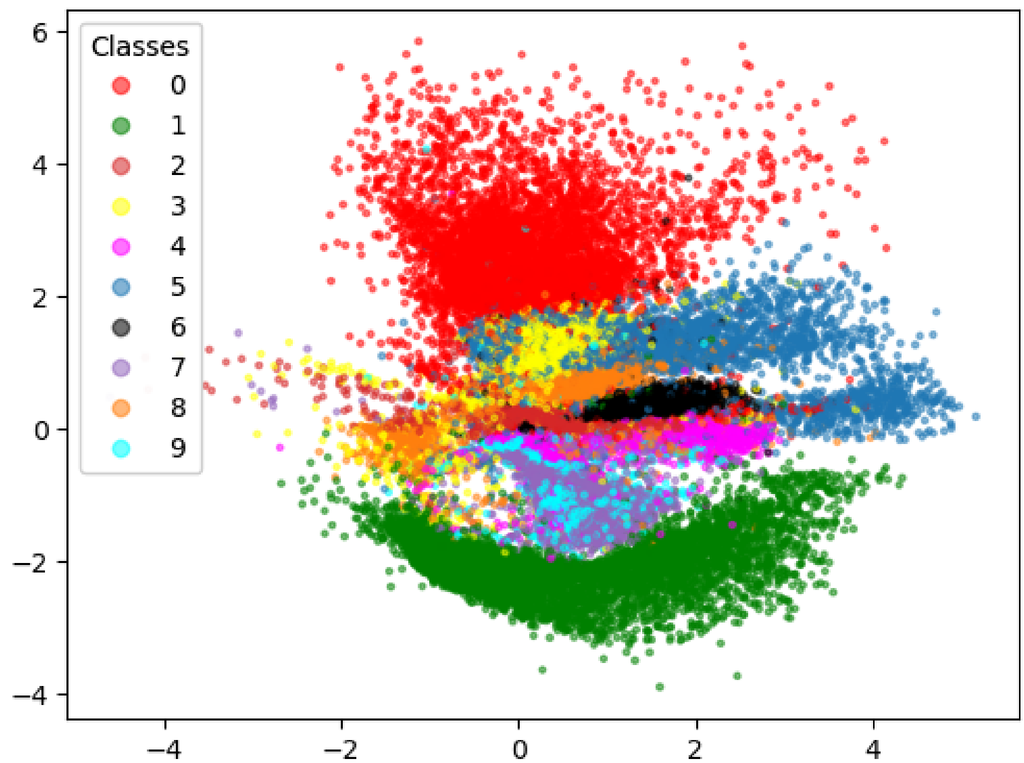
[14]:
```
# add your code section here !
```

[15]:
```
from scipy import stats

index = 612
img = (x_train / 255)[index:index+1]

enc = autoencoder.encoder(img)
fig, axs = plt.subplots(2, 2)
fig.tight_layout()

for ax, (mu, logvar) in zip(axs.flatten(), np.stack([e.numpy().reshape(-1) for␣
 ↪e in enc]).T):
    x = np.linspace(-3, 3, 1000)
    sigma = np.exp(logvar * .5)
    ax.plot(x, stats.norm.pdf(x, mu, sigma))

axs[1, 0].set_title("inputs image")
axs[1, 0].imshow(img[0])

sampled_data = latent_sampling().call(*enc)
axs[1, 1].set_title("reconstructed image")
axs[1, 1].imshow(autoencoder.decoder(sampled_data)[0, :, :, 0])
plt.show()
```



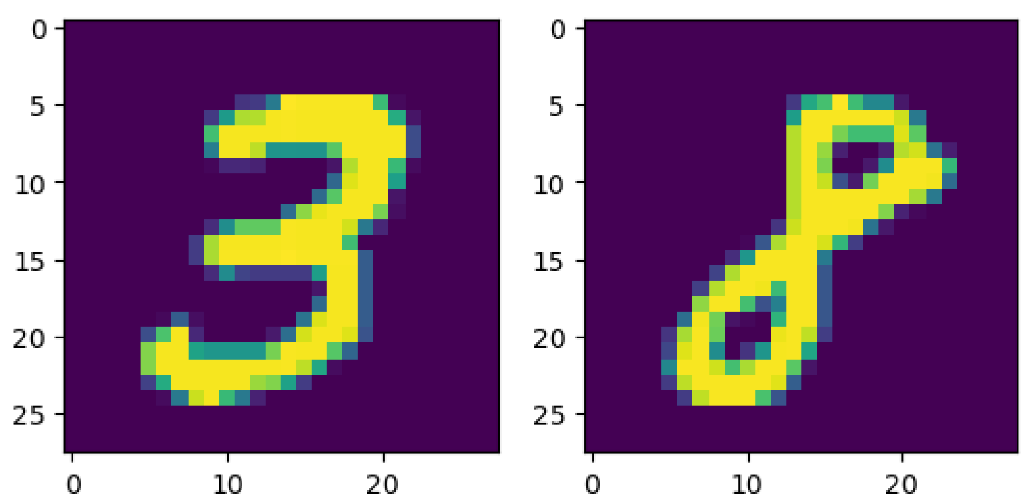
[10]:
```python
from matplotlib.colors import ListedColormap

def make_scatter_plot(mu, y_train, **scater_kwargs):
    fig, ax = plt.subplots()
    colormap = ListedColormap(["red", "green", "tab:red", "yellow", "magenta",␣
 ↪"tab:blue", "black", "tab:purple", "tab:orange", "cyan"])

    scatter = ax.scatter(*np.array(mu).T, c=y_train,
                         cmap=colormap,
                         **scater_kwargs)

    # produce a legend with the unique colors from the scatter
    legend1 = ax.legend(*scatter.legend_elements(),
                        loc="best", title="Classes")
    ax.add_artist(legend1)
    return fig, ax
```

[16]:
```python
mu, logvar = autoencoder.encoder(x_train / 255)
fig, ax = make_scatter_plot(mu, y_train, s=5, alpha=.5)
plt.show()
```



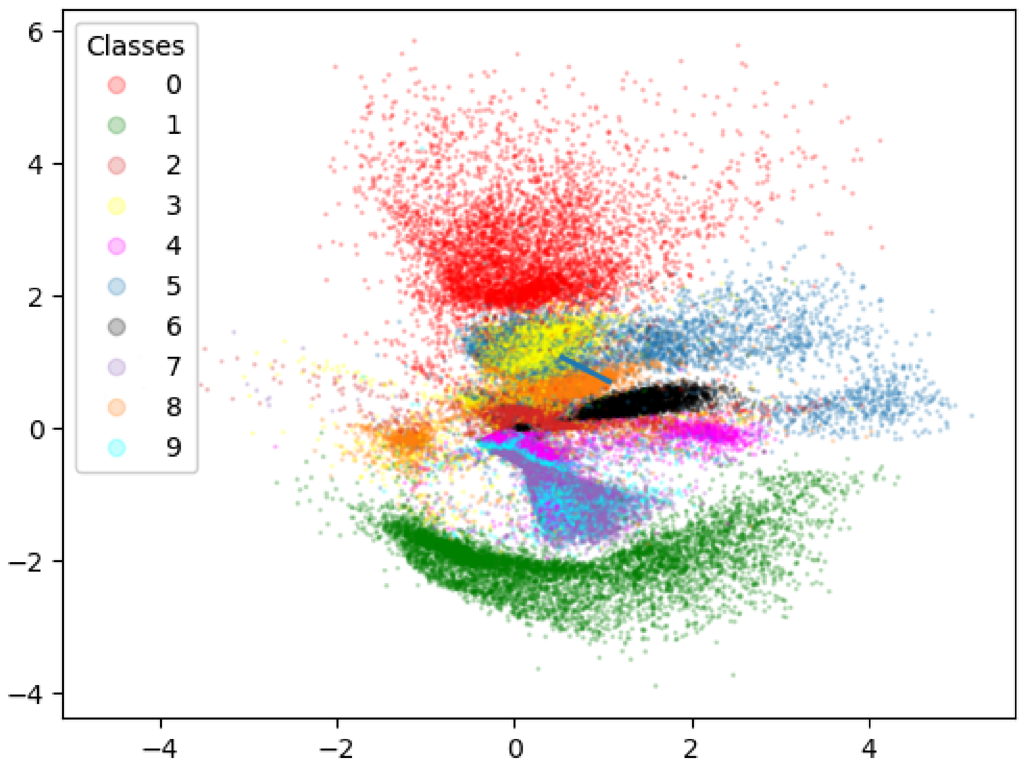

```
[17]: fig, axs = plt.subplots(1, 2)
      axs[0].imshow(x_train[7])
      axs[1].imshow(x_train[31])
```

```
[17]: <matplotlib.image.AxesImage at 0x7fab1466fa90>
```



[18]:

```
mu3, logvar3 = autoencoder.encoder(x_train[7:8] / 255)
mu8, logvar8 = autoencoder.encoder(x_train[31:32] /255)

mu, logvar = autoencoder.encoder(x_train / 255)
fig, ax = make_scatter_plot(mu, y_train, alpha=.2, s=1)
ax.plot(*np.stack([mu3, mu8])[:, 0, :].T, lw=2)
plt.show()
```



[19]:

```
vec38 = mu8 - mu3
length = np.linalg.norm(vec38)
mu3, vec38, length
```

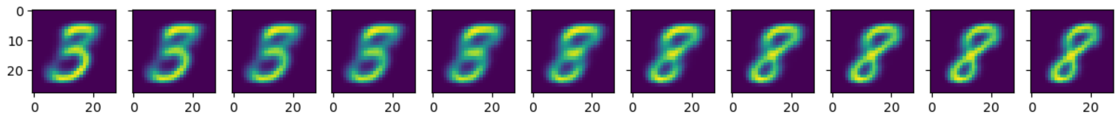
[19]:

```
(<tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[0.5450834, 1.0702983]],
dtype=float32)>,
 <tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[ 0.52775824, -0.3650753
]], dtype=float32)>,
 0.6417233)
```

[20]:

```
# transform 3 to an 8
steps = 10
fig, axs = plt.subplots(1, steps + 1, sharey=True, figsize=(15, 6))
```

```
for i in range(steps + 1):
    axs[i].imshow(autoencoder.decode(mu3 + vec38 * (i / steps))[0, :, :, 0] *␣
 ↪255)
```



[21]:
```
class SimpleVAE(VAE):

    def __init__(self, latent_dim):
        super().__init__(latent_dim)

    def call(self, inputs):
        z_mean, z_log_var = self.encoder(inputs)
        sampled_output = self.sampling(z_mean, z_log_var)
        output = self.decoder(sampled_output)
        return output

autoencoder_rl = SimpleVAE(2)
autoencoder_rl.compile(tf.keras.optimizers.Nadam(),loss = recon_loss,␣
 ↪metrics=["accuracy", recon_loss])
autoencoder_rl.fit(final_dataset, batch_size=256, epochs=30)
```

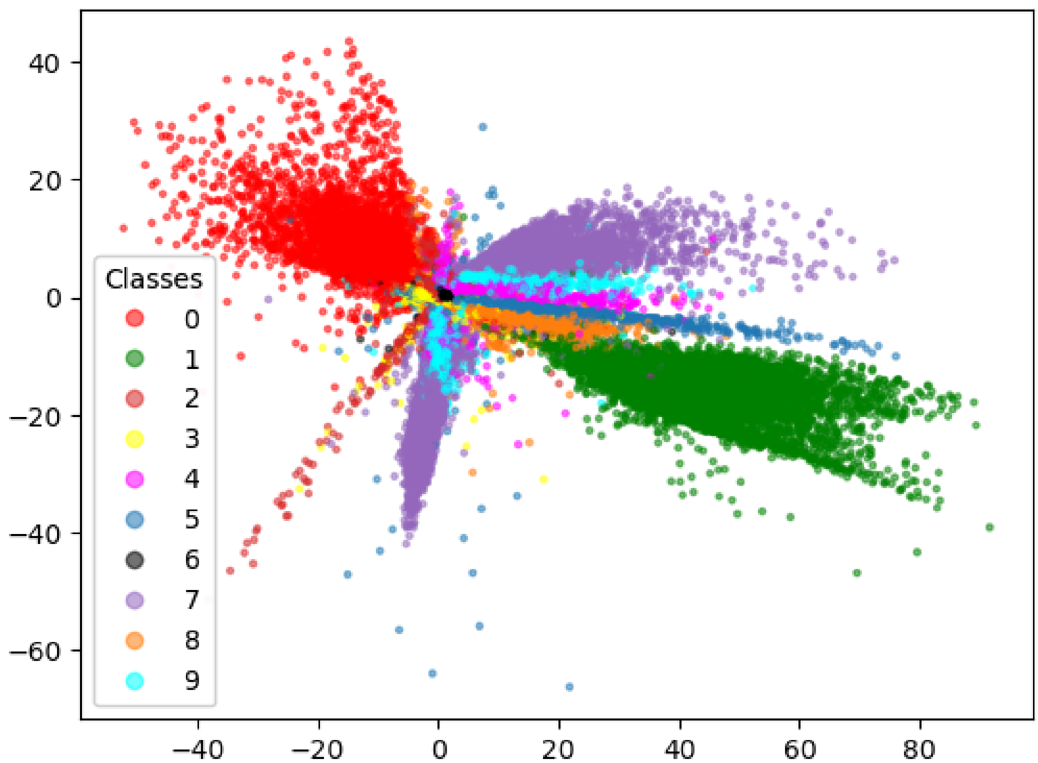
```
Epoch 1/30
937/937 [==============================] - 72s 76ms/step - loss: 171.6586 -
accuracy: 0.7922 - recon_loss: 171.6586
Epoch 2/30
937/937 [==============================] - 71s 76ms/step - loss: 156.9735 -
accuracy: 0.7940 - recon_loss: 156.9735
Epoch 3/30
937/937 [==============================] - 68s 72ms/step - loss: 152.9373 -
accuracy: 0.7949 - recon_loss: 152.9373
Epoch 4/30
937/937 [==============================] - 67s 71ms/step - loss: 150.3996 -
accuracy: 0.7954 - recon_loss: 150.3996
Epoch 5/30
937/937 [==============================] - 74s 79ms/step - loss: 148.5307 -
accuracy: 0.7959 - recon_loss: 148.5307
Epoch 6/30
937/937 [==============================] - 69s 73ms/step - loss: 146.6259 -
accuracy: 0.7963 - recon_loss: 146.6259
Epoch 7/30
937/937 [==============================] - 73s 78ms/step - loss: 145.4773 -
```

```
accuracy: 0.7966 - recon_loss: 145.4773
Epoch 8/30
937/937 [==============================] - 73s 78ms/step - loss: 144.4462 -
accuracy: 0.7968 - recon_loss: 144.4462
Epoch 9/30
937/937 [==============================] - 75s 80ms/step - loss: 143.5263 -
accuracy: 0.7970 - recon_loss: 143.5263
Epoch 10/30
937/937 [==============================] - 67s 72ms/step - loss: 143.1402 -
accuracy: 0.7971 - recon_loss: 143.1402
Epoch 11/30
937/937 [==============================] - 75s 81ms/step - loss: 142.0425 -
accuracy: 0.7974 - recon_loss: 142.0425
Epoch 12/30
937/937 [==============================] - 72s 77ms/step - loss: 141.5271 -
accuracy: 0.7976 - recon_loss: 141.5271
Epoch 13/30
937/937 [==============================] - 75s 81ms/step - loss: 140.8691 -
accuracy: 0.7976 - recon_loss: 140.8691
Epoch 14/30
937/937 [==============================] - 72s 77ms/step - loss: 140.0132 -
accuracy: 0.7979 - recon_loss: 140.0132
Epoch 15/30
937/937 [==============================] - 67s 72ms/step - loss: 140.1476 -
accuracy: 0.7979 - recon_loss: 140.1476
Epoch 16/30
937/937 [==============================] - 76s 81ms/step - loss: 139.2444 -
accuracy: 0.7981 - recon_loss: 139.2444
Epoch 17/30
937/937 [==============================] - 70s 75ms/step - loss: 138.9846 -
accuracy: 0.7981 - recon_loss: 138.9846
Epoch 18/30
937/937 [==============================] - 75s 80ms/step - loss: 138.7413 -
accuracy: 0.7982 - recon_loss: 138.7413
Epoch 19/30
937/937 [==============================] - 74s 79ms/step - loss: 138.4829 -
accuracy: 0.7982 - recon_loss: 138.4829
Epoch 20/30
937/937 [==============================] - 73s 78ms/step - loss: 137.9688 -
accuracy: 0.7984 - recon_loss: 137.9688
Epoch 21/30
937/937 [==============================] - 63s 67ms/step - loss: 138.2464 -
accuracy: 0.7984 - recon_loss: 138.2464
Epoch 22/30
937/937 [==============================] - 75s 80ms/step - loss: 137.7760 -
accuracy: 0.7984 - recon_loss: 137.7760
Epoch 23/30
937/937 [==============================] - 68s 73ms/step - loss: 137.8123 -
```

```
accuracy: 0.7984 - recon_loss: 137.8123
Epoch 24/30
937/937 [==============================] - 67s 71ms/step - loss: 137.9104 -
accuracy: 0.7984 - recon_loss: 137.9104
Epoch 25/30
937/937 [==============================] - 77s 82ms/step - loss: 137.8315 -
accuracy: 0.7984 - recon_loss: 137.8315
Epoch 26/30
937/937 [==============================] - 74s 79ms/step - loss: 137.5948 -
accuracy: 0.7985 - recon_loss: 137.5948
Epoch 27/30
937/937 [==============================] - 74s 79ms/step - loss: 137.2370 -
accuracy: 0.7986 - recon_loss: 137.2370
Epoch 28/30
937/937 [==============================] - 74s 79ms/step - loss: 137.1545 -
accuracy: 0.7985 - recon_loss: 137.1545
Epoch 29/30
937/937 [==============================] - 77s 83ms/step - loss: 136.6813 -
accuracy: 0.7986 - recon_loss: 136.6813
Epoch 30/30
937/937 [==============================] - 72s 77ms/step - loss: 136.2618 -
accuracy: 0.7988 - recon_loss: 136.2618
```

[21]:
```
<keras.callbacks.History at 0x7fab16858ed0>
```

[22]:
```python
mu, logvar = autoencoder_rl.encoder(x_train / 255)
fig, ax = make_scatter_plot(mu, y_train, s=5, alpha=.5)
```



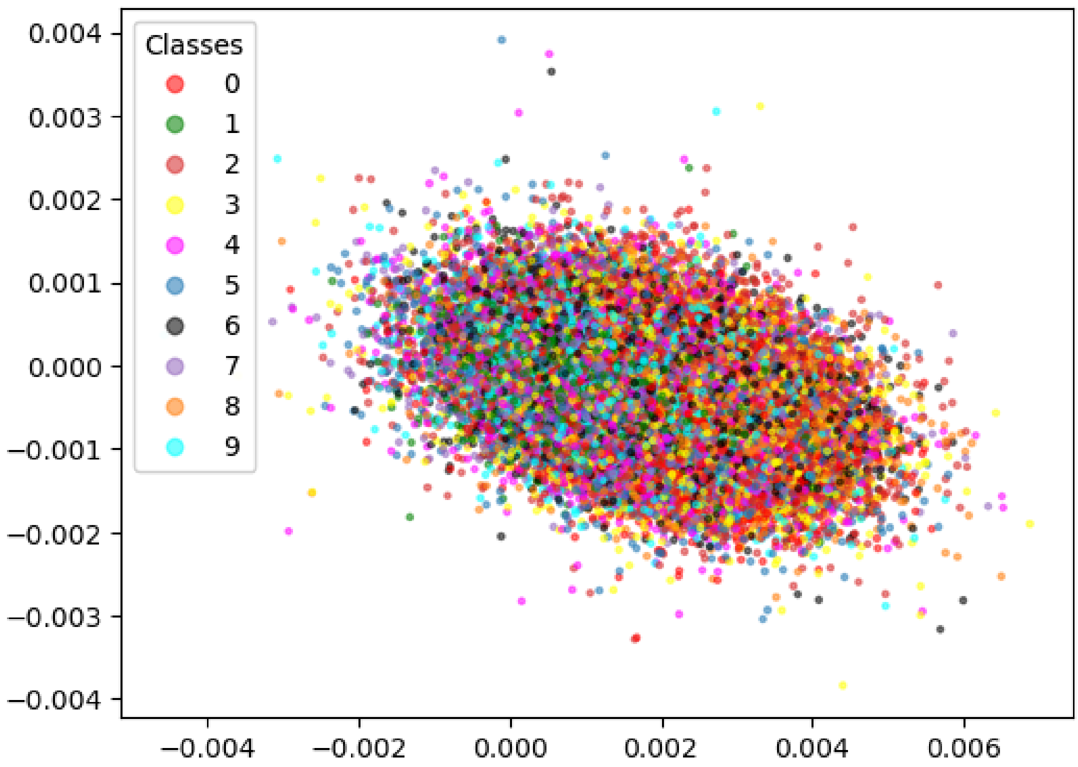
[12]:
```
autoencoder_kl = VAE(2)
autoencoder_kl.compile(tf.keras.optimizers.Nadam(), loss= lambda x, y: 0,␣
↪metrics=["accuracy"])
autoencoder_kl.fit(final_dataset, batch_size=256, epochs=5)
```

```
Epoch 1/5
WARNING:tensorflow:Gradients do not exist for variables ['dec_dense_1/kernel:0',
'dec_dense_1/bias:0', 'dec_dense_2/kernel:0', 'dec_dense_2/bias:0',
'dec_dense_3/kernel:0', 'dec_dense_3/bias:0'] when minimizing the loss. If
you're using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['dec_dense_1/kernel:0',
'dec_dense_1/bias:0', 'dec_dense_2/kernel:0', 'dec_dense_2/bias:0',
'dec_dense_3/kernel:0', 'dec_dense_3/bias:0'] when minimizing the loss. If
you're using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['dec_dense_1/kernel:0',
'dec_dense_1/bias:0', 'dec_dense_2/kernel:0', 'dec_dense_2/bias:0',
'dec_dense_3/kernel:0', 'dec_dense_3/bias:0'] when minimizing the loss. If
you're using `model.compile()`, did you forget to provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['dec_dense_1/kernel:0',
'dec_dense_1/bias:0', 'dec_dense_2/kernel:0', 'dec_dense_2/bias:0',
'dec_dense_3/kernel:0', 'dec_dense_3/bias:0'] when minimizing the loss. If
you're using `model.compile()`, did you forget to provide a `loss` argument?
```

```
937/937 [==============================] - 30s 31ms/step - loss: 0.0044 -
accuracy: 0.3939 - kl_loss_metric: 0.0044
Epoch 2/5
937/937 [==============================] - 39s 42ms/step - loss: 2.7704e-05 -
accuracy: 0.3938 - kl_loss_metric: 2.7704e-05
Epoch 3/5
937/937 [==============================] - 41s 44ms/step - loss: 1.3141e-05 -
accuracy: 0.3937 - kl_loss_metric: 1.3141e-05
Epoch 4/5
937/937 [==============================] - 31s 33ms/step - loss: 6.7413e-06 -
accuracy: 0.3939 - kl_loss_metric: 6.7413e-06
Epoch 5/5
937/937 [==============================] - 45s 48ms/step - loss: 3.6658e-06 -
accuracy: 0.3939 - kl_loss_metric: 3.6658e-06
```

[12]: <keras.callbacks.History at 0x7f3e3c749410>

[13]:
```
mu, logvar = autoencoder_kl.encoder(x_train / 255)
fig, ax = make_scatter_plot(mu, y_train, s=5, alpha=.5)
plt.show()
```



[25]:
```
from sklearn.manifold import TSNE
```

[26]:
```
autoencoder3d = VAE(3)

autoencoder3d.compile(tf.keras.optimizers.Nadam(),loss = recon_loss, metrics =
 ↪[recon_loss, 'accuracy'])
autoencoder3d.fit(final_dataset, batch_size = 256, epochs =30)
```

```
Epoch 1/30
937/937 [==============================] - 77s 81ms/step - loss: 167.4568 -
recon_loss: 160.7381 - accuracy: 0.7939 - kl_loss_metric: 6.7186
Epoch 2/30
937/937 [==============================] - 74s 79ms/step - loss: 152.2176 -
recon_loss: 144.8506 - accuracy: 0.7969 - kl_loss_metric: 7.3670
Epoch 3/30
937/937 [==============================] - 68s 73ms/step - loss: 147.9819 -
recon_loss: 140.2288 - accuracy: 0.7978 - kl_loss_metric: 7.7532
Epoch 4/30
937/937 [==============================] - 70s 75ms/step - loss: 145.3789 -
recon_loss: 137.3445 - accuracy: 0.7984 - kl_loss_metric: 8.0345
Epoch 5/30
937/937 [==============================] - 71s 76ms/step - loss: 143.3012 -
recon_loss: 135.0583 - accuracy: 0.7989 - kl_loss_metric: 8.2428
Epoch 6/30
937/937 [==============================] - 68s 73ms/step - loss: 141.7202 -
recon_loss: 133.3374 - accuracy: 0.7994 - kl_loss_metric: 8.3829
Epoch 7/30
937/937 [==============================] - 71s 76ms/step - loss: 140.4082 -
recon_loss: 131.8803 - accuracy: 0.7997 - kl_loss_metric: 8.5279
Epoch 8/30
937/937 [==============================] - 76s 82ms/step - loss: 139.4824 -
recon_loss: 130.8406 - accuracy: 0.8000 - kl_loss_metric: 8.6418
Epoch 9/30
937/937 [==============================] - 75s 80ms/step - loss: 138.7017 -
recon_loss: 129.9822 - accuracy: 0.8002 - kl_loss_metric: 8.7195
Epoch 10/30
937/937 [==============================] - 75s 80ms/step - loss: 137.8387 -
recon_loss: 129.0171 - accuracy: 0.8005 - kl_loss_metric: 8.8216
Epoch 11/30
937/937 [==============================] - 84s 90ms/step - loss: 137.1180 -
recon_loss: 128.2190 - accuracy: 0.8007 - kl_loss_metric: 8.8990
Epoch 12/30
937/937 [==============================] - 77s 83ms/step - loss: 136.6233 -
recon_loss: 127.6727 - accuracy: 0.8008 - kl_loss_metric: 8.9505
Epoch 13/30
937/937 [==============================] - 79s 85ms/step - loss: 136.2528 -
recon_loss: 127.2514 - accuracy: 0.8009 - kl_loss_metric: 9.0013
Epoch 14/30
937/937 [==============================] - 82s 88ms/step - loss: 135.6930 -
```

```
recon_loss: 126.6568 - accuracy: 0.8011 - kl_loss_metric: 9.0361
Epoch 15/30
937/937 [==============================] - 63s 68ms/step - loss: 135.2268 -
recon_loss: 126.1516 - accuracy: 0.8012 - kl_loss_metric: 9.0751
Epoch 16/30
937/937 [==============================] - 81s 86ms/step - loss: 134.9568 -
recon_loss: 125.8389 - accuracy: 0.8013 - kl_loss_metric: 9.1180
Epoch 17/30
937/937 [==============================] - 75s 80ms/step - loss: 134.8733 -
recon_loss: 125.7105 - accuracy: 0.8013 - kl_loss_metric: 9.1629
Epoch 18/30
937/937 [==============================] - 74s 79ms/step - loss: 134.4916 -
recon_loss: 125.2989 - accuracy: 0.8014 - kl_loss_metric: 9.1927
Epoch 19/30
937/937 [==============================] - 79s 84ms/step - loss: 134.1295 -
recon_loss: 124.9153 - accuracy: 0.8015 - kl_loss_metric: 9.2142
Epoch 20/30
937/937 [==============================] - 70s 74ms/step - loss: 133.9973 -
recon_loss: 124.7518 - accuracy: 0.8015 - kl_loss_metric: 9.2453
Epoch 21/30
937/937 [==============================] - 74s 79ms/step - loss: 133.6031 -
recon_loss: 124.3347 - accuracy: 0.8016 - kl_loss_metric: 9.2684
Epoch 22/30
937/937 [==============================] - 73s 78ms/step - loss: 133.4664 -
recon_loss: 124.1798 - accuracy: 0.8017 - kl_loss_metric: 9.2866
Epoch 23/30
937/937 [==============================] - 73s 78ms/step - loss: 133.1581 -
recon_loss: 123.8423 - accuracy: 0.8018 - kl_loss_metric: 9.3158
Epoch 24/30
937/937 [==============================] - 74s 79ms/step - loss: 132.9082 -
recon_loss: 123.5902 - accuracy: 0.8018 - kl_loss_metric: 9.3180
Epoch 25/30
937/937 [==============================] - 74s 79ms/step - loss: 132.8698 -
recon_loss: 123.5075 - accuracy: 0.8019 - kl_loss_metric: 9.3622
Epoch 26/30
937/937 [==============================] - 76s 81ms/step - loss: 132.4619 -
recon_loss: 123.0668 - accuracy: 0.8020 - kl_loss_metric: 9.3952
Epoch 27/30
937/937 [==============================] - 72s 77ms/step - loss: 132.3114 -
recon_loss: 122.9164 - accuracy: 0.8021 - kl_loss_metric: 9.3950
Epoch 28/30
937/937 [==============================] - 70s 75ms/step - loss: 132.2445 -
recon_loss: 122.8379 - accuracy: 0.8020 - kl_loss_metric: 9.4066
Epoch 29/30
937/937 [==============================] - 78s 83ms/step - loss: 132.1173 -
recon_loss: 122.7170 - accuracy: 0.8021 - kl_loss_metric: 9.4003
Epoch 30/30
937/937 [==============================] - 80s 85ms/step - loss: 131.8980 -
```

```
recon_loss: 122.4427 - accuracy: 0.8021 - kl_loss_metric: 9.4553
```

[26]:
```
<keras.callbacks.History at 0x7faad504f350>
```

[27]:
```python
mu, logvar = autoencoder3d.encoder(x_train / 255)
mu = mu.numpy()
mu.shape
```

[27]:
```
(60000, 3)
```

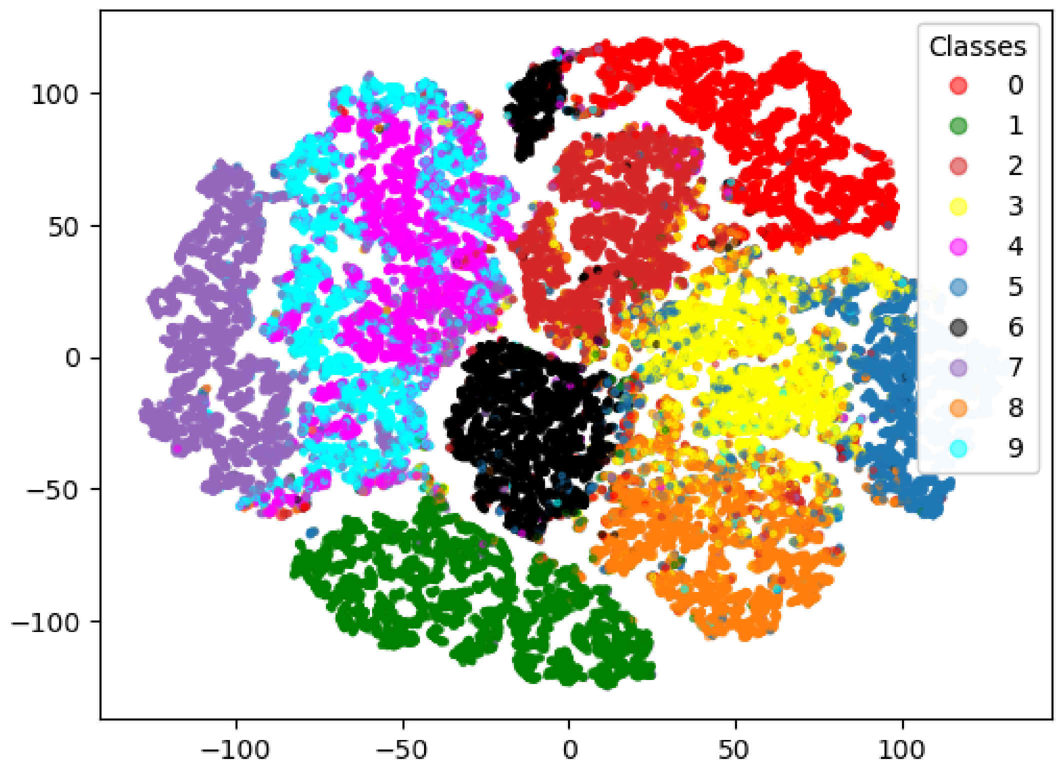
[28]:
```python
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, verbose=1, random_state=123)
z = tsne.fit_transform(mu)
```

```
[t-SNE] Computing 91 nearest neighbors…
[t-SNE] Indexed 60000 samples in 0.030s…
[t-SNE] Computed neighbors for 60000 samples in 1.001s…
[t-SNE] Computed conditional probabilities for sample 1000 / 60000
[t-SNE] Computed conditional probabilities for sample 2000 / 60000
[t-SNE] Computed conditional probabilities for sample 3000 / 60000
[t-SNE] Computed conditional probabilities for sample 4000 / 60000
[t-SNE] Computed conditional probabilities for sample 5000 / 60000
[t-SNE] Computed conditional probabilities for sample 6000 / 60000
[t-SNE] Computed conditional probabilities for sample 7000 / 60000
[t-SNE] Computed conditional probabilities for sample 8000 / 60000
[t-SNE] Computed conditional probabilities for sample 9000 / 60000
[t-SNE] Computed conditional probabilities for sample 10000 / 60000
[t-SNE] Computed conditional probabilities for sample 11000 / 60000
[t-SNE] Computed conditional probabilities for sample 12000 / 60000
[t-SNE] Computed conditional probabilities for sample 13000 / 60000
[t-SNE] Computed conditional probabilities for sample 14000 / 60000
[t-SNE] Computed conditional probabilities for sample 15000 / 60000
[t-SNE] Computed conditional probabilities for sample 16000 / 60000
[t-SNE] Computed conditional probabilities for sample 17000 / 60000
[t-SNE] Computed conditional probabilities for sample 18000 / 60000
[t-SNE] Computed conditional probabilities for sample 19000 / 60000
[t-SNE] Computed conditional probabilities for sample 20000 / 60000
[t-SNE] Computed conditional probabilities for sample 21000 / 60000
[t-SNE] Computed conditional probabilities for sample 22000 / 60000
[t-SNE] Computed conditional probabilities for sample 23000 / 60000
[t-SNE] Computed conditional probabilities for sample 24000 / 60000
[t-SNE] Computed conditional probabilities for sample 25000 / 60000
[t-SNE] Computed conditional probabilities for sample 26000 / 60000
[t-SNE] Computed conditional probabilities for sample 27000 / 60000
[t-SNE] Computed conditional probabilities for sample 28000 / 60000
[t-SNE] Computed conditional probabilities for sample 29000 / 60000
[t-SNE] Computed conditional probabilities for sample 30000 / 60000
```

```
[t-SNE] Computed conditional probabilities for sample 31000 / 60000
[t-SNE] Computed conditional probabilities for sample 32000 / 60000
[t-SNE] Computed conditional probabilities for sample 33000 / 60000
[t-SNE] Computed conditional probabilities for sample 34000 / 60000
[t-SNE] Computed conditional probabilities for sample 35000 / 60000
[t-SNE] Computed conditional probabilities for sample 36000 / 60000
[t-SNE] Computed conditional probabilities for sample 37000 / 60000
[t-SNE] Computed conditional probabilities for sample 38000 / 60000
[t-SNE] Computed conditional probabilities for sample 39000 / 60000
[t-SNE] Computed conditional probabilities for sample 40000 / 60000
[t-SNE] Computed conditional probabilities for sample 41000 / 60000
[t-SNE] Computed conditional probabilities for sample 42000 / 60000
[t-SNE] Computed conditional probabilities for sample 43000 / 60000
[t-SNE] Computed conditional probabilities for sample 44000 / 60000
[t-SNE] Computed conditional probabilities for sample 45000 / 60000
[t-SNE] Computed conditional probabilities for sample 46000 / 60000
[t-SNE] Computed conditional probabilities for sample 47000 / 60000
[t-SNE] Computed conditional probabilities for sample 48000 / 60000
[t-SNE] Computed conditional probabilities for sample 49000 / 60000
[t-SNE] Computed conditional probabilities for sample 50000 / 60000
[t-SNE] Computed conditional probabilities for sample 51000 / 60000
[t-SNE] Computed conditional probabilities for sample 52000 / 60000
[t-SNE] Computed conditional probabilities for sample 53000 / 60000
[t-SNE] Computed conditional probabilities for sample 54000 / 60000
[t-SNE] Computed conditional probabilities for sample 55000 / 60000
[t-SNE] Computed conditional probabilities for sample 56000 / 60000
[t-SNE] Computed conditional probabilities for sample 57000 / 60000
[t-SNE] Computed conditional probabilities for sample 58000 / 60000
[t-SNE] Computed conditional probabilities for sample 59000 / 60000
[t-SNE] Computed conditional probabilities for sample 60000 / 60000
[t-SNE] Mean sigma: 0.062719
[t-SNE] KL divergence after 250 iterations with early exaggeration: 86.847862
[t-SNE] KL divergence after 1000 iterations: 1.668219
```

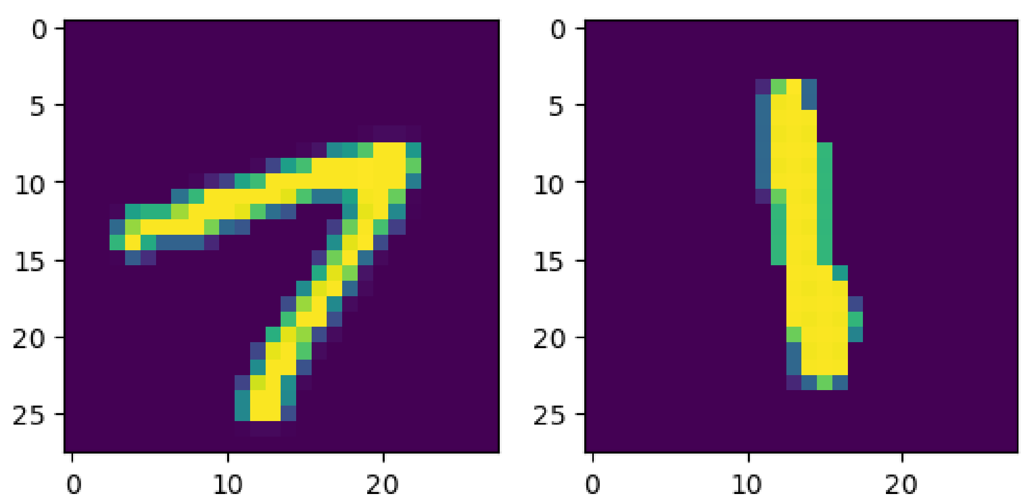
[29]:

```
fig, ax = make_scatter_plot(z, y_train, s=5, alpha=.5)
plt.show()
```



[30]:
```
fig, axs = plt.subplots(1, 2)
axs[0].imshow(x_train[101])
axs[1].imshow(x_train[200])
```

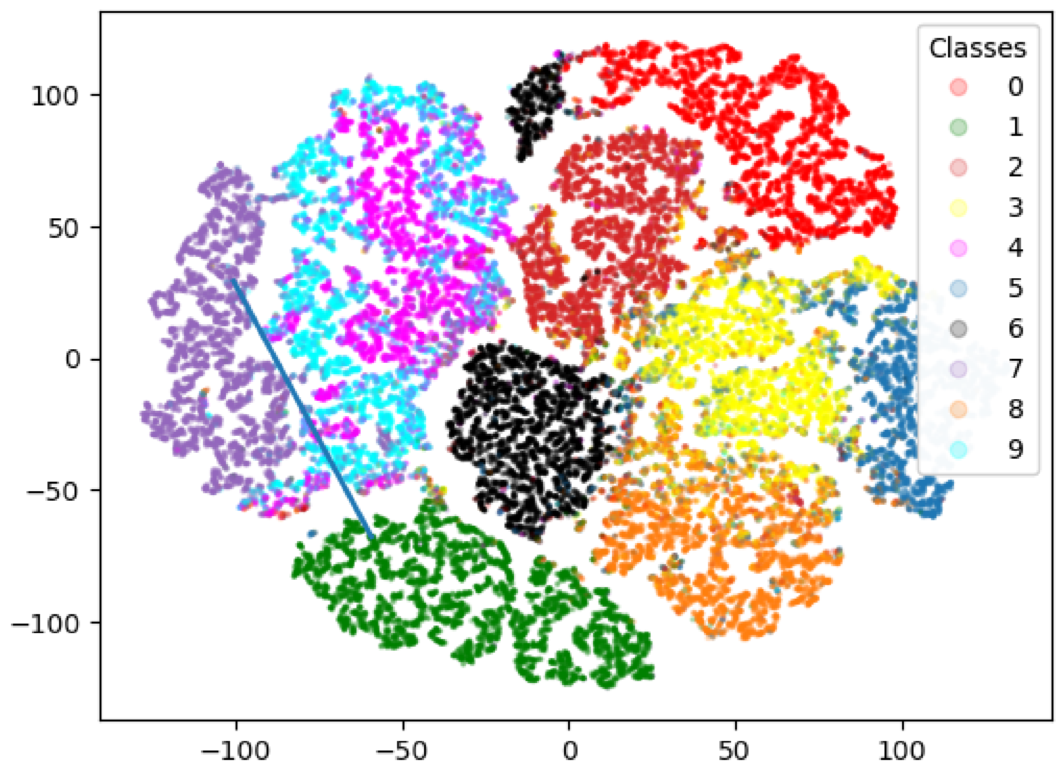
[30]: <matplotlib.image.AxesImage at 0x7faad4e1bc90>



[31]:
```
mu7, _ = autoencoder3d.encoder(x_train[101:102] / 255)
mu1, _ = autoencoder3d.encoder(x_train[200:201] / 255)
mu1, mu7
```

[31]:
```
(<tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[-1.4310837 ,
0.77846795, -0.5304799 ]], dtype=float32)>,
 <tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[-0.13459326,  0.977983
,  2.117529  ]], dtype=float32)>)
```

[32]:
```
fig, ax = make_scatter_plot(z, y_train, s=2, alpha=.2)
ax.plot(*np.stack((z[101], z[200])).T, lw=2)
plt.show()
```

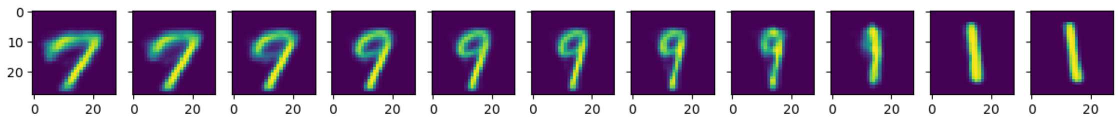


[33]:
```
vec71 = mu1 - mu7
length = np.linalg.norm(vec71)
vec71, length
```

[33]:
```
(<tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[-1.2964904 ,
-0.19951504, -2.6480088 ]], dtype=float32)>,
 2.9551048)
```

[34]:

```
steps = 10
fig, axs = plt.subplots(1, steps + 1, figsize=(15, 6), sharey=True)
for ax, step in zip(axs, range(steps + 1)):
    ax.imshow(autoencoder3d.decode(mu7 + vec71 * (step / steps))[0, :, :, 0])
```



[ ]: