VAE_dist_MNIST_StaticPlot_TrainTest

March 26, 2021

1 MLP Variational Autoencoder for MNIST dataset

2 +

3 Static plots for training and latent interpolation

implement probability and sampling with torch.distribution package, modified from this link

[1]:
```
%matplotlib inline
import os, sys
import numpy as np
import pickle
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torch.distributions as torchD

import torch, seaborn as sns
import pandas as pd

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap

from utils import *
```

[2]:
```
# # set seed for reproducibility
# seed = 24
# torch.manual_seed(seed)
# np.random.seed(seed)
```

3.1 Load Dataset

[3]:
```
# !wget www.di.ens.fr/~lelarge/MNIST.tar.gz
# !tar -zxvf MNIST.tar.gz
```

[4]:
```
batch_size = 128

transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor()])

train_dataset = torchvision.datasets.MNIST(
    root="../data/", train=True, transform=transform, download=True)
train_loader = torch.utils.data.DataLoader(train_dataset,␣
↪batch_size=batch_size, shuffle=True)

test_dataset = torchvision.datasets.MNIST(
    root="../data/", train=False, transform=transform, download=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,␣
↪shuffle=True)
```

3.2 Variational Autoencoder

A MLP variational autoencoder is used. The mean and s.d. of the approximate posterior are outputs of the encoding MLP. See more details at VAE paper

[5]:
```
class MLP_V_Encoder(nn.Module):
    def __init__(self, **kwargs):
        super(MLP_V_Encoder, self).__init__()
        self.model = nn.Sequential(
            nn.Flatten(),
            nn.Linear(in_features=np.prod(kwargs["input_shape"]),␣
↪out_features=kwargs["enc_dim"]),
            nn.ReLU(),
#             nn.Linear(in_features=400, out_features=kwargs["enc_dim"]),
#             nn.ReLU(),
        )


    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        enc_out = self.model(x)
        return enc_out
```

[6]:
```
class MLP_V_Decoder(nn.Module):
    def __init__(self, **kwargs):
        super(MLP_V_Decoder, self).__init__()
        self.model = nn.Sequential(
```

```
            nn.Linear(in_features=kwargs["latent_dim"],␣
↪out_features=kwargs["enc_dim"]),
            nn.ReLU(),
            nn.Linear(in_features=kwargs["enc_dim"], out_features=np.
↪prod(kwargs["input_shape"])),
            nn.Sigmoid() # push the pixels in range (0,1)
        )
        self.output_shape = kwargs["input_shape"]

    def forward(self, latent):
        x_bar = self.model(latent)
        x_bar = x_bar.view([-1]+ self.output_shape)
        return x_bar
```

[7]:
```
class MLP_VAE(nn.Module):
    """
    TODO: check whether to use sum or mean for the probability part
    """
    def __init__(self, **kwargs):
        # kwargs["input_shape"] = [1,28,28]
        # kwargs["latent_dim"] = 4
        super(MLP_VAE, self).__init__()
        self.encoder = MLP_V_Encoder(**kwargs)
        self.decoder = MLP_V_Decoder(**kwargs)

        # distribution layers
        self.enc_dim = kwargs["enc_dim"]
        self.latent_dim = kwargs["latent_dim"]
        self.enc_to_mean = nn.Linear(self.enc_dim, self.latent_dim)
        self.enc_to_logvar = nn.Linear(self.enc_dim, self.latent_dim)

    def encode(self, x):
        enc_out = self.encoder(x)
        mean = self.enc_to_mean(enc_out)
        logvar = self.enc_to_logvar(enc_out)
        return mean, logvar

    def decode(self, latent):
        return self.decoder(latent)

    def pxz_likelihood(self, x, x_bar, scale=1., dist_type="Gaussian"):
        """
        compute the likelihood p(x|z) based on predefined distribution, given a␣
↪latent vector z
        default scale = 1, can be broadcasted to the shape of x_bar
        """
        if dist_type == "Gaussian":
```

```
            dist = torch.distributions.Normal(loc=x_bar, scale=scale)
        else:
            raise NotImplementedError("unknown distribution for p(x|z) {}".
↪format(dist_type))

        log_pxz = dist.log_prob(x)
        return log_pxz.sum() # log_pxz.sum((1,2,3))

    def kl_divergence(self, mean, logvar):
        """
        Monte Carlo way to solve KL divergence
        """
        pz = torchD.Normal(torch.zeros_like(mean), scale=1)
        std = torch.exp(0.5*logvar)
        qzx = torchD.Normal(loc=mean, scale=std)

        z = qzx.rsample() # reparameterized sampling, shape [32,2]

        # clamp the log prob to avoid -inf
        qzx_lp = qzx.log_prob(z).clamp(min=-1e10, max=0.)
        pz_lp = pz.log_prob(z).clamp(min=-1e10, max=0.)

        kl = qzx_lp - pz_lp
        if torch.isnan(qzx_lp).any():
            print("nan in qzx_lp")
            print("qzx_lp")
            print(qzx_lp)
            print("z")
            print(z)
            print("mean")
            print(mean)
            print("logvar")
            print(logvar)
            plot_p_q(mean, logvar)
            raise ValueError
        if torch.isnan(pz_lp).any():
            print("nan in pz_lp")
            print("pz_lp")
            print(pz_lp)
            print("z")
            print(z)
            print("mean")
            print(mean)
            print("logvar")
            print(logvar)
            plot_p_q(mean, logvar)
            raise ValueError
```

```
        if torch.isnan(kl.mean()).any():
            print(qzx_lp)
            print(pz_lp)
            print(z)

        return kl.sum()

    def reparameterize(self, mean, logvar):
        # assume Gaussian for p(epsilon)
        sd = torch.exp(0.5*logvar)
        # use randn_like to sample N(0,1) of the same size as std/mean
        # default only sample once, otherwise should try sample multiple times␣
↪take mean
        eps = torch.randn_like(sd)
        return mean + sd * eps

    def sample_latent_embedding(self, mean, logvar, method="reparameterize"):
        """
        Write a sampling function to make function name consistent
        """
        if method=="reparameterize":
            return self.reparameterize(mean, logvar)
        else:
            raise NotImplementedError("Unrecognized method for sampling latent␣
↪embedding {}".format(method))

    def forward(self, x, if_plot_pq=False):
        latent_mean, latent_logvar = self.encode(x)
        latent = self.reparameterize(latent_mean, latent_logvar)
        x_bar = self.decoder(latent)

        if if_plot_pq:
            plot_p_q(latent_mean, latent_logvar)

        return latent, x_bar, latent_mean, latent_logvar
```

3.3 Training process

[8]:
```
def train(model, device, train_loader, num_epochs=5, learning_rate=1e-3,␣
↪use_scheduler=False, w_kl=1e-4):
    recon_loss_fn = nn.BCELoss(reduction="sum")

    optimizer = optim.Adam(model.parameters(),
                           lr=learning_rate,
```

```
                        # weight_decay=5e-4,
                        )
    if use_scheduler:
        step_size = 5
        gamma = 0.1
        scheduler = optim.lr_scheduler.StepLR(optimizer,step_size=step_size,␣
↪gamma=gamma)

    epoch_train_loss = []
    epoch_train_kl_loss = []
    epoch_train_recon_loss = []
    epoch_sample_img = []
    epoch_sample_reconstruction = []
    epoch_sample_latent = []
    epoch_train_kl_divergence = []
    epoch_train_pxz_likelihood = []
    epoch_train_elbo = []

    model.train()
    for epoch in range(num_epochs):

        model.train()

        train_loss = 0
        train_kl_loss = 0
        train_recon_loss = 0
        train_kl_divergence = 0
        train_pxz_likelihood = 0
        train_elbo = 0


        for img, label in train_loader:
            optimizer.zero_grad()

            img, label = img.to(device), label.to(device)

            latent, reconstruction, latent_mean, latent_logvar = model(img)
            kl_loss = -0.5 * torch.sum(1 + latent_logvar - latent_mean.pow(2) -␣
↪latent_logvar.exp())
            kl_divergence = model.kl_divergence(latent_mean, latent_logvar)

            recon_loss = recon_loss_fn(reconstruction, img)
            pxz_likelihood = model.pxz_likelihood(img, reconstruction)
            elbo = pxz_likelihood - kl_divergence # should be maximized

            loss = w_kl*kl_loss + recon_loss
```

```
            loss.backward()
            train_loss += loss.item()
            train_kl_loss += kl_loss.item()
            train_recon_loss += recon_loss.item()
            train_kl_divergence += kl_divergence.item()
            train_pxz_likelihood += pxz_likelihood.item()
            train_elbo += elbo.item()

            optimizer.step()


        if use_scheduler:
            scheduler.step()

        train_loss = train_loss/len(train_loader)
        train_kl_loss = train_kl_loss/len(train_loader)
        train_recon_loss = train_recon_loss/len(train_loader)
        train_kl_divergence = train_kl_divergence/len(train_loader)
        train_pxz_likelihood = train_pxz_likelihood/len(train_loader)
        train_elbo = train_elbo/len(train_loader)


        if (epoch<5) or (epoch%5 == 0):
            print("Epoch {}, Loss {:.4f}, kl_loss {:.4f}, recon_loss {:.4f},␣
↪kl_divergence {:.4f}".format(epoch+1, float(train_loss),␣
↪float(train_kl_loss), float(train_recon_loss), float(train_kl_divergence)))
            plot_latent(label, latent, dtype="tensor", suptitle_app="_train")
            plot_p_q(latent_mean, latent_logvar, suptitle_app="_train")

            if use_scheduler:
                print("current learning rate {}".format(scheduler.
↪get_last_lr()))

            # test dataset, plot latent for one batch
            model.eval()
            for img, label in test_loader:
                img, label = img.to(device), label.to(device)
                latent, reconstruction, latent_mean, latent_logvar = model(img)
                plot_latent(label, latent, dtype="tensor", suptitle_app="_test")
                plot_p_q(latent_mean, latent_logvar, suptitle_app="_test")
                break


        epoch_train_loss.append(train_loss)
        epoch_train_kl_loss.append(train_kl_loss)
        epoch_train_recon_loss.append(train_recon_loss)
```

```
        epoch_sample_img.append(img.cpu().detach().numpy())
        epoch_sample_reconstruction.append(reconstruction.cpu().detach().
↪numpy())
        epoch_sample_latent.append(latent.cpu().detach().numpy())

        epoch_train_kl_divergence.append(train_kl_divergence)
        epoch_train_pxz_likelihood.append(train_pxz_likelihood)
        epoch_train_elbo.append(train_elbo)

        results_dict = {
            "train_loss": np.array(epoch_train_loss),
            "train_kl_loss": np.array(epoch_train_kl_loss),
            "train_recon_loss": np.array(epoch_train_recon_loss),
            "sample_img": np.array(epoch_sample_img),
            "sample_reconstruction": np.array(epoch_sample_reconstruction),
            "sample_latent": np.array(epoch_sample_latent),
            "train_kl_divergence": np.array(epoch_train_kl_divergence),
            "train_pxz_likelihood": np.array(epoch_train_pxz_likelihood),
            "train_elbo": np.array(epoch_train_elbo)
        }

    return model, results_dict
```

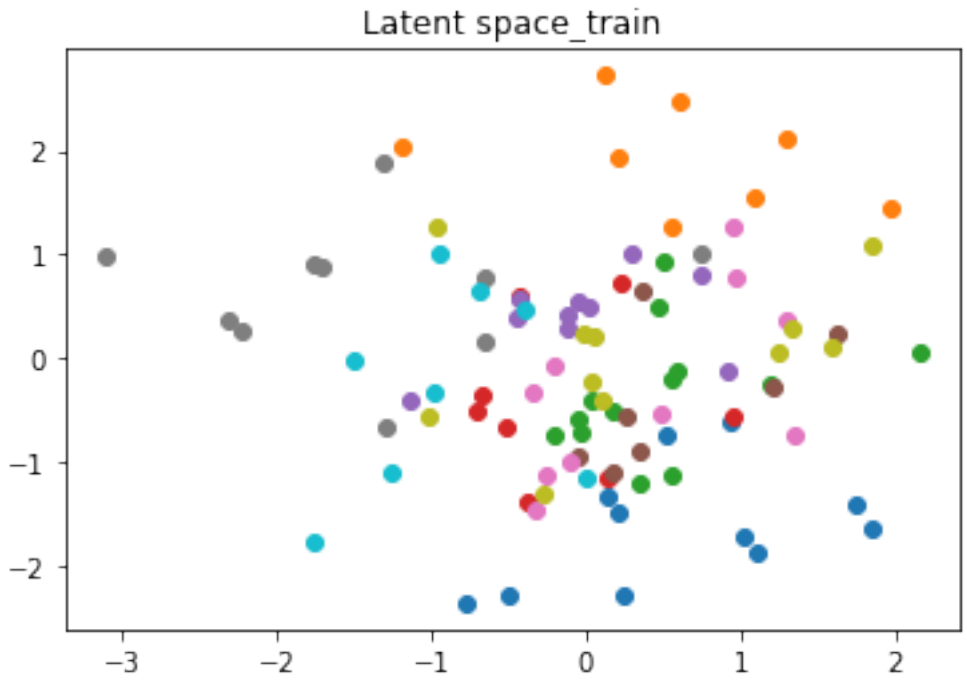
[9]:
```
# use gpu if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
input_shape = [1, 28, 28]
enc_dim = 400
latent_dim = 2
num_epochs = 50
learning_rate = 1e-3
logDir = "models_and_stats/"
w_kl = 0
model_name = "MLP_VAE_dist_l2_wkl_{}".format(w_kl)
model_path = logDir + model_name + ".pt"
dict_name = model_name + '.pkl'
```

[10]:
```
pretrain = False
```

[11]:
```
%%time
if pretrain:
    # load the pretrained model
    model = MLP_VAE(input_shape=input_shape, enc_dim=enc_dim,␣
↪latent_dim=latent_dim)
    model.load_state_dict(torch.load(model_path))
    model.to(device)
    results_dict = pickle.load(open(logDir + dict_name, 'rb'))
```

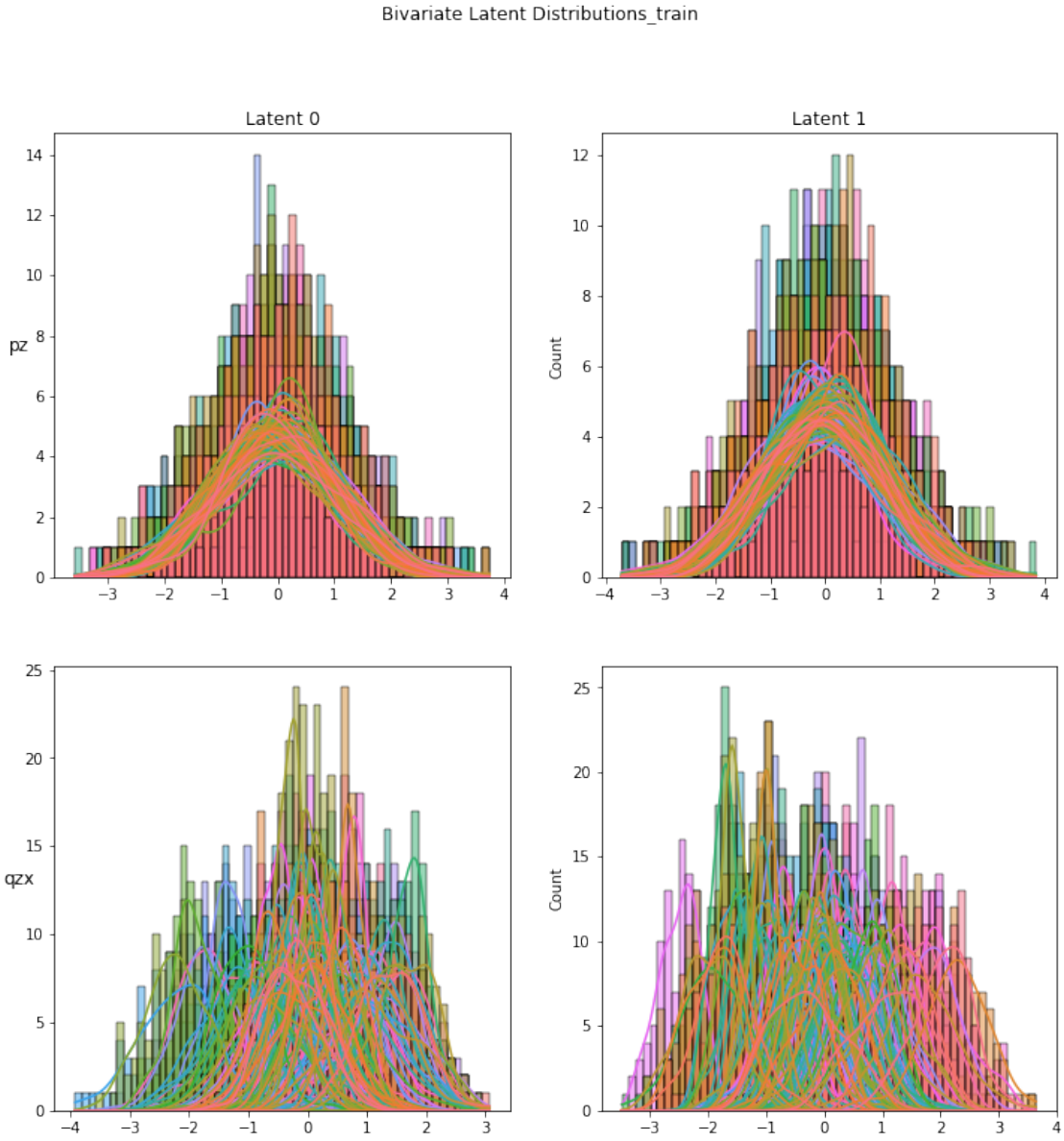
```
else:
    # train and save the model
    for w_kl in [10]:
        model_name = "MLP_VAE_dist_l2_wkl_{}".format(w_kl)
        model_path = logDir + model_name + ".pt"
        dict_name = model_name + '.pkl'
        model = MLP_VAE(input_shape=input_shape, enc_dim=enc_dim,␣
↪latent_dim=latent_dim).to(device)
        model, results_dict = train(model, device, train_loader,␣
↪num_epochs=num_epochs, learning_rate=learning_rate, use_scheduler=False,␣
↪w_kl=w_kl)
        torch.save(model.state_dict(), model_path)
        pickle.dump(results_dict, open(logDir + dict_name, 'wb'))
        print("dump results dict to {}".format(dict_name))

model.eval()
```

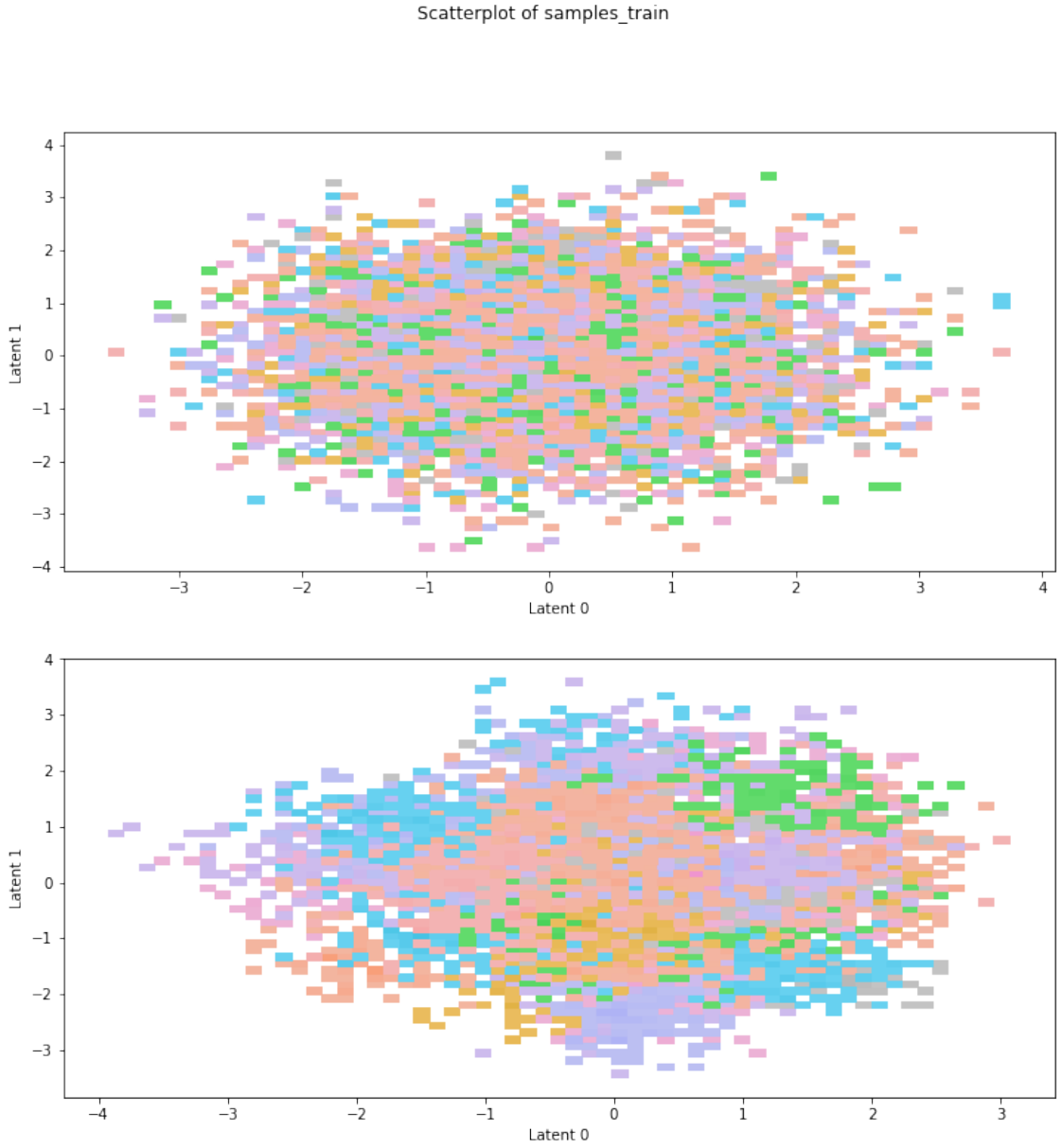
```
Epoch 1, Loss 26516.7278, kl_loss 166.6709, recon_loss 24850.0187, kl_divergence
165.1188
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```



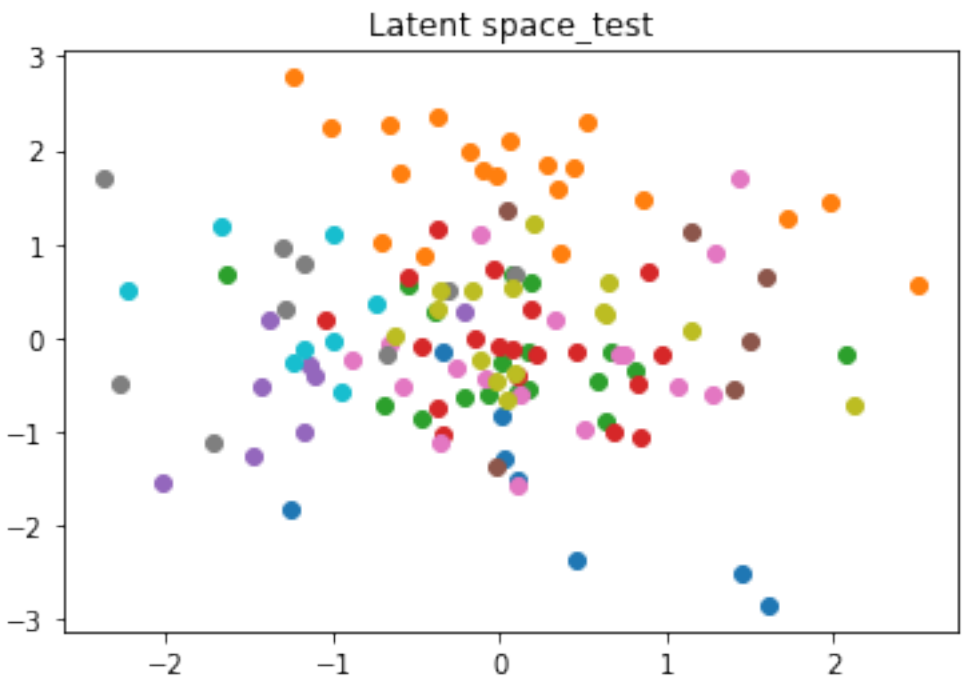

```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
```

check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)

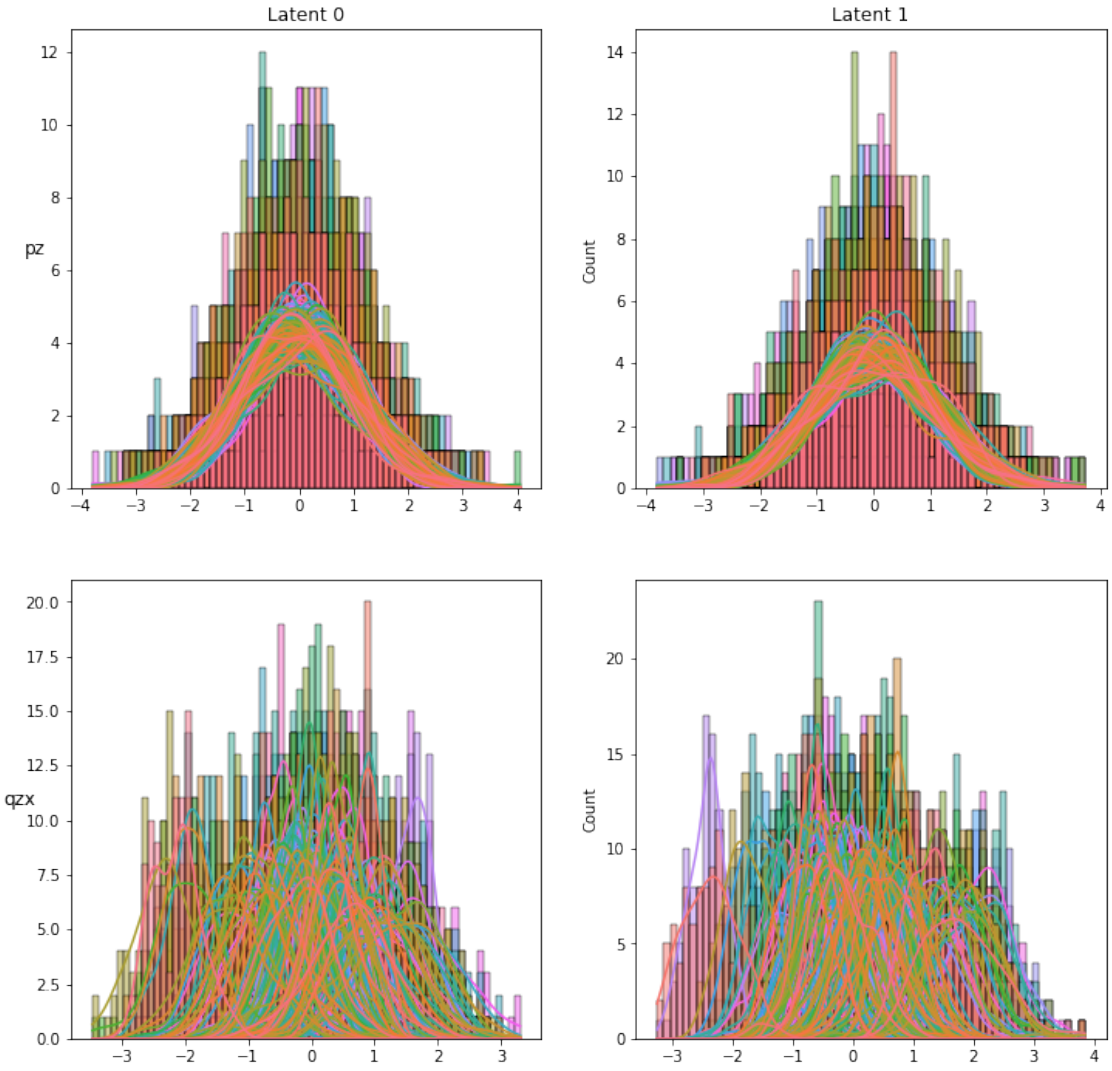


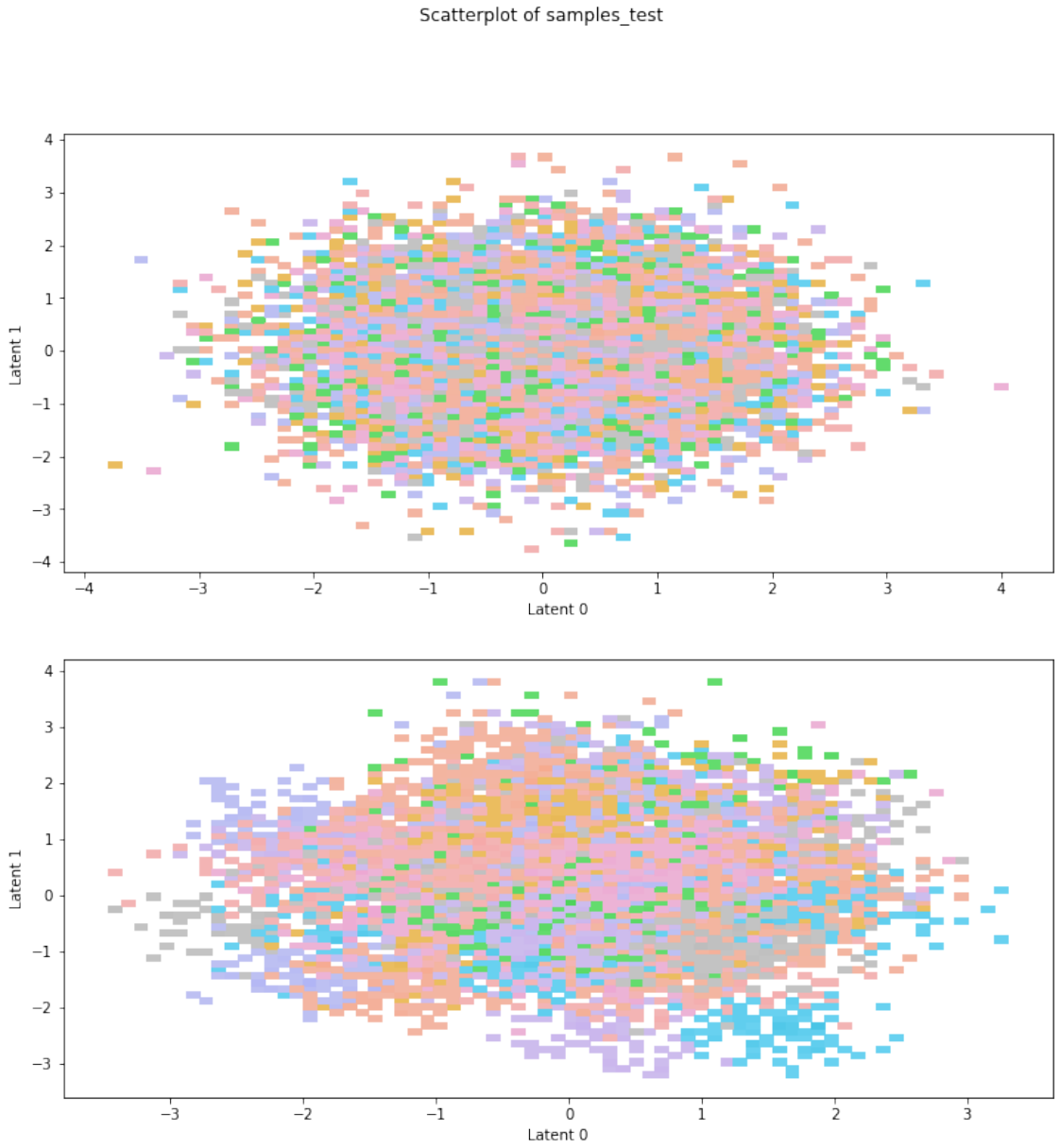



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```

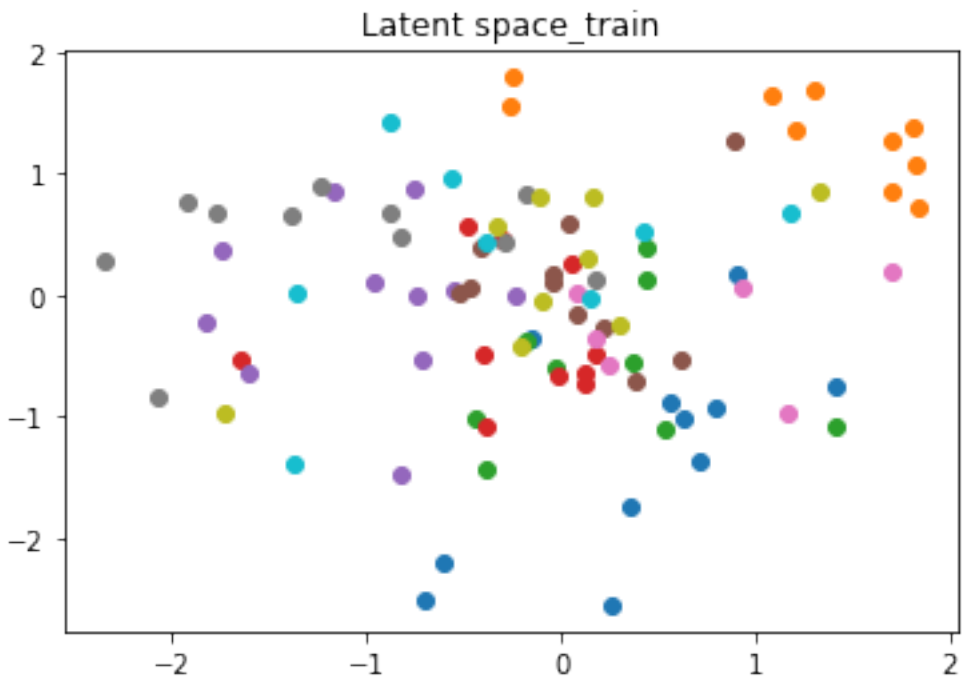



```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

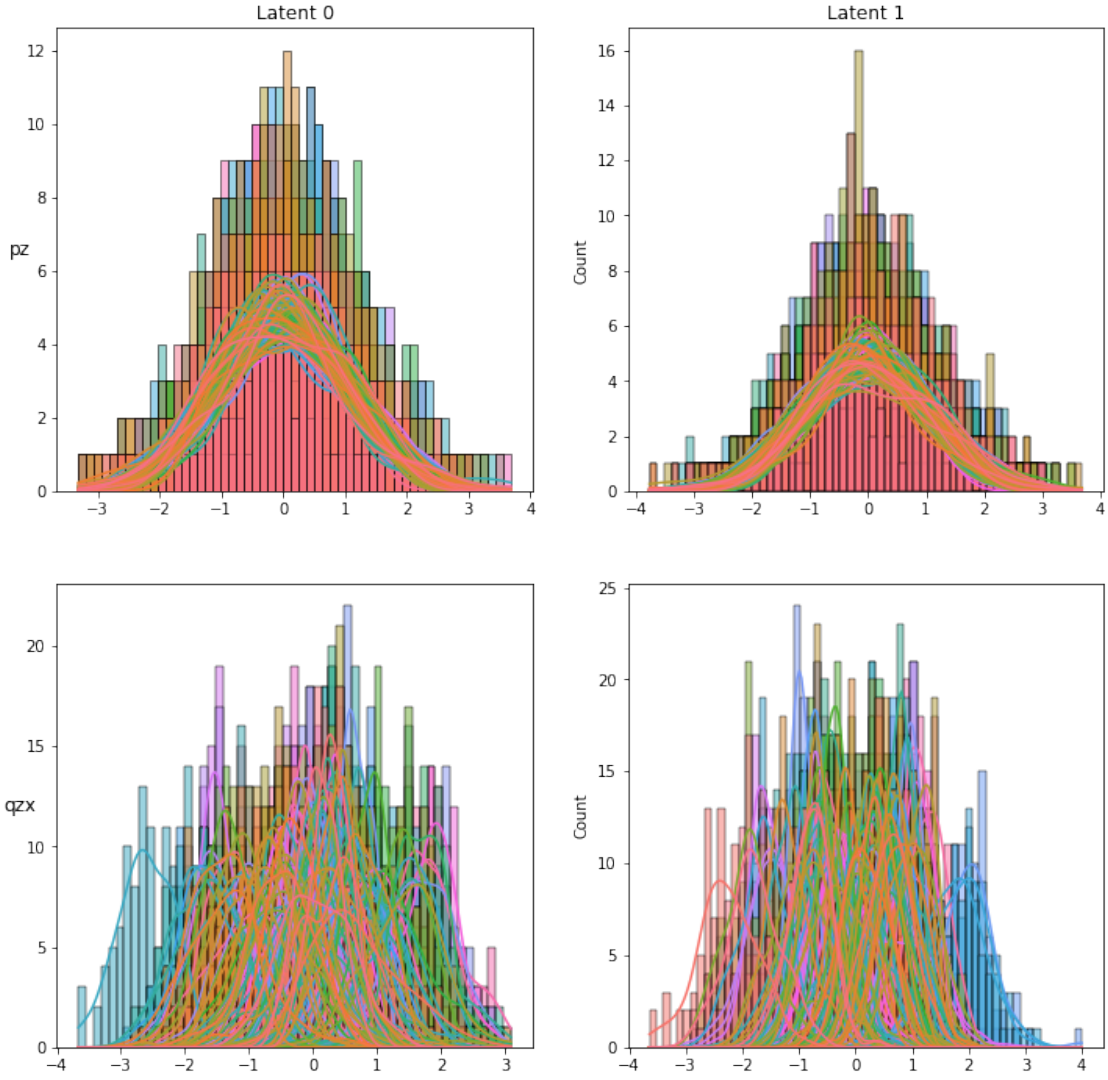

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count

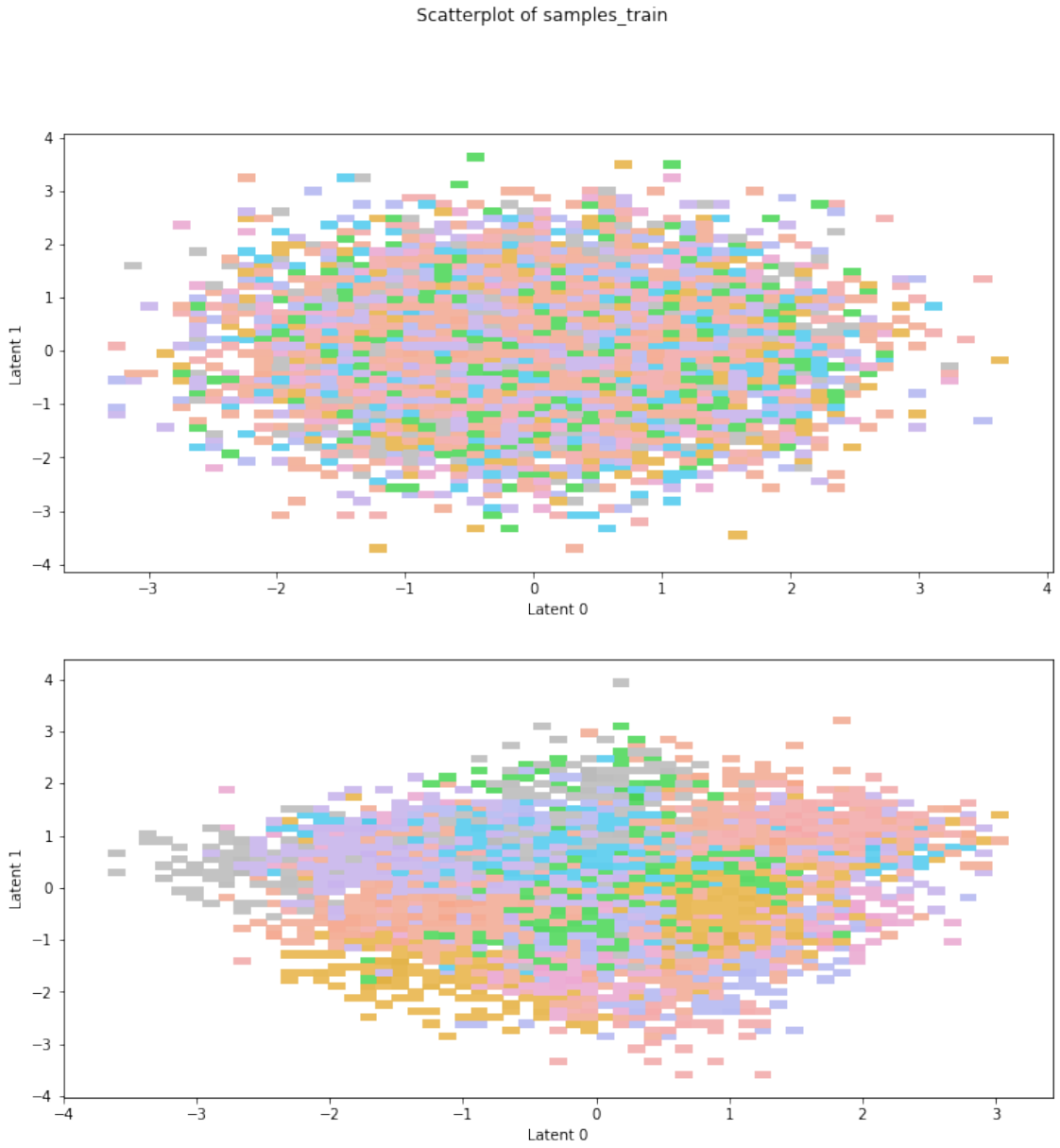



```
Epoch 2, Loss 24556.3994, kl_loss 223.1297, recon_loss 22325.1020, kl_divergence
217.7933
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

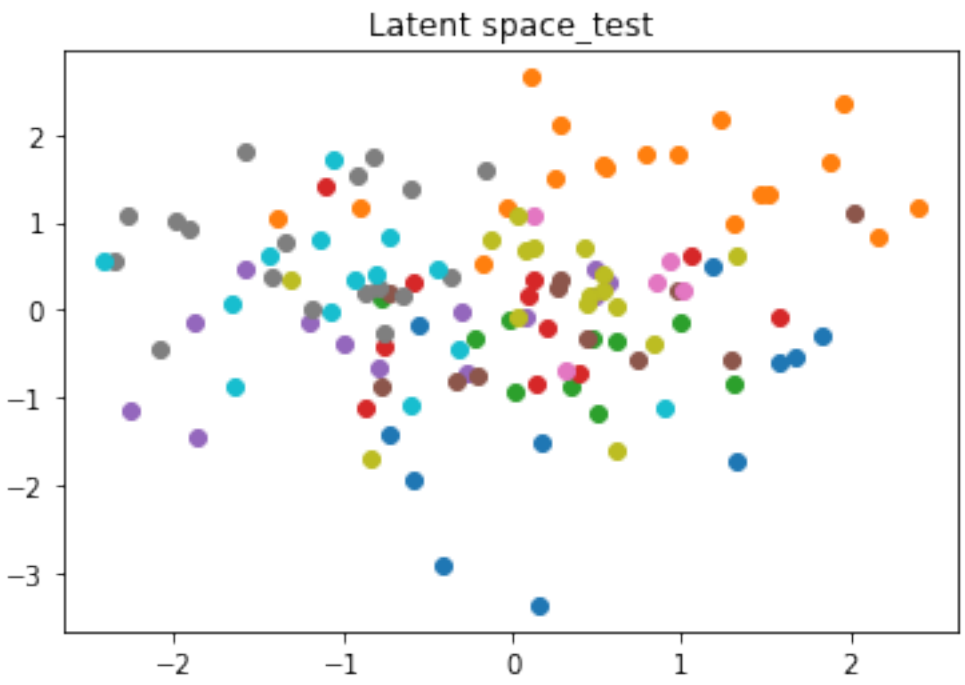



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

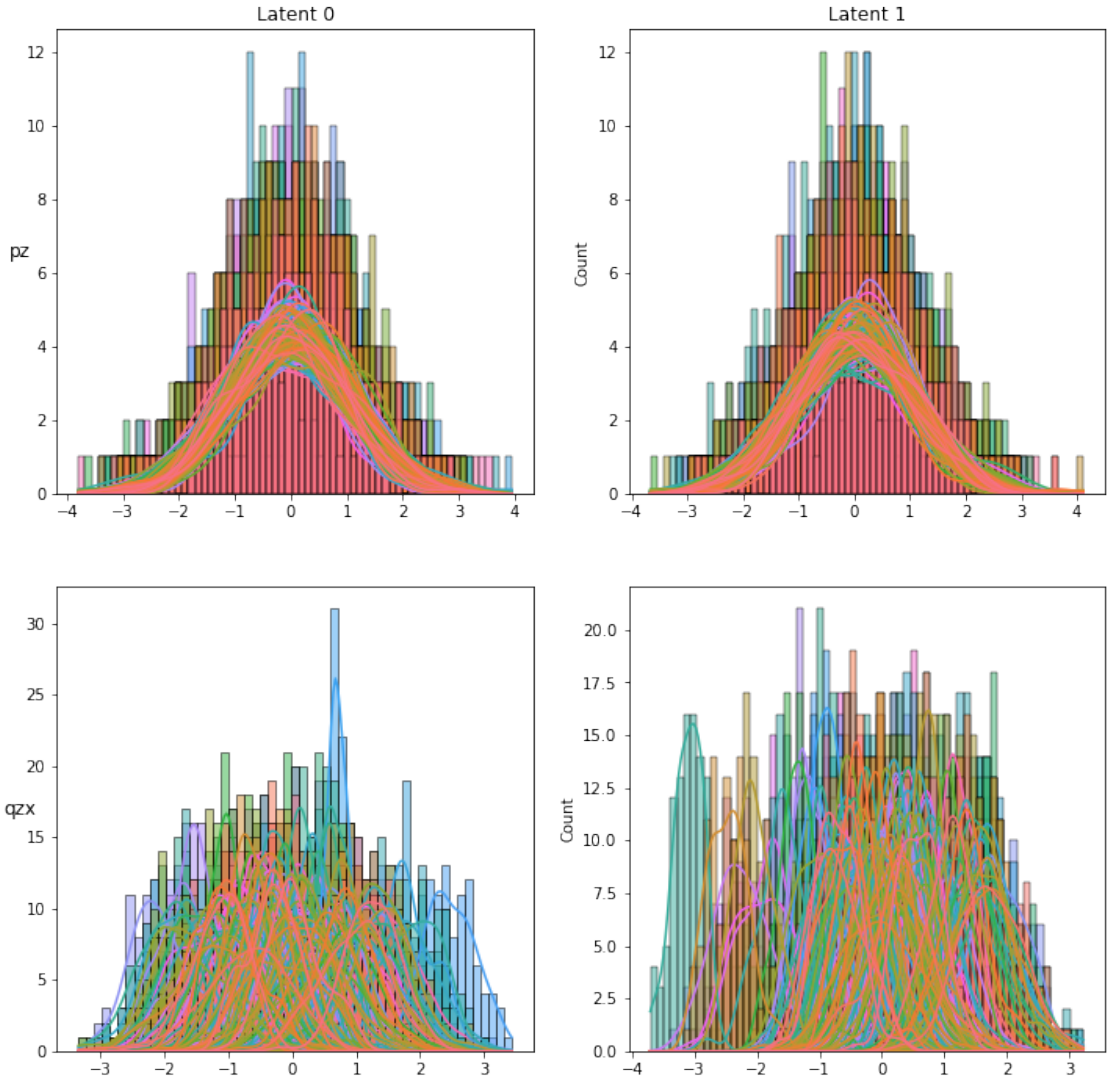

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
qzx
Count
Count

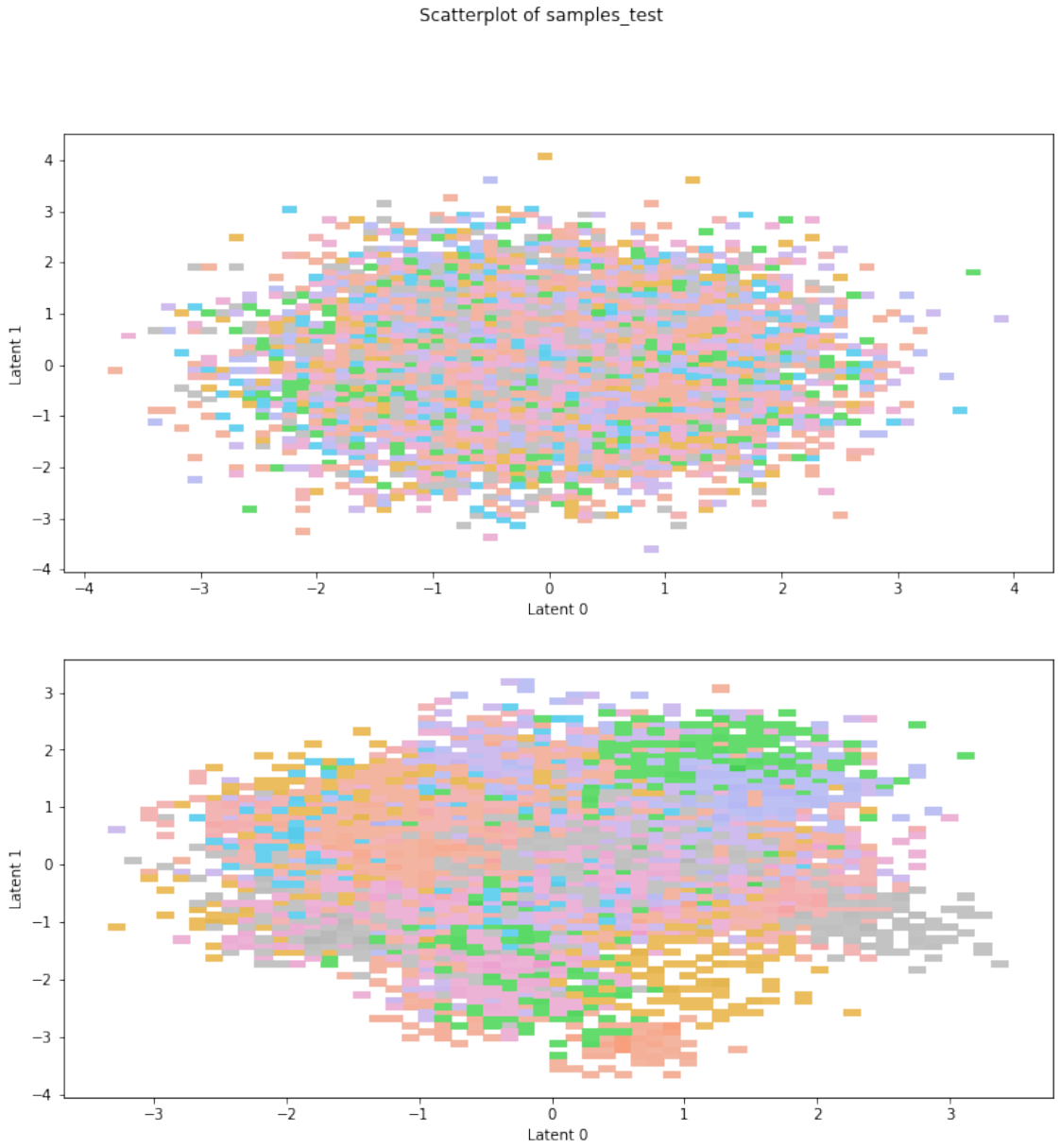



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```

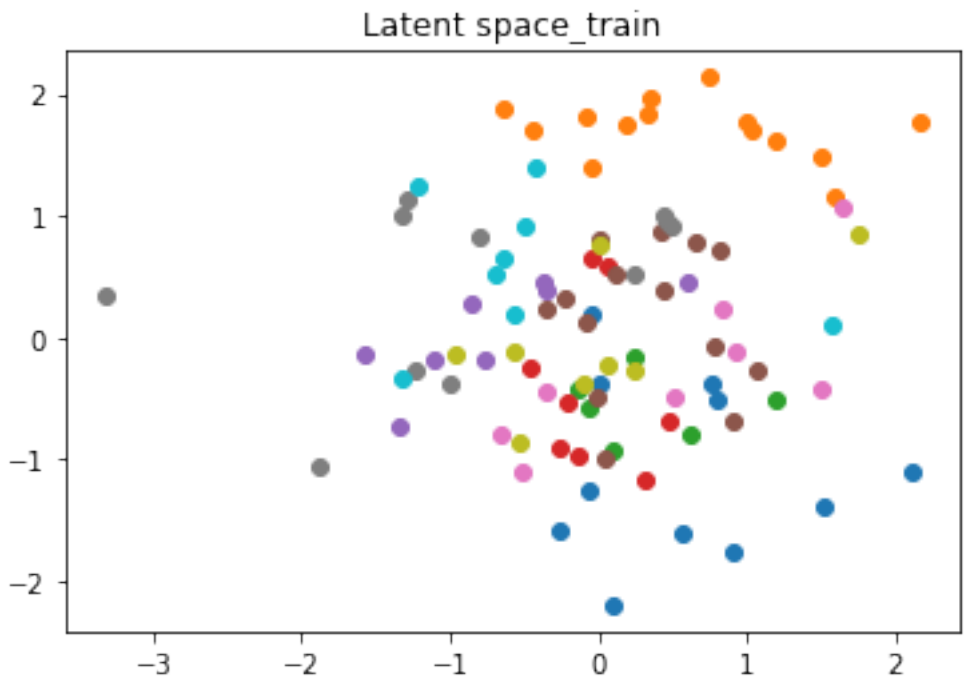



```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

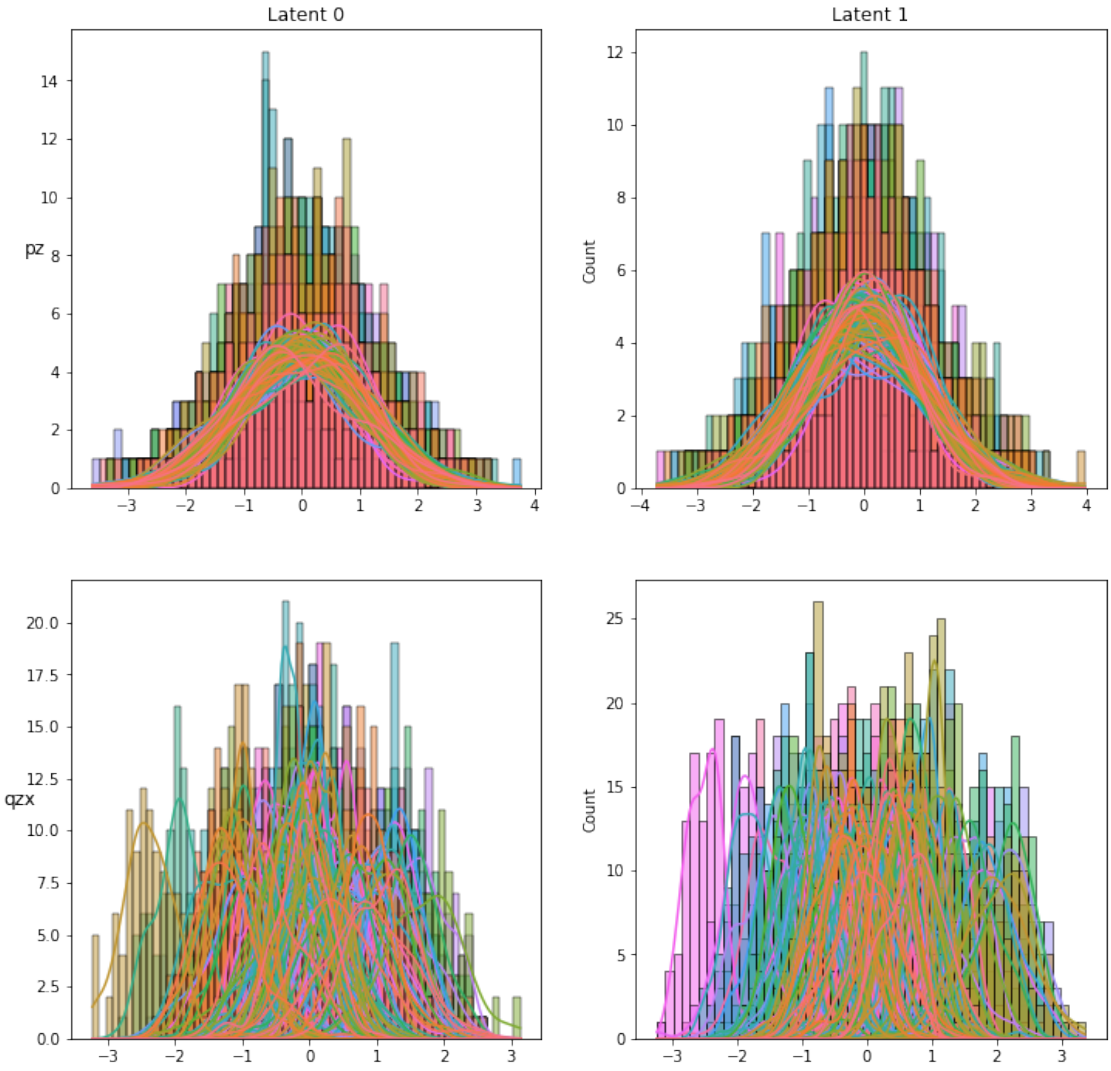

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count

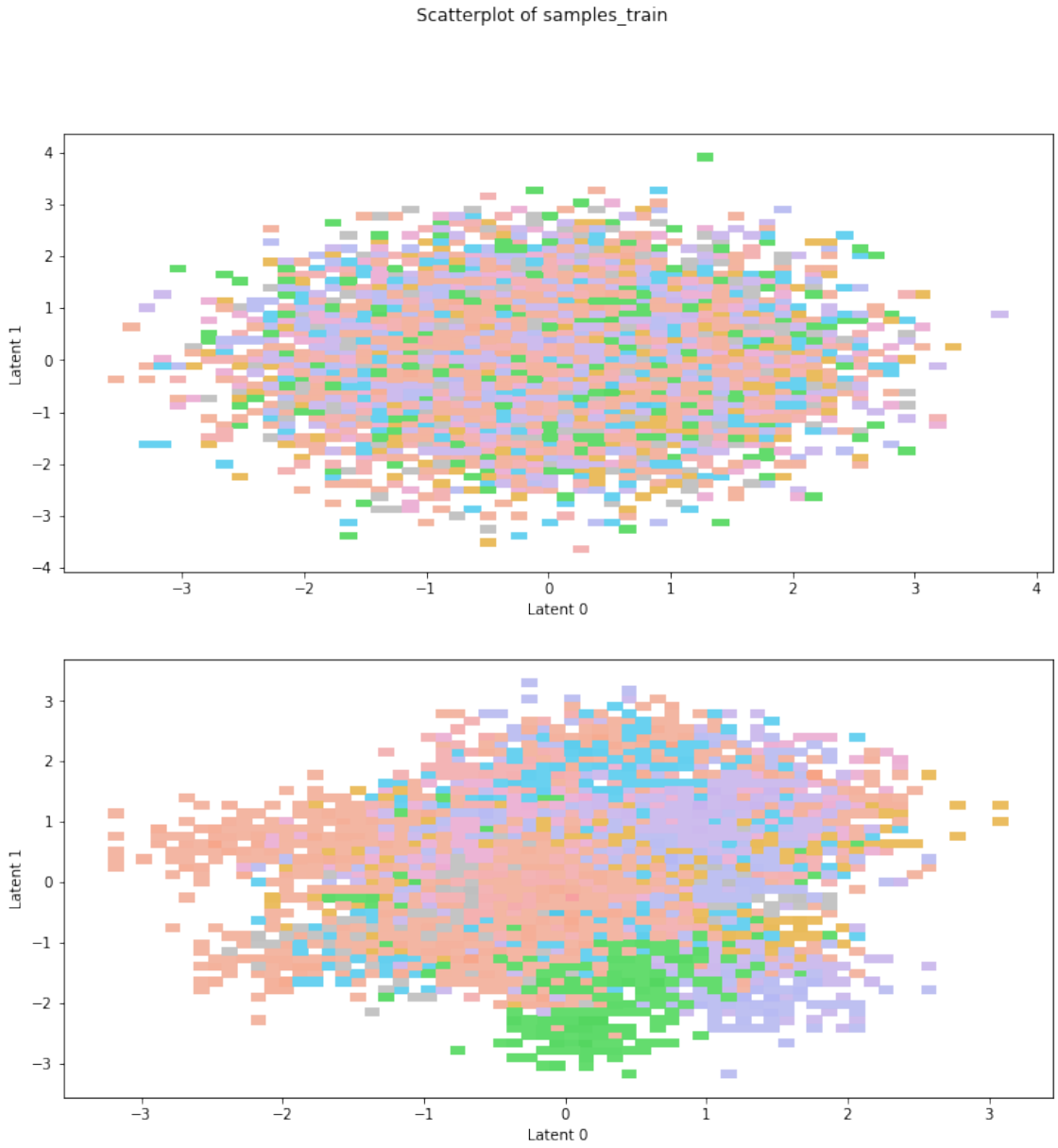



```
Epoch 3, Loss 24415.7664, kl_loss 242.3862, recon_loss 21991.9046, kl_divergence
233.7920
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

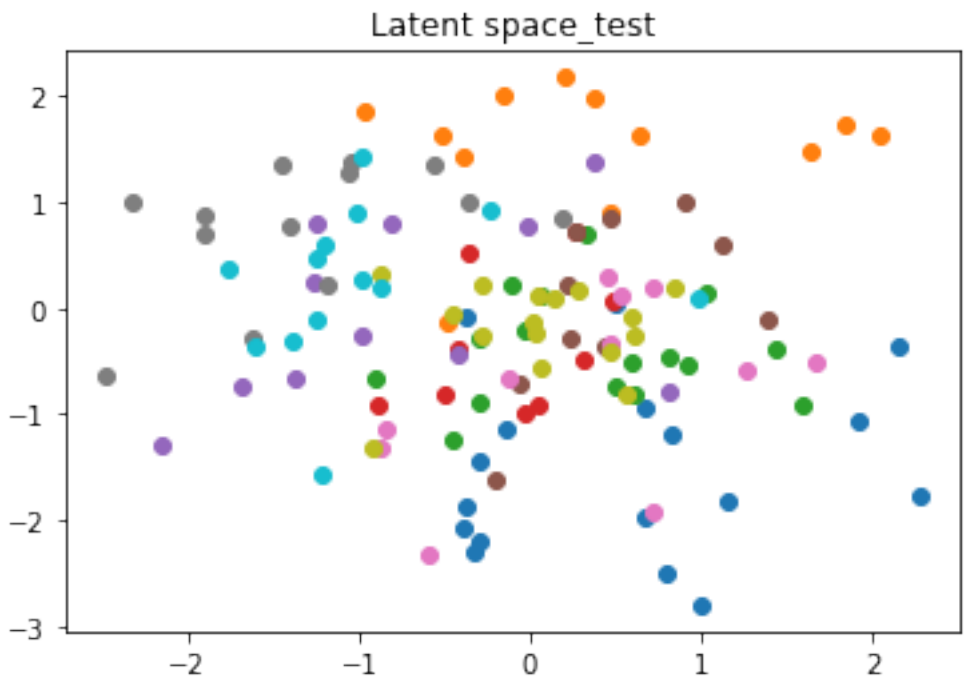



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

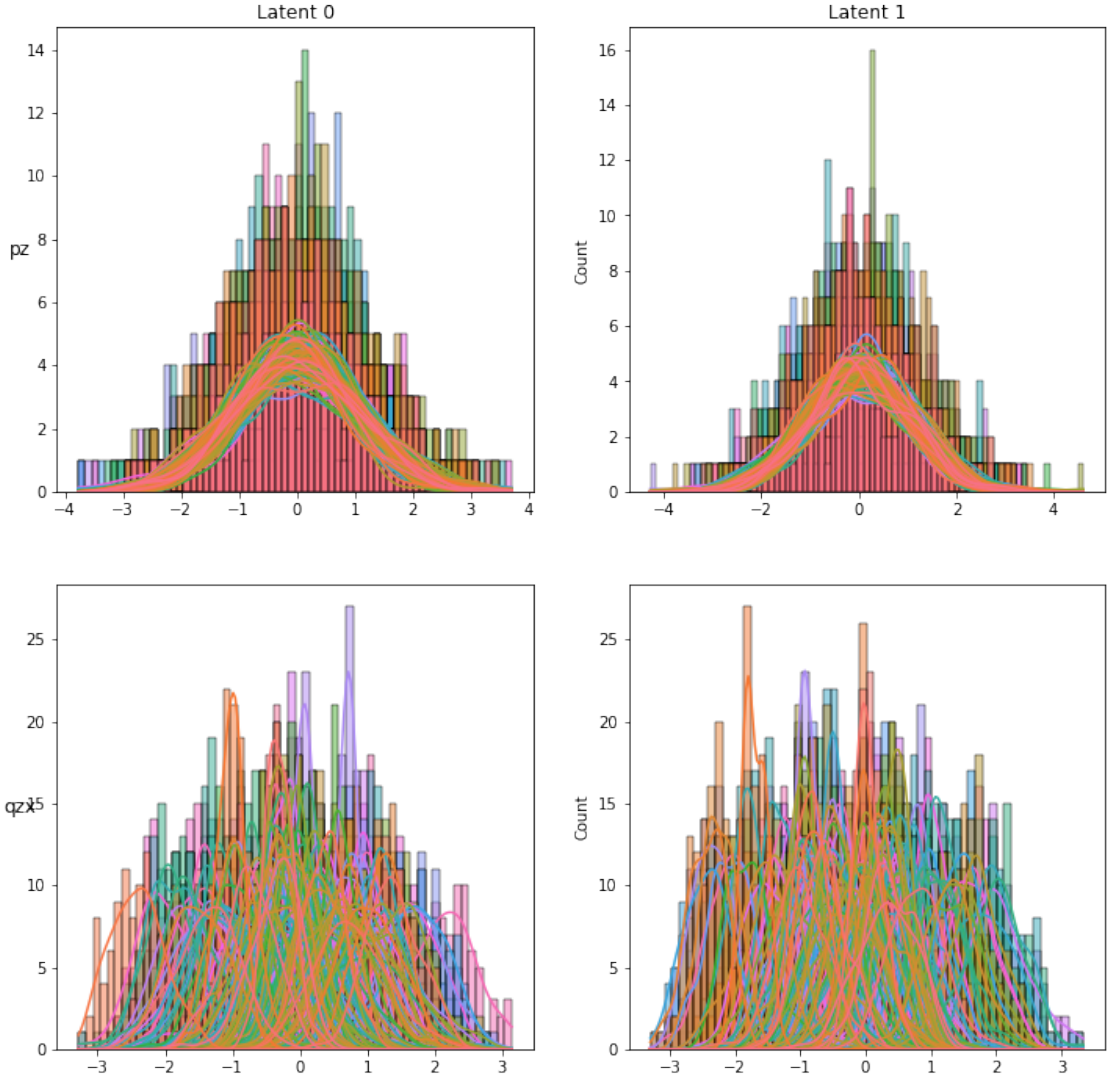

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

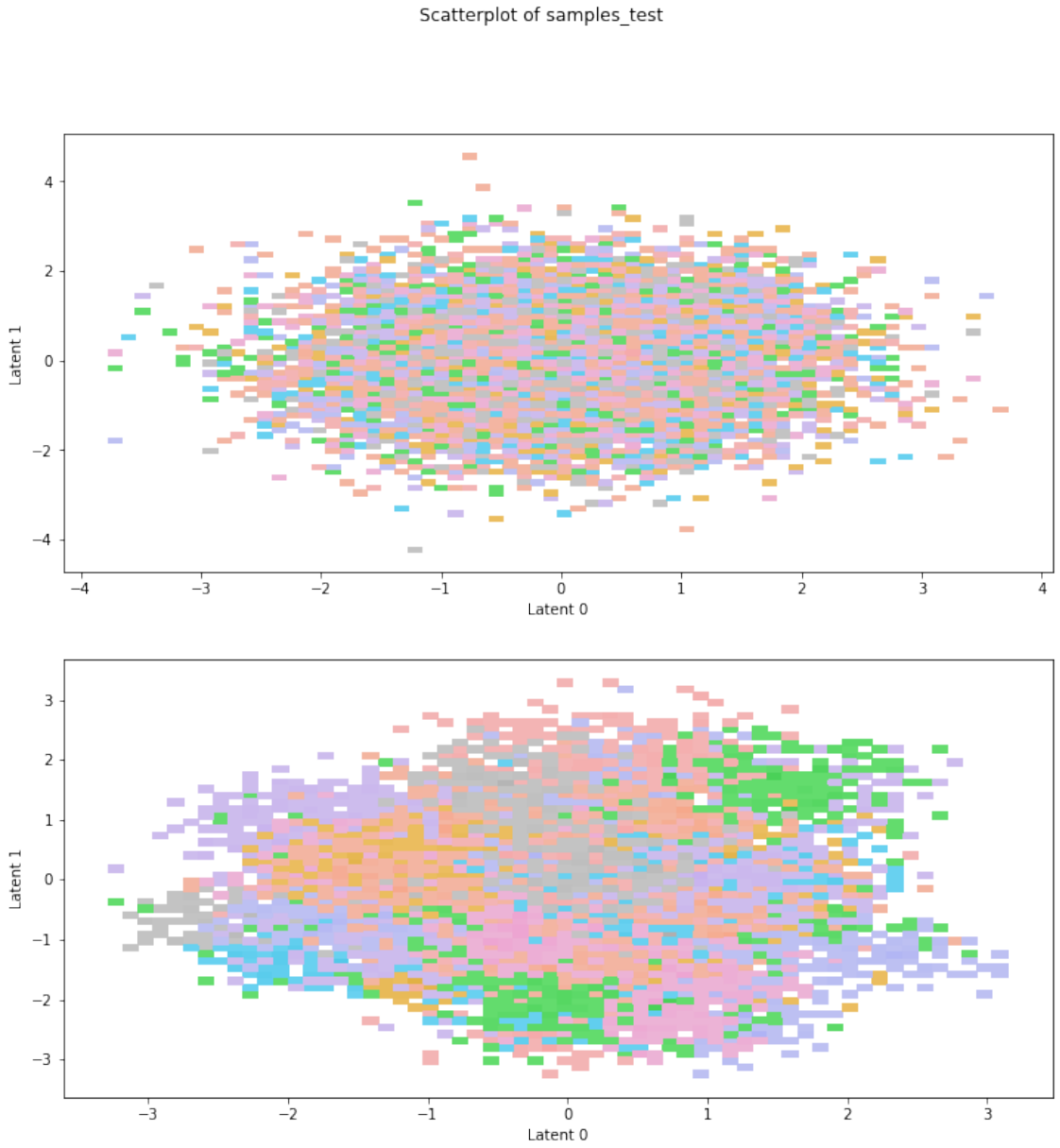



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```

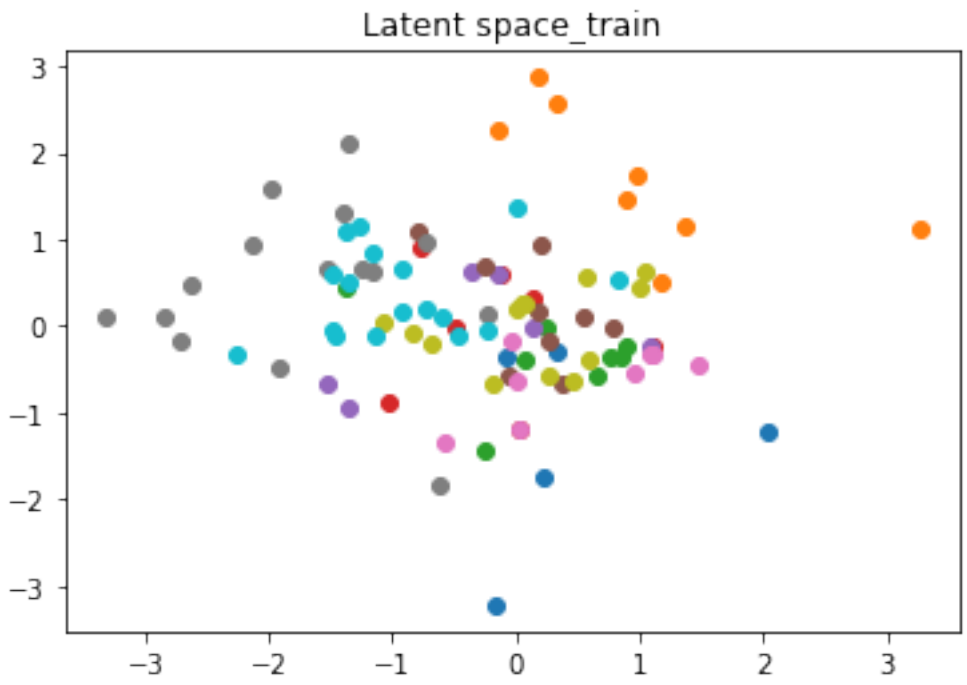



```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

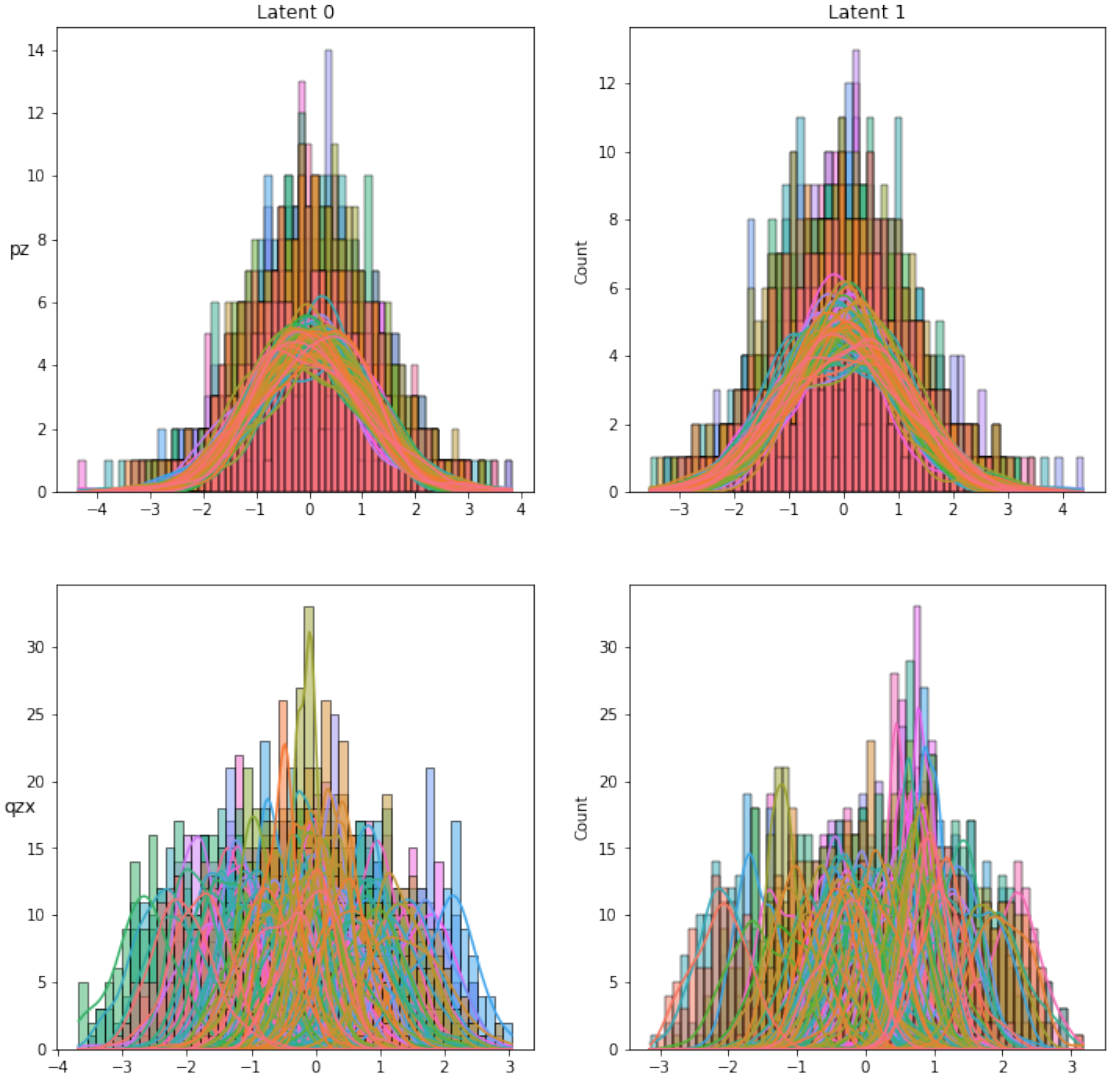

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count

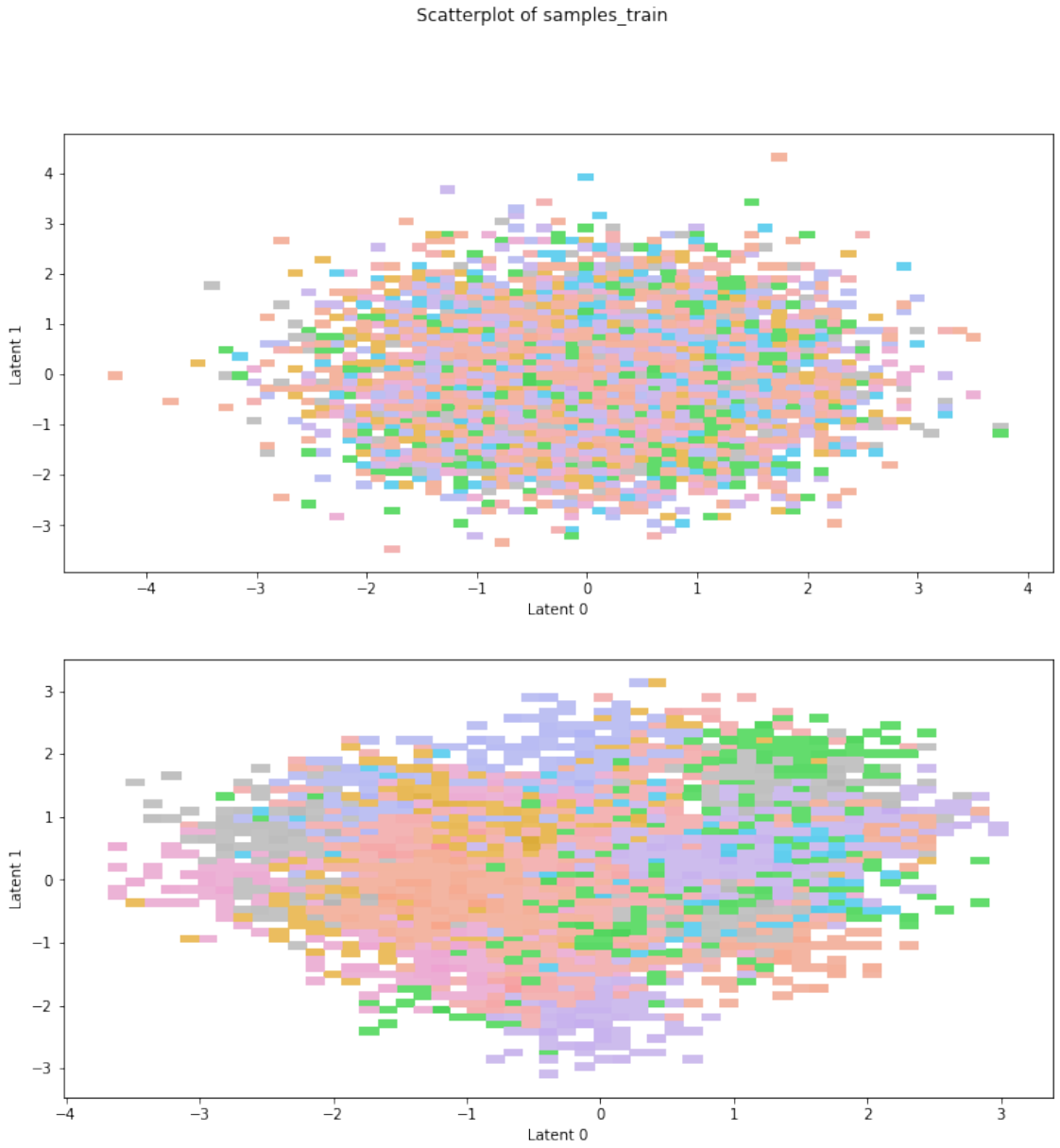



```
Epoch 4, Loss 24330.6896, kl_loss 253.4965, recon_loss 21795.7245, kl_divergence
242.4884
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

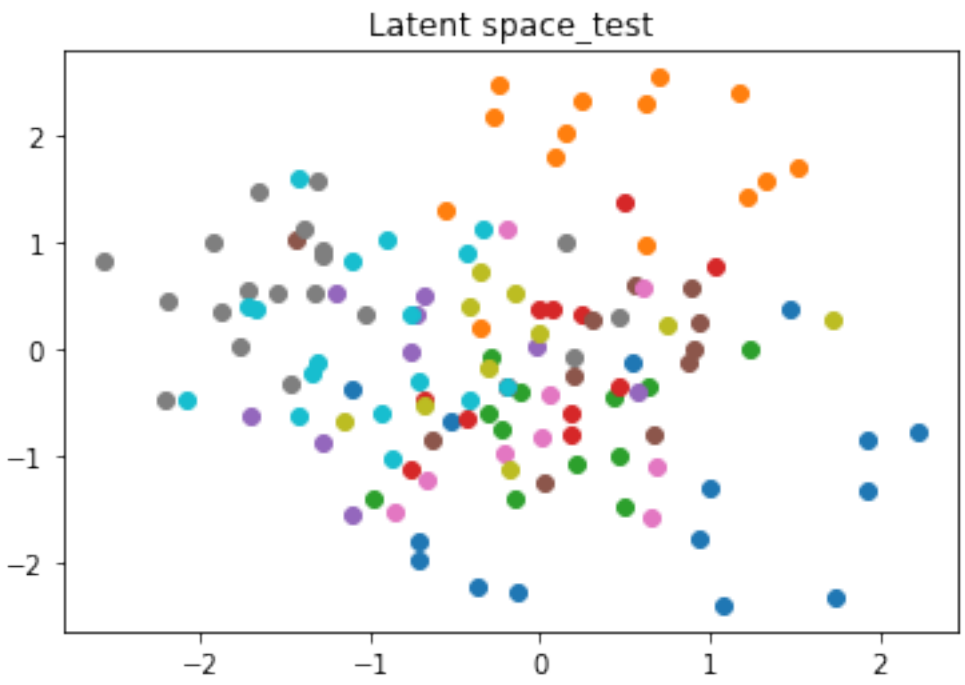



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

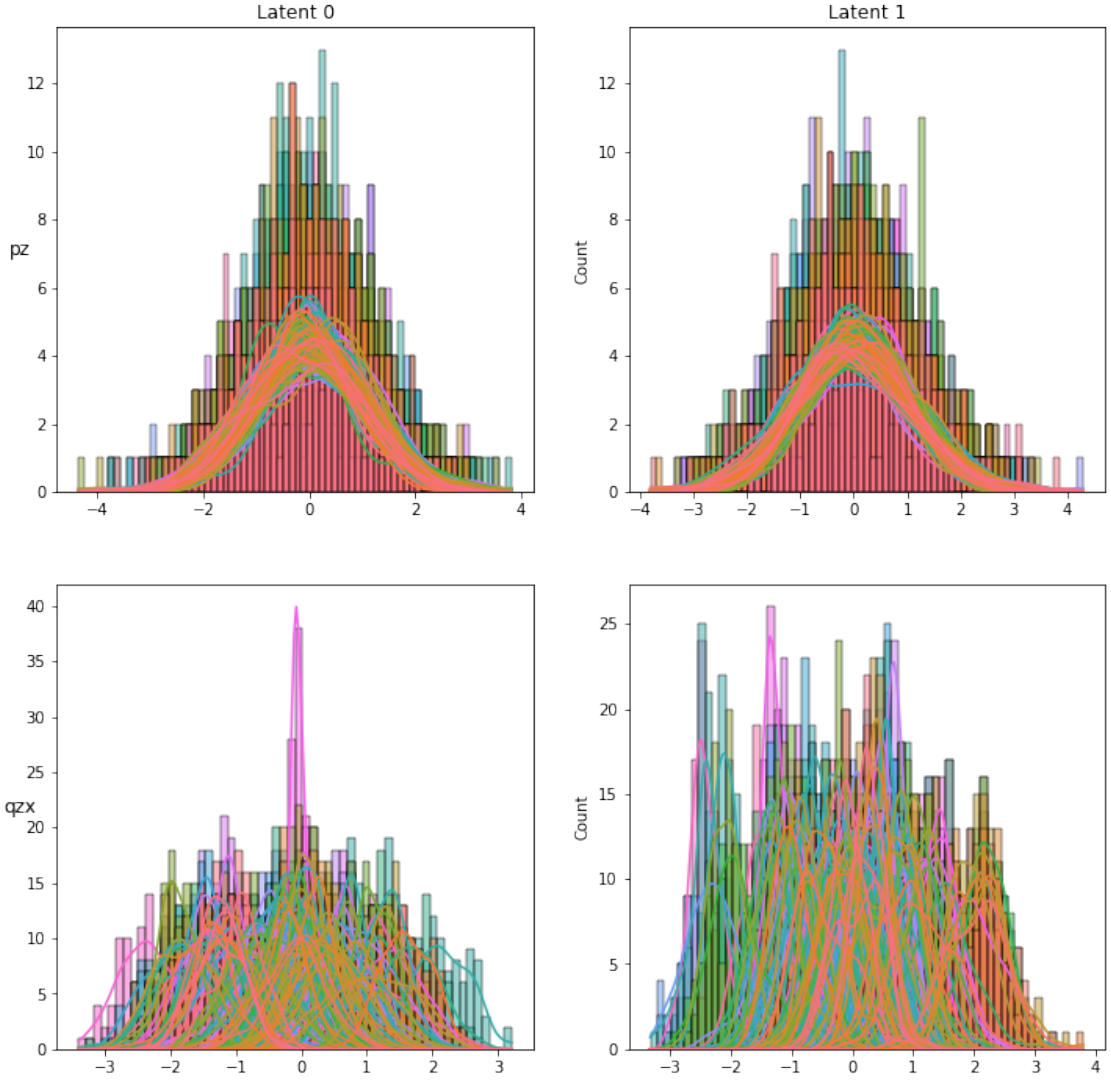

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

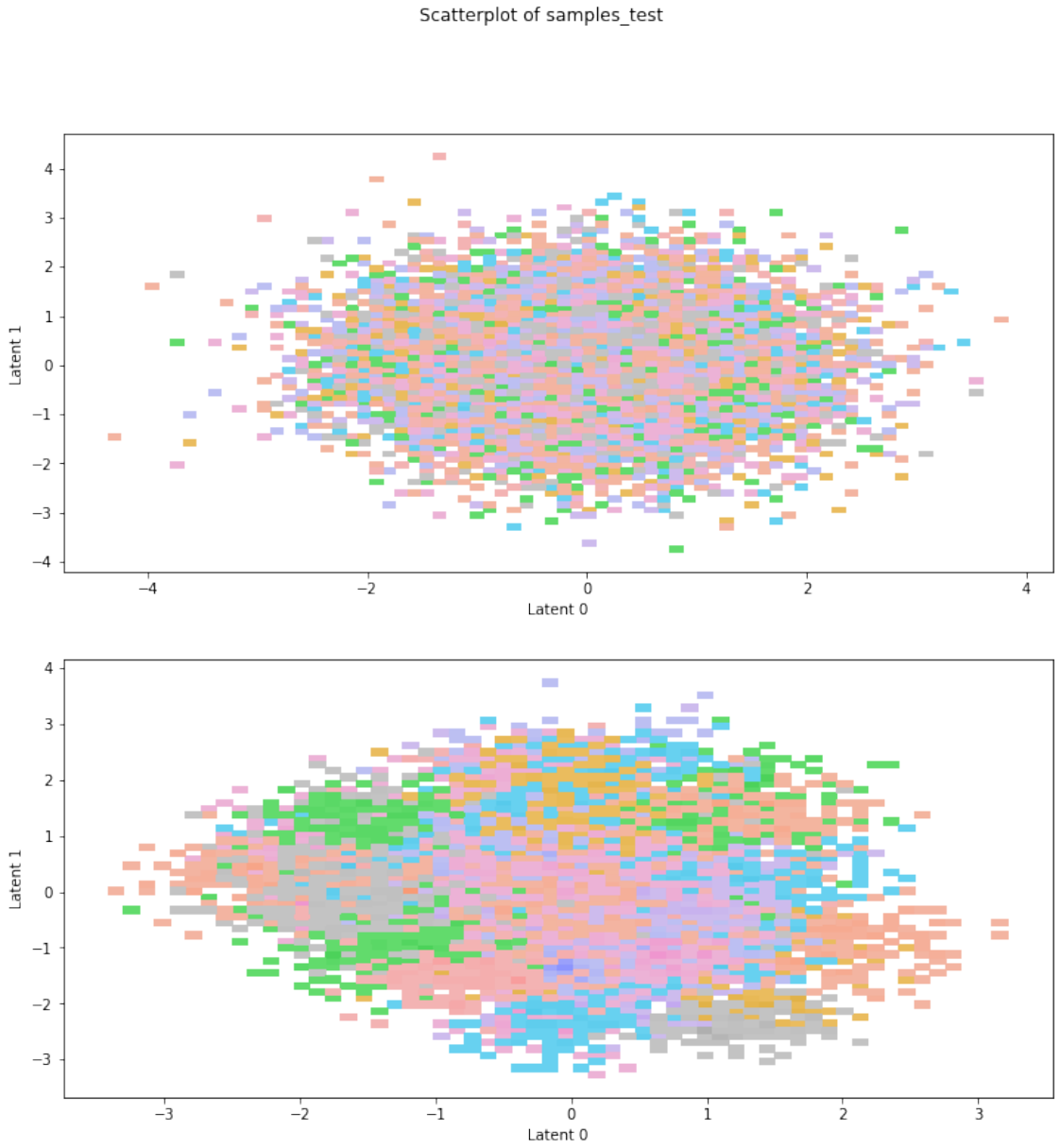



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```

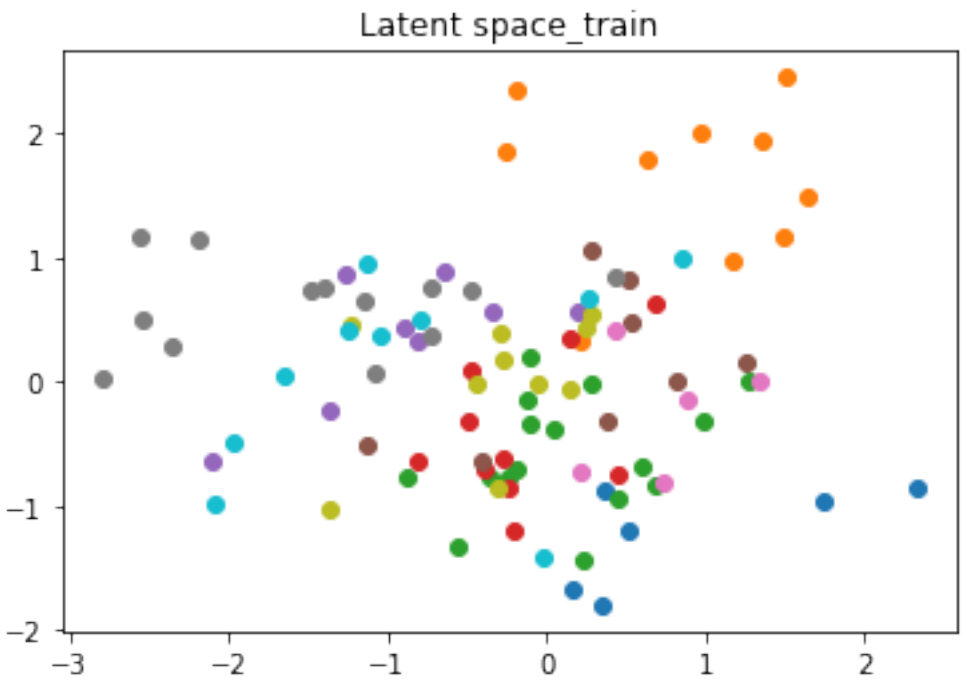



```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

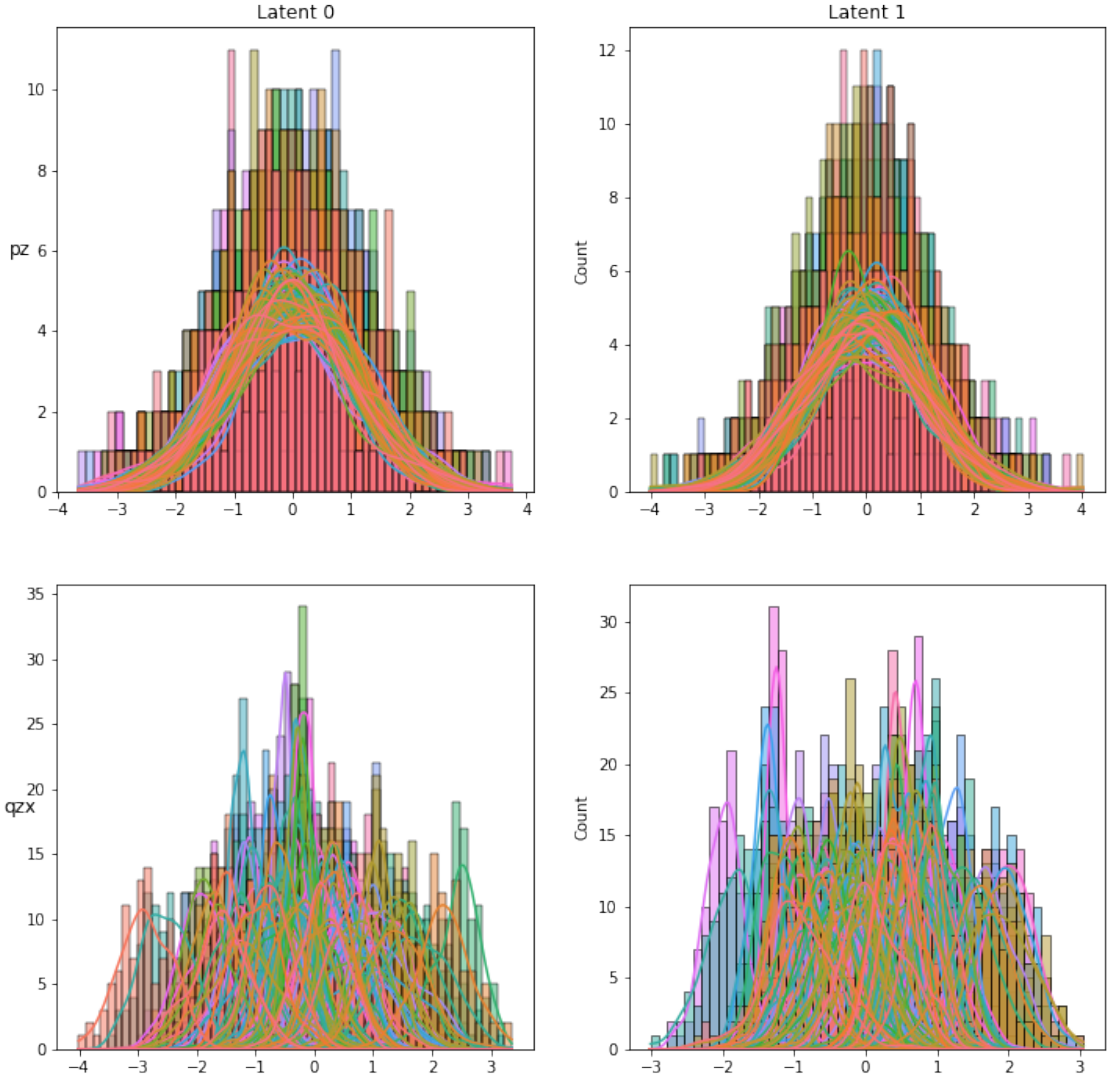

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
qzx
Count
Count

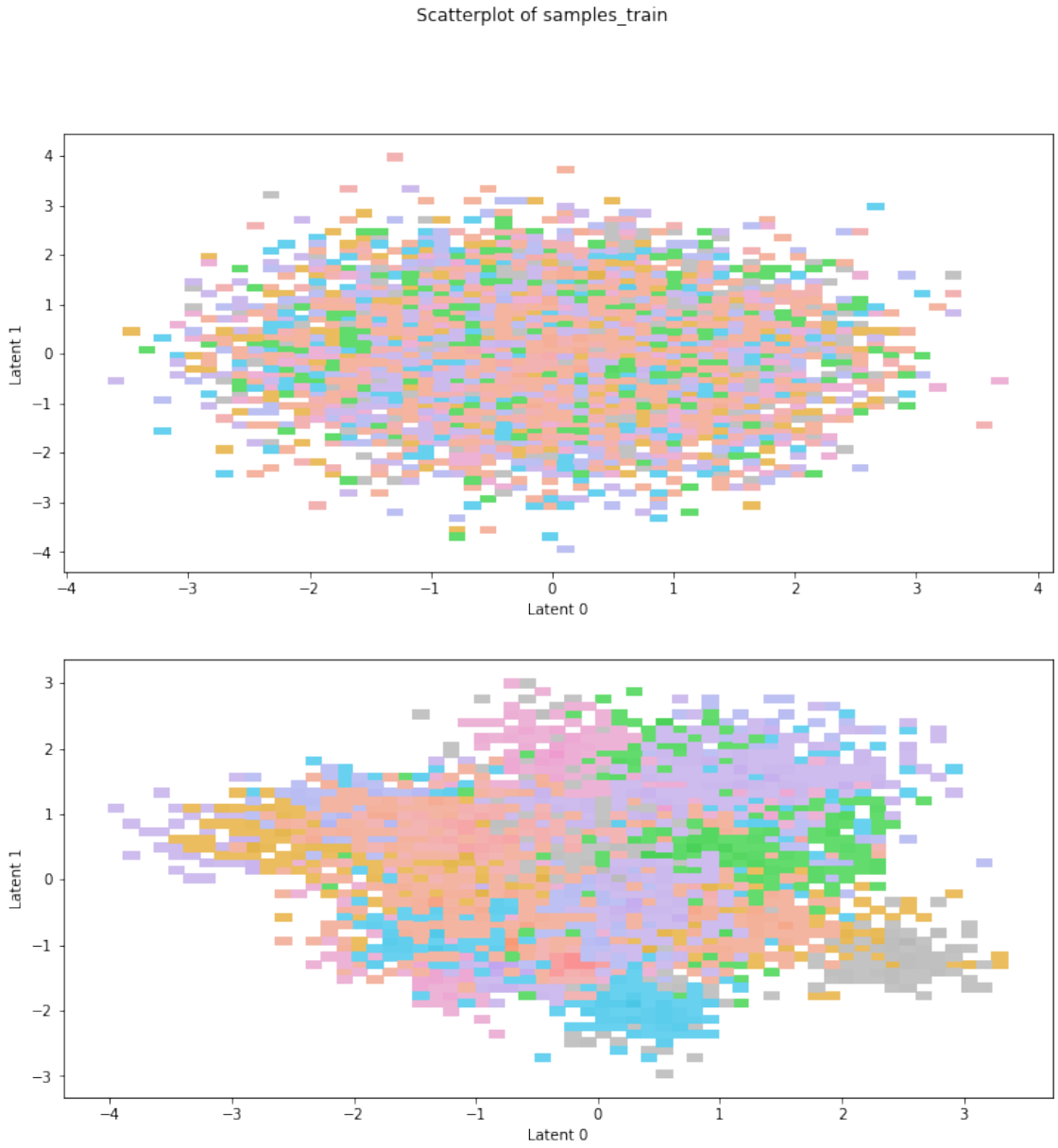



```
Epoch 5, Loss 24263.2728, kl_loss 261.7018, recon_loss 21646.2549, kl_divergence
247.3936
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

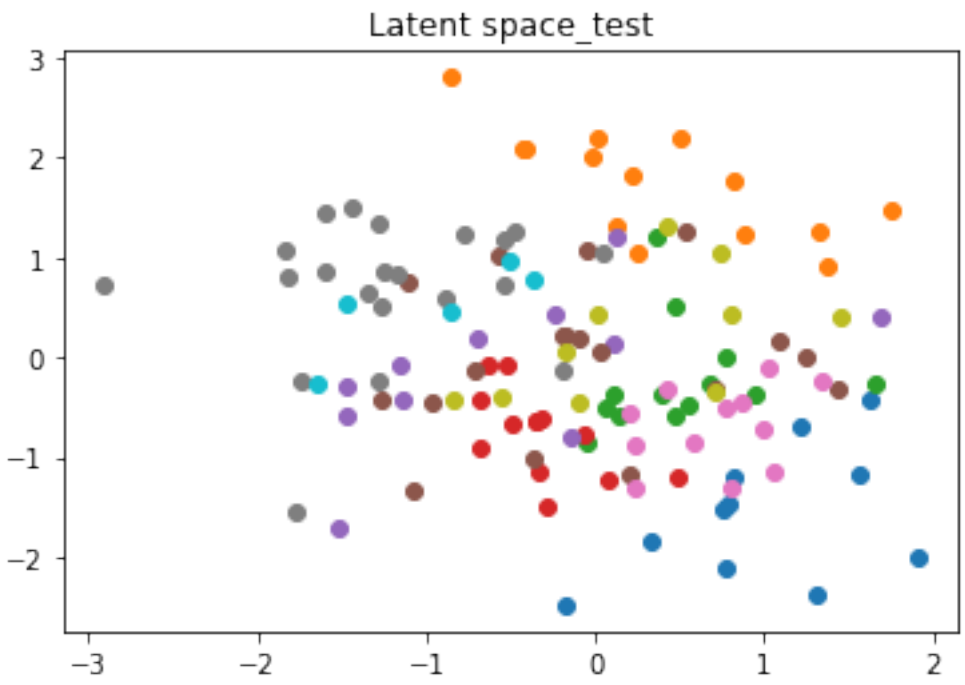



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

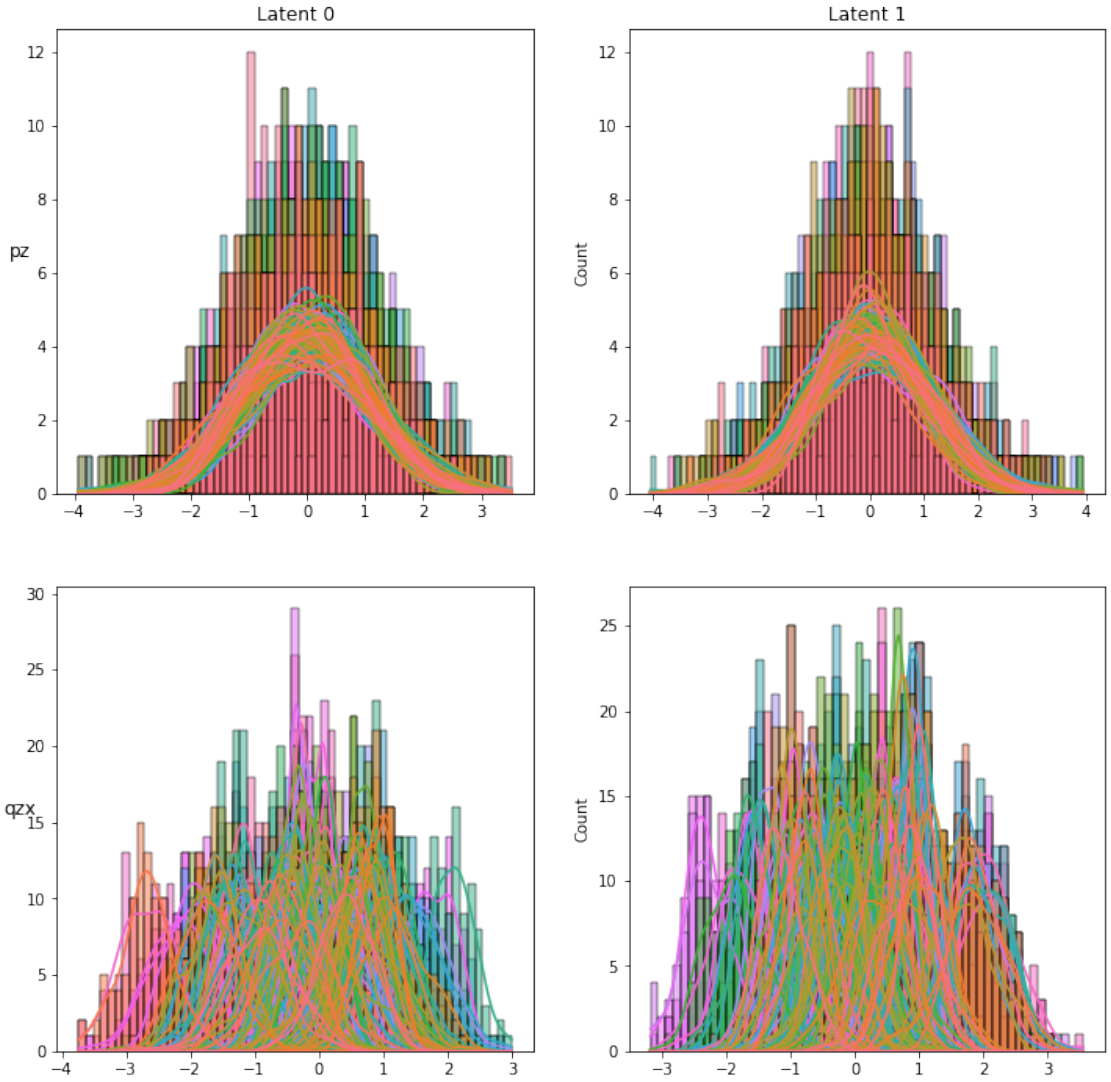

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

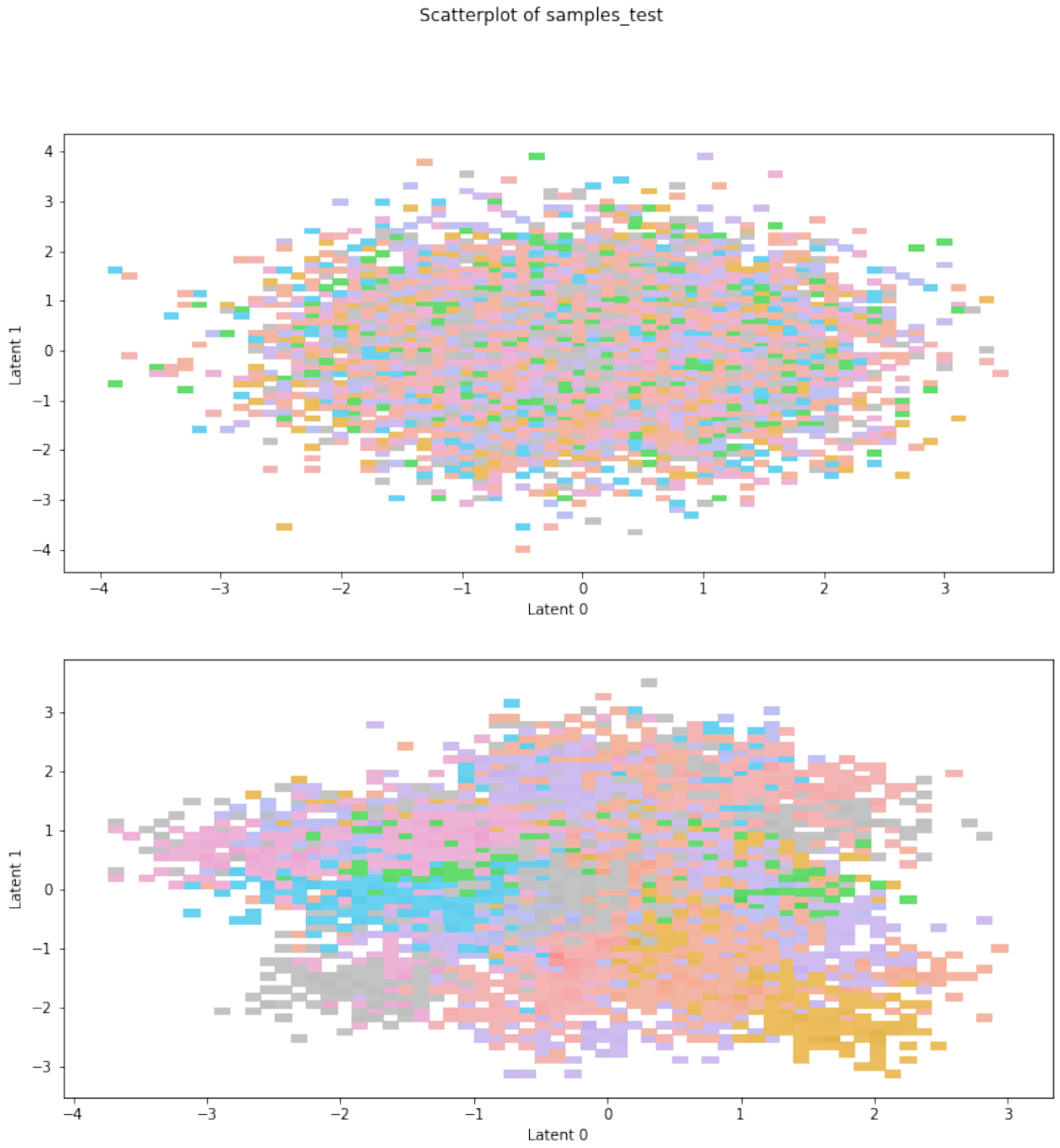



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```

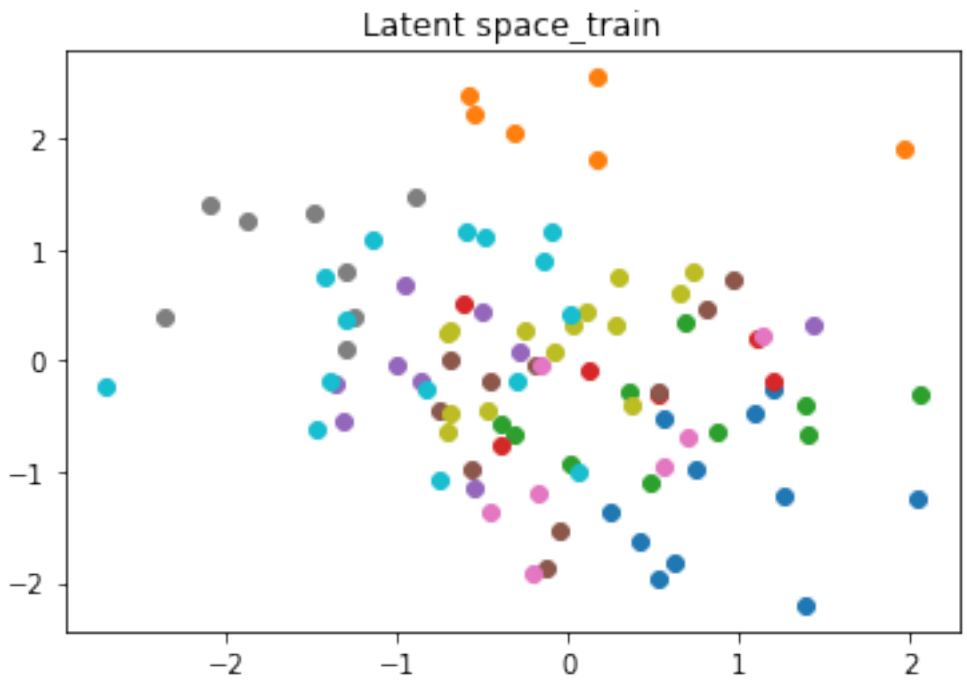



```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

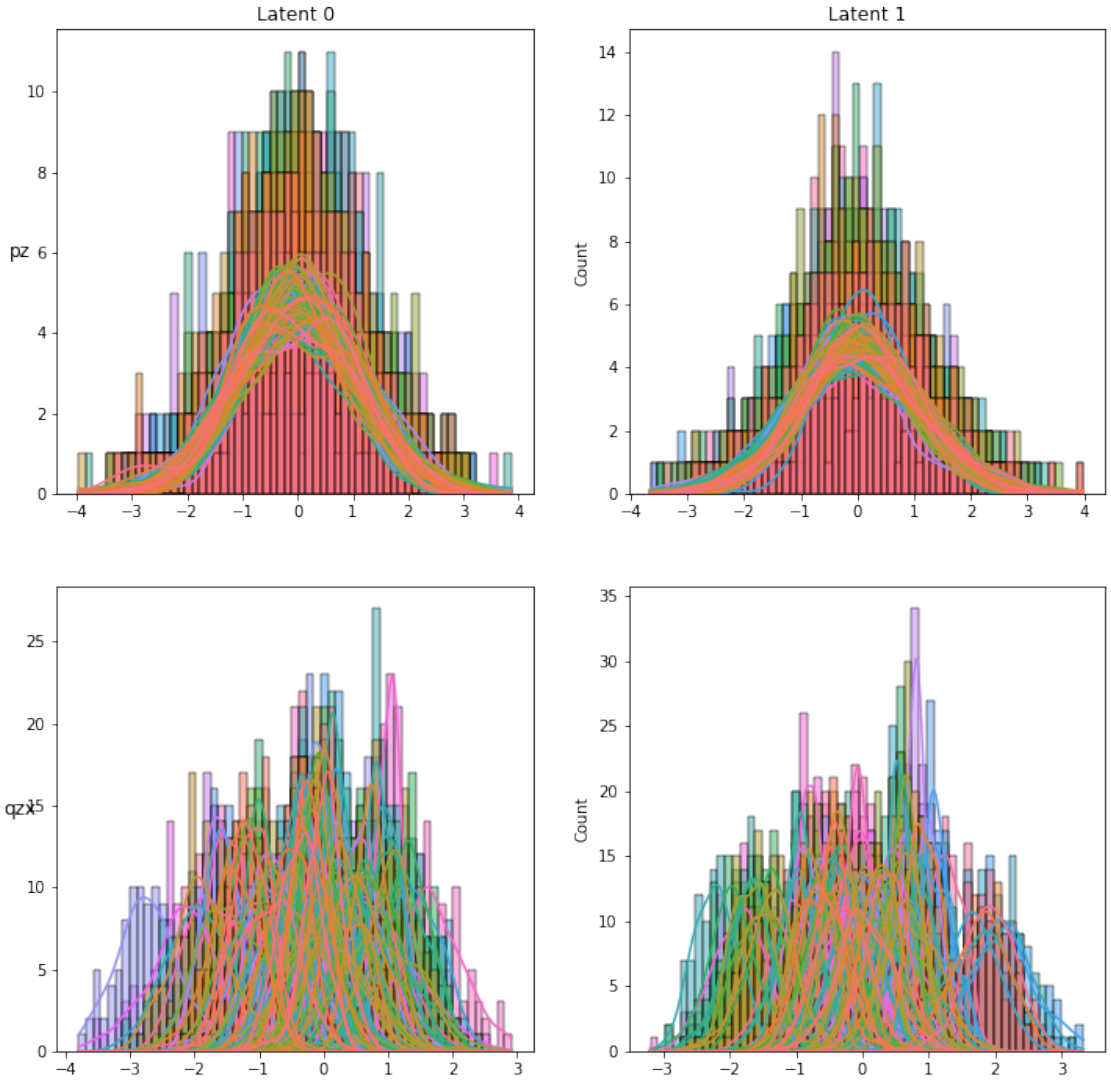

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
qzx
Count
Count

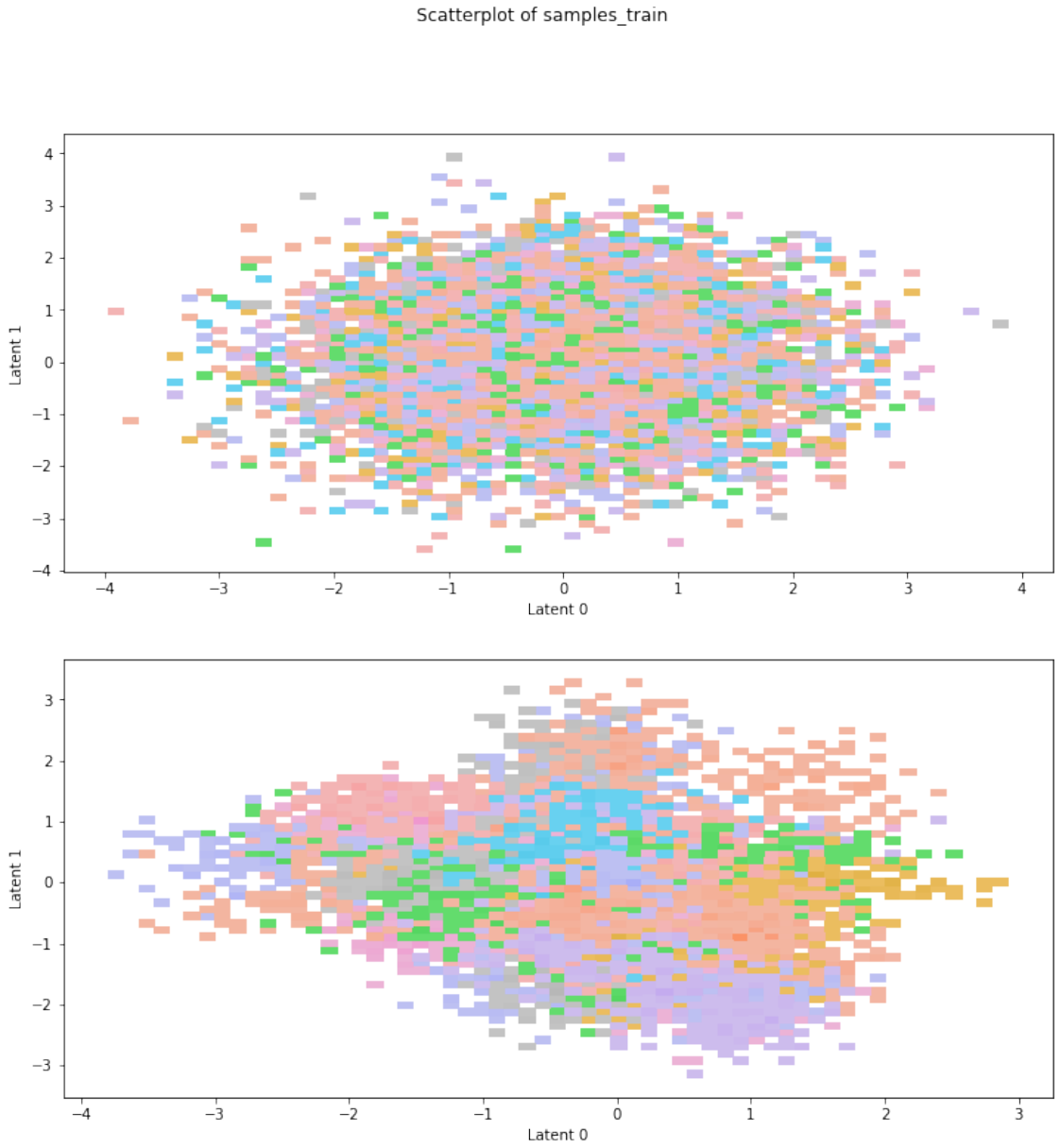



```
Epoch 6, Loss 24224.8780, kl_loss 269.5320, recon_loss 21529.5577, kl_divergence
253.0020
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

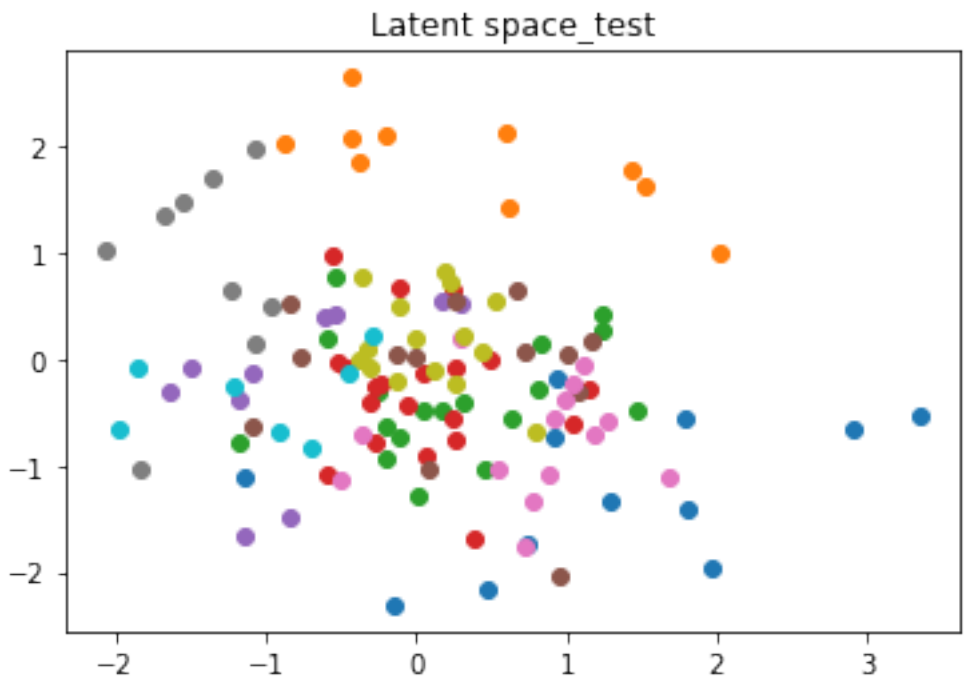



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

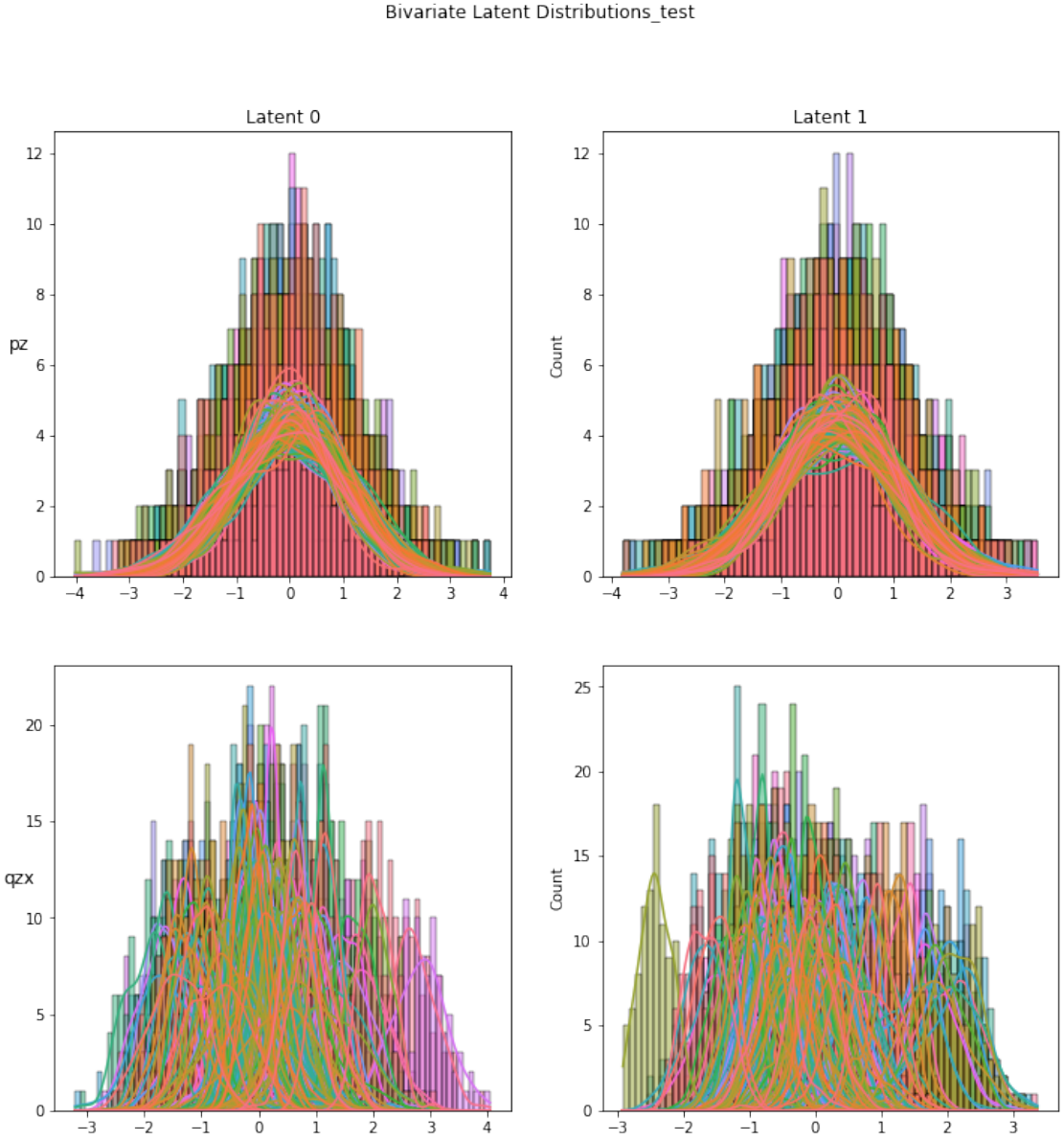

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
qz|x
Count
Count

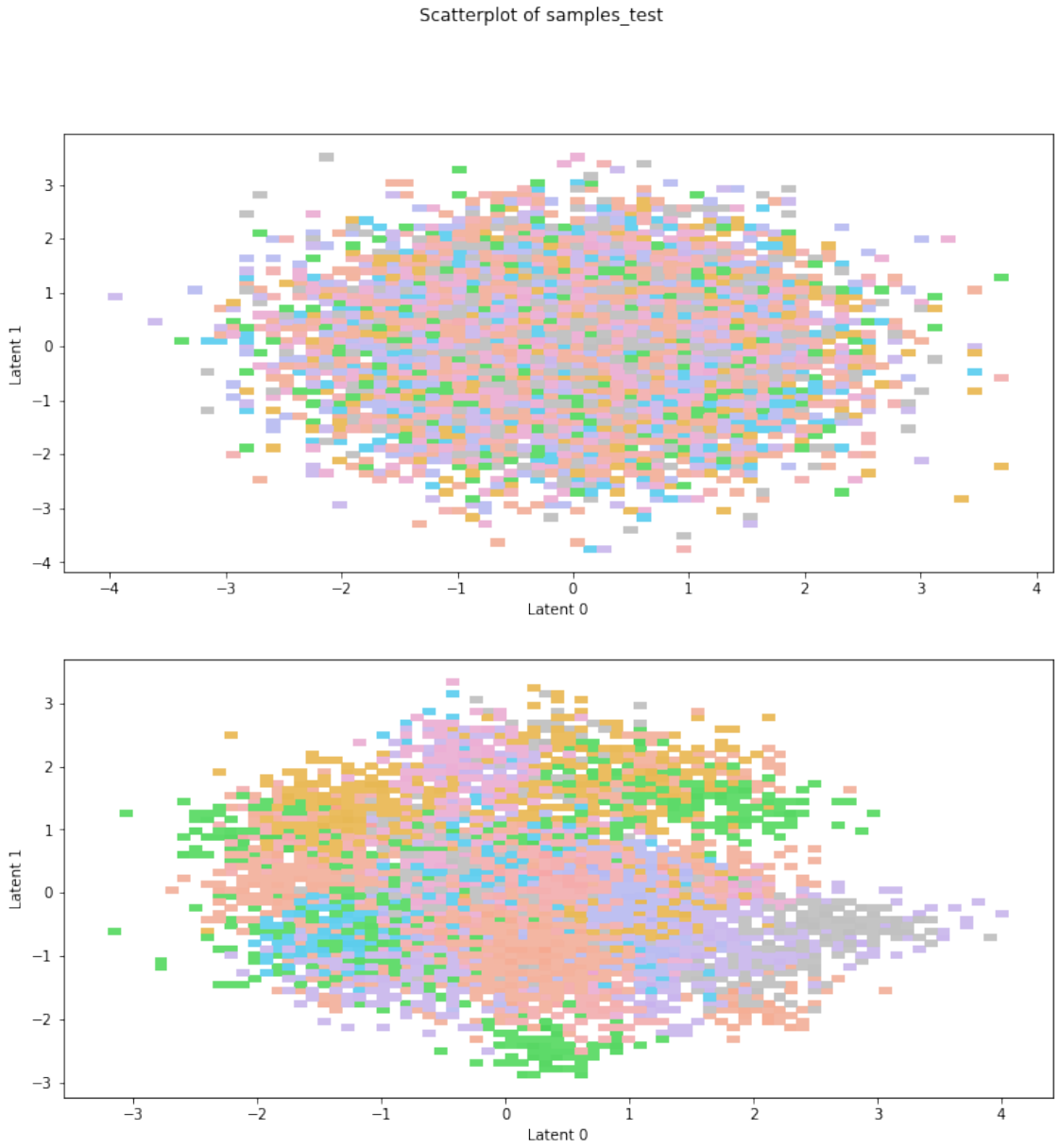



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

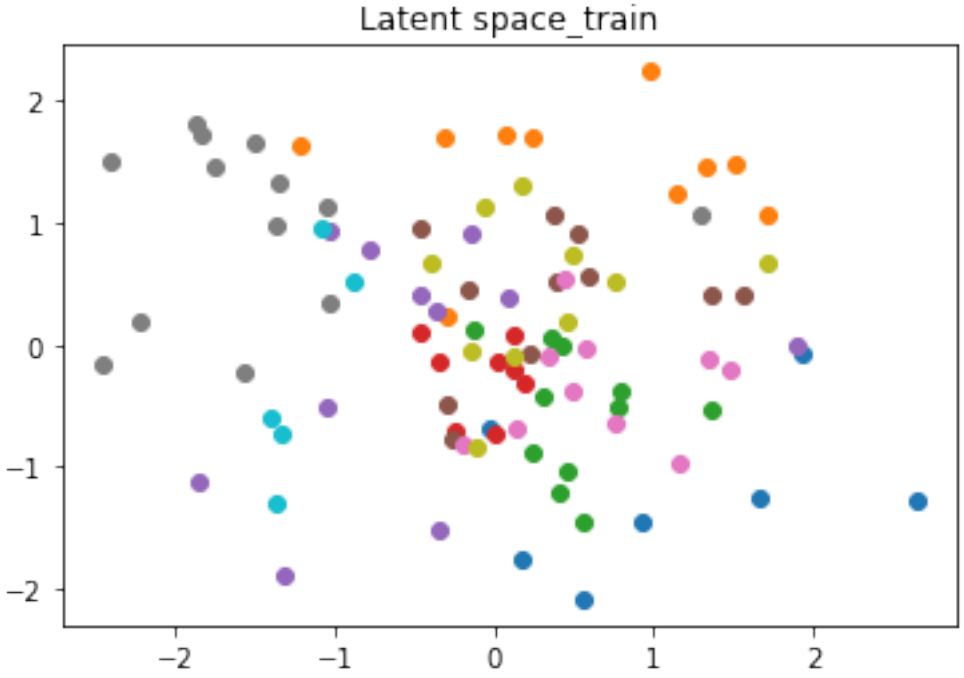

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count



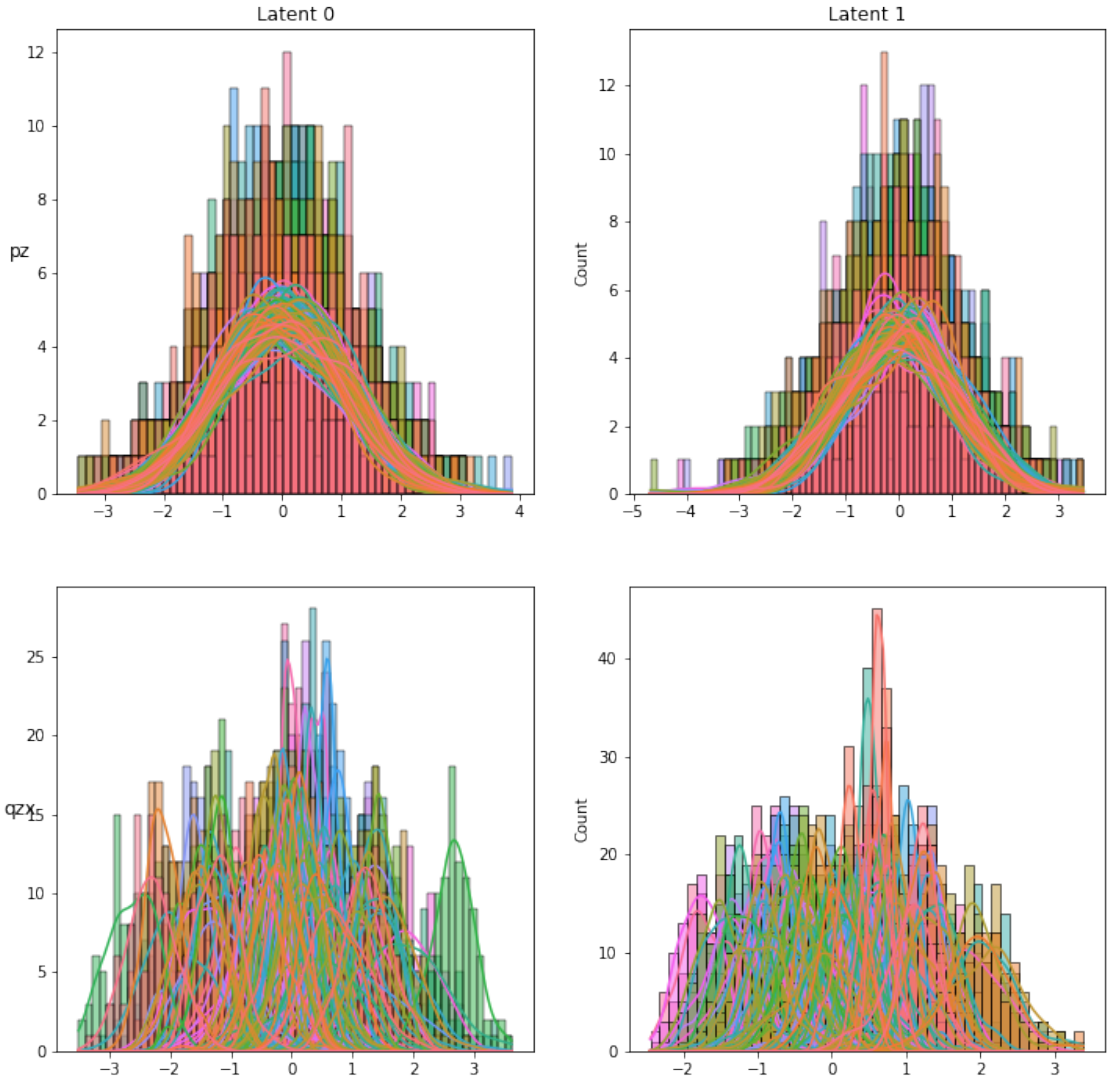

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

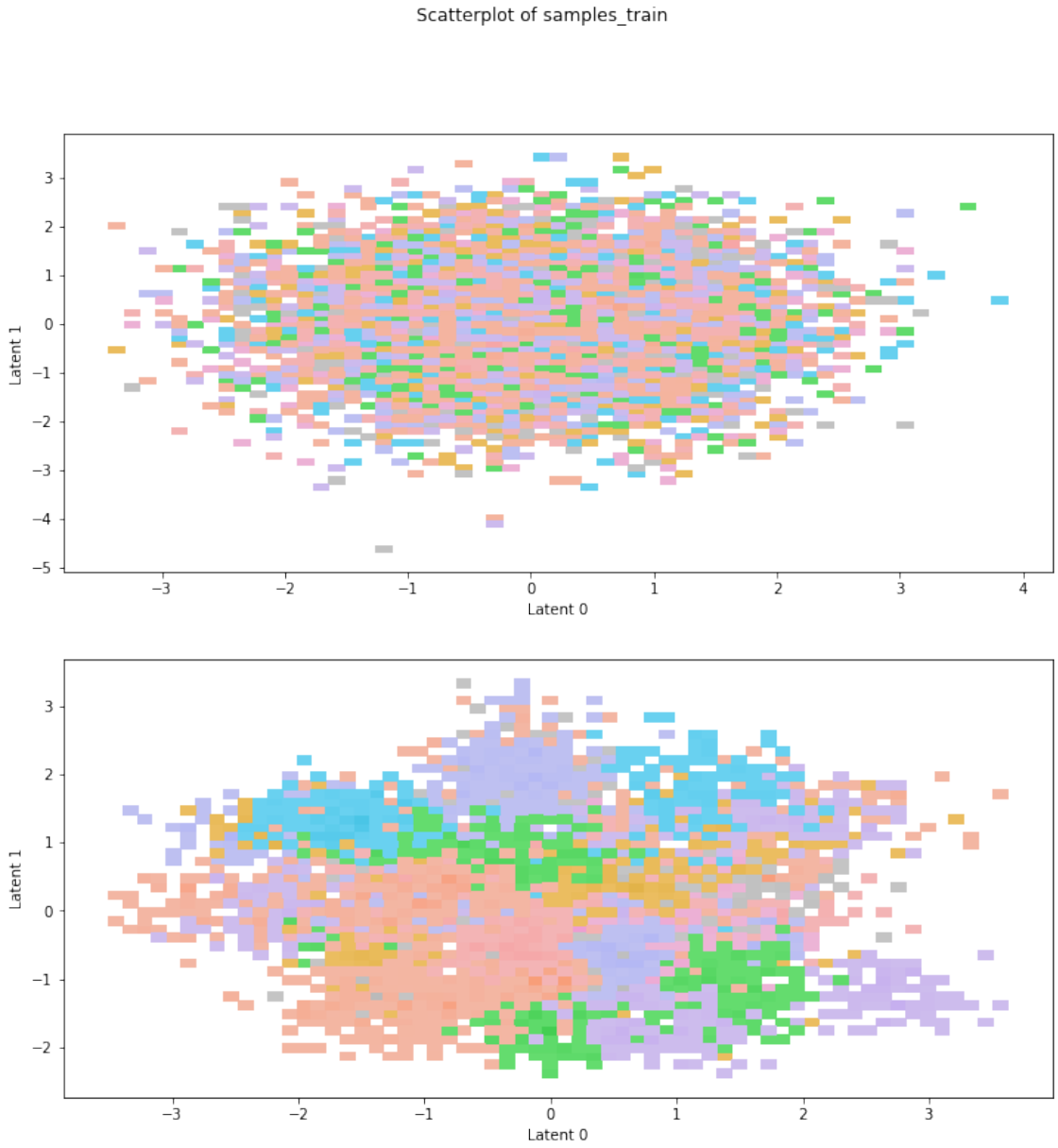
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 11, Loss 24072.9779, kl_loss 293.8677, recon_loss 21134.3006,
kl_divergence 266.7414
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

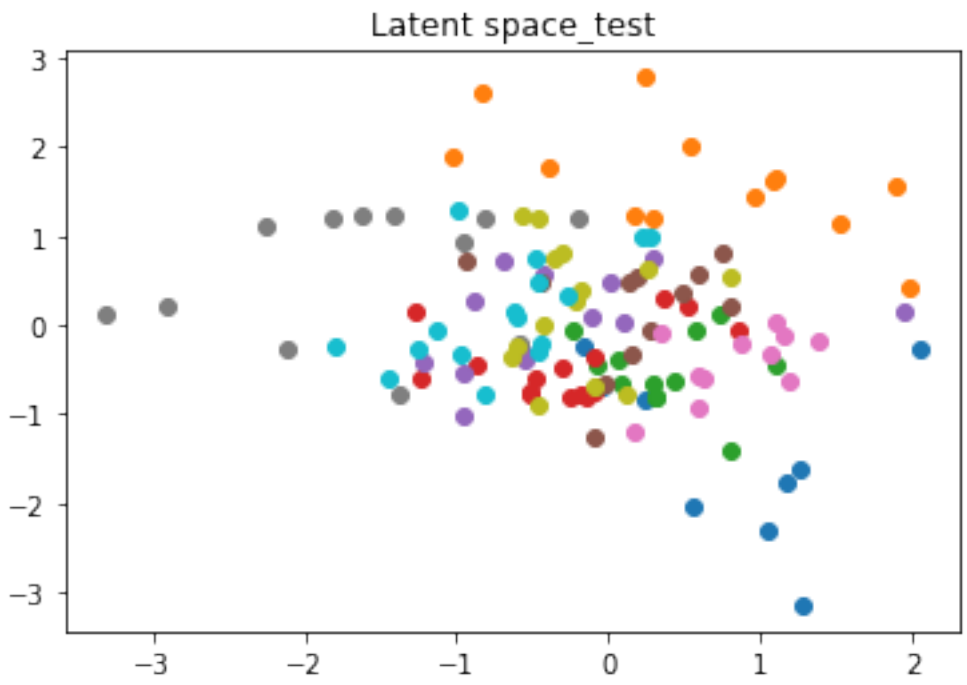



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

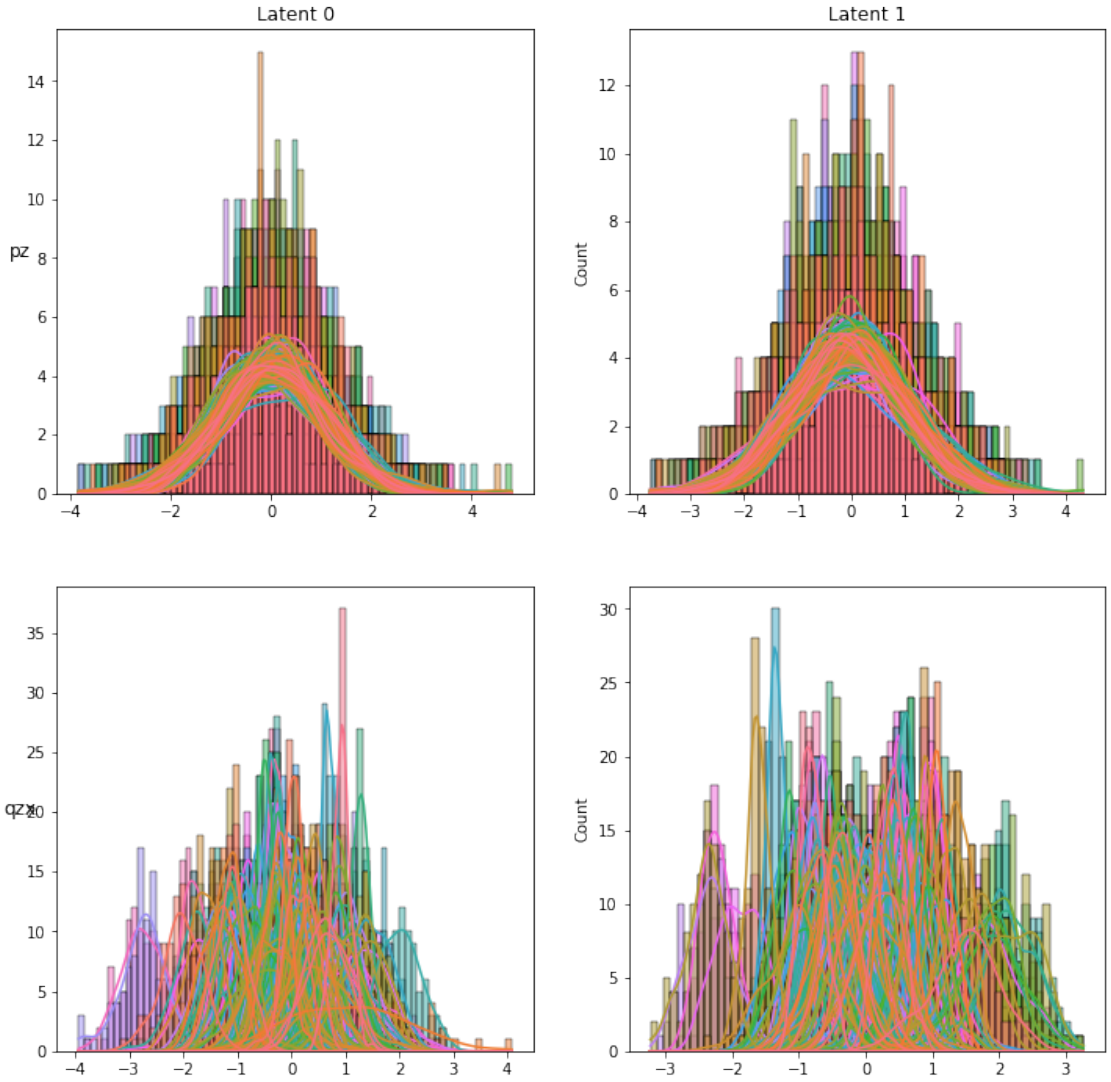

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count




```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```



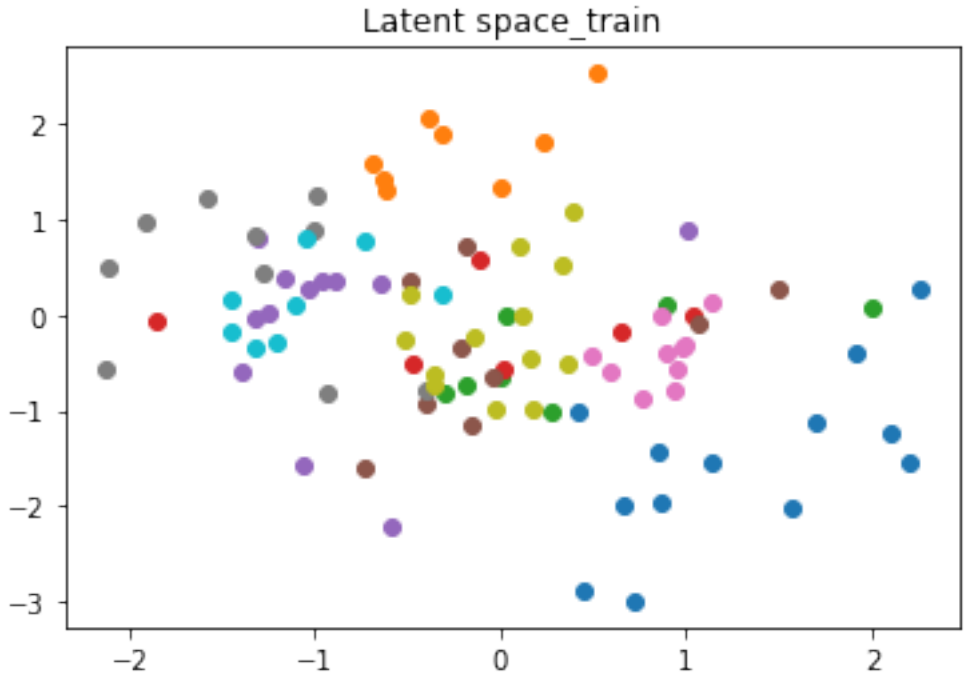

```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```


Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qz|x
Count

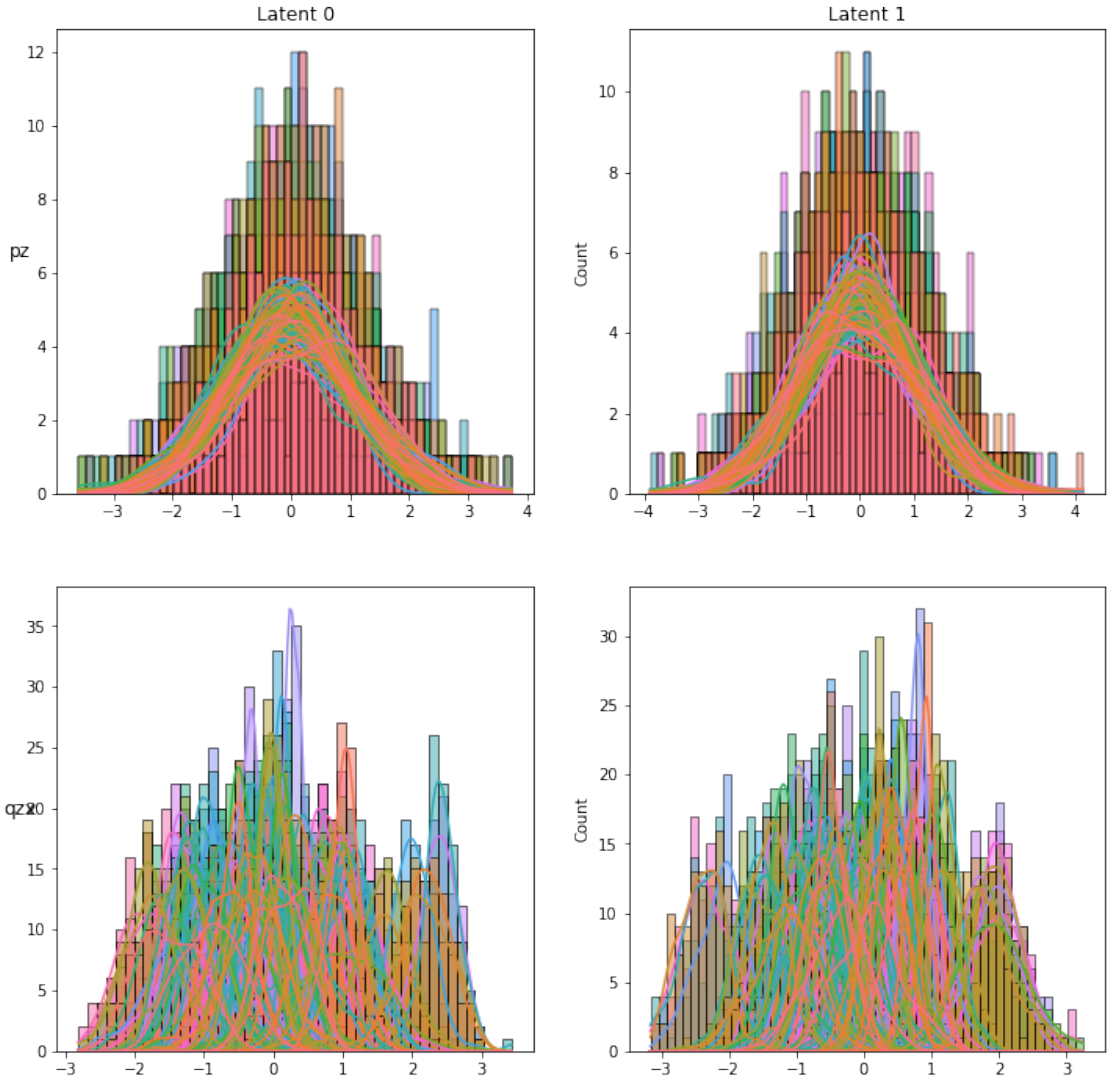

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

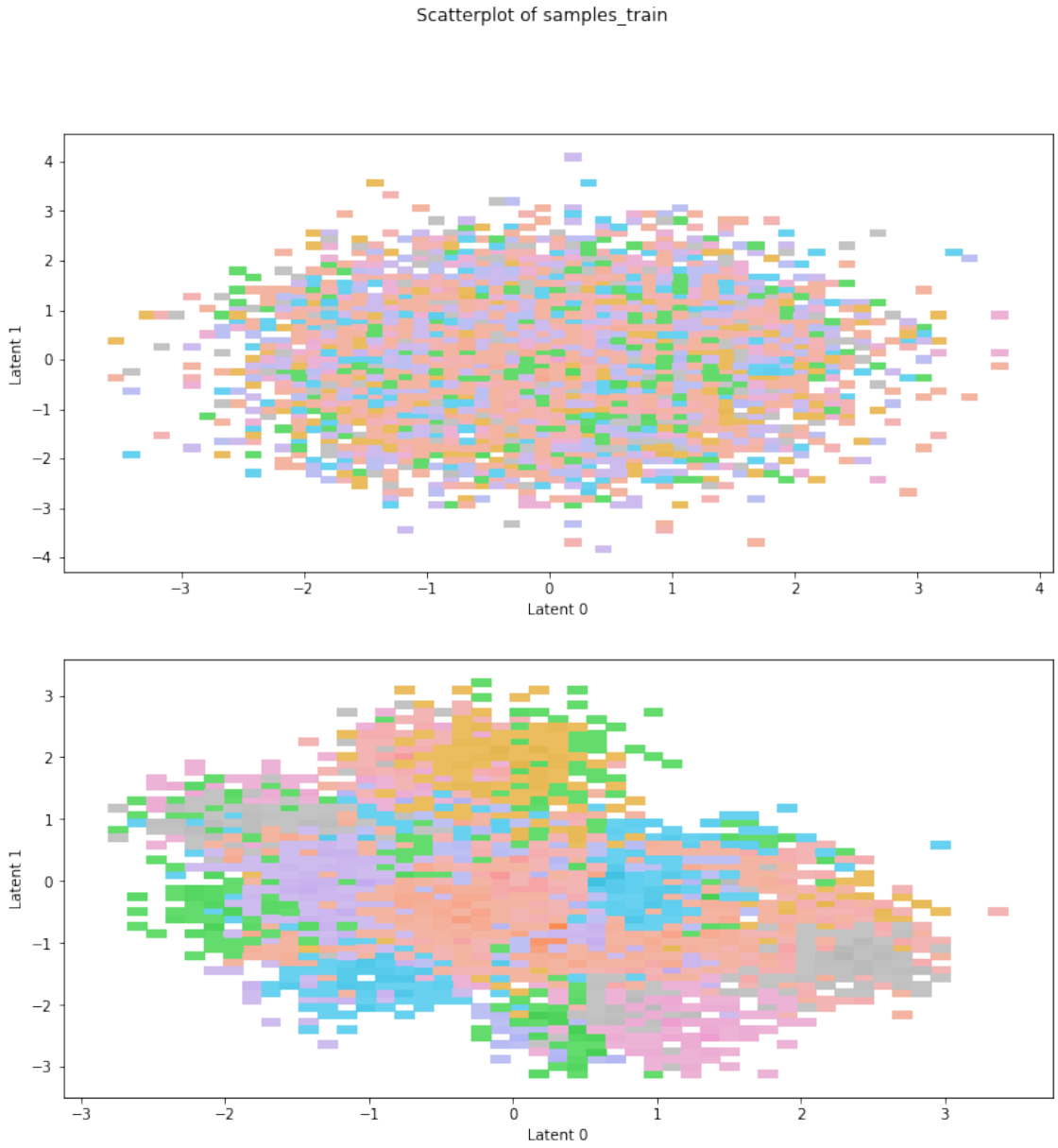
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 16, Loss 23987.4009, kl_loss 307.6212, recon_loss 20911.1891,
kl_divergence 273.4875
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

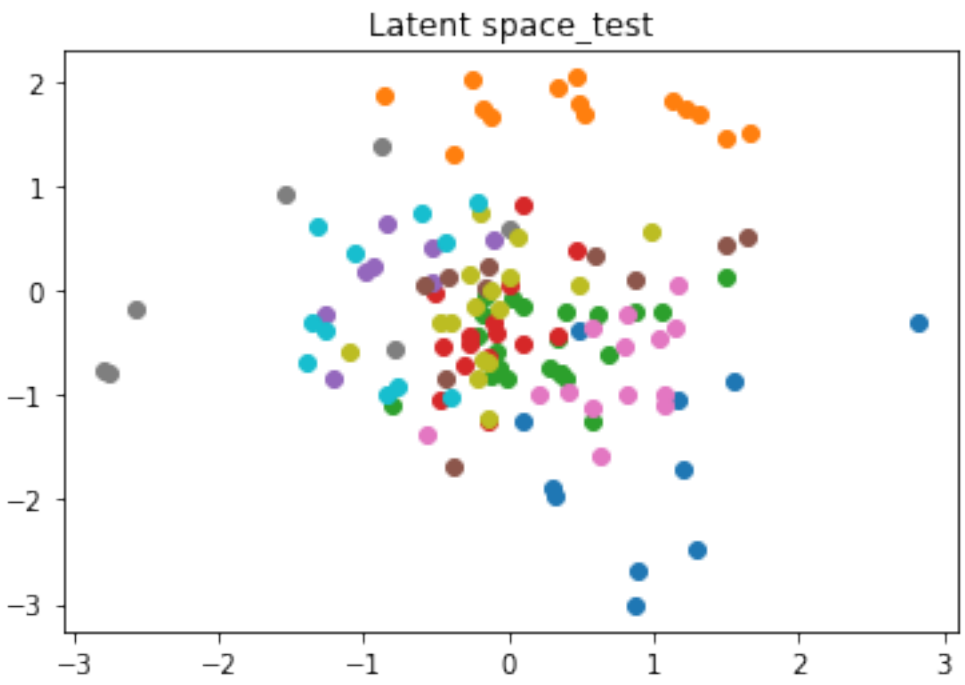



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

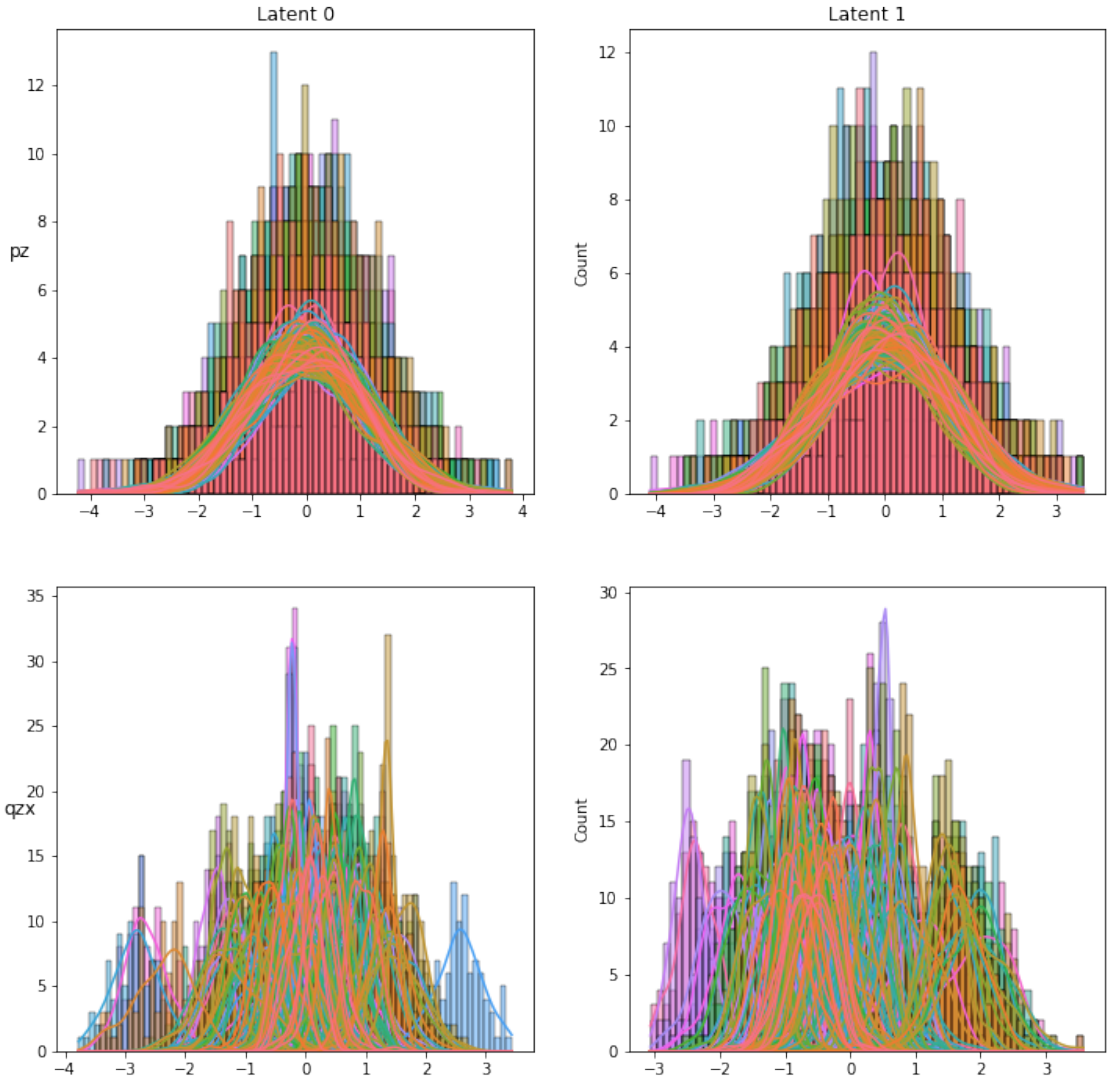

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qz
Count

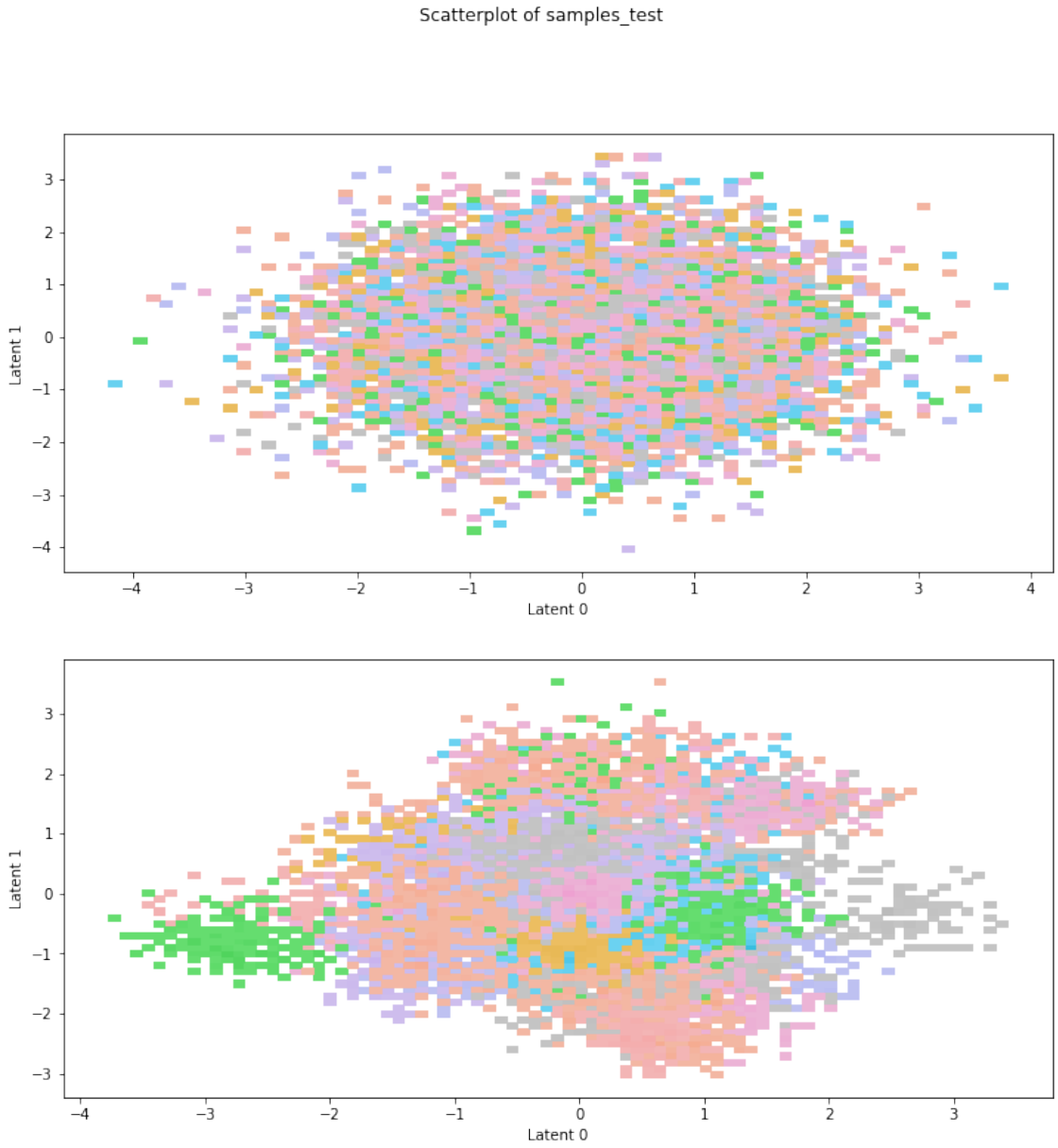



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

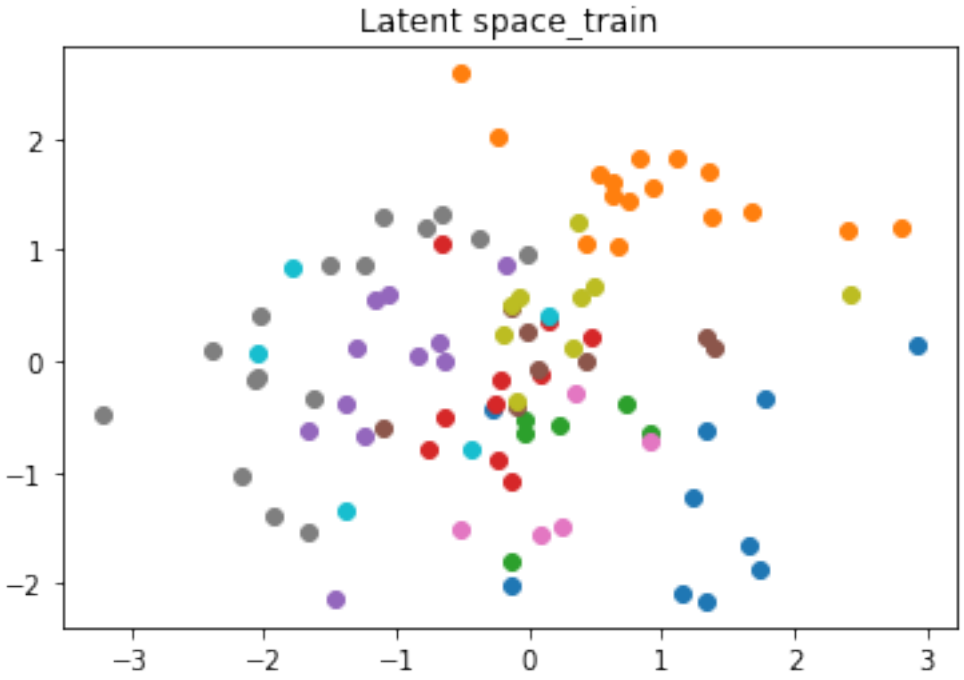

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
qzx
Count
Count



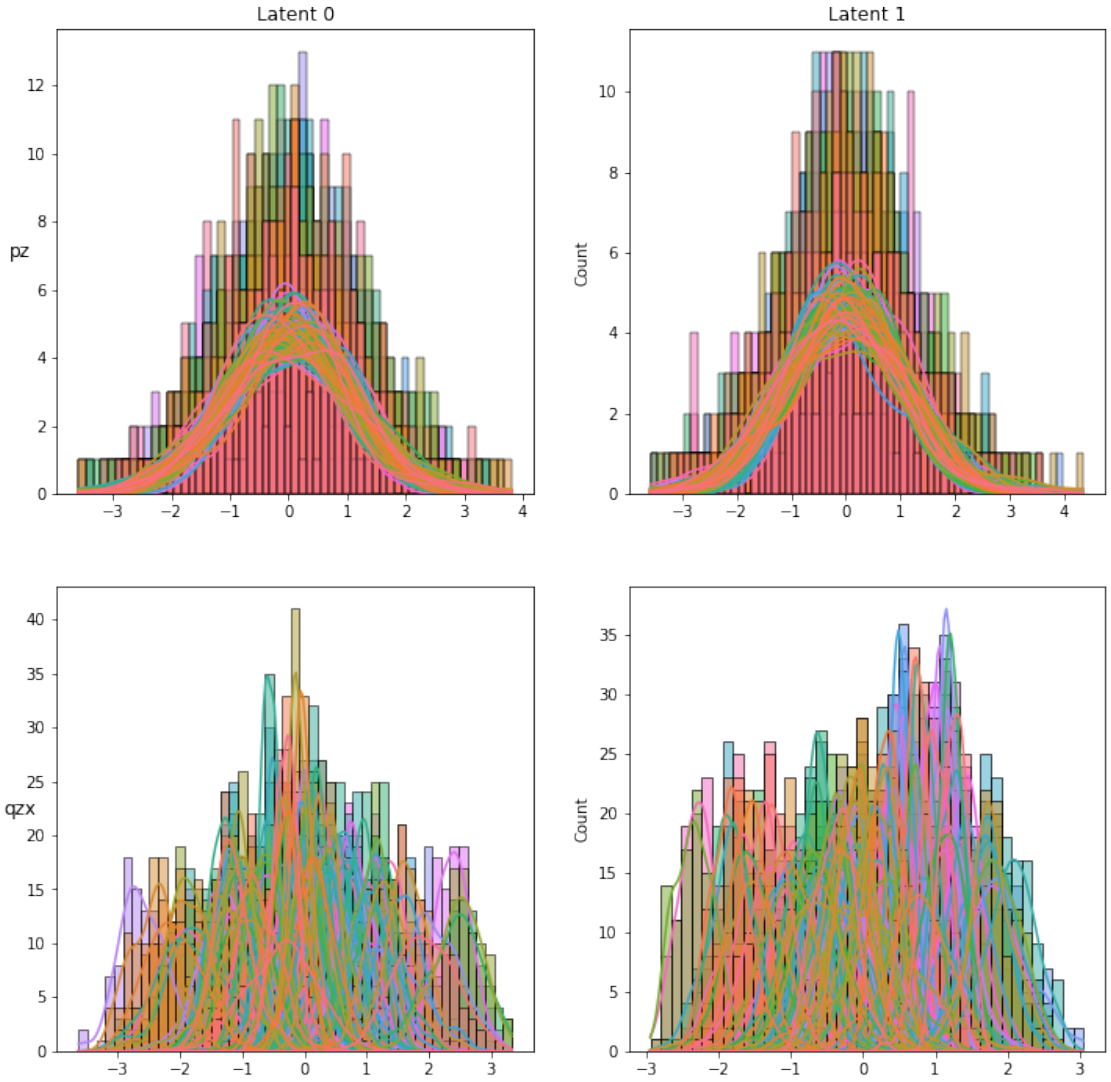

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

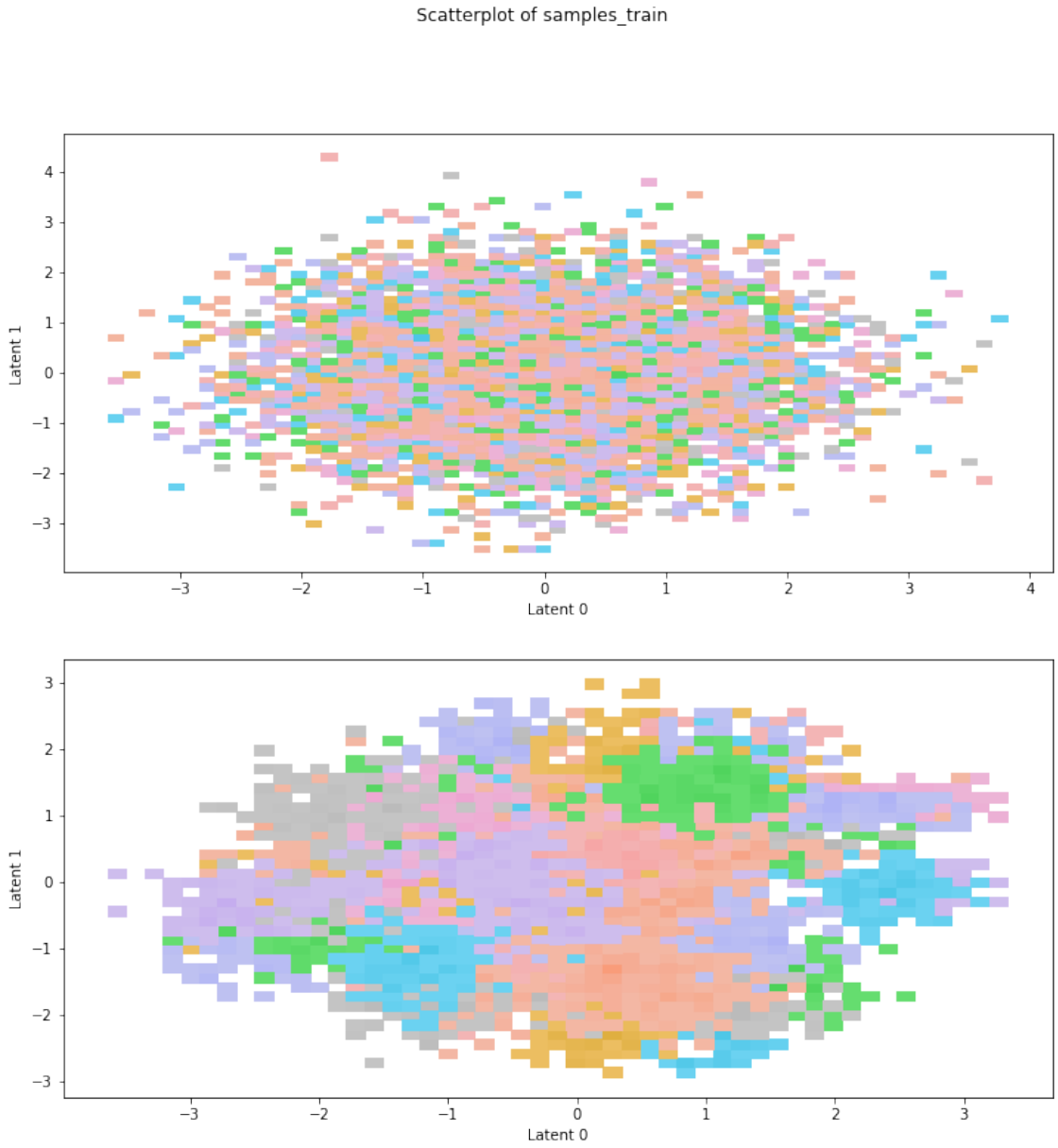
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 21, Loss 23925.1056, kl_loss 316.2605, recon_loss 20762.5010,
kl_divergence 276.9648
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

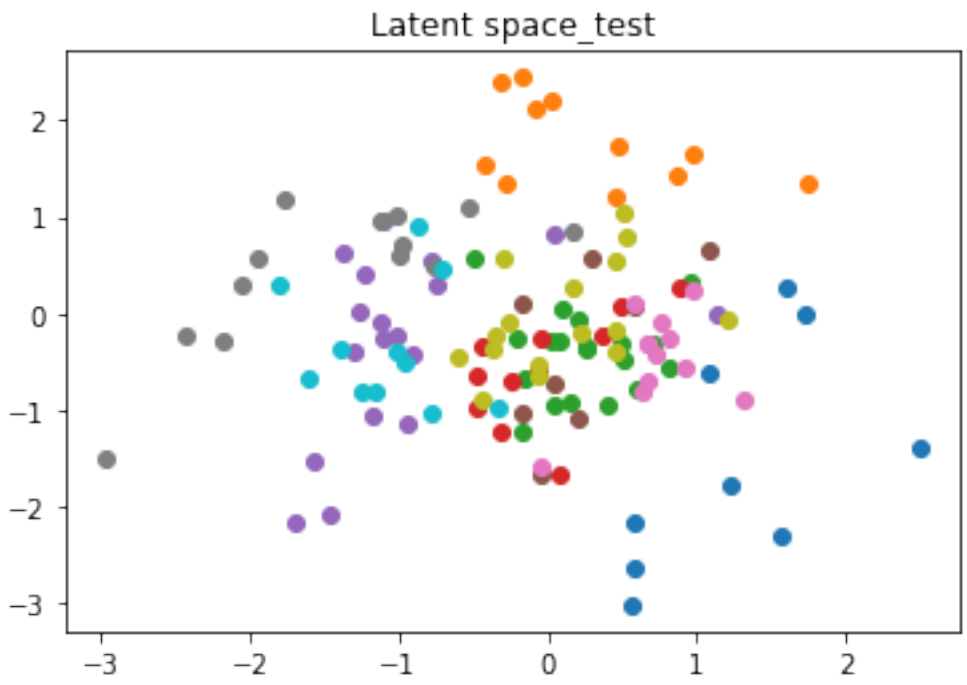



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

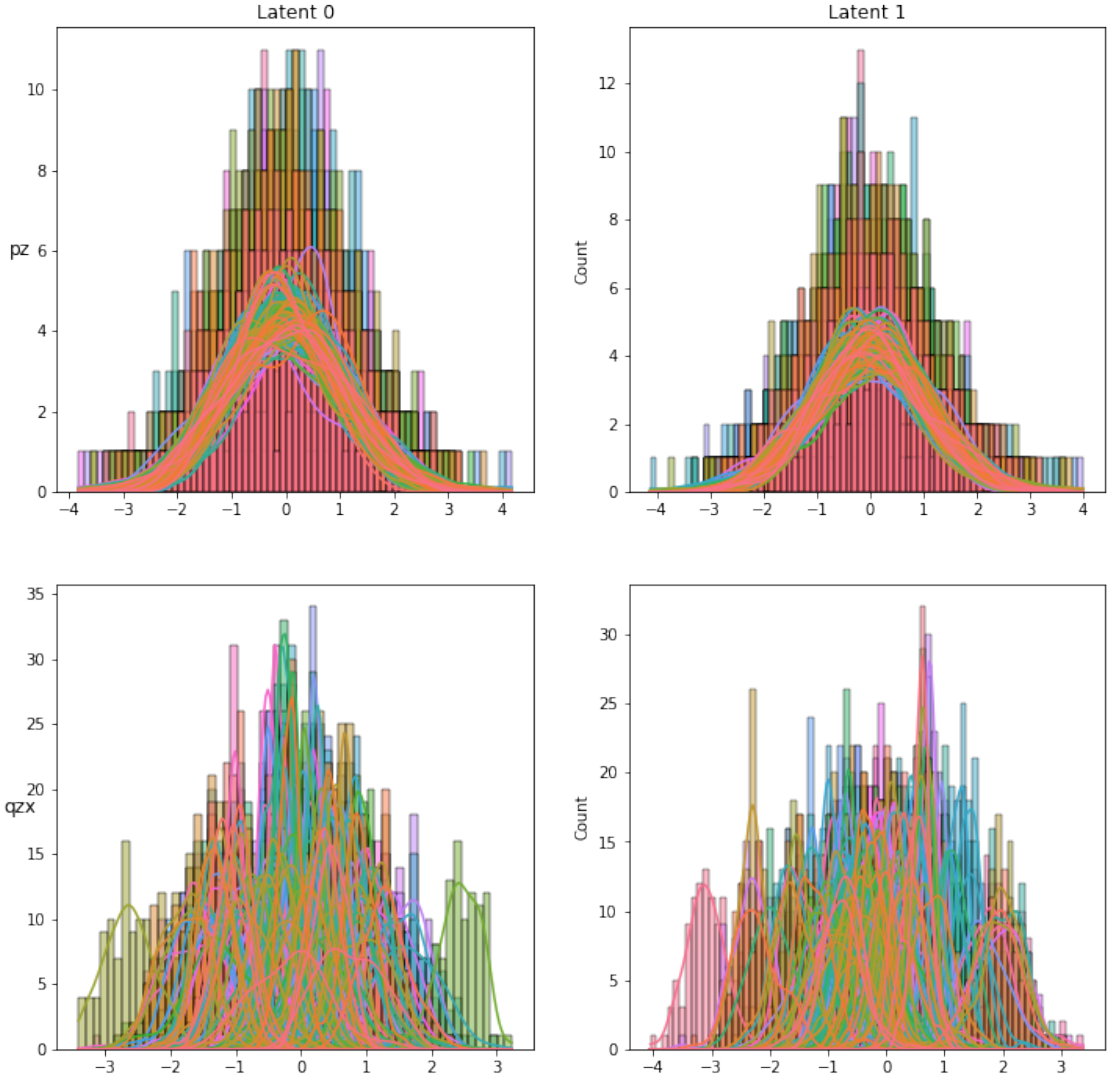

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
qzx
Count
Count

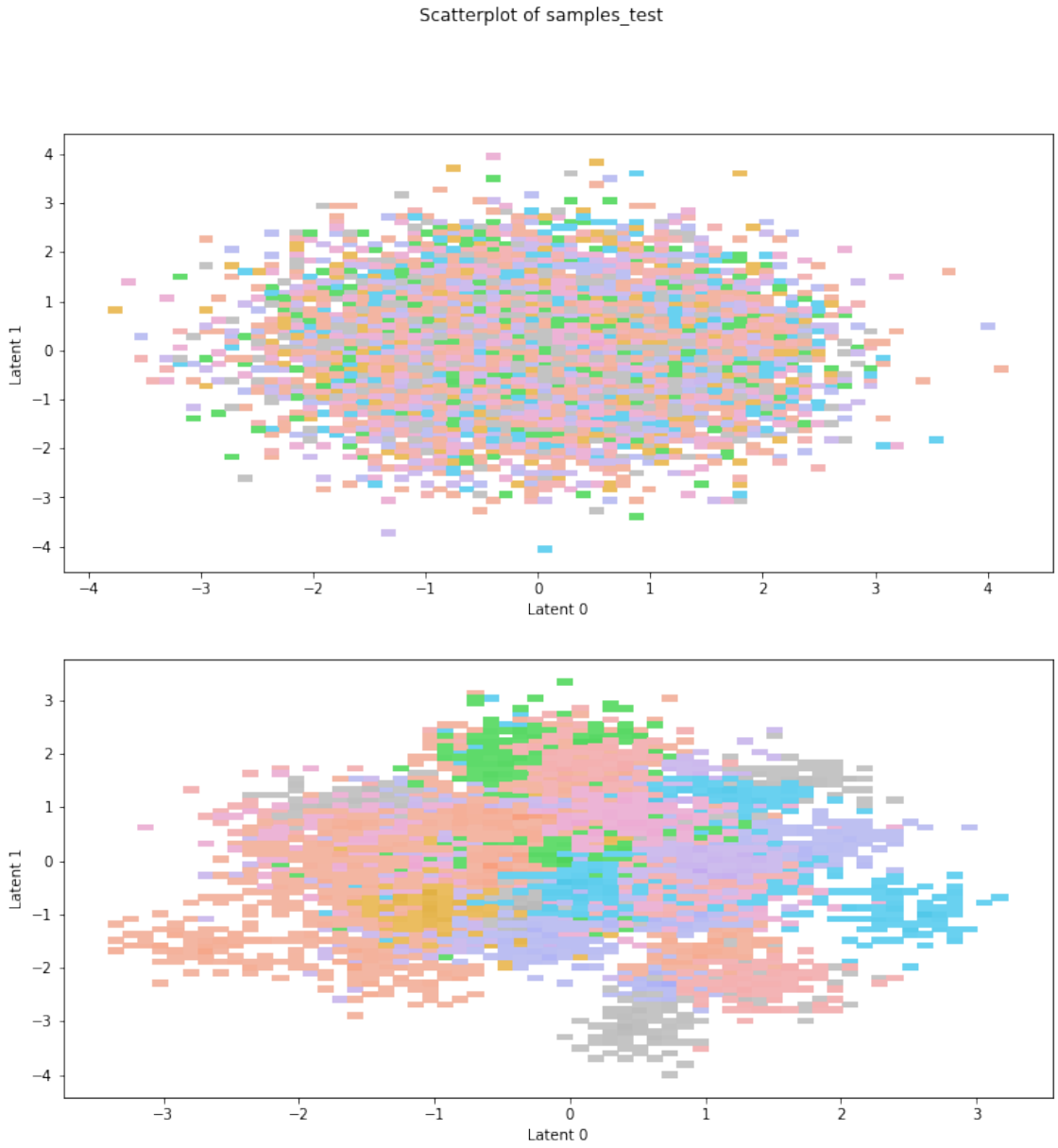



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

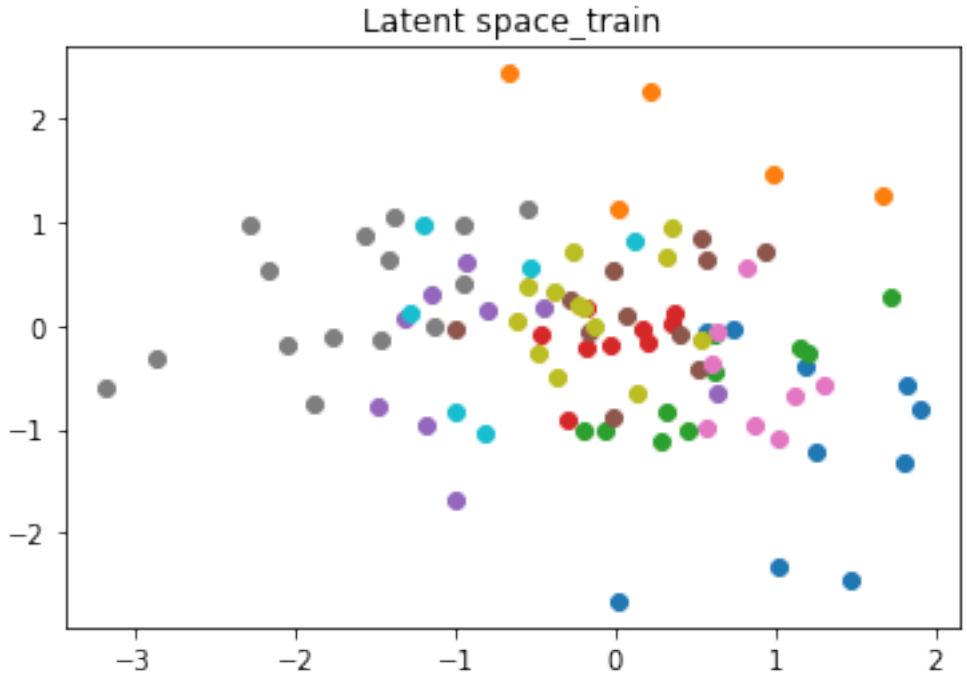

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count



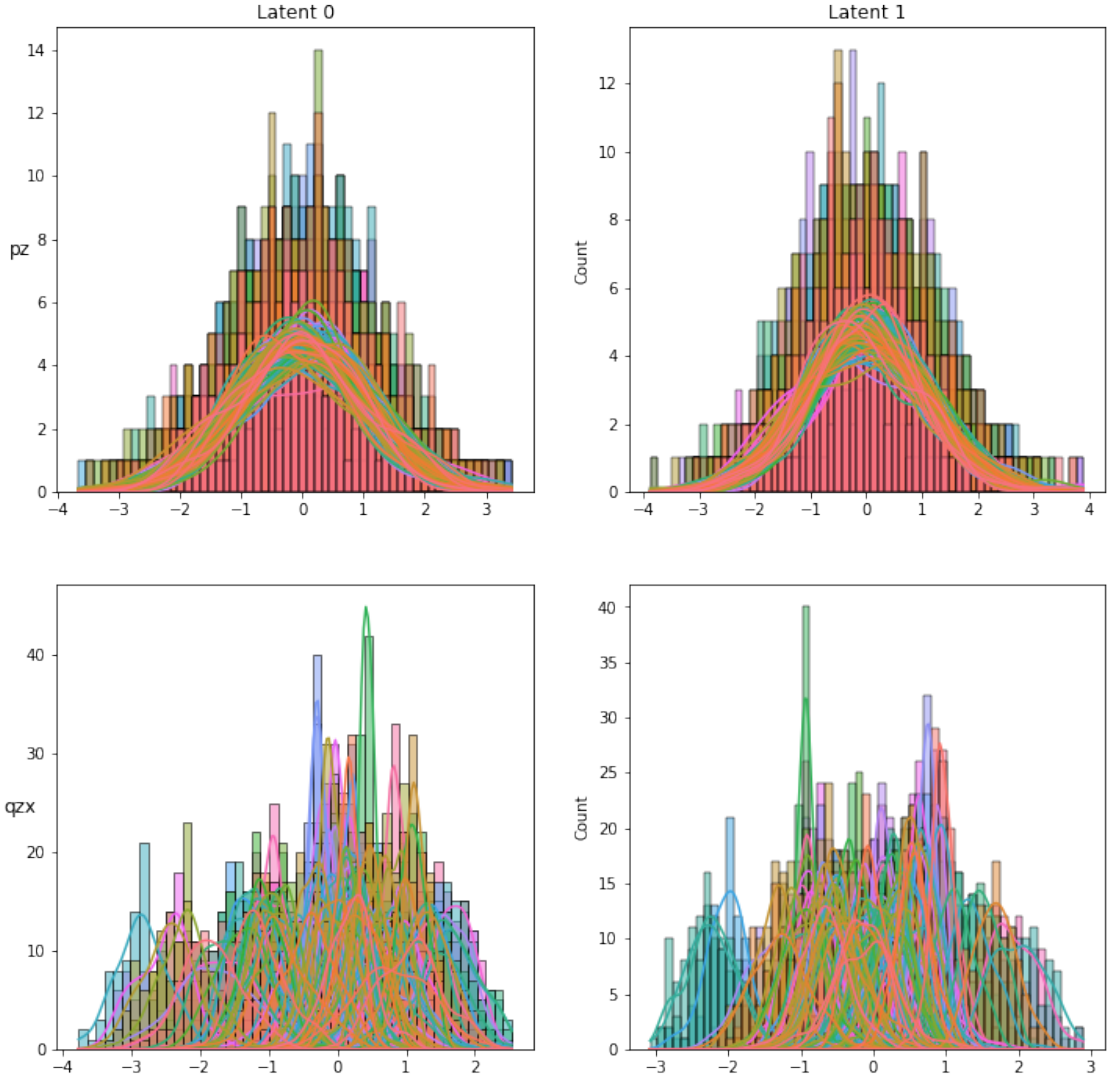

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

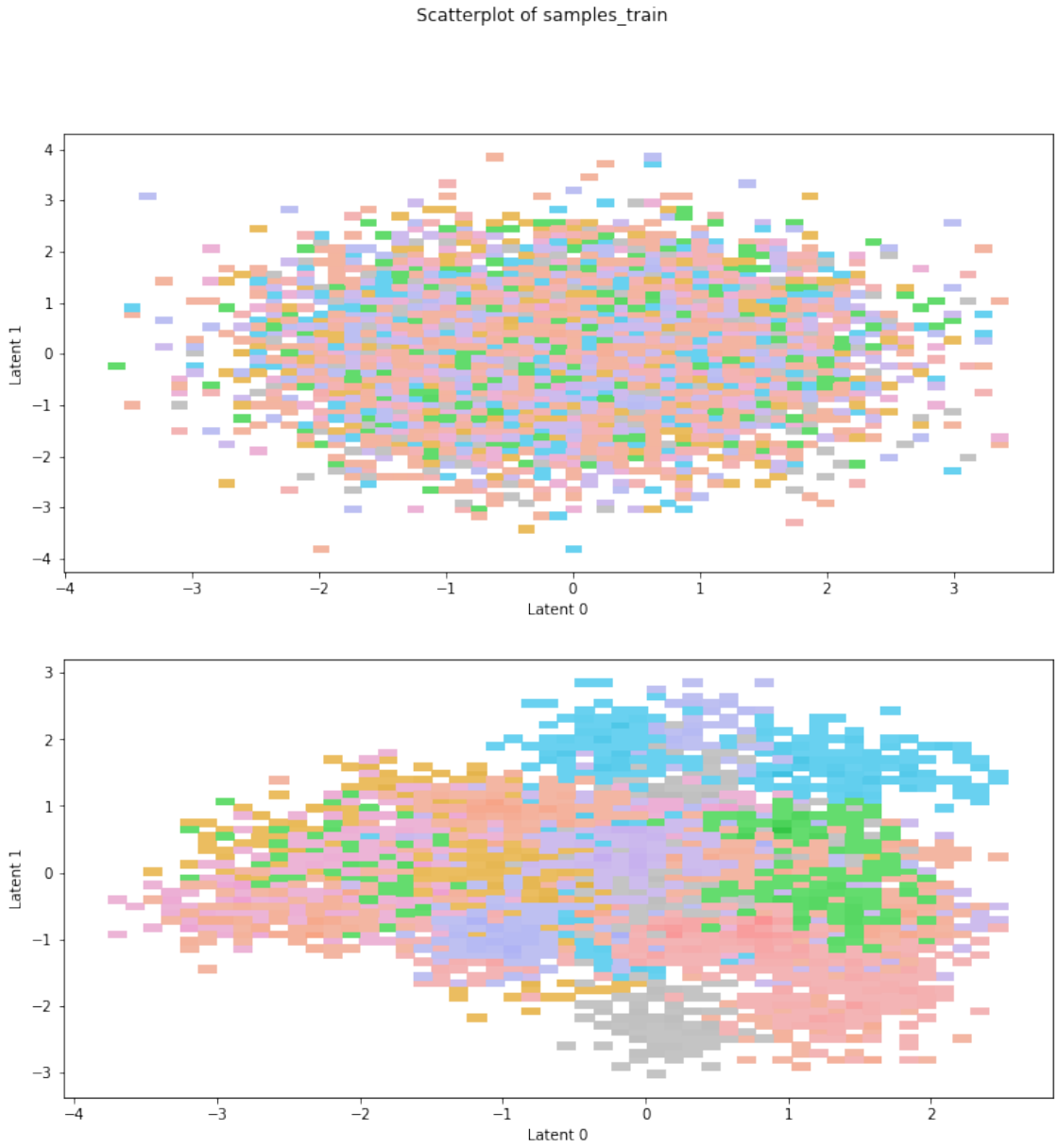
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 26, Loss 23880.3052, kl_loss 322.6754, recon_loss 20653.5512,
kl_divergence 279.9738
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

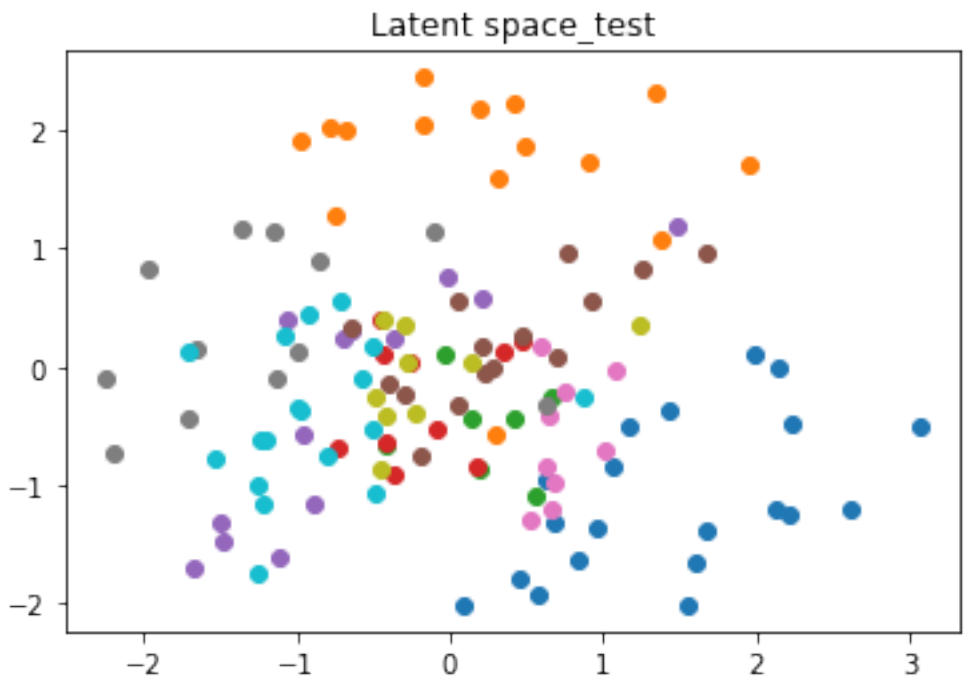



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

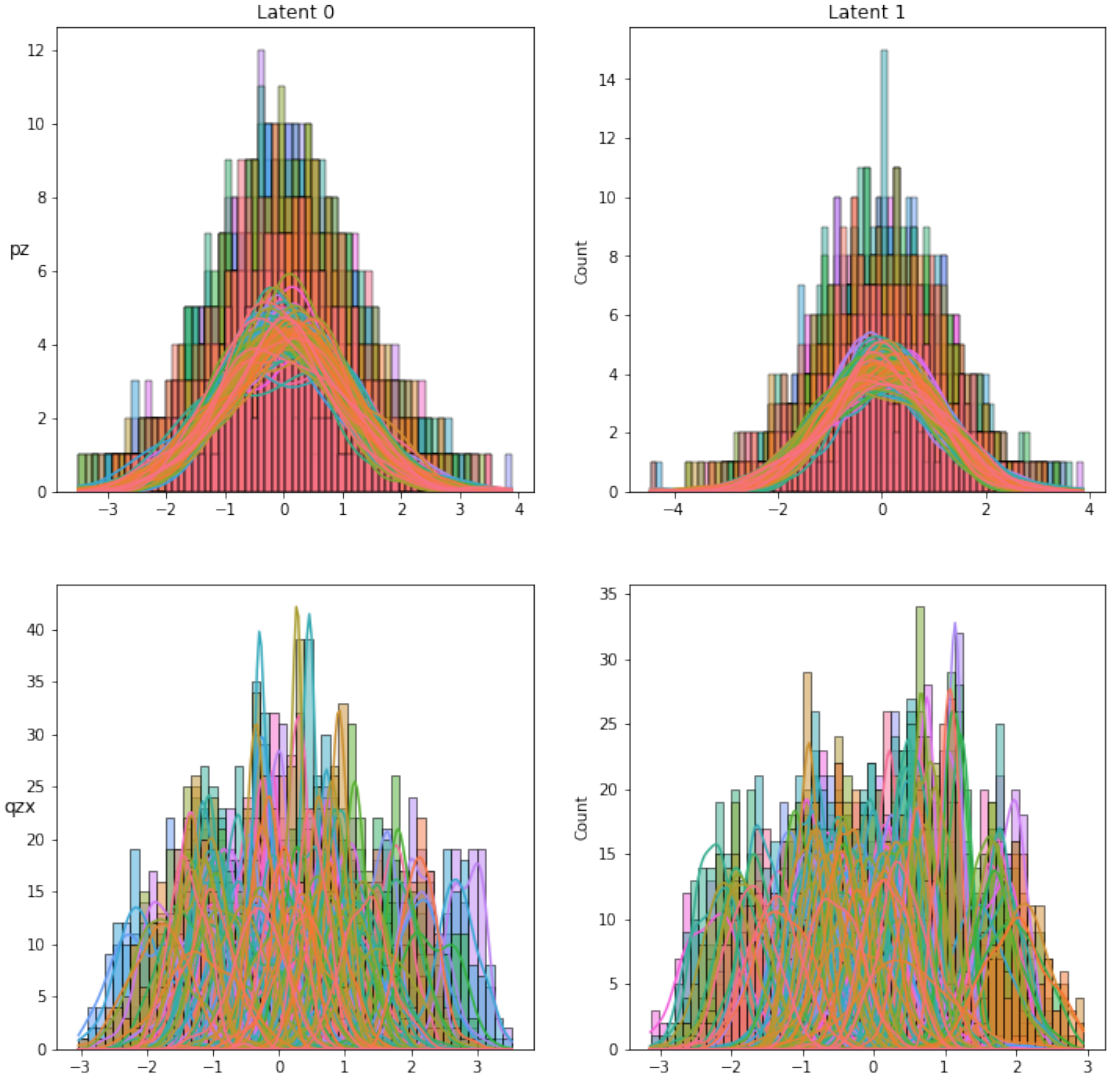

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
qzx
Count
Count

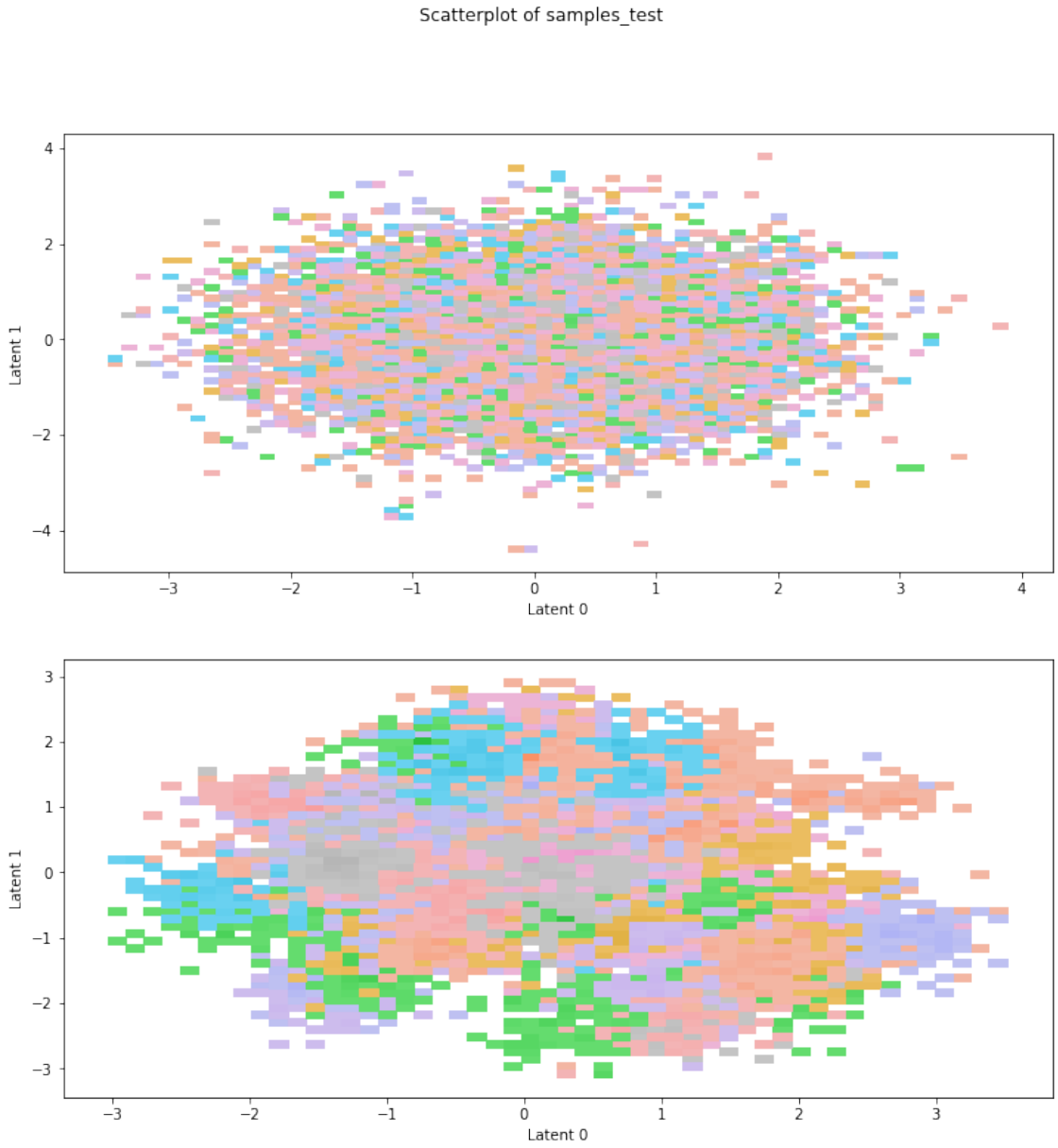



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

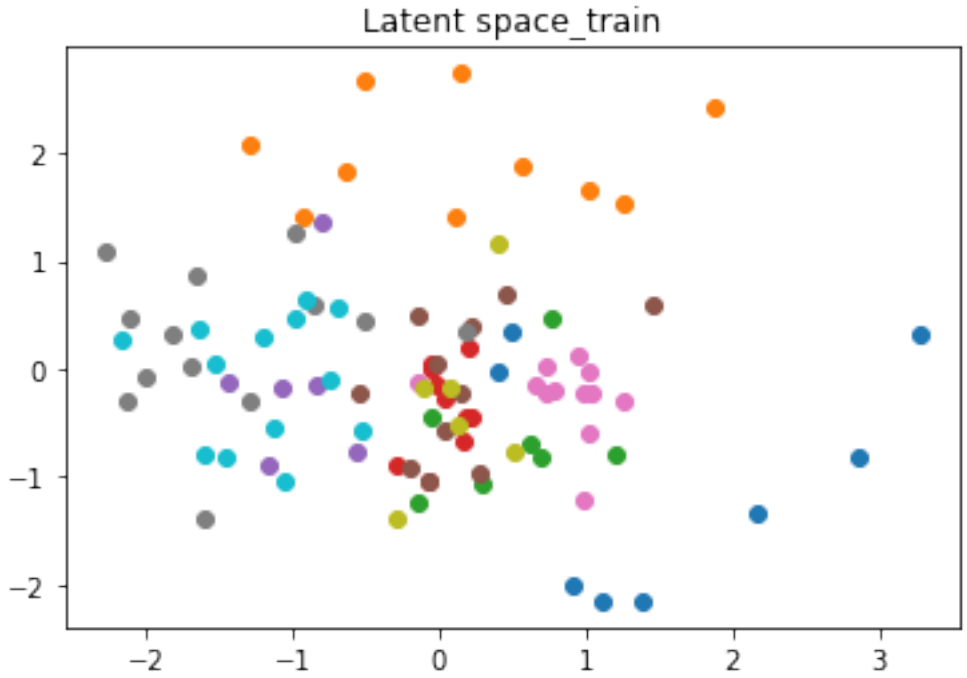

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count



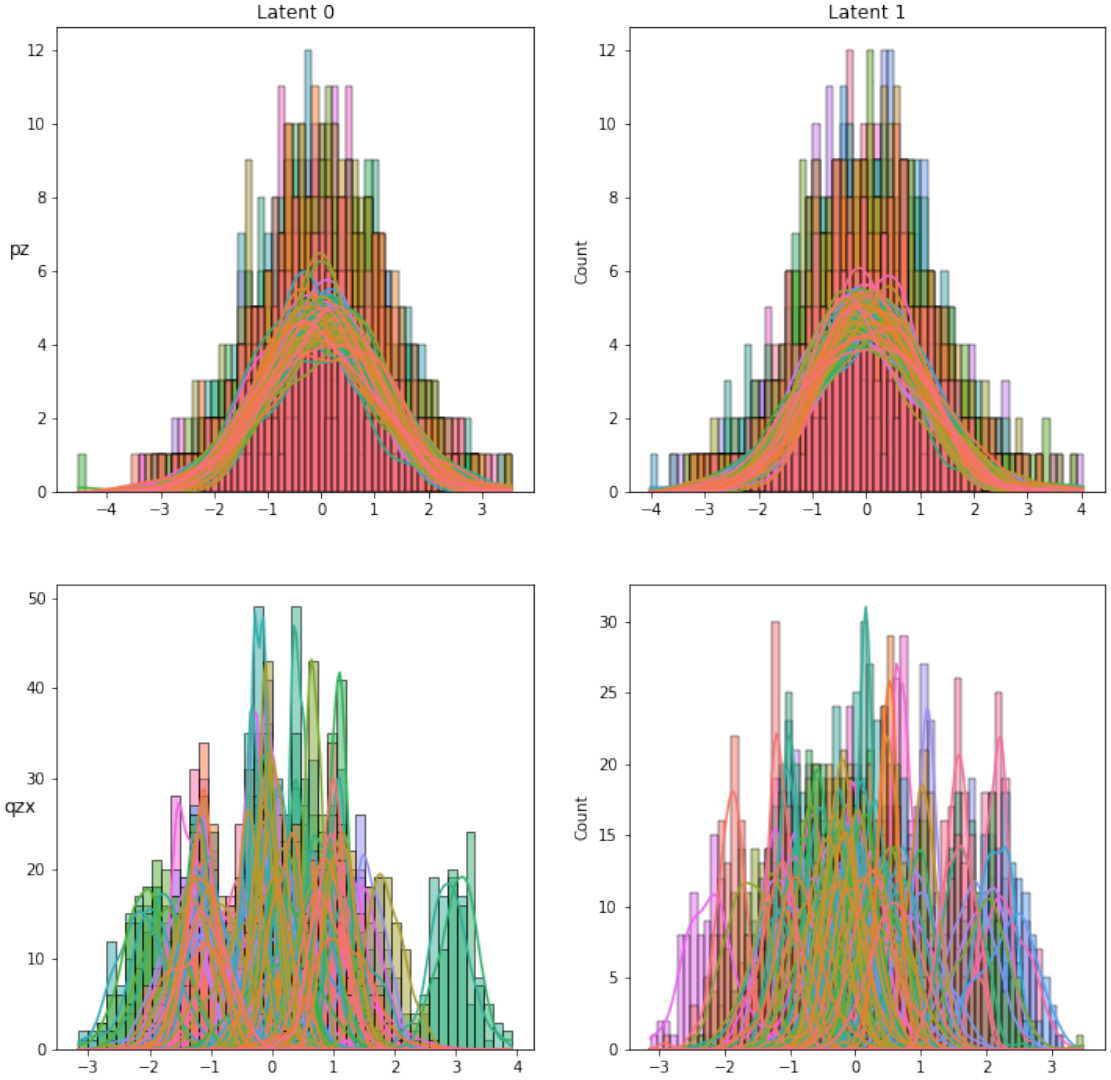

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

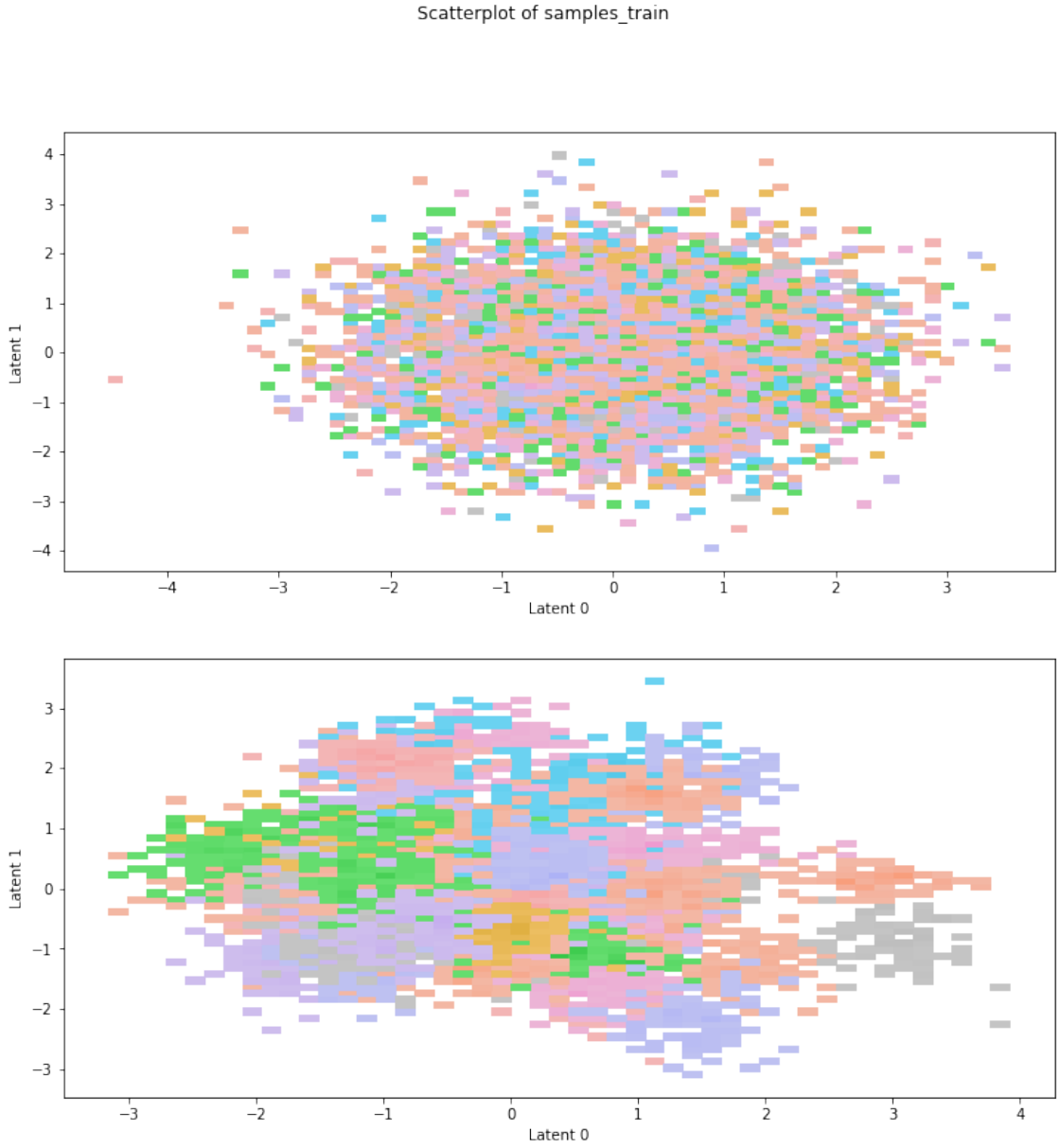
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 31, Loss 23849.2130, kl_loss 328.0221, recon_loss 20568.9921,
kl_divergence 281.1865
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

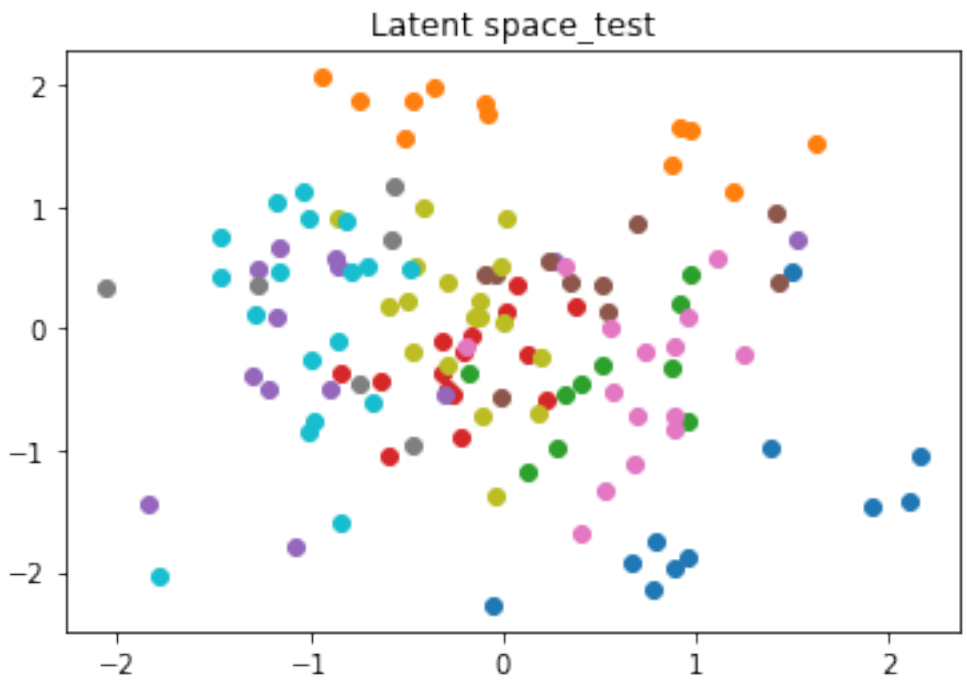



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

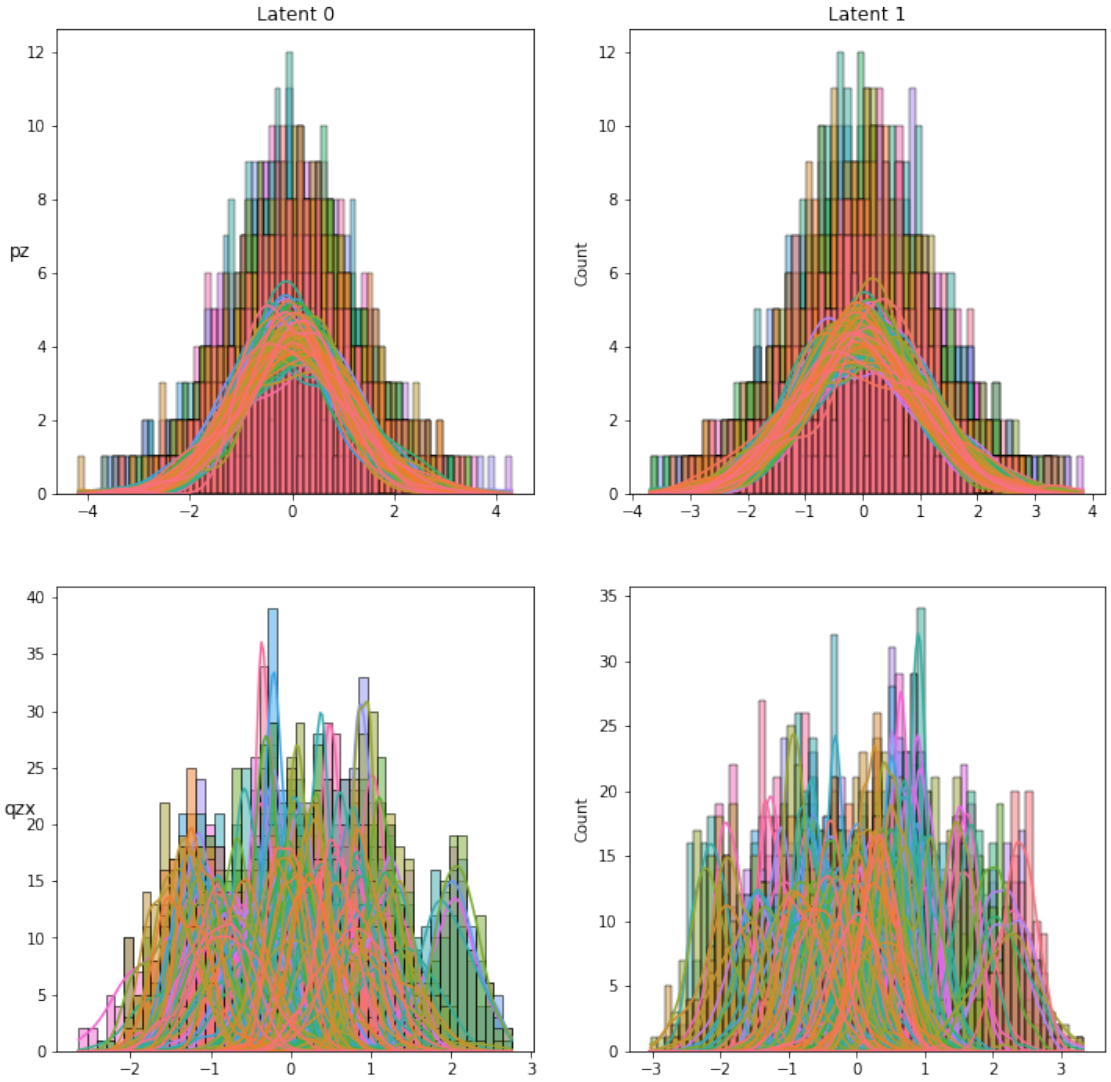

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

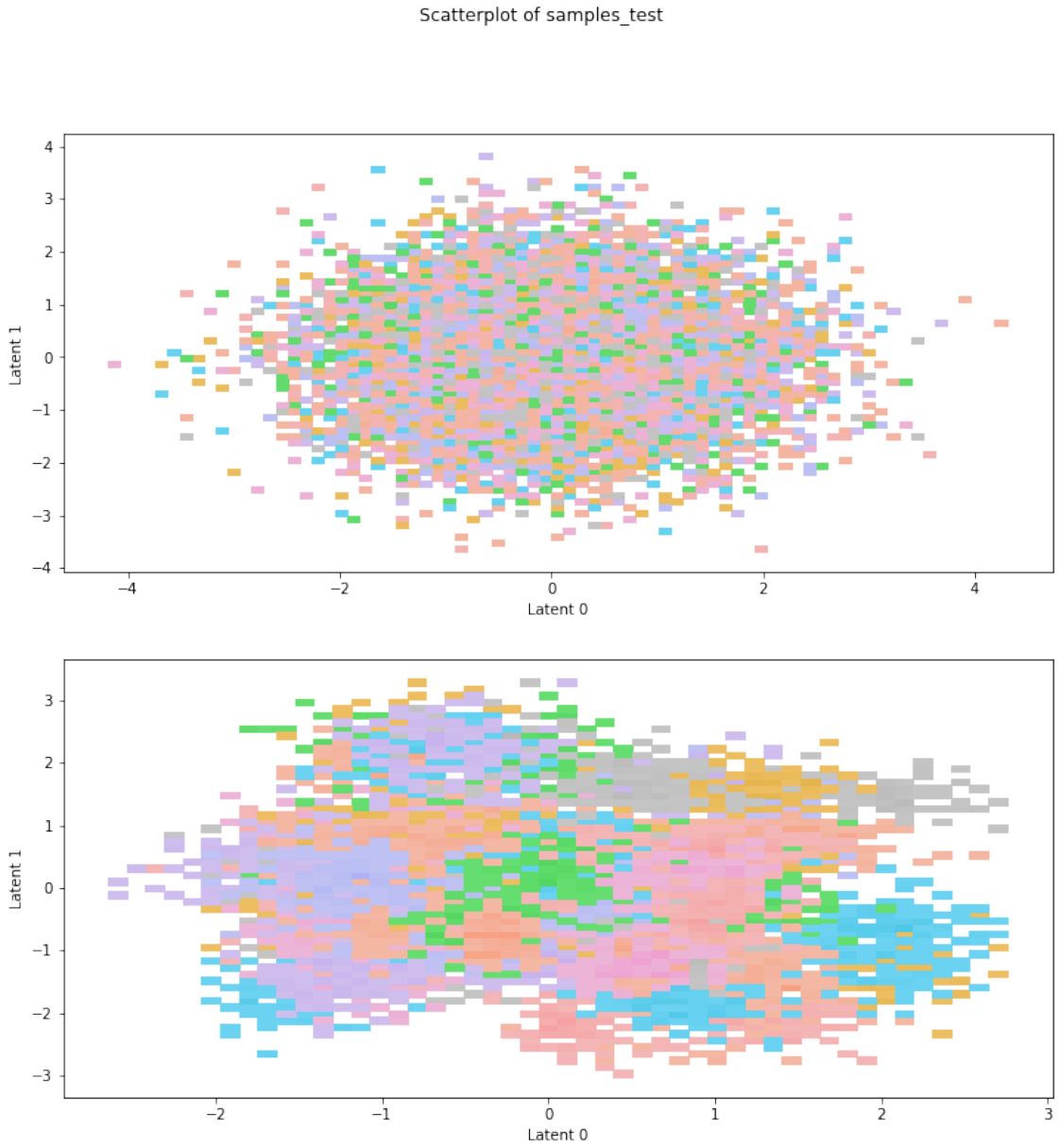



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

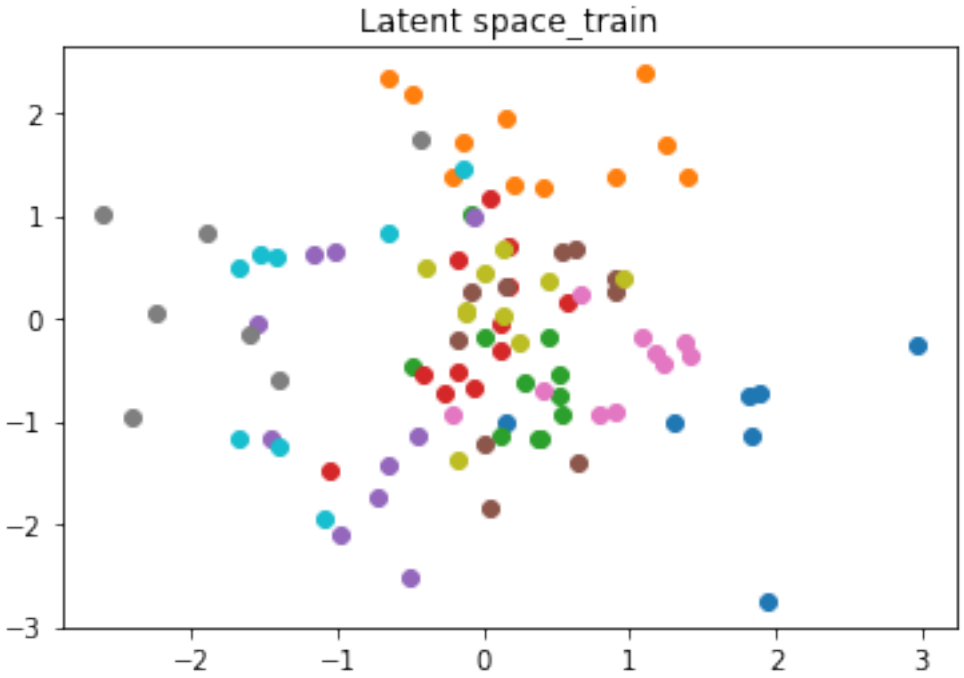

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
qzx
Count
Count



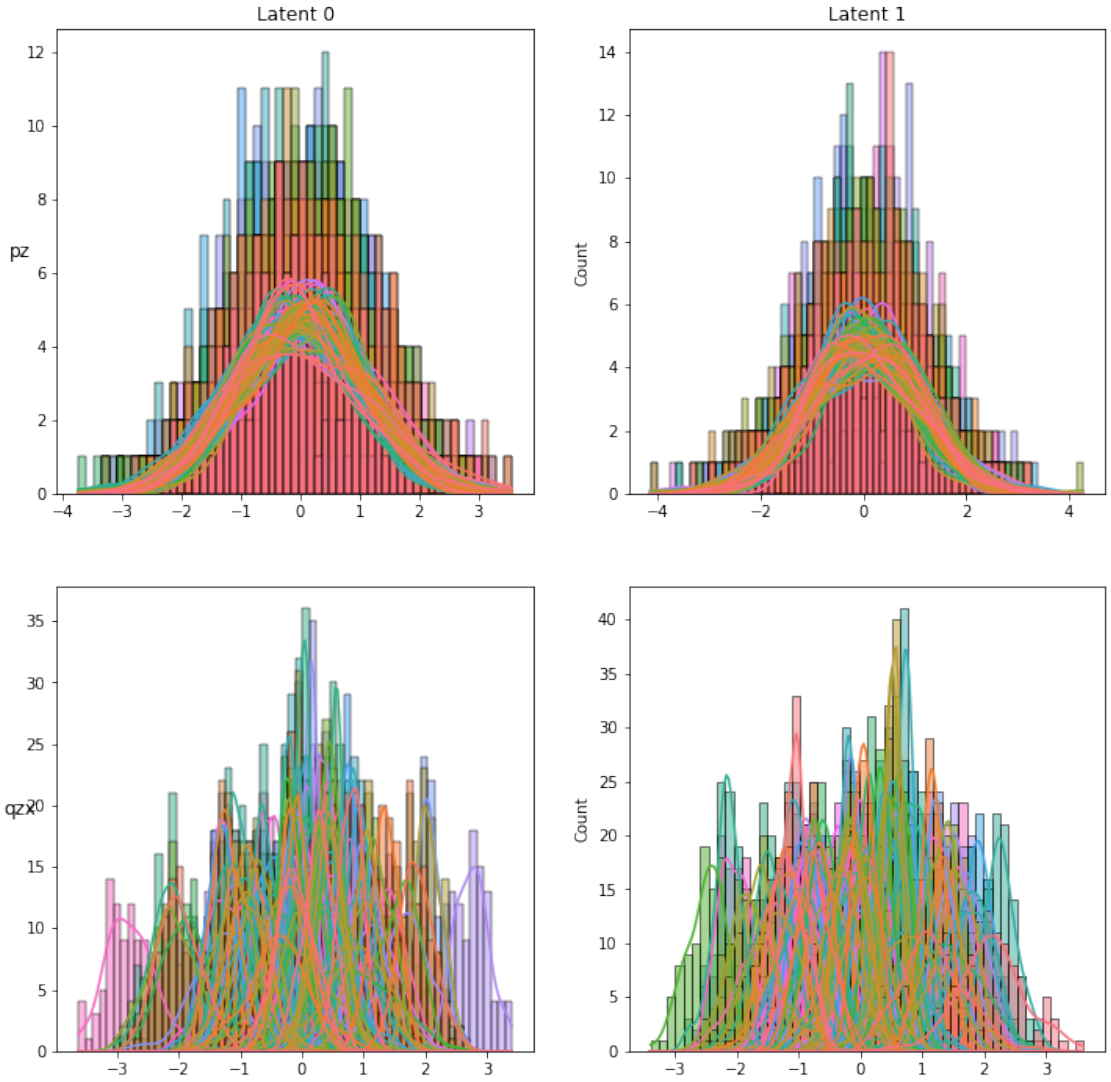

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

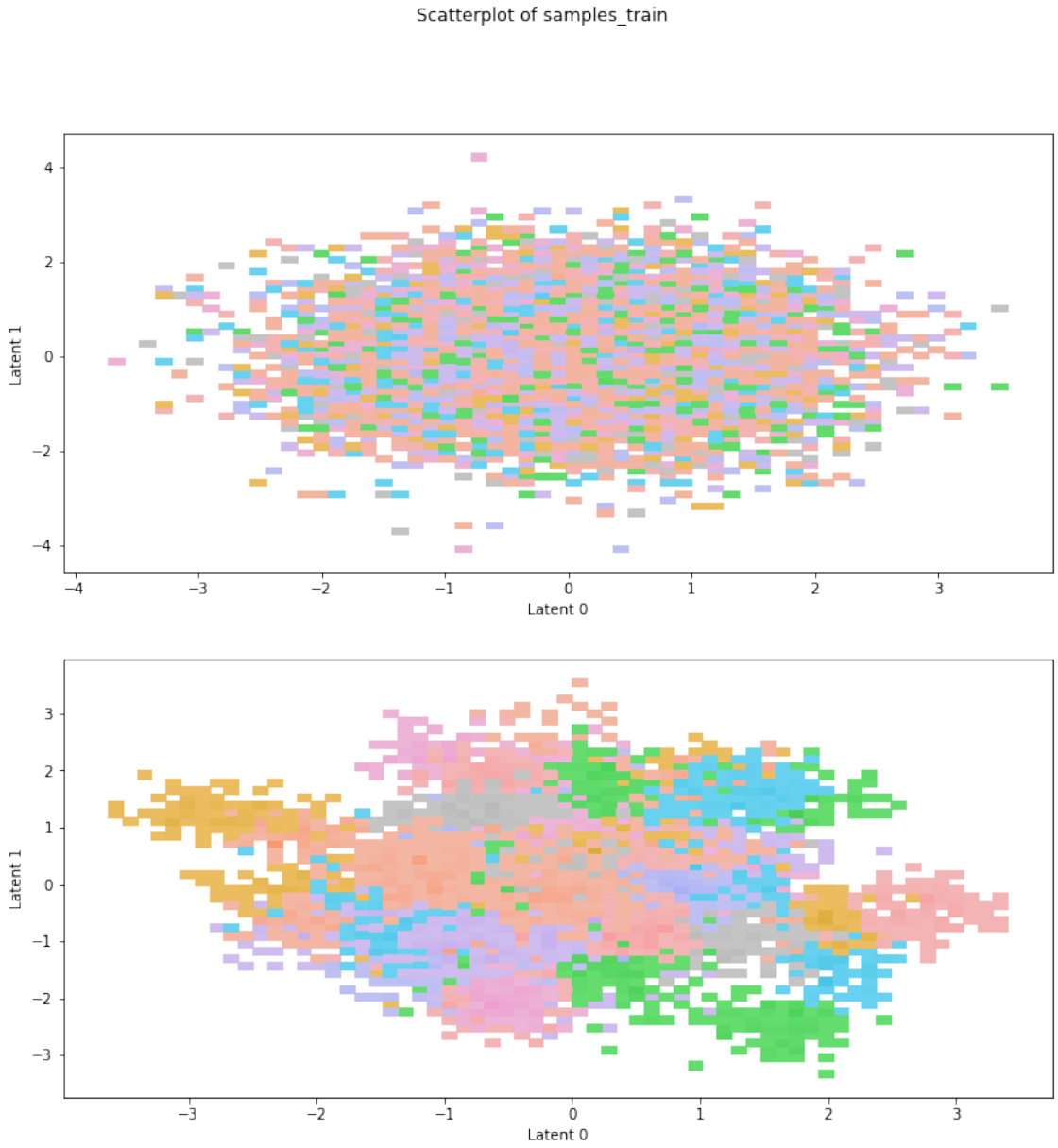
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 36, Loss 23818.8979, kl_loss 333.5830, recon_loss 20483.0683,
kl_divergence 284.0076
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

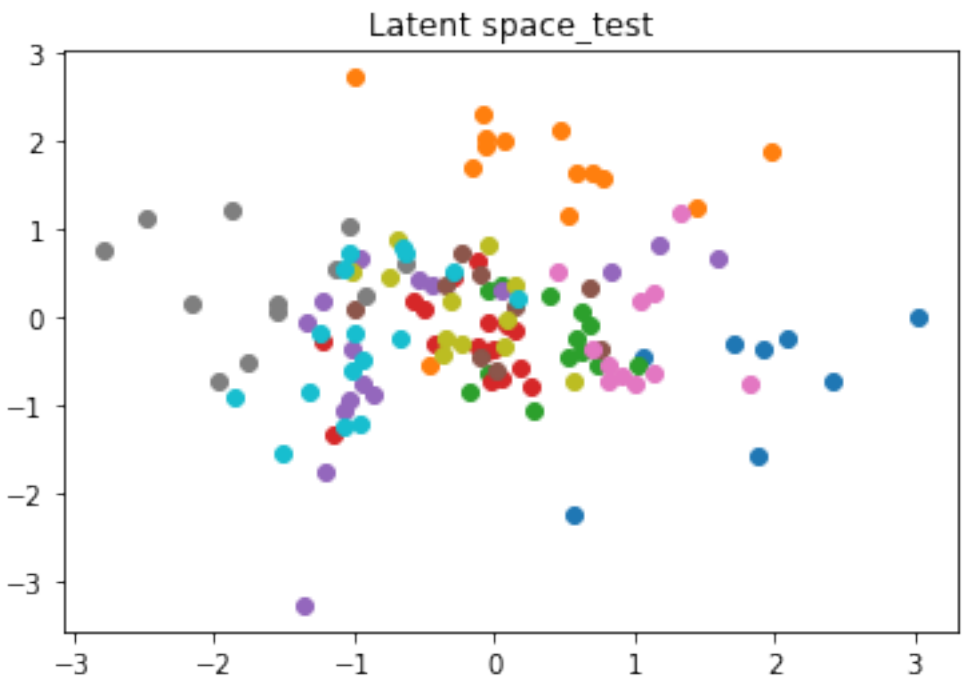



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

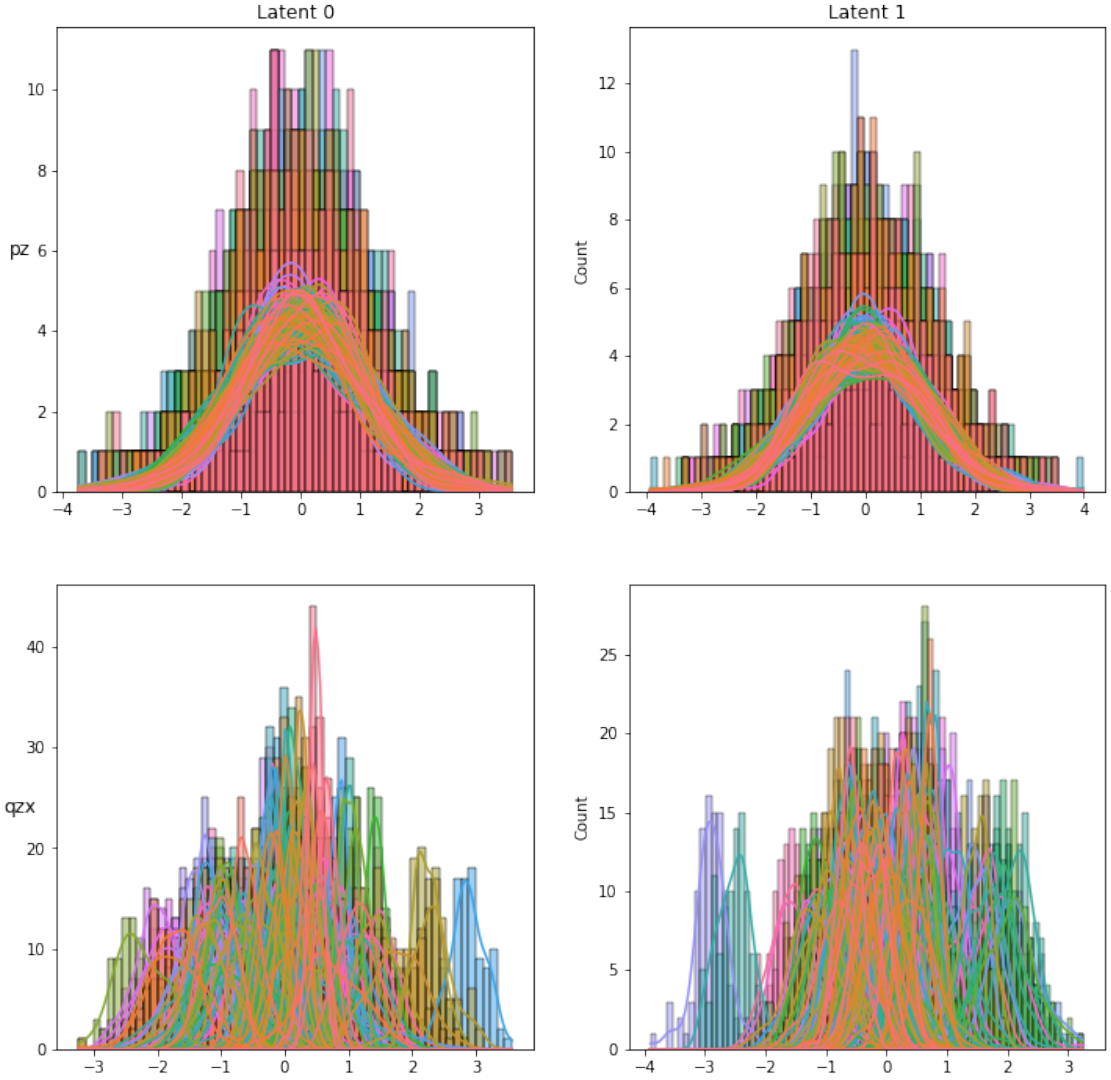

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

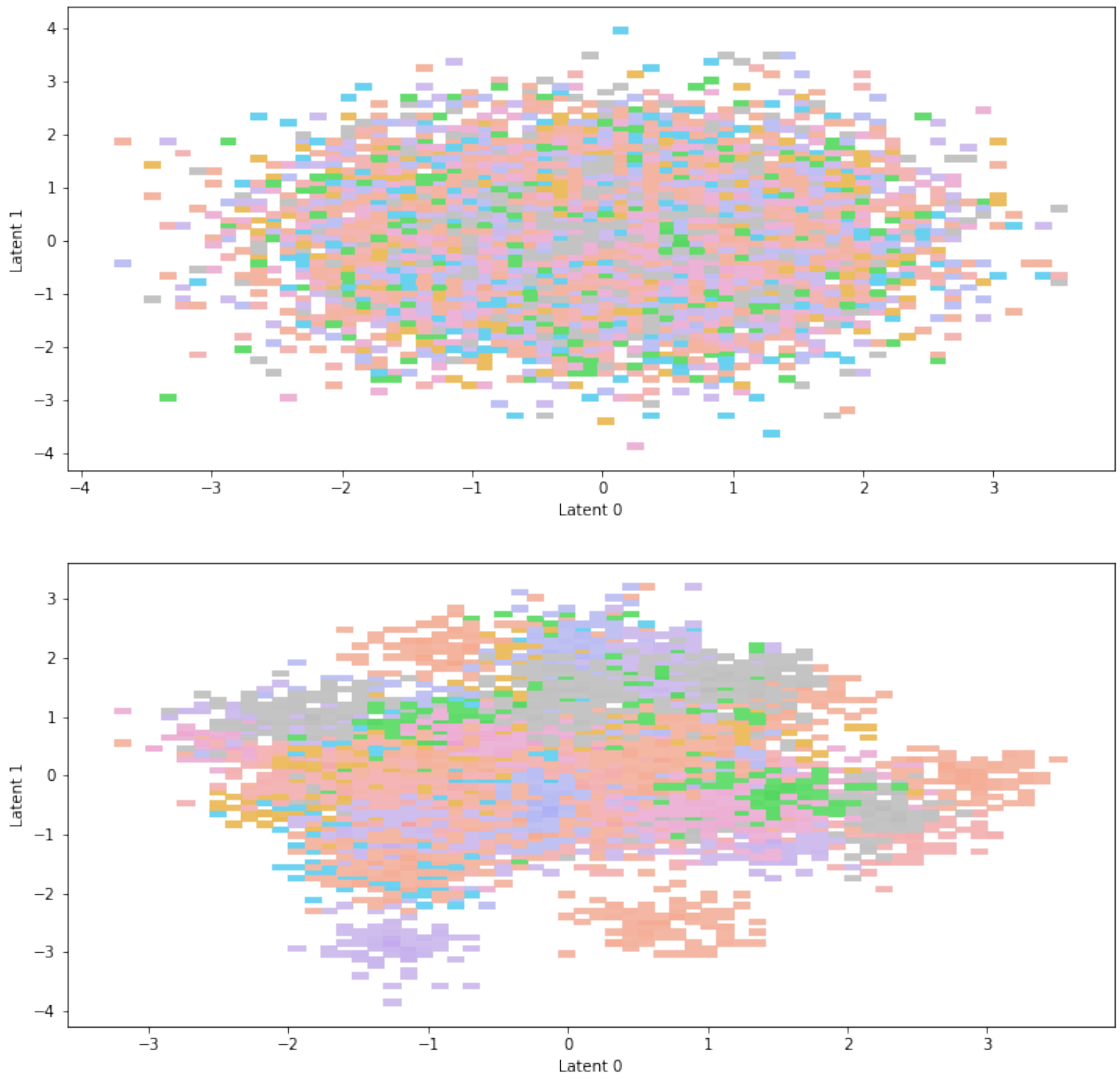



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

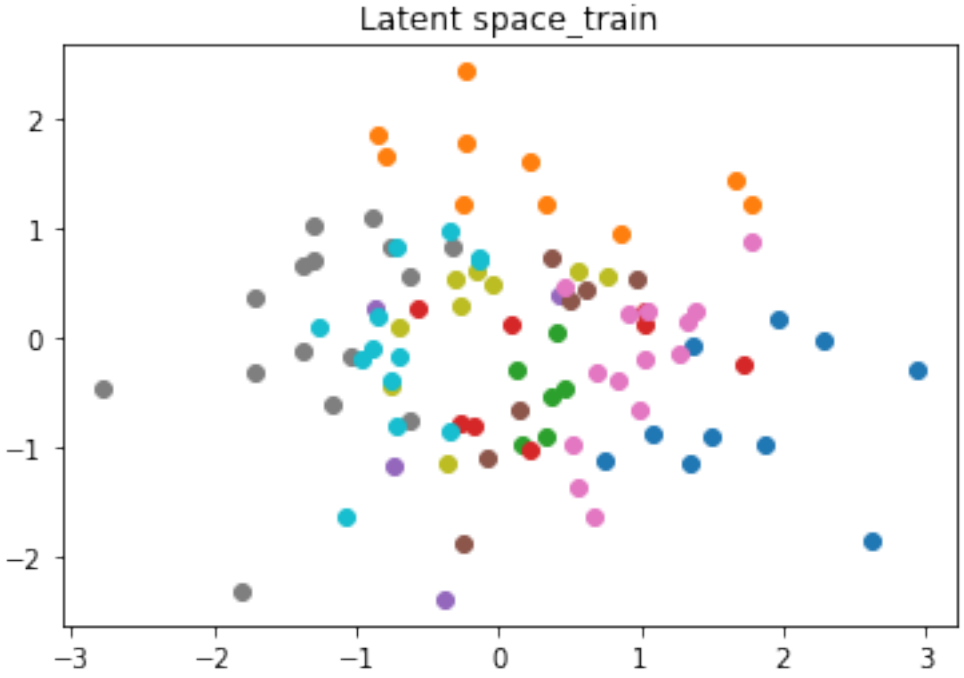

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count



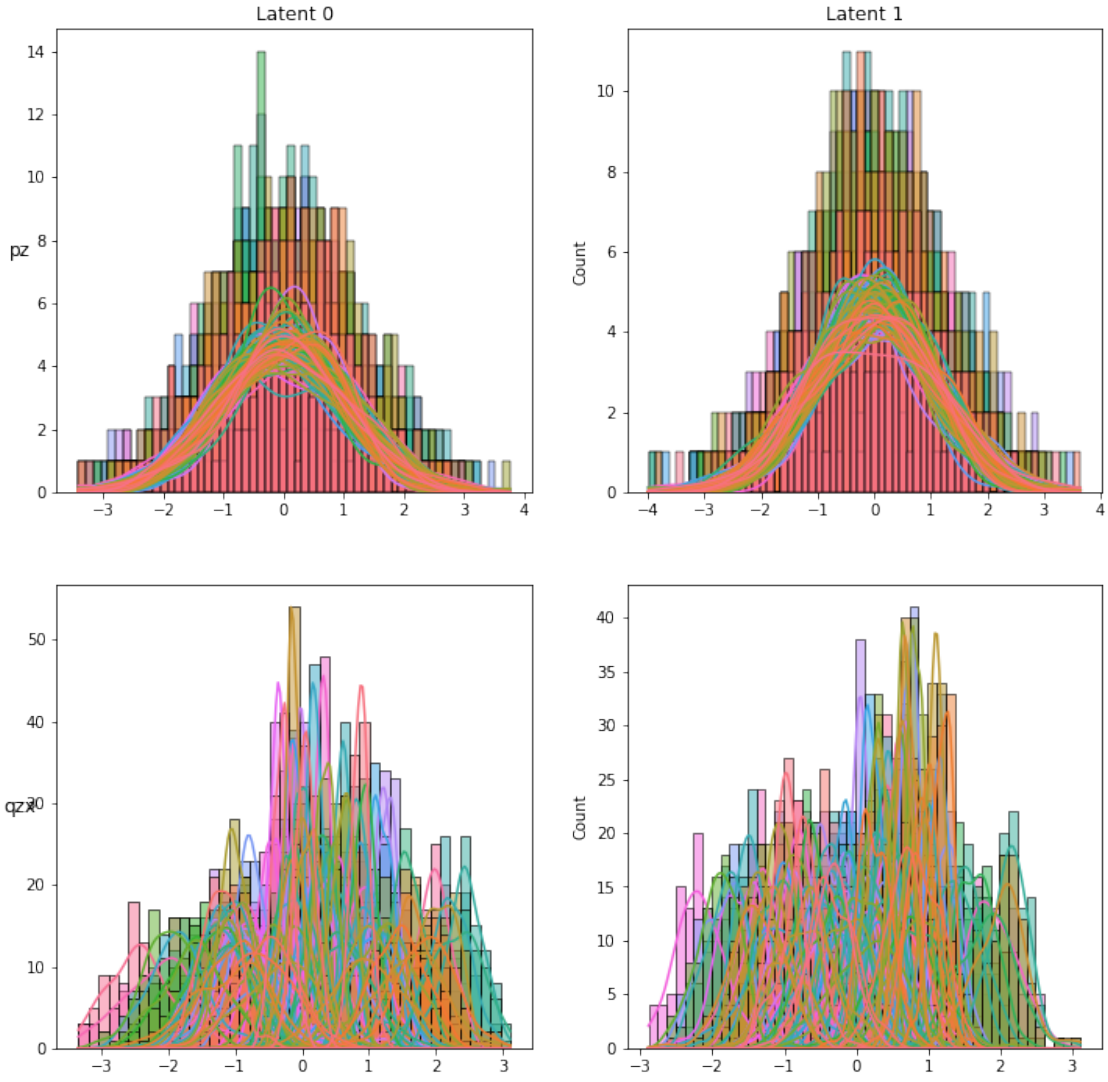

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

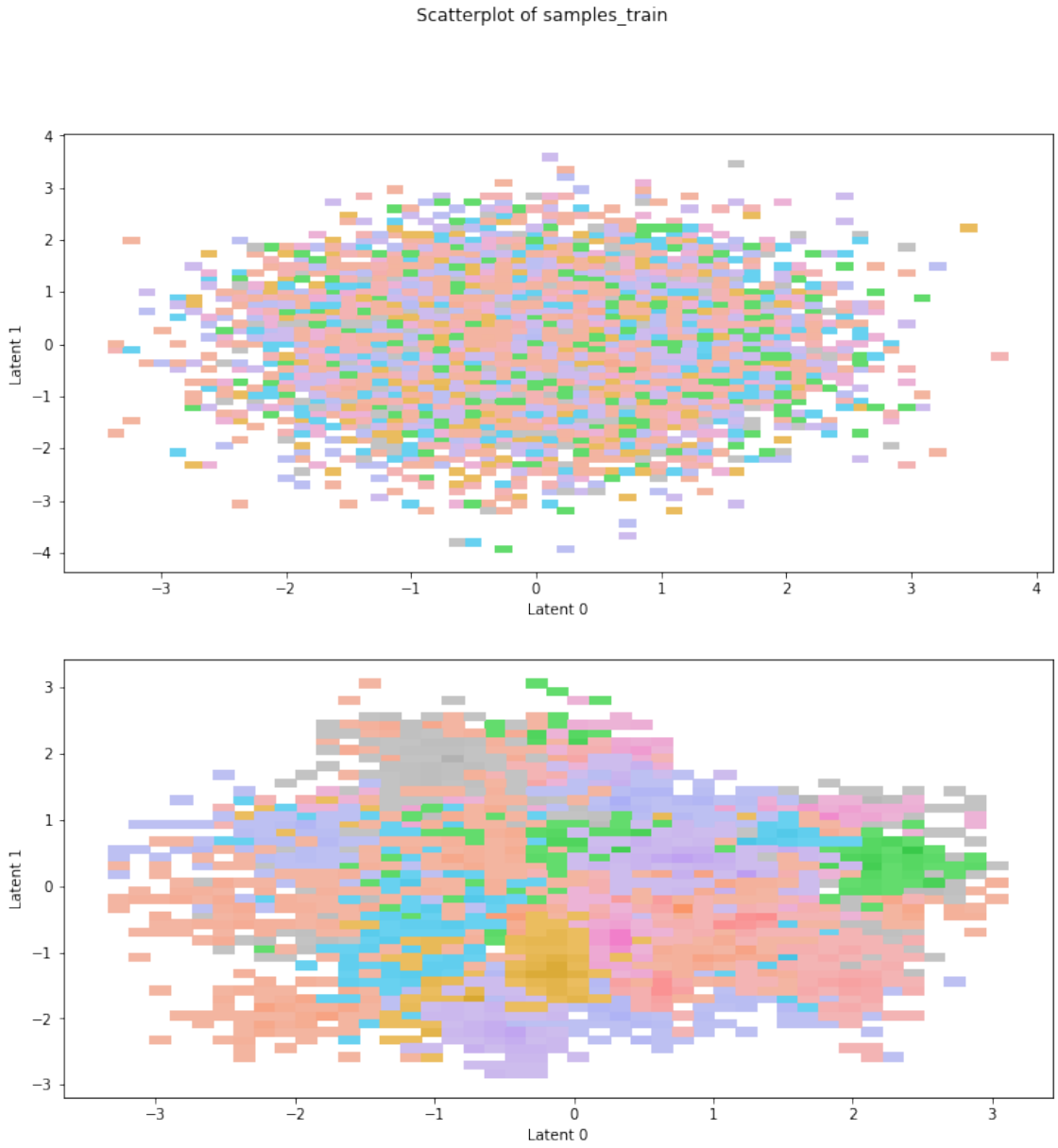
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 41, Loss 23781.7332, kl_loss 336.4745, recon_loss 20416.9883,
kl_divergence 286.3459
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

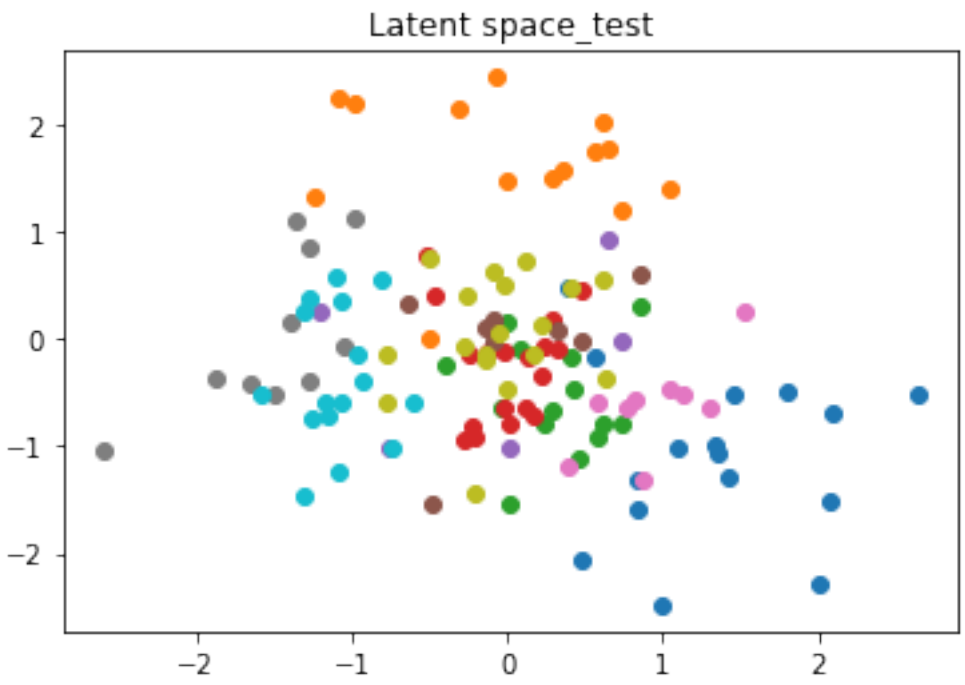



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

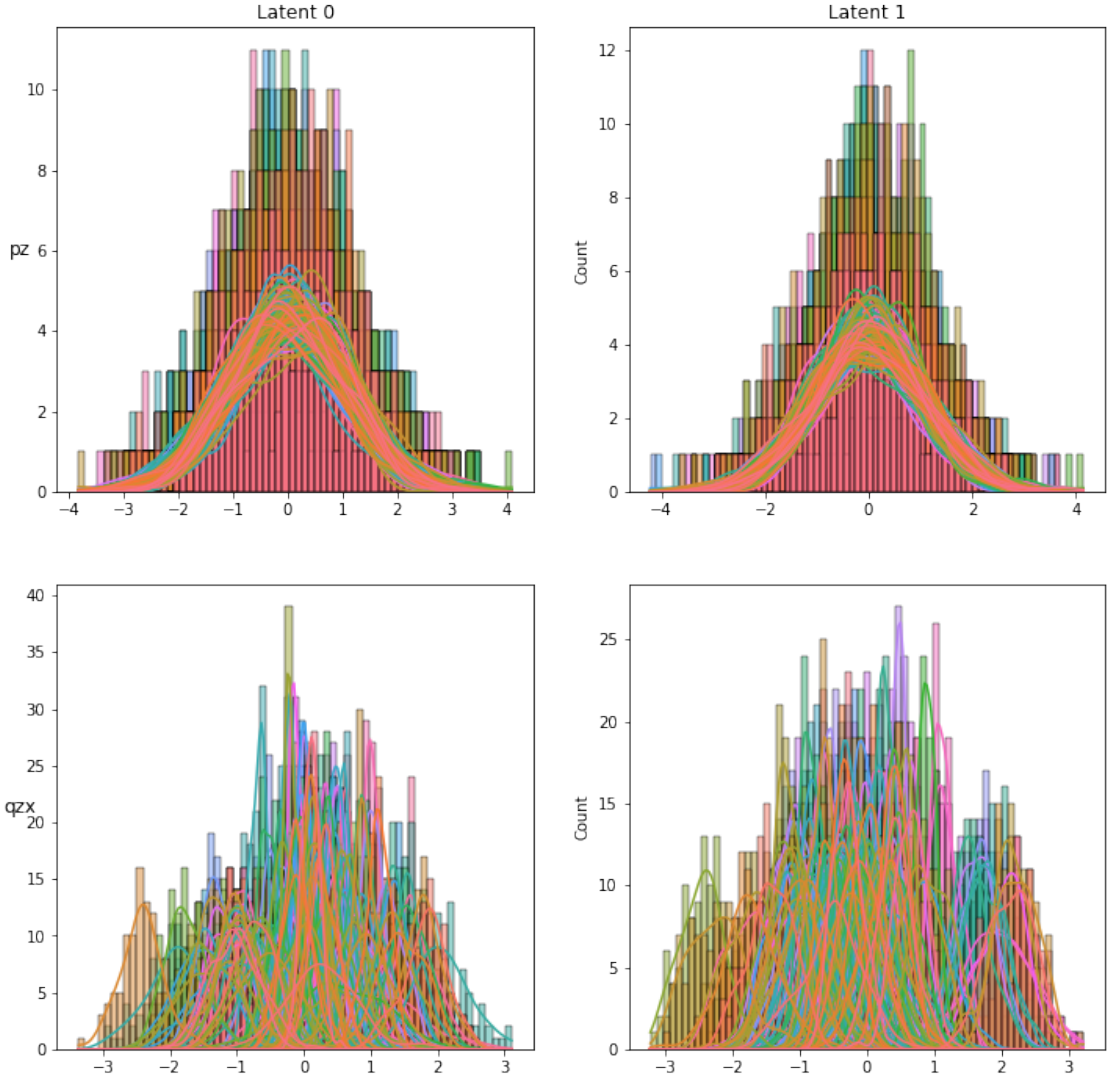

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qz|x
Count

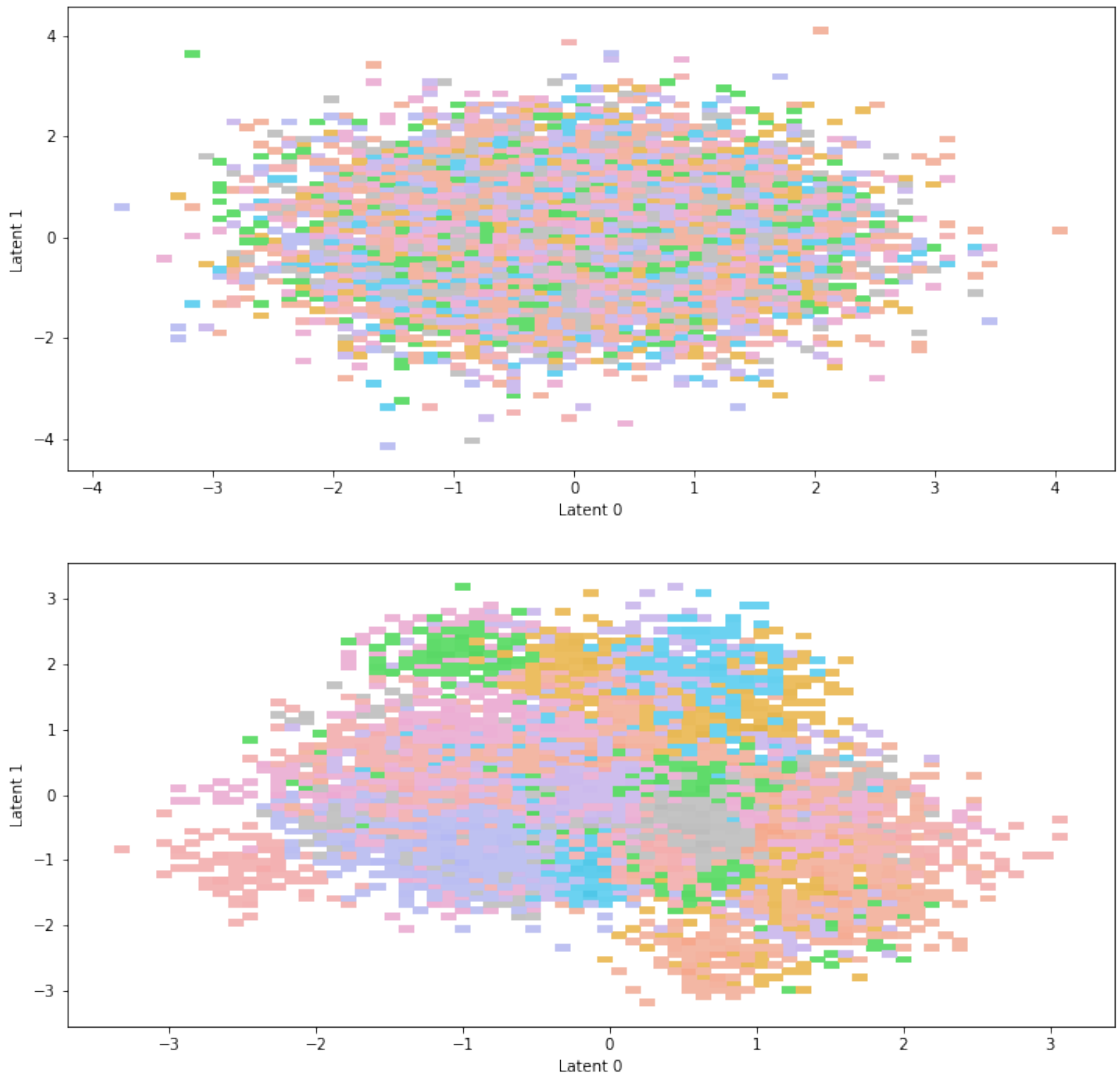



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```

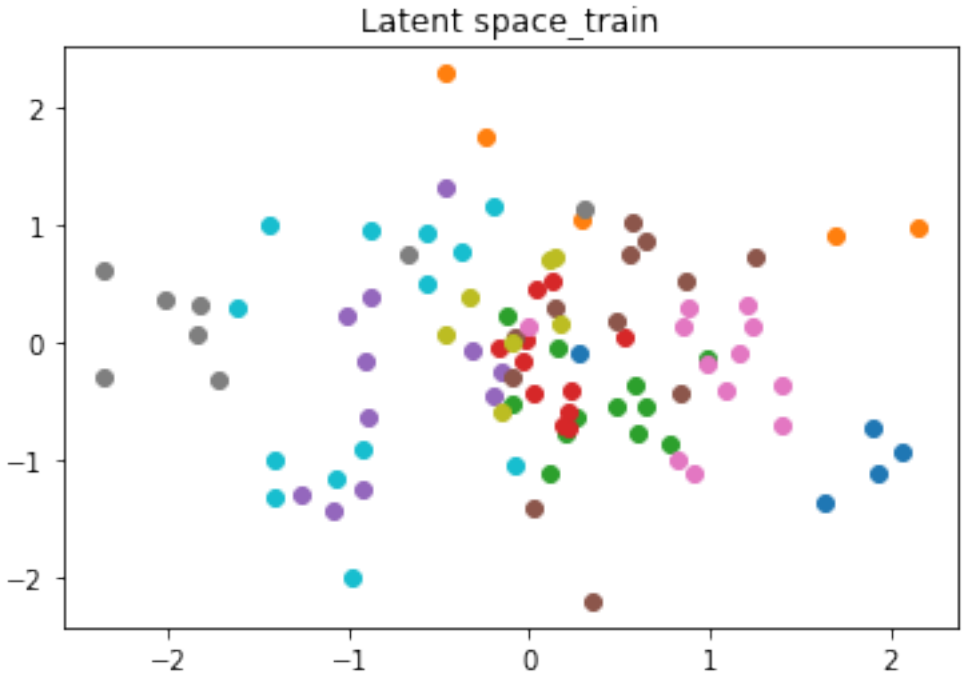

Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count



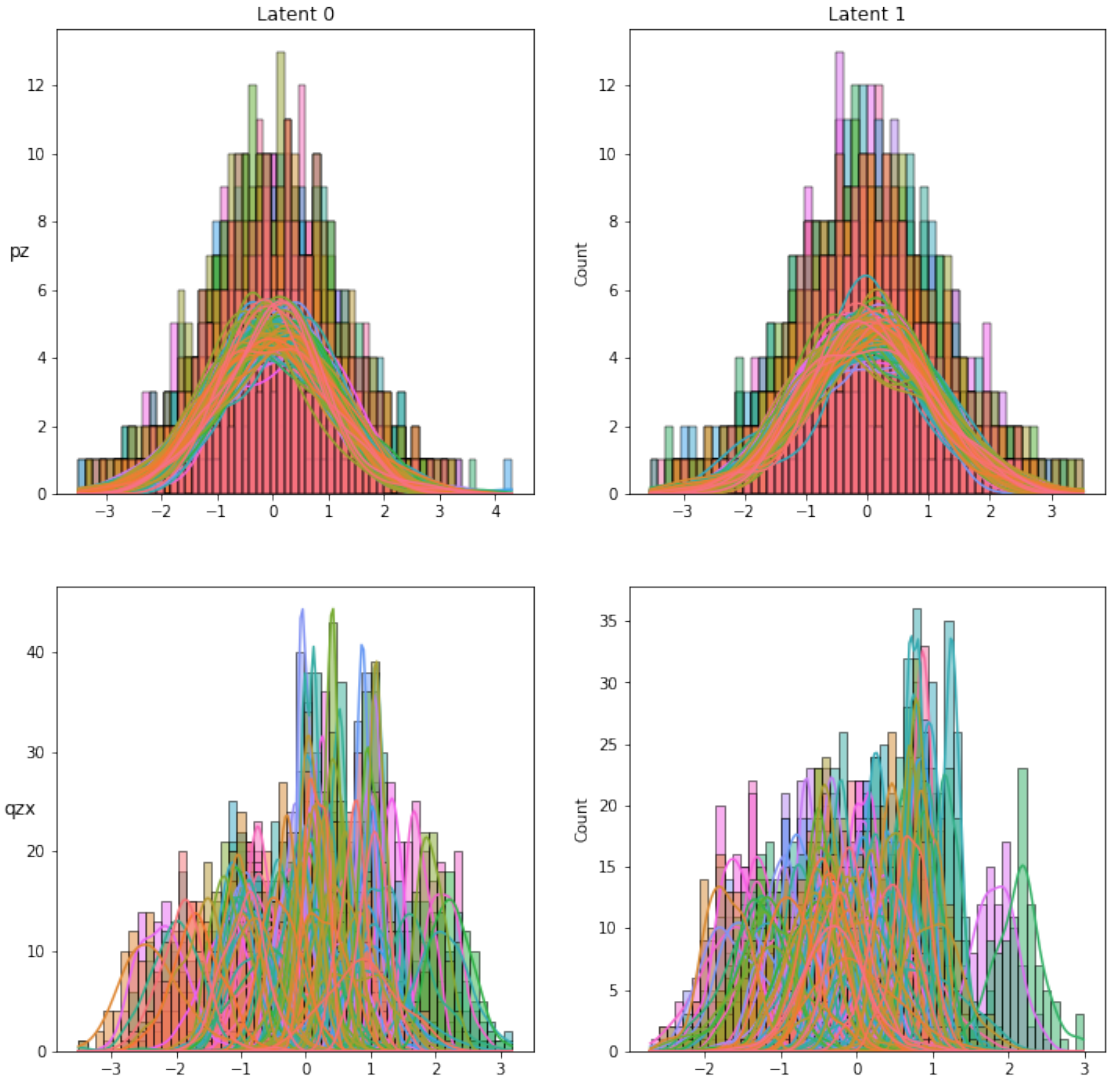

```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

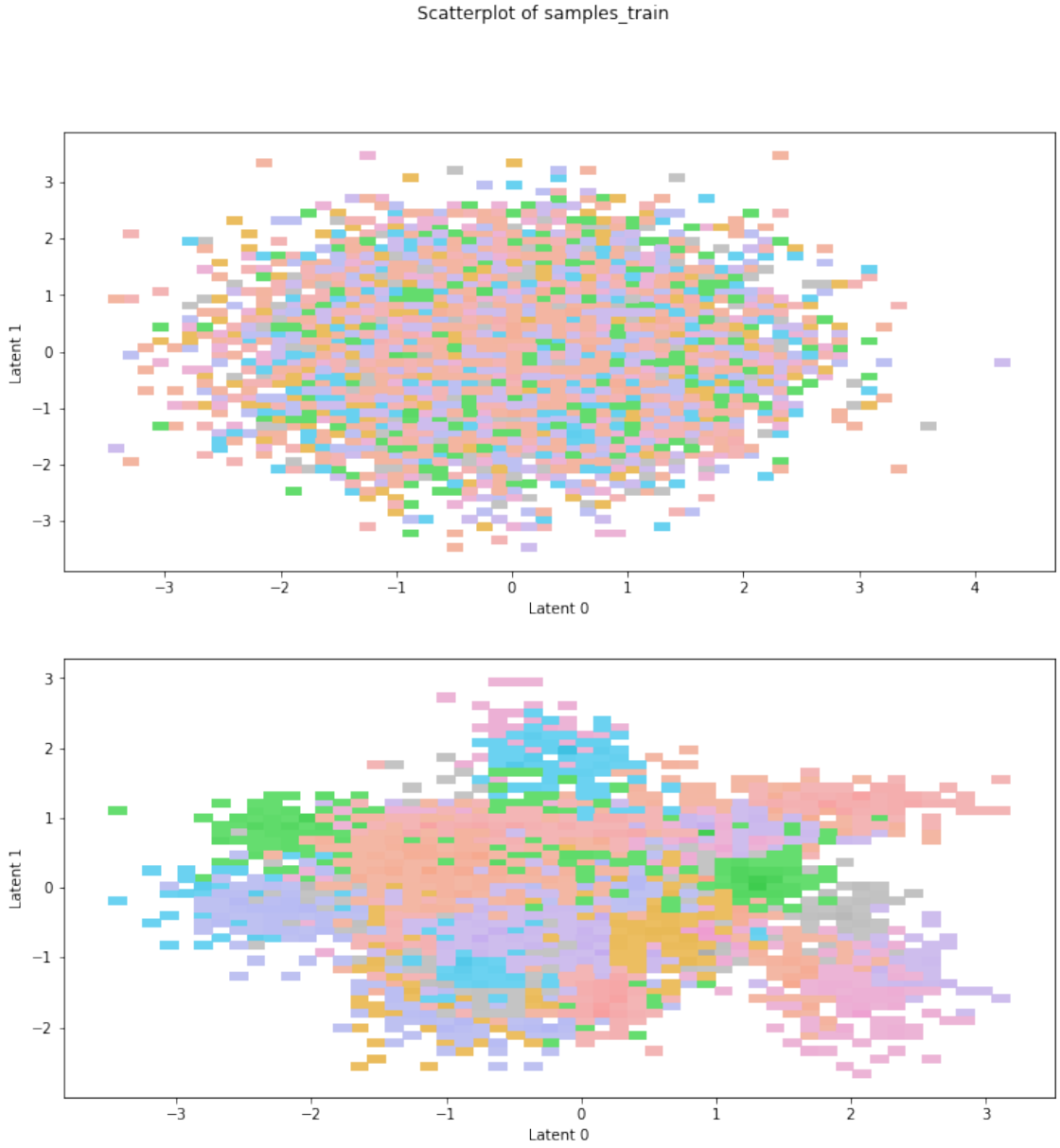
```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),
```

```
Epoch 46, Loss 23755.1689, kl_loss 340.6944, recon_loss 20348.2248,
kl_divergence 286.7706
labels <class 'numpy.ndarray'> (96,)
latents <class 'numpy.ndarray'> (96, 2)
```

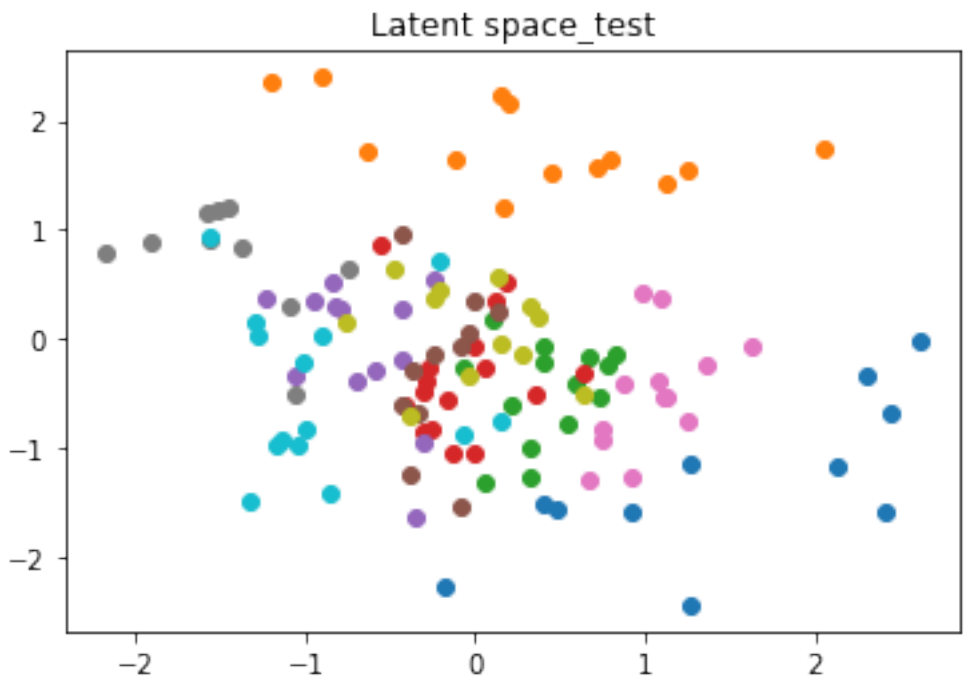



```
Plot bivariate latent distributions
pz batch_shape torch.Size([96, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([96, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 96, 2), qzx (100, 96, 2)
```

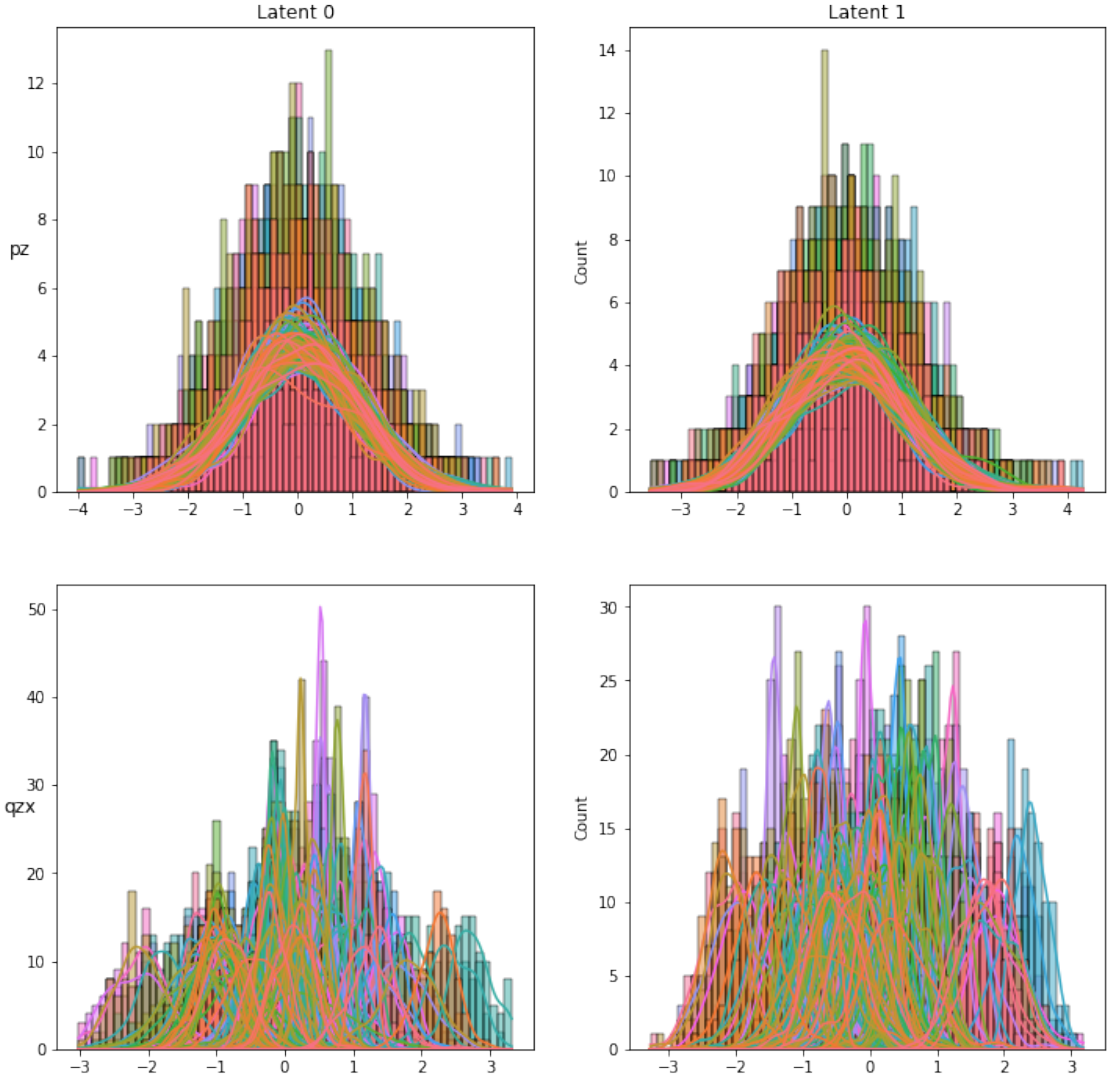

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

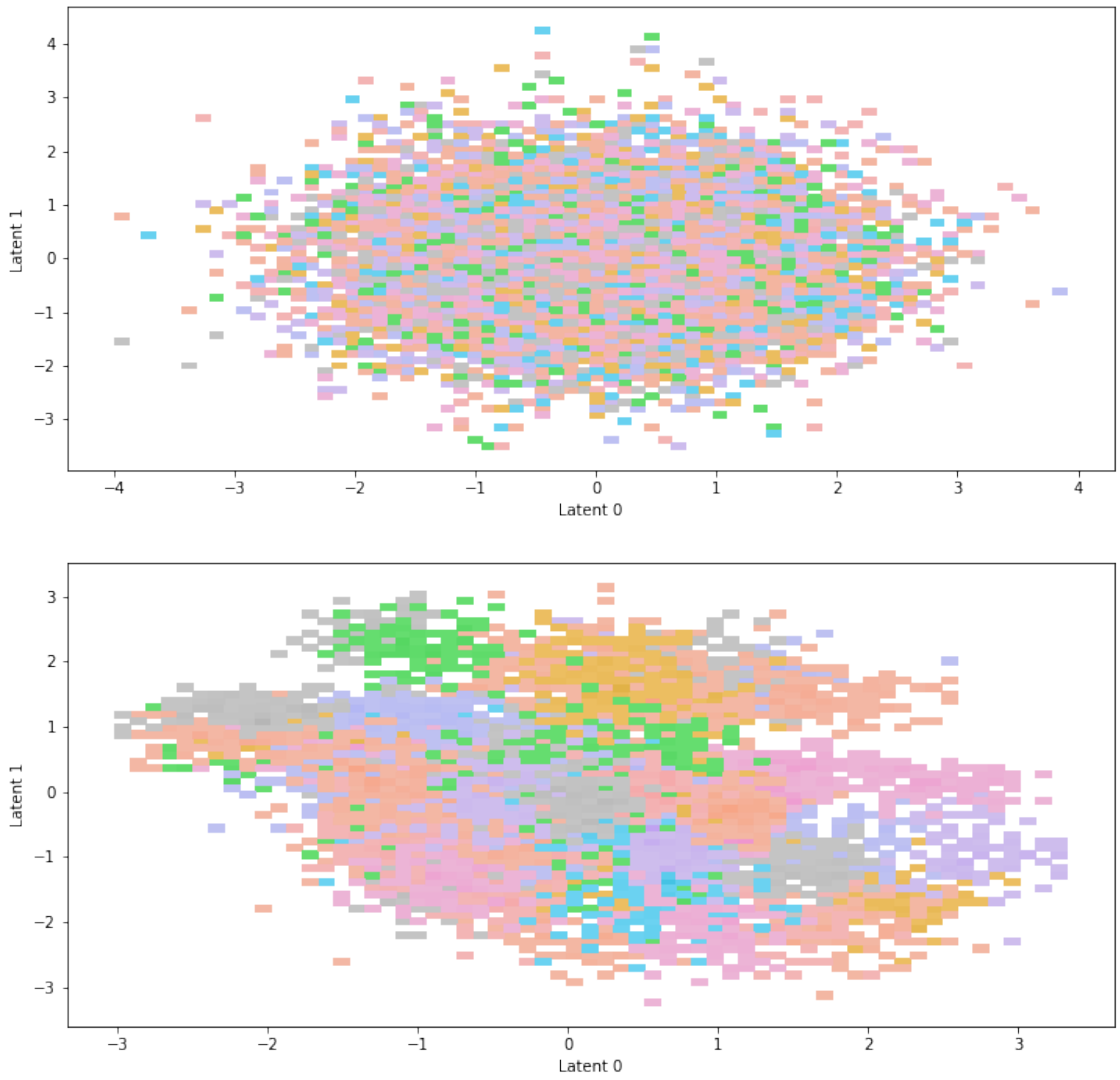



```
labels <class 'numpy.ndarray'> (128,)
latents <class 'numpy.ndarray'> (128, 2)
```




```
Plot bivariate latent distributions
pz batch_shape torch.Size([128, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([128, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 128, 2), qzx (100, 128, 2)
```


Bivariate Latent Distributions_test
Latent 0
Latent 1
pz
Count
qzx
Count




```
<ipython-input-8-d81aeaf6b3a3>:107: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_img": np.array(epoch_sample_img),
<ipython-input-8-d81aeaf6b3a3>:108: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_reconstruction": np.array(epoch_sample_reconstruction),
```

```
<ipython-input-8-d81aeaf6b3a3>:109: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  "sample_latent": np.array(epoch_sample_latent),

dump results dict to MLP_VAE__dist_l2_wkl_10.pkl
CPU times: user 23min 40s, sys: 4.55 s, total: 23min 44s
Wall time: 20min 43s
```

[11]:
```
MLP_VAE(
  (encoder): MLP_V_Encoder(
    (model): Sequential(
      (0): Flatten(start_dim=1, end_dim=-1)
      (1): Linear(in_features=784, out_features=400, bias=True)
      (2): ReLU()
    )
  )
  (decoder): MLP_V_Decoder(
    (model): Sequential(
      (0): Linear(in_features=2, out_features=400, bias=True)
      (1): ReLU()
      (2): Linear(in_features=400, out_features=784, bias=True)
      (3): Sigmoid()
    )
  )
  (enc_to_mean): Linear(in_features=400, out_features=2, bias=True)
  (enc_to_logvar): Linear(in_features=400, out_features=2, bias=True)
)
```

[12]:
```
# problem: distribution.log_prob return positive and nan values, randomly
```

4 Test log_porb function

[13]:
```
mean = torch.tensor([0.,0.])
pz = torch.distributions.Normal(torch.zeros_like(mean), scale=1.)
```

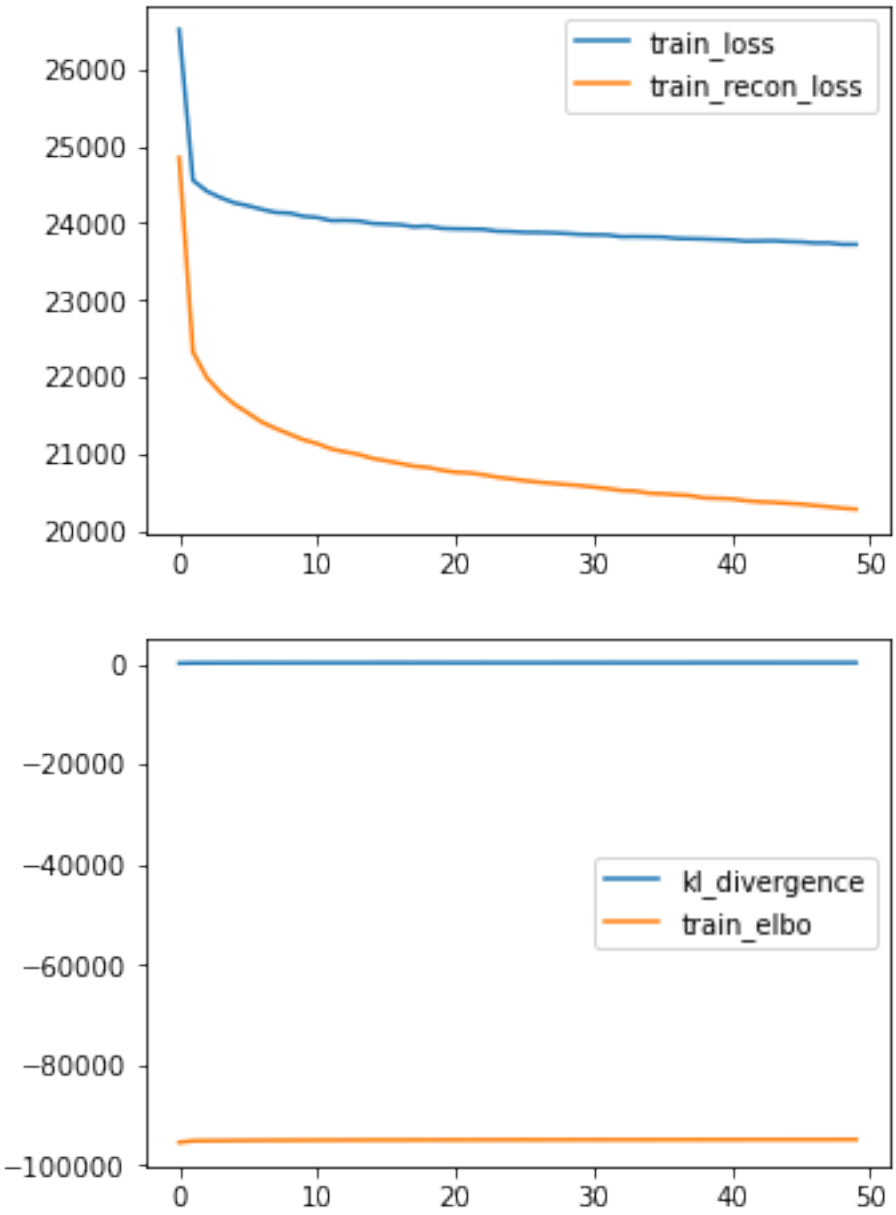
[14]:
```
a = torch.tensor([100, -500]).float()
print(pz.log_prob(a))
```

```
tensor([  -5000.9189, -125000.9219])
```

4.1 Visualization of the training process and results

4.1.1 plot the learning curve

[15]:
```
epoch_train_loss = results_dict["train_loss"]
epoch_train_kl_loss = results_dict["train_kl_loss"]
epoch_train_recon_loss = results_dict["train_recon_loss"]

epoch_train_kl_divergence = results_dict["train_kl_divergence"]
epoch_train_pxz_likelihood = results_dict["train_pxz_likelihood"]
epoch_train_elbo = results_dict["train_elbo"]

fig, axes = plt.subplots(2,1, figsize=(5,8))
assert len(epoch_train_loss)==num_epochs, "check num_epochs"
axes[0].plot(np.arange(num_epochs), epoch_train_loss, label="train_loss")
# axes[0].plot(np.arange(num_epochs), epoch_train_kl_loss,␣
 ↪label="train_kl_loss")
axes[0].plot(np.arange(num_epochs), epoch_train_recon_loss,␣
 ↪label="train_recon_loss")
axes[0].legend()
axes[1].plot(np.arange(num_epochs), epoch_train_kl_divergence,␣
 ↪label="kl_divergence")
# axes[1].plot(np.arange(num_epochs), epoch_train_pxz_likelihood,␣
 ↪label="pxz_likelihood")
axes[1].plot(np.arange(num_epochs), epoch_train_elbo, label="train_elbo")
axes[1].legend()

plt.show()
```



[16]:
```
print(len(epoch_train_recon_loss), len(epoch_train_kl_divergence),␣
↪len(epoch_train_elbo))
```

```
50 50 50
```

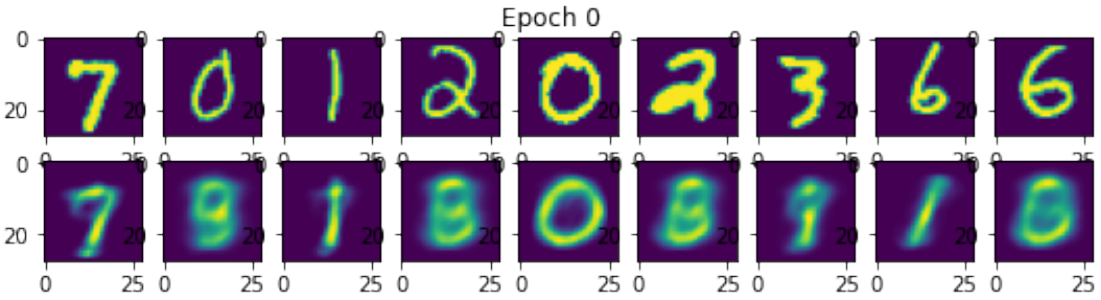
[17]:
```
print(epoch_train_elbo)
```

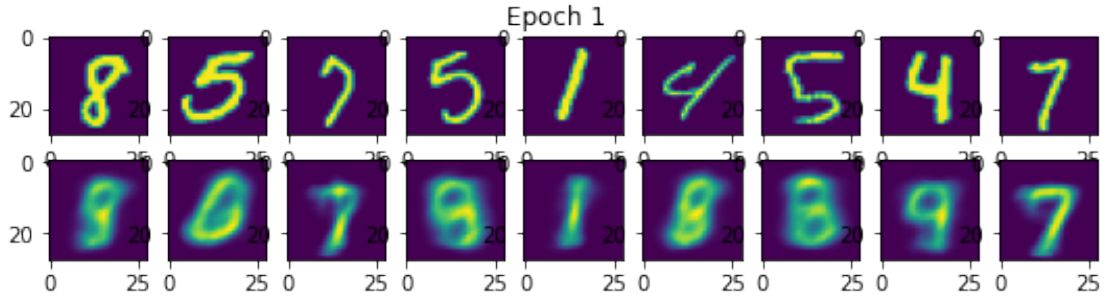
```
[-95419.37828158 -95111.39005864 -95075.36905317 -95051.94059835
 -95032.59021855 -95018.29442631 -95001.96083755 -94991.3139992
 -94982.82052905 -94971.79359342 -94965.96641791 -94954.24878398
```

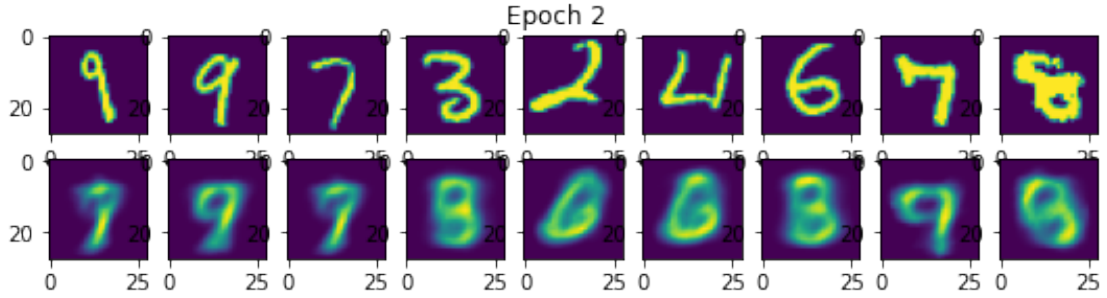
```
-94950.81531517 -94946.50038313 -94938.21388593 -94934.79342684
-94929.65921509 -94924.03174973 -94922.71115405 -94915.93590085
-94912.87548308 -94911.29266058 -94908.08084022 -94904.07189499
-94901.88121335 -94896.04114472 -94893.0338153  -94890.24190432
-94889.52242138 -94886.89797108 -94883.55343817 -94880.54021189
-94876.74958356 -94876.73016058 -94872.25478078 -94870.69329691
-94867.97394723 -94866.65503398 -94863.17825493 -94861.68846615
-94861.51577492 -94855.65709955 -94854.8374367  -94853.81969616
-94851.90421775 -94849.83583755 -94847.23494136 -94844.39443963
-94840.90406783 -94839.43936567]
```

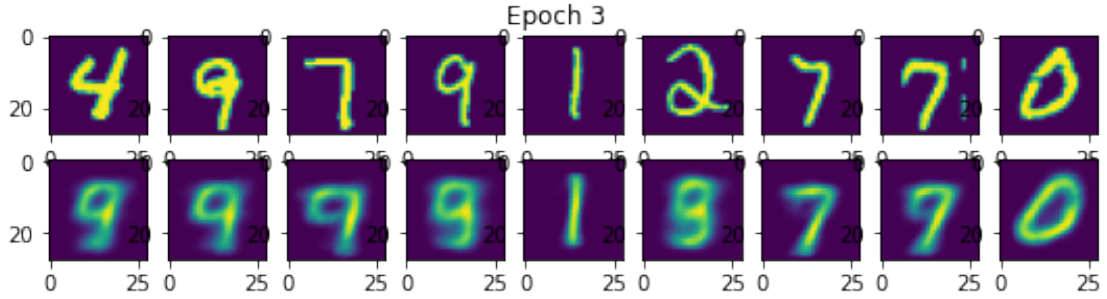
4.1.2 plot the evolution of reconstruction through epochs

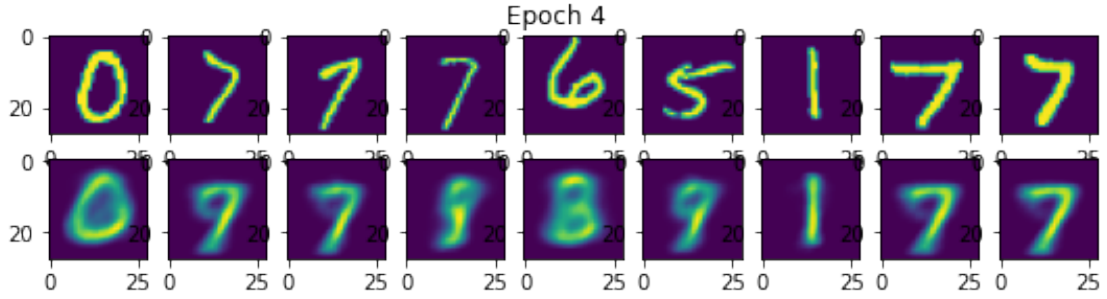
[18]:
```
num_cols = 9
selected_epochs = np.concatenate((np.arange(5),np.arange(10,num_epochs,10)))
for epoch in selected_epochs:
    figure = plt.figure(figsize=(num_cols,2))
    figure.suptitle("Epoch {}".format(epoch))
    imgs = results_dict["sample_img"][epoch]
    reconstructions = results_dict["sample_reconstruction"][epoch]
    for i, item in enumerate(imgs):
        # plot only first few images
        if i>=num_cols: break
        plt.subplot(2,num_cols, i+1)
        plt.imshow(item[0])

    for i, item in enumerate(reconstructions):
        if i>=num_cols: break
        plt.subplot(2, num_cols, num_cols+i+1)
        plt.imshow(item[0])
```

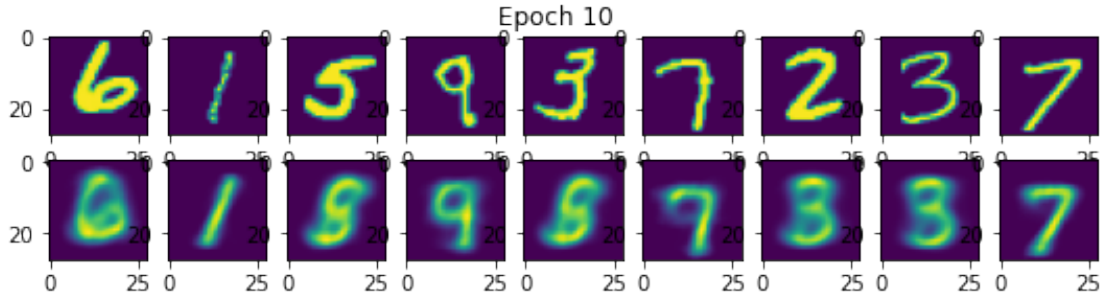


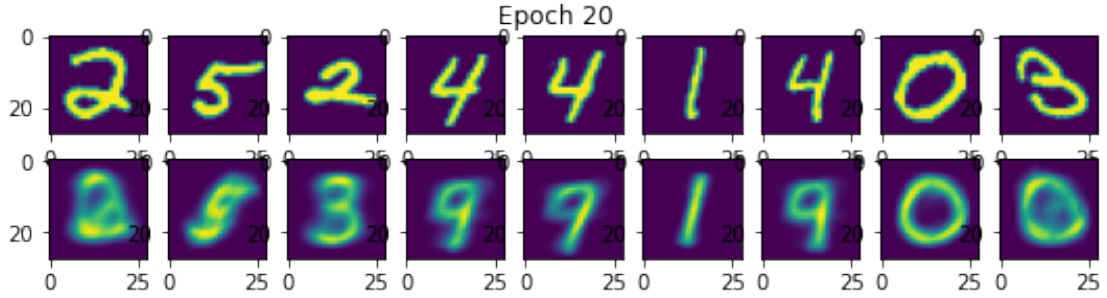

Epoch 1

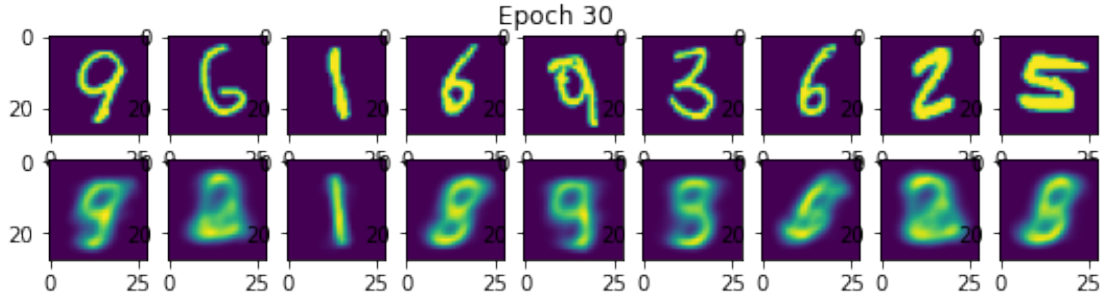

Epoch 2

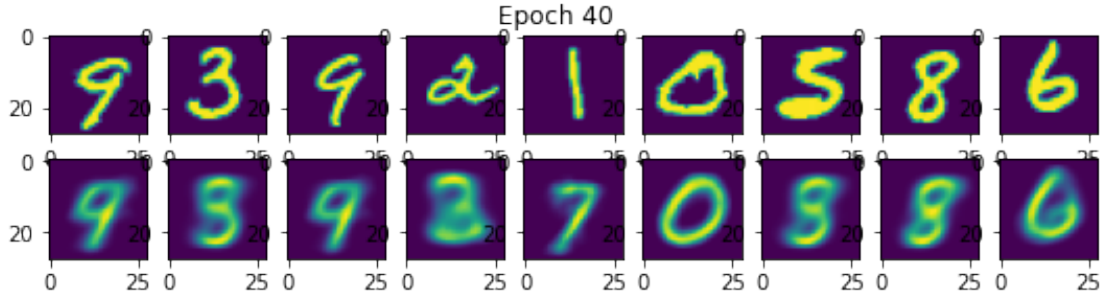

Epoch 3


Epoch 4


Epoch 10


Epoch 20


Epoch 30


Epoch 40

4.1.3 plot the latent space

The hover part takes reference from this post

[19]:
```
train_imgs, train_labels = train_dataset.data, train_dataset.targets
train_imgs = torch.unsqueeze(train_imgs, 1).to(device).float()
print("train_imgs", type(train_imgs))
print("train_labels", type(train_labels))
print(train_imgs.shape, train_labels.shape)
train_latents, train_reconstructions, train_latent_mean, train_latent_logvar =␣
↪model(train_imgs)
train_latents, train_reconstructions = train_latents.cpu().detach().numpy(),␣
↪train_reconstructions.cpu().detach().numpy()
print("train_latents {} {}, \ntrain_reconstructions {} {}".
↪format(type(train_latents), train_latents.shape,␣
↪type(train_reconstructions), train_reconstructions.shape))
train_labels = train_labels.cpu().detach().numpy()
print("train_labels {} {}".format(type(train_labels), train_labels.shape))
```

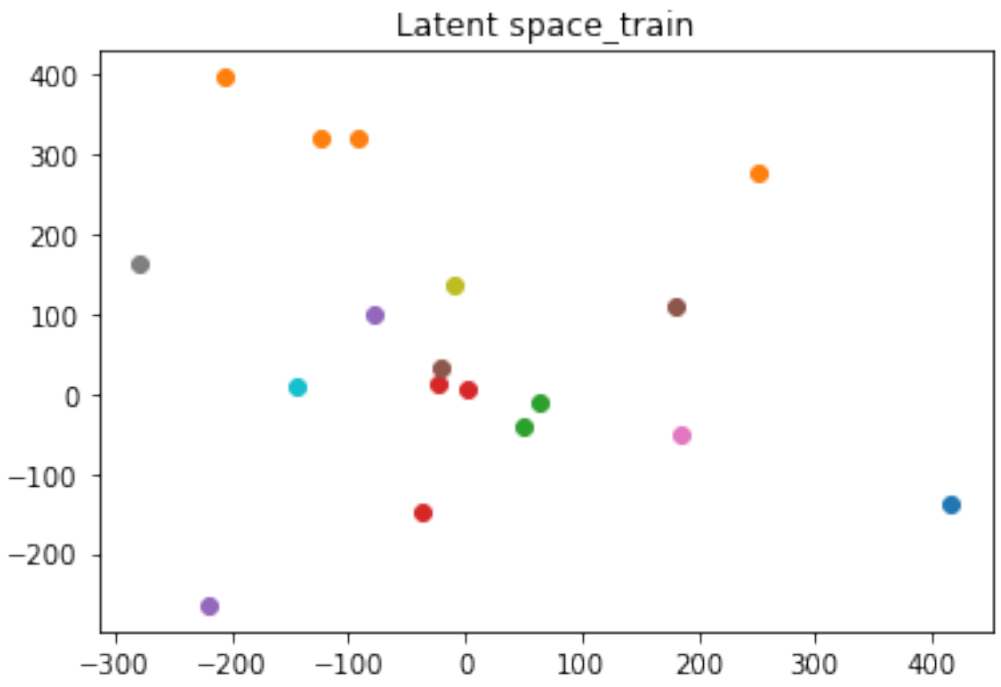
```
train_imgs <class 'torch.Tensor'>
train_labels <class 'torch.Tensor'>
torch.Size([60000, 1, 28, 28]) torch.Size([60000])
train_latents <class 'numpy.ndarray'> (60000, 2),
train_reconstructions <class 'numpy.ndarray'> (60000, 1, 28, 28)
train_labels <class 'numpy.ndarray'> (60000,)
```

[20]:
```
sample_train_labels = train_labels[:18,...]
sample_train_latents = train_latents[:18,...]
sample_train_latent_mean = train_latent_mean[:18,...]
sample_train_latent_logvar = train_latent_logvar[:18,...]
sample_train_latent_std = (0.5*sample_train_latent_logvar).exp()
```

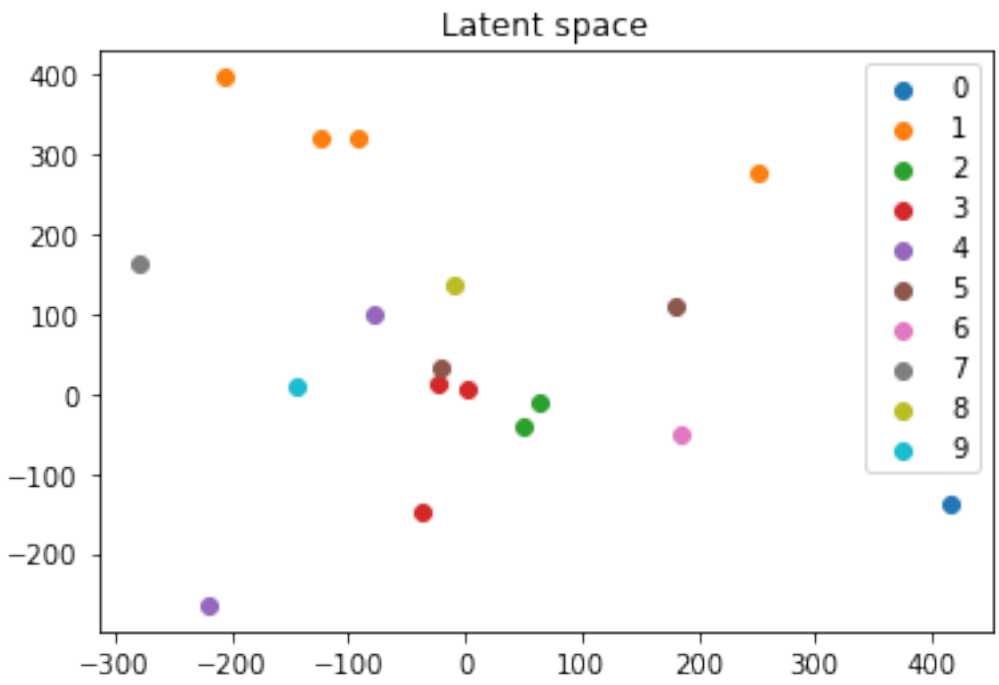
[21]:
```
print("mean")
print(sample_train_latent_mean)
print("std")
print(sample_train_latent_std)
plot_latent(sample_train_labels, sample_train_latents)
```

```
mean
tensor([[ -20.0732,   32.5483],
        [ 416.9071, -138.2627],
        [-220.1731, -264.6514],
        [ 250.6994,  275.4350],
        [-145.5950,   10.8437],
        [  64.8372,  -10.6862],
        [-206.2195,  396.7052],
        [ -23.1264,   12.9825],
        [-124.1428,  321.3107],
```

```
        [ -77.1736,  100.6335],
        [   1.7513,    6.3637],
        [ 179.4392,  110.3283],
        [ -37.2863, -147.6292],
        [ 183.9504,  -50.2837],
        [ -91.6600,  320.0708],
        [-279.7747,  163.4902],
        [  49.0321,  -41.7944],
        [ -10.2450,  136.9720]], device='cuda:0', grad_fn=<SliceBackward>)
std
tensor([[0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [1.4053e-29, 2.9599e-37],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.4428e-40],
        [4.8632e-32, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.7690e-38],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00]], device='cuda:0', grad_fn=<ExpBackward>)
labels <class 'numpy.ndarray'> (18,)
latents <class 'numpy.ndarray'> (18, 2)
```



[22]:
```
fig, ax = plt.subplots()
# ax.scatter(test_latents[:,0], test_latents[:,1], c=test_labels) # fail to set␣
 ↪labels
for y in np.unique(train_labels):
    i = np.where(sample_train_labels == y)
    ax.scatter(sample_train_latents[i,0], sample_train_latents[i,1], label=y,␣
 ↪cmap="tab10")
ax.legend()
plt.title("Latent space")
plt.show()
```

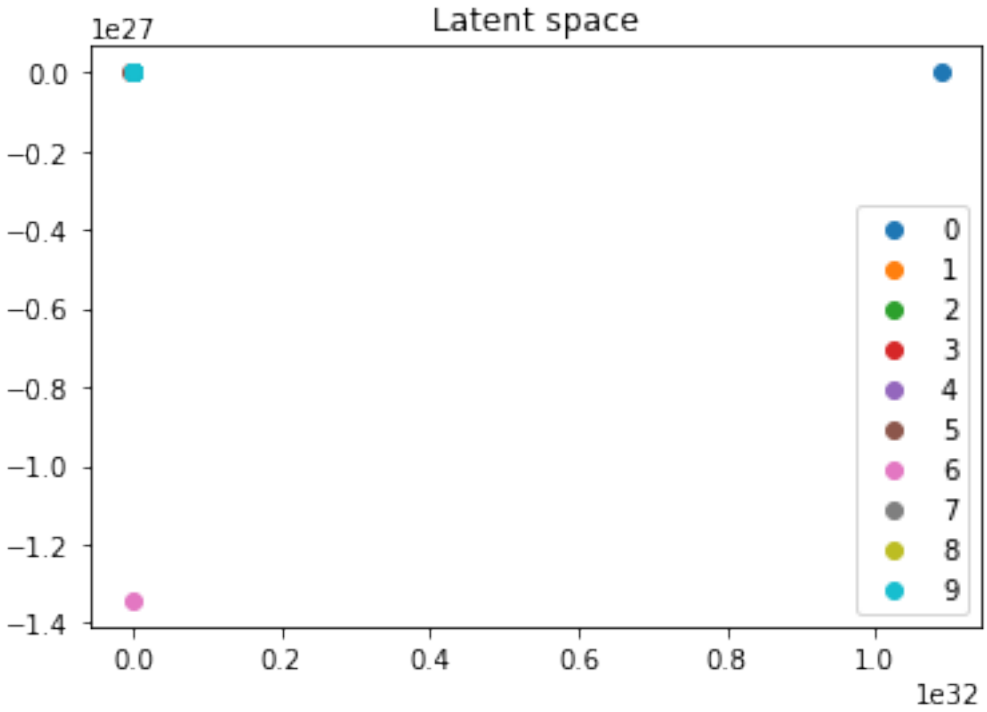


[23]:
```
test_imgs, test_labels = test_dataset.data, test_dataset.targets
test_imgs = torch.unsqueeze(test_imgs, 1).to(device).float()
print("test_imgs", type(test_imgs))
print("test_labels", type(test_labels))
print(test_imgs.shape, test_labels.shape)
test_latents, test_reconstructions, test_latent_mean, test_latent_logvar =␣
 ↪model(test_imgs)
test_latents, test_reconstructions = test_latents.cpu().detach().numpy(),␣
 ↪test_reconstructions.cpu().detach().numpy()
print("test_latents {} {}, \ntest_reconstructions {} {}".
 ↪format(type(test_latents), test_latents.shape, type(test_reconstructions),␣
 ↪test_reconstructions.shape))
test_labels = test_labels.cpu().detach().numpy()
print("test_labels {} {}".format(type(test_labels), test_labels.shape))
```

```
test_imgs <class 'torch.Tensor'>
test_labels <class 'torch.Tensor'>
torch.Size([10000, 1, 28, 28]) torch.Size([10000])
test_latents <class 'numpy.ndarray'> (10000, 2),
test_reconstructions <class 'numpy.ndarray'> (10000, 1, 28, 28)
test_labels <class 'numpy.ndarray'> (10000,)
```

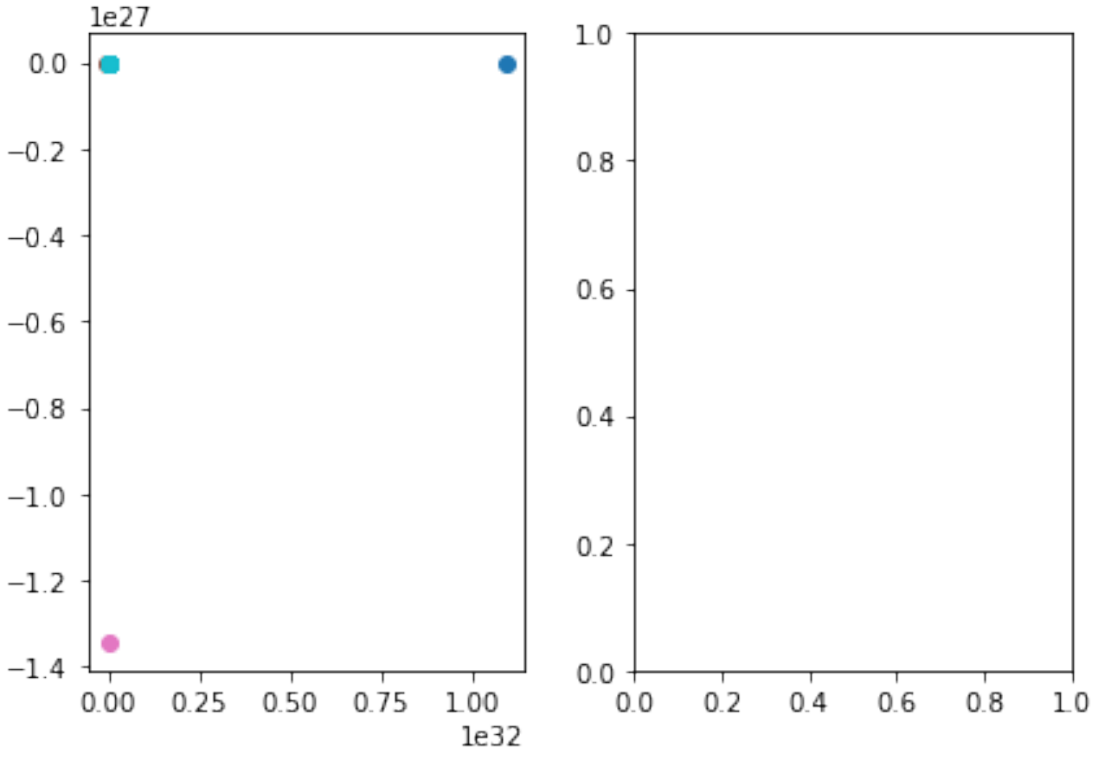
[24]:
```
fig, ax = plt.subplots()
```

```
# ax.scatter(test_latents[:,0], test_latents[:,1], c=test_labels) # fail to set␣
 ↪labels
for y in np.unique(test_labels):
    i = np.where(test_labels == y)
    ax.scatter(test_latents[i,0], test_latents[i,1], label=y, cmap="tab10")
ax.legend()
plt.title("Latent space")
plt.show()
```



[25]:
```
import PyQt5
%matplotlib qt
# interactive hovering
fig, ax = plt.subplots(1, 2)
plt.tight_layout()
for y in np.unique(test_labels):
    i = np.where(test_labels == y)
    ax[0].scatter(test_latents[i,0], test_latents[i,1], label=y, cmap="tab10")
#     ax[0].legend(loc='lower center', bbox_to_anchor=(0.5, -0.
 ↪05),fancybox=True, shadow=True, ncol=5)

def onclick(event):
    global flag
    if event.xdata is None or event.ydata is None:
```

```
        return
    ix, iy = int(event.xdata), int(event.ydata)

    latent_vector = np.array([[ix, iy]])
    latent_vector = torch.from_numpy(latent_vector).float().to(device)
    decoded_img = model.decoder(latent_vector)
    decoded_img = decoded_img.cpu().detach().numpy()[0][0] # [1,1,28,28] =>␣
↪[28,28]
    ax[1].imshow(decoded_img)
    plt.draw()

# button_press_event
# motion_notify_event
cid = fig.canvas.mpl_connect('motion_notify_event', onclick)

plt.show()
```

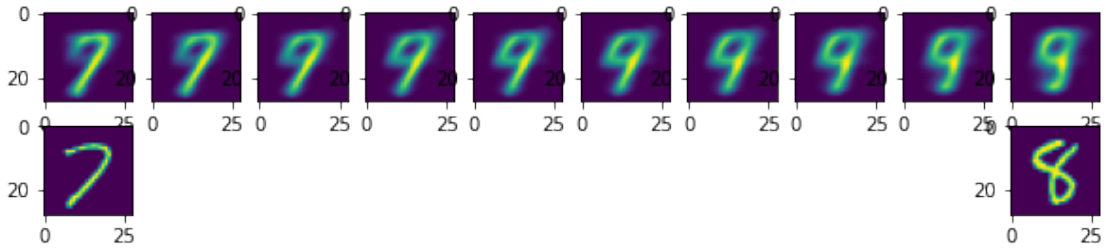


4.1.4 Interpolation between any two images

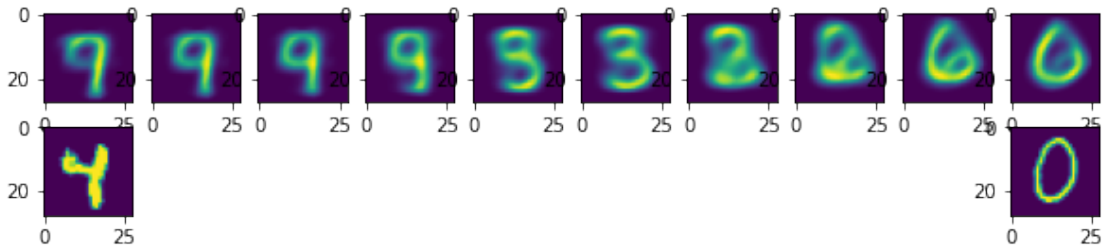
[26]:
```
%matplotlib inline
```

[27]:
```
def interpolate(index1, index2):
    x1 = results_dict["sample_img"][-1][index1]
    x2 = results_dict["sample_img"][-1][index2]
    x1, x2 = torch.from_numpy(x1).float().to(device), torch.from_numpy(x2).
↪float().to(device)
    x = torch.stack([x1, x2])
    latent_mean, latent_logvar = model.encode(x)
    embedding = model.sample_latent_embedding(latent_mean, latent_logvar)
    e1 = embedding[0]
    e2 = embedding[1]

    embedding_values = []
    for i in range(10):
        e = e1 * (i/10) + e2 * (10-i)/10
        embedding_values.append(e)
    embedding_values = torch.stack(embedding_values)
    recon_from_embeddings = model.decoder(embedding_values) # shape [10, 1, 28,␣
↪28]

    plt.figure(figsize=(10,2))
    for i, recon in enumerate(recon_from_embeddings.cpu().detach().numpy()):
        plt.subplot(2, 10, i+1)
        plt.imshow(recon[0])
    # plot two original images
    plt.subplot(2, 10, 11)
    plt.imshow(x2.cpu().detach().numpy()[0])
    plt.subplot(2, 10, 20)
    plt.imshow(x1.cpu().detach().numpy()[0])
```

[28]:
```
interpolate(3,5)
```



[29]:
```
interpolate(2,7)
```

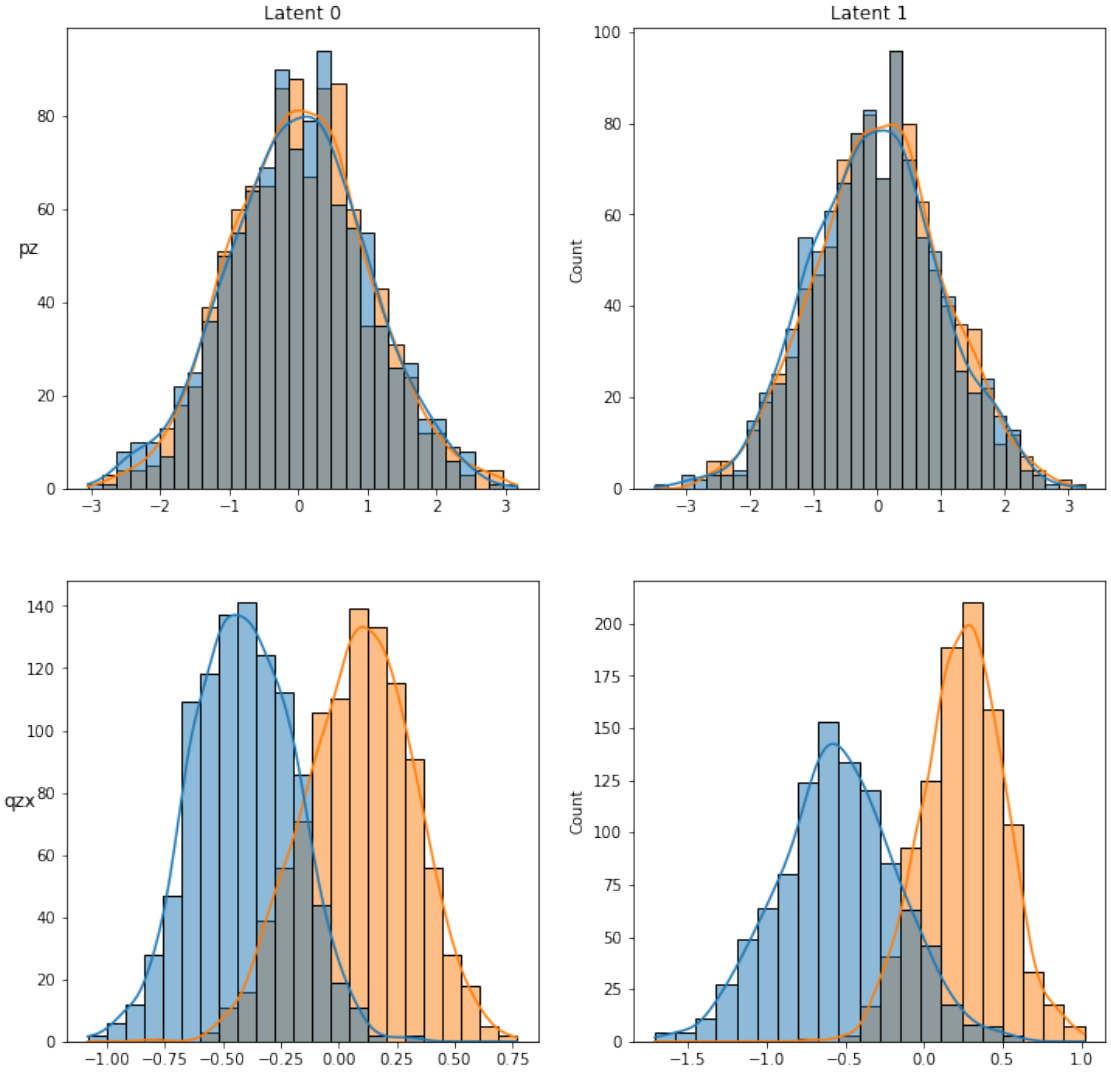


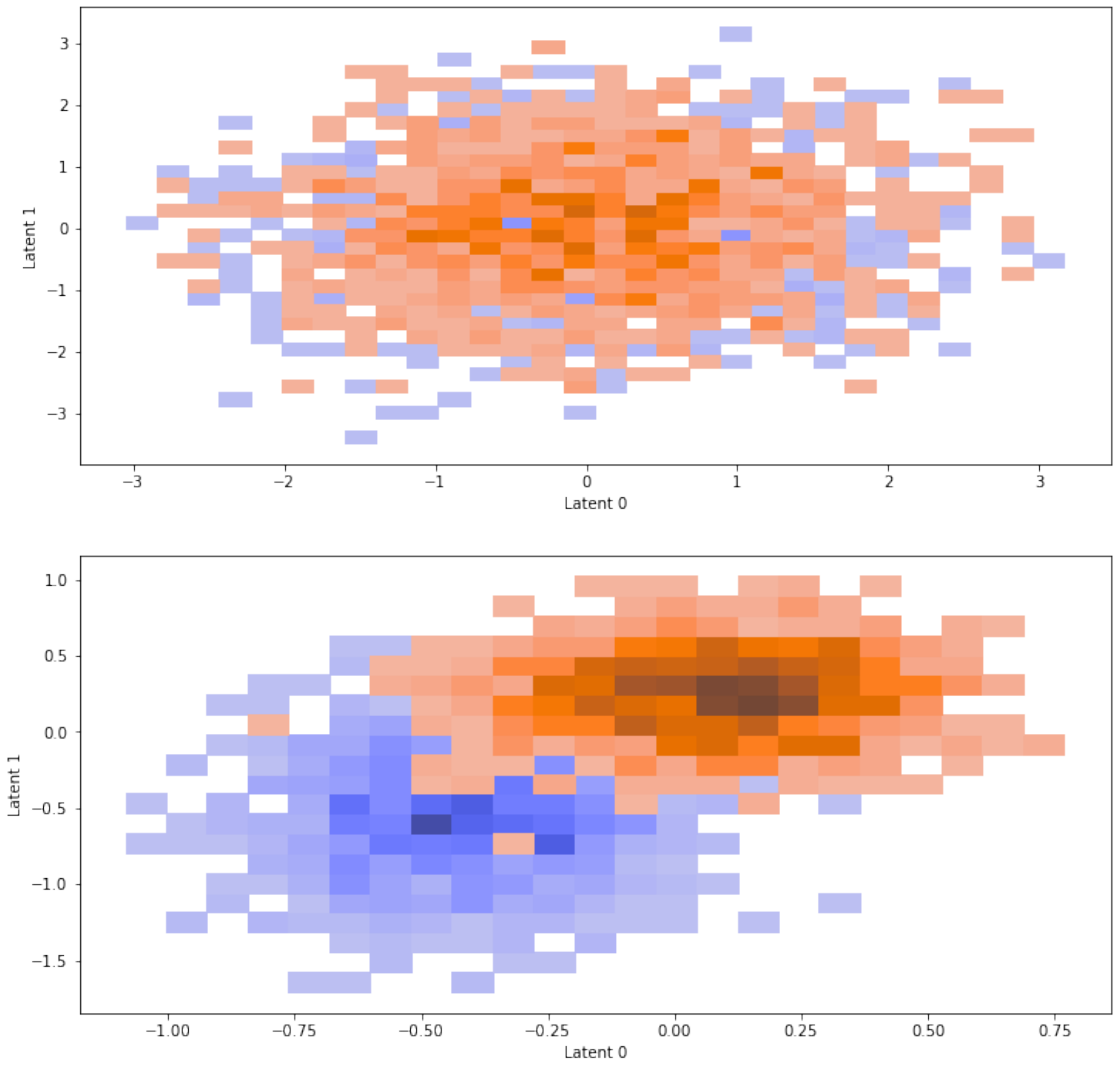
4.1.5 Interactive scroll bar for latent space

see AE_MNIST_Interactive_ScrollBar

[30]:
```
index1 = 3
index2 = 15
x1 = results_dict["sample_img"][-1][index1]
x2 = results_dict["sample_img"][-1][index2]
x1, x2 = torch.from_numpy(x1).float().to(device), torch.from_numpy(x2).float().
 ↪to(device)
x = torch.stack([x1, x2])
latent_mean, latent_logvar = model.encode(x)
plot_p_q(latent_mean, latent_logvar, N_samples=1000)
print("mean")
print(latent_mean)
print("logvar")
print(latent_logvar)
latent_std = torch.sqrt(torch.exp(latent_logvar))
print("std")
print(latent_std)
```

```
Plot bivariate latent distributions
pz batch_shape torch.Size([2, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([2, 2]), event_shape torch.Size([])
check p, q shape, pz (1000, 2, 2), qzx (1000, 2, 2)
```

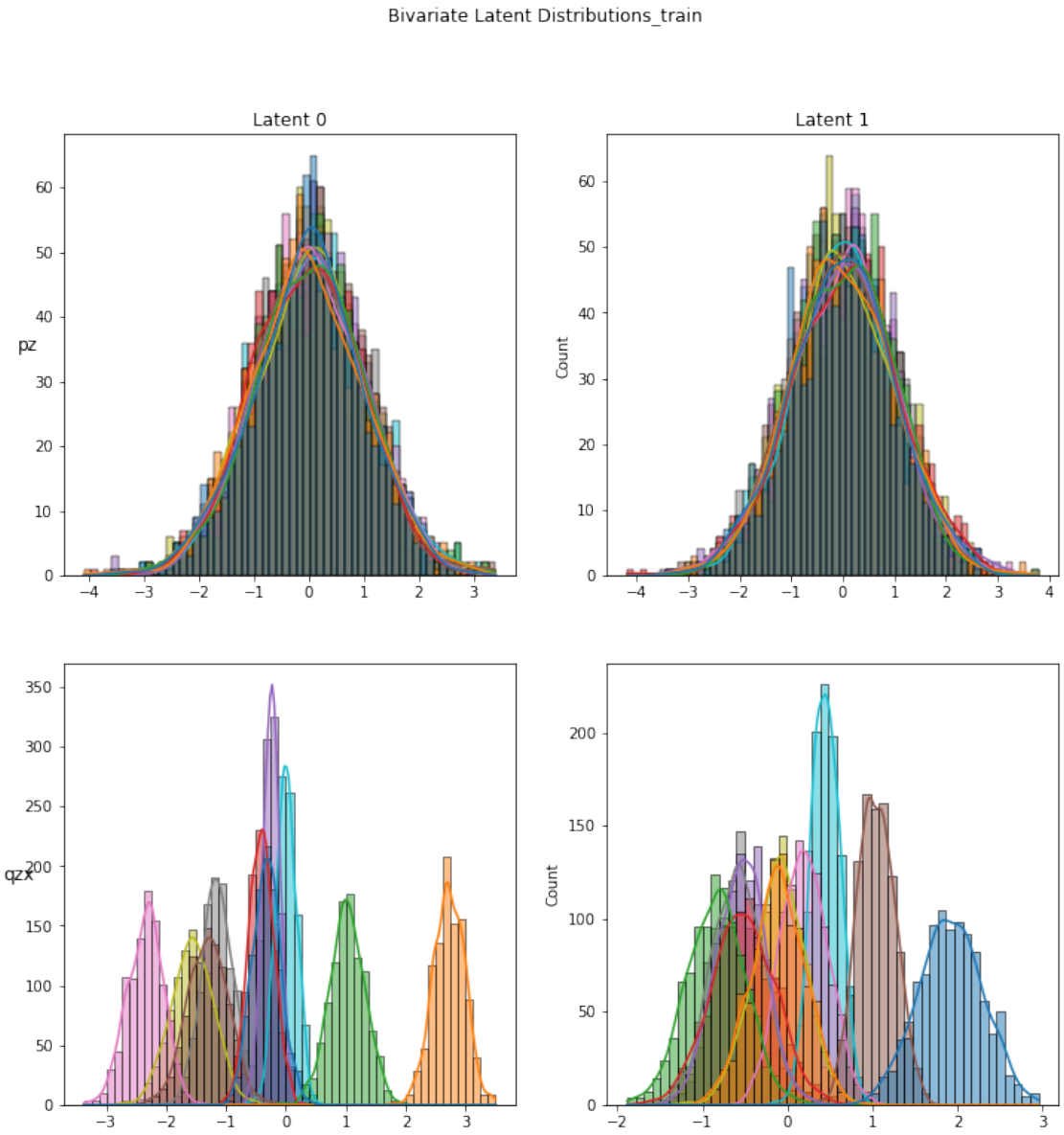

Bivariate Latent Distributions_train
Latent 0
Latent 1
pz
Count
qzx
Count

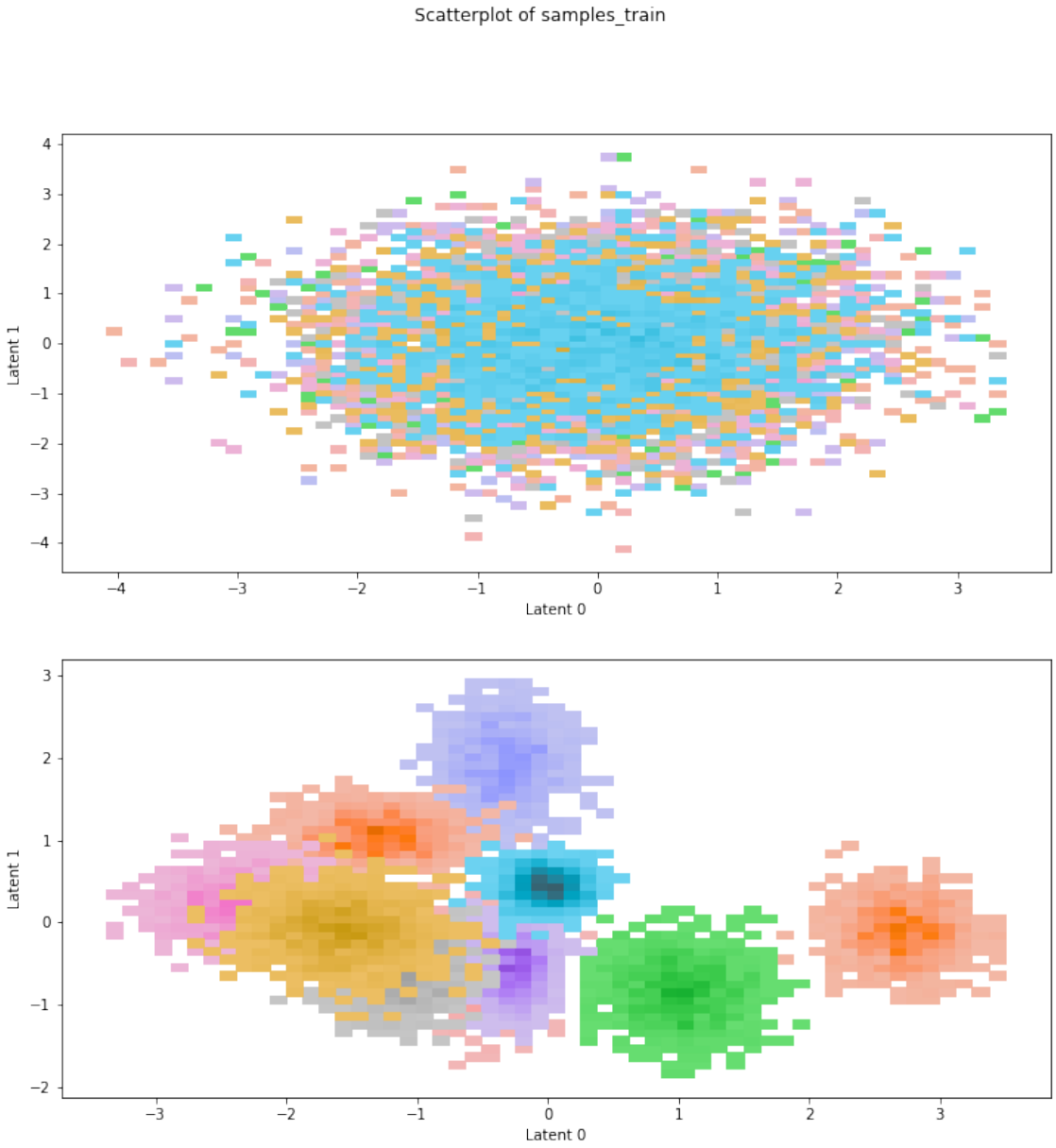



```
mean
tensor([[-0.4070, -0.5264],
        [ 0.0847,  0.2538]], device='cuda:0', grad_fn=<AddmmBackward>)
logvar
tensor([[-3.1127, -1.9761],
        [-2.9104, -2.7014]], device='cuda:0', grad_fn=<AddmmBackward>)
std
tensor([[0.2109, 0.3723],
        [0.2334, 0.2591]], device='cuda:0', grad_fn=<SqrtBackward>)
```

[31]:
```
x = torch.from_numpy(results_dict["sample_img"][-1][:10,...]).float().to(device)
latent_mean, latent_logvar = model.encode(x)
plot_p_q(latent_mean, latent_logvar, N_samples=1000)
```

```
Plot bivariate latent distributions
pz batch_shape torch.Size([10, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([10, 2]), event_shape torch.Size([])
check p, q shape, pz (1000, 10, 2), qzx (1000, 10, 2)
```





[32]:
```
def plot_latent(test_labels, test_latents):

#     test_labels = test_labels.cpu().detach().numpy()
    print("test_labels {} {}".format(type(test_labels), test_labels.shape))
#     test_latents = test_latents.cpu().detach().numpy()
    print("test_latents {} {}".format(type(test_latents), test_latents.shape))

    fig, ax = plt.subplots()
    # ax.scatter(test_latents[:,0], test_latents[:,1], c=test_labels) # fail to␣
 ↪set labels
```

```python
    for y in np.unique(test_labels):
        i = np.where(test_labels == y)
        ax.scatter(test_latents[i,0], test_latents[i,1], label=y, cmap="tab10")
    # ax.legend()
    plt.title("Latent space")
    plt.show()
```

[ ]:

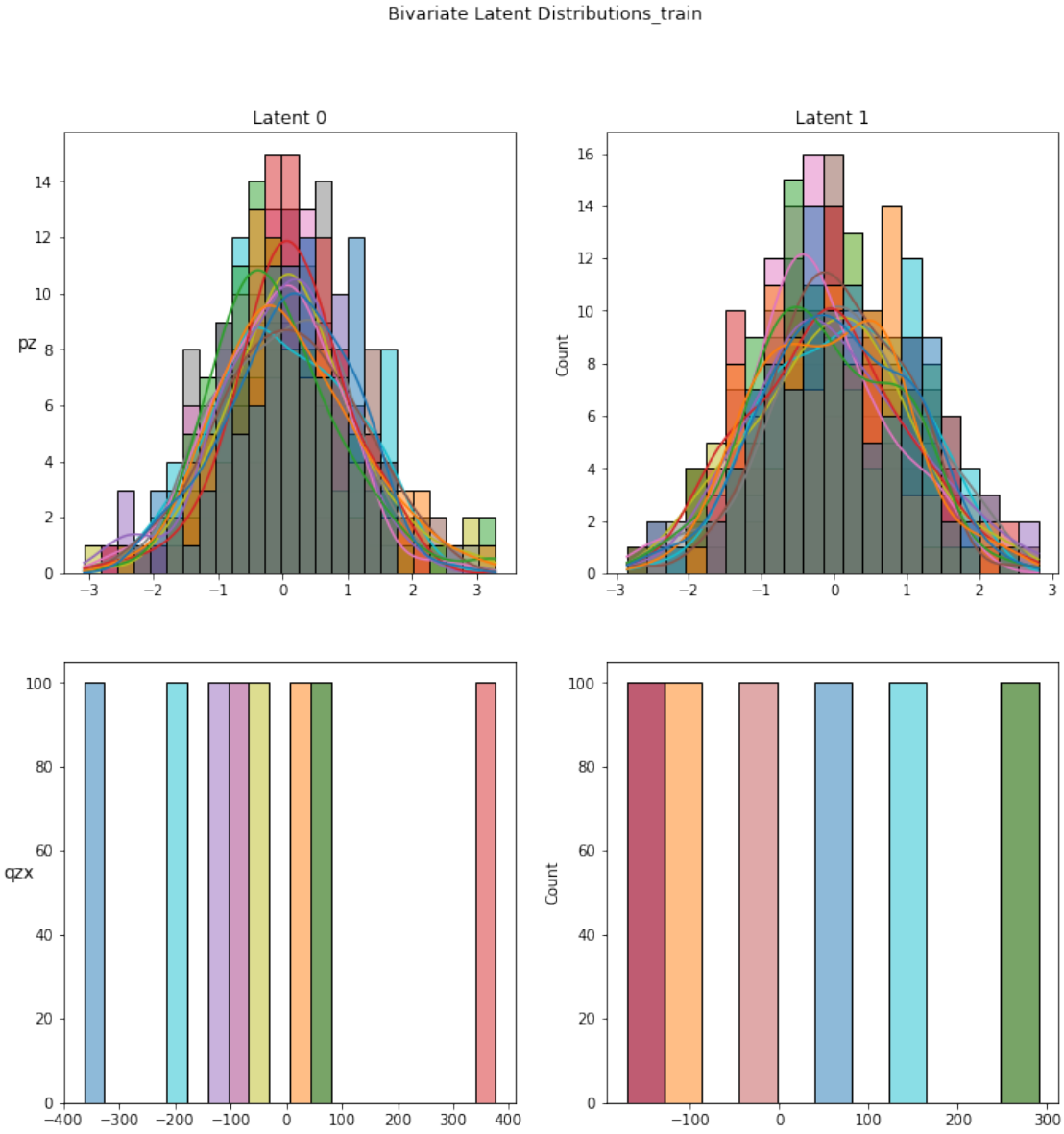
[33]:
```python
plot_p_q(test_latent_mean[:10,...], test_latent_logvar[:10,...])
# plot_latent(test_labels[:100,...], test_reconstructions[:100,...])
```

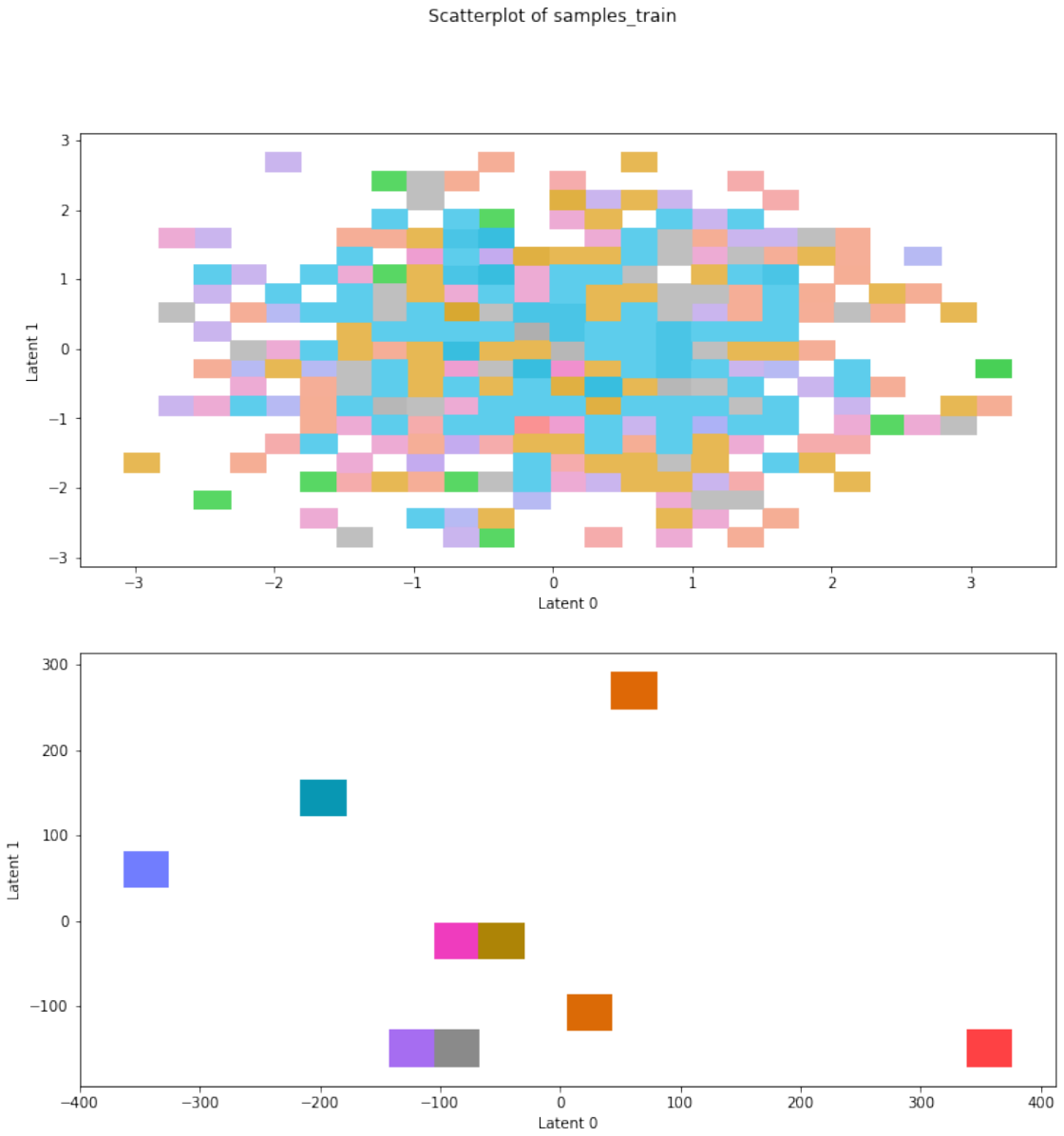
```
Plot bivariate latent distributions
pz batch_shape torch.Size([10, 2]), event_shape torch.Size([])
qzx batch_shape torch.Size([10, 2]), event_shape torch.Size([])
check p, q shape, pz (100, 10, 2), qzx (100, 10, 2)
```

```
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
```

```
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
  warnings.warn(msg, UserWarning)
/home/ruihan/anaconda3/envs/slayer/lib/python3.8/site-
packages/seaborn/distributions.py:306: UserWarning: Dataset has 0 variance;
skipping density estimate.
```

```
warnings.warn(msg, UserWarning)
```



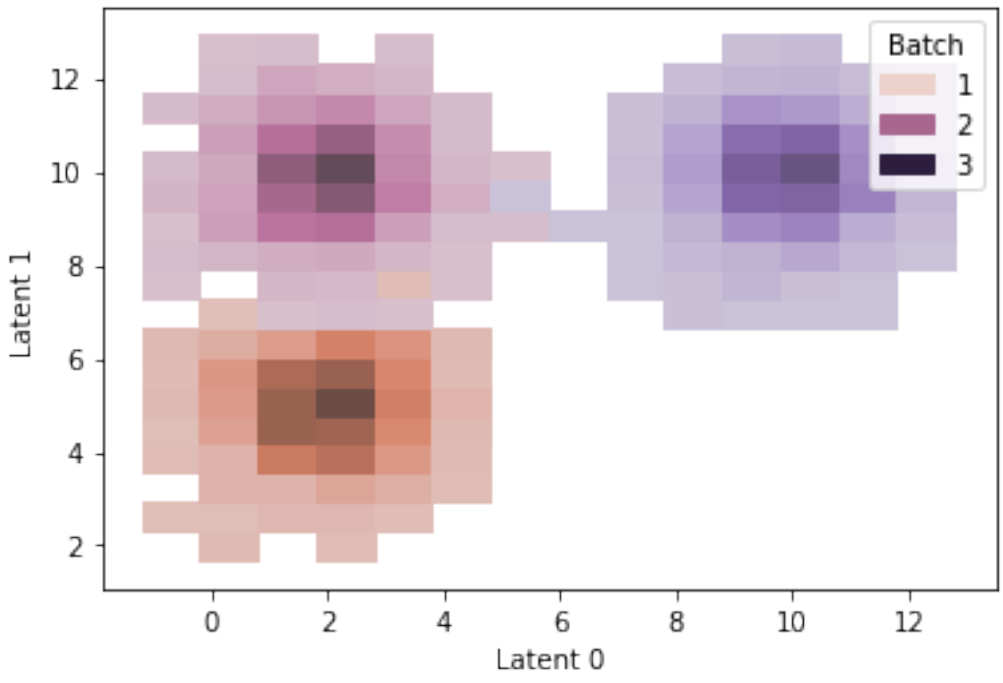


[34]:
```
# Explore reshape numpy array into panda frame and plot with seaborn
import numpy as np
import pandas as pd
np.random.seed(2016)

sample_shape = 1000
batch_shape = 3
latent_shape = 2
# input shape (sample_shape, batch_shape, latent_shape)
# output shape seaborn plot with x=latent_shape[0], y=latent_shape[1],␣
 ↪hue=batch_shape
```

```
sigma = 1
mu = [[2, 5],[2,10], [10,10]]
arr = np.random.normal(mu, sigma, (sample_shape, batch_shape, latent_shape))
print("arr", arr.shape)


df = pd.DataFrame(arr.reshape(-1, latent_shape), columns=["Latent {}".format(i)␣
 ↪for i in range(latent_shape)]) #
df.index = np.tile(np.arange(arr.shape[1]), arr.shape[0]) + 1
df.index.name = 'Batch'
index = np.tile(np.arange(arr.shape[1]), arr.shape[0]) + 1
# print("arr")
# print(arr)
# print("df")
# print(df)
# print("index")
# print(index)

sns.histplot(df, x="Latent 0", y="Latent 1", hue="Batch", kde=True)
```

```
arr (1000, 3, 2)
```

[34]: <AxesSubplot:xlabel='Latent 0', ylabel='Latent 1'>



[ ]: