DT8122 - Assignment

Deadline: 2023 August 15 AoE (Anywhere on Earth)

Send a zip file with the notebook both as a .ipynb and as a .pdf file to dt8122@idi.ntnu.no. Label the file with your full name.

The task is to implement conditional DDPM for MNIST images. Your implementation should take as input a digit and be able to generate 28x28 grayscale handwritten image of said digit.

You can add additional cells anywhere in the notebook to make your code more readable.

DDPM: https://arxiv.org/abs/2006.11239

Classifier-free conditional DDPM: https://arxiv.org/abs/2207.12598

The notebook should be run when it is turned in so all plots are visible. All code should be contained in the notebook.

Install necessary libraries

Any additional libraries you make use of should be installed in this cell.

Import libraries

All import statements should be contained here.

In [1]:
```
import random
import math

import matplotlib.pyplot as plt
import torch

from torch.utils.data import DataLoader
from torchvision.transforms import Compose, ToTensor, Lambda
from torchvision.datasets.mnist import MNIST

from typing import Optional, Tuple, TypeVar
import numpy as np
from tqdm import tqdm
```

Define constants here

Constants such as number of epochs, device, and learning rate and other hyperparameters should be defined here.

In [2]:
```
SEED = 0
random.seed(SEED)
torch.manual_seed(SEED)

batch_size = 128
##number of steps for defusion and denoising
N = 500
BMIN = 0.0001
BMAX = 0.02
guidence_w = 0.5
device = "cuda:0"
```

Functionality for loading and visualizing dataset

We have provided some functionality for loading and visualizing the dataset. You may add more cells/functions here.

In [3]:
```
## from 0-1 to -1-1
transform = Compose([
    ToTensor(),
    Lambda(lambda x: (x - 0.5) * 2)]
)
dataset = MNIST("./datasets", download=True, train=True, transform=transform)
loader = DataLoader(dataset, batch_size, shuffle=True)
```

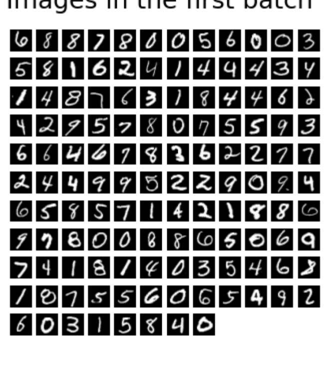
In [4]:
```
"""
This function plots images in a grid. Input is a Tensor.
See show_first_batch to see how it is used.
"""
def show_images(images, title=""):
    images = images.detach().cpu().numpy()
    fig = plt.figure(figsize=(4, 4))
    cols = math.ceil(len(images) ** (1 / 2))
    rows = math.ceil(len(images) / cols)
    for r in range(rows):
        for c in range(cols):
            idx = cols * r + c
            ax = fig.add_subplot(rows, cols, idx + 1)
            ax.axis('off')
            if idx < len(images):
                ax.imshow(images[idx][0], cmap="gray")
    fig.suptitle(title, fontsize=18)
    plt.show()

## we gnerate the 10*n_sample conditioned images, so we resize it to the same size in the plt
def show_images_custom(images, title=""):
    images = images.detach().cpu().numpy()
    fig = plt.figure(figsize=(4, 8))
    rows = 11
    cols = math.ceil(len(images) / rows)
    for r in range(rows):
        for c in range(cols):
            plot_idx = r * cols + c
            ax = fig.add_subplot(rows, cols, plot_idx + 1)
            ax.axis('off')
            img_idx = plot_idx
            if img_idx < len(images):
                ax.imshow(images[img_idx][0], cmap="gray")
    fig.suptitle(title, fontsize=18)
    plt.show()
```

In [5]:
```
def show_first_batch(loader):
    for batch_feats,batch_labels in loader:
        print(batch_labels)
        print(batch_feats.shape)
        show_images(batch_feats, "Images in the first batch")
        break
show_first_batch(loader)
```

```
tensor([6, 8, 8, 7, 8, 0, 0, 5, 6, 0, 0, 3, 5, 8, 1, 6, 2, 4, 1, 4, 9, 4, 3, 4,
        1, 4, 8, 7, 6, 3, 1, 8, 4, 4, 6, 2, 4, 2, 9, 5, 7, 8, 0, 7, 5, 5, 9, 3,
        6, 6, 4, 6, 7, 8, 3, 6, 2, 2, 7, 7, 2, 4, 4, 9, 9, 5, 2, 2, 9, 0, 9, 4,
        6, 5, 8, 5, 7, 1, 4, 2, 1, 8, 8, 6, 9, 7, 8, 0, 0, 8, 8, 6, 5, 0, 6, 9,
        7, 4, 1, 8, 1, 4, 0, 3, 5, 4, 6, 8, 1, 8, 7, 5, 5, 6, 0, 6, 5, 4, 9, 2,
        6, 0, 3, 1, 5, 8, 4, 0])
torch.Size([128, 1, 28, 28])
```

Images in the first batch



Your conditional DDPM implementation should go here

This includes functionality for adding noise to the image.

In [6]:
```
class Forward_pass:
    def __init__(self) -> None:
        #number of steps
        self.N = N
        self.beta_min = BMIN
        self.beta_max = BMAX
        self._betas, self._alphas, self._alpha_bars = (0, 0 ,0)
        self.get_coefs()

    ##linear steps
    def get_coefs(self):
        """Get the coefficients for the noise schedule.
        Returns:
            Tuple[torch.FloatTensor, torch.FloatTensor, torch.FloatTensor]:
                betas, alphas, alpha_bars
        """
        self._betas = torch.linspace(self.beta_min, self.beta_max, self.N)
        self._alphas = 1 - self._betas
        self._alpha_bars = torch.cumprod(self._alphas, dim=0)

    #return the coef for initializing the denoising model
    def ret_coefs(self) -> Tuple[torch.FloatTensor, torch.FloatTensor, torch.FloatTensor]:

        return (self._betas,self._alphas,self._alpha_bars)

    def _q_mean(
        self,
        z: torch.FloatTensor,
        t: torch.LongTensor,
    ) -> torch.FloatTensor:
        # mean of the distribution q(z_t | z_0)
        sqrt_alpha_bars = torch.sqrt(self._alpha_bars[t.tolist()])
        return sqrt_alpha_bars.view(-1,1,1,1) * z

    def _q_std(
        self,
        z: torch.FloatTensor,
        t: torch.LongTensor,
    ) -> torch.FloatTensor:
        # std of the distribution q(z_t | z_0)
        std = torch.sqrt(1.0 - self._alpha_bars[t.tolist()])
        return std.view(-1,1,1,1)

    def q_sample(
        self,
        z: torch.FloatTensor,
        t: torch.LongTensor,
    ) -> Tuple[torch.FloatTensor, torch.FloatTensor]:
        # Sample from q(z_t | z_0)

        mean = self._q_mean(z, t)
        std = self._q_std(z, t)
        epsilon = torch.randn_like(z)
        z_t = epsilon * std + mean

        return z_t, epsilon

    ##for training, sample random n timesteps for each input, return the repeated features, times-steps sampl
    def q_sample_random(
        self,
        z: torch.FloatTensor,
        lables: torch.FloatTensor,
        n_timesteps = 5,
    ) -> Tuple[torch.FloatTensor, torch.FloatTensor,   torch.IntTensor, torch.IntTensor]:
        # Sample from q(z_t | z_0)
        #we use the same times steps for all the samples in this batch for simplicity
        times = list(torch.randint(0, self.N-1, (n_timesteps,)))
        print(times)
        times_list = [torch.Tensor([i]).repeat(z.shape[0]).to(torch.int64) for i in times ]
        times_repeated = torch.cat(times_list)
        lables_repeated = lables.repeat(n_timesteps)
        z_repeated = z.repeat(n_timesteps,1,1,1) #shape=(batch_size*5, 1, 28, 28)
        mean = self._q_mean(z_repeated, times_repeated)  #shape=(batch_size, 1, 28, 28)
        std = self._q_std(z_repeated, times_repeated)
        epsilon = torch.randn_like(z_repeated)

        z_out = epsilon * std + mean

        return z_out, epsilon, times_repeated, lables_repeated


    ##for training, sample randomly one timestep for each img, return the repeated features, times-steps samp
    def q_sample_random2(
        self,
        z: torch.FloatTensor,
        lables: torch.FloatTensor,
    ) -> Tuple[torch.FloatTensor, torch.FloatTensor,  torch.IntTensor, torch.IntTensor]:
        # Sample from q(z_t | z_0)
        #we use the same times steps for all the samples in this batch for simplicity
        times = torch.randint(0, self.N-1, (z.shape[0],))
        mean = self._q_mean(z, times)  #shape=(batch_size, 1, 28, 28)
        std = self._q_std(z, times)
        epsilon = torch.randn_like(z)

        z_out = epsilon * std + mean

        return z_out, epsilon, times, lables
```

The implementation of the neural network used to estimate the noise should go here

The network should make use of both time and context embedding.

Any functions/methods used for time and context embeddings should also go here.

In [7]:
```
## I tried some simpler models but they don't seeem to work well, so finally I use the same Unet implementati
## The difference is we use simple embedding that has only one linear layer and adds to the input image featu
class ResidualConvBlock(torch.nn.Module):
    def __init__(
        self, in_channels: int, out_channels: int, is_res: bool = False
    ) -> None:
        super().__init__()
        '''
        standard ResNet style convolutional block
        '''
        self.same_channels = in_channels==out_channels
        self.is_res = is_res
        self.conv1 = torch.nn.Sequential(
            torch.nn.Conv2d(in_channels, out_channels, 3, 1, 1),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.GELU(),
        )
        self.conv2 = torch.nn.Sequential(
            torch.nn.Conv2d(out_channels, out_channels, 3, 1, 1),
            torch.nn.BatchNorm2d(out_channels),
            torch.nn.GELU(),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        if self.is_res:
            x1 = self.conv1(x)
            x2 = self.conv2(x1)
            # this adds on correct residual in case channels have increased
            if self.same_channels:
                out = x + x2
            else:
                out = x1 + x2
            return out / 1.414
        else:
            x1 = self.conv1(x)
            x2 = self.conv2(x1)
            return x2

class UnetDown(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UnetDown, self).__init__()
        '''
        process and downscale the image feature maps
        '''
        layers = [ResidualConvBlock(in_channels, out_channels), torch.nn.MaxPool2d(2)]
        self.model = torch.nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)


class UnetUp(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UnetUp, self).__init__()
        '''
        process and upscale the image feature maps
        '''
        layers = [
            torch.nn.ConvTranspose2d(in_channels, out_channels, 2, 2),
            ResidualConvBlock(out_channels, out_channels),
            ResidualConvBlock(out_channels, out_channels),
        ]
        self.model = torch.nn.Sequential(*layers)

    def forward(self, x, skip):
        x = torch.cat((x, skip), 1)
        x = self.model(x)
        return x


class EmbedFC(torch.nn.Module):
    def __init__(self, input_dim, emb_dim):
        super(EmbedFC, self).__init__()
        '''
        generic one layer FC NN for embedding things
        '''
        self.input_dim = input_dim
        layers = [
            torch.nn.Linear(input_dim, emb_dim),
            torch.nn.GELU(),
            torch.nn.Linear(emb_dim, emb_dim),
        ]
        self.model = torch.nn.Sequential(*layers)

    def forward(self, x):
        x = x.view(-1, self.input_dim)
        return self.model(x)

class ConditionedBackbone(torch.nn.Module):
    def __init__(self, in_channels = 1, hidden_channels = 256, image_size = 28, n_classes=11, num_timesteps=N
        super(ConditionedBackbone, self).__init__()

        self.in_channels = in_channels
        self.n_feat = hidden_channels
        self.image_size = image_size
        self.num_timesteps = num_timesteps
        self.n_classes = n_classes  #the classes are 0-9 plus a "empty/unconditional" lable, they are encodec

        ## trainable embeddings for lable and time, not working if using a simple linear transformation, so t
        #self.time_emb = torch.nn.Linear(1, self.n_feat*2)
        #self.lable_emb = torch.nn.Linear(self.n_classes, self.n_feat*2)

        self.time_emb = EmbedFC(1, self.n_feat * 2)
        self.lable_emb = EmbedFC(self.n_classes, self.n_feat * 2)

        self.init_conv = ResidualConvBlock(in_channels, self.n_feat, is_res=True)
        self.down1 = UnetDown(self.n_feat, self.n_feat)
        self.down2 = UnetDown(self.n_feat, 2 * self.n_feat)

        self.to_vec = torch.nn.Sequential(torch.nn.AvgPool2d(7), torch.nn.GELU())

        self.up0 = torch.nn.Sequential(
            torch.nn.ConvTranspose2d(6 * self.n_feat, 2 * self.n_feat, 7, 7),
            torch.nn.GroupNorm(8, 2 * self.n_feat),
            torch.nn.ReLU(),
        )

        self.up1 = UnetUp(4 * self.n_feat, self.n_feat)
        self.up2 = UnetUp(2 * self.n_feat, self.n_feat)
        self.out = torch.nn.Sequential(
            torch.nn.Conv2d(2 * self.n_feat, self.n_feat, 3, 1, 1),
            torch.nn.GroupNorm(8, self.n_feat),
            torch.nn.ReLU(),
            torch.nn.Conv2d(self.n_feat, self.in_channels, 3, 1, 1),
        )


    def forward(self, x, lables, t) -> float:
        # x is (noisy) image batch (from the Forward_pass for training and from the N(0,1) for sampling),  (b
        # diff_noise are the corresponding noise in the forward pass,
        #labels are the numbers from 0-10, 10 represents the unconditioned sample, t is the timestep usually


        x = self.init_conv(x)
        down1 = self.down1(x)
        down2 = self.down2(down1)
        hiddenvec = self.to_vec(down2)

        #I tried several embedding methods, the concatenation seems to be the best
        #map to the vector size
        time_emb = self.time_emb(t.type(torch.float32).view(-1,1)).view(-1, self.n_feat * 2, 1 ,1)
        lable_one_hot = torch.nn.functional.one_hot(lables, num_classes=self.n_classes).type(torch.float)
        lable_emb  = self.lable_emb(lable_one_hot).view(-1, self.n_feat * 2, 1 ,1)

        #hiddenvec = hiddenvec + lable_emb

        up1 = self.up0(torch.cat((time_emb, lable_emb, hiddenvec), 1))
        up2 = self.up1(up1, down2)
        up3 = self.up2(up2, down1)
        e_pred = self.out(torch.cat((up3, x), 1))

        return e_pred

class Reverse_pass(torch.nn.Module):
    def __init__(self, model, alphas, alpha_bars, betas, num_classes = 11 ) -> None:
        super().__init__()
        self.model = model.to(device)
        self.N = N
        self.num_classes = num_classes
        ##for computing the samples
        self.alphas = alphas
        self.sqrt_1_minus_alpha_bars = torch.sqrt(1-alpha_bars)
        ##we use the simpler version of sigma for the reverse pass
        self.sigmas = torch.sqrt(betas)

        #loss
        self.mse = torch.nn.MSELoss()

    def sample(self, n_samples = 3, image_size = 28, guidence_w = guidence_w):
        ## sample "n" samples for each number “0-9” (including the unditional label "10") using 'Classifier-F
        with torch.no_grad():
            #sample noised images and arrange labels
            batch_size = n_samples*self.num_classes
            #x_batch = torch.randn(batch_size, 1, image_size, image_size).to(device)
            x_batch = torch.randn(batch_size, 1, image_size, image_size).to(device)
            ##labels = [0,0,0,1,1,1,2,2,2....]
            labels = [ torch.tensor([i], device=device).repeat(n_samples) for i in range(self.num_classes)]
            #labels = torch.cat(labels).to(device)
            labels = torch.cat(labels)
            # unconditioned labels for interpolation (label = 10)
            labels_uncon = torch.full_like(labels, self.num_classes-1)
            labels = torch.cat((labels, labels_uncon))
            last_batch = x_batch
            for i in range(self.N, 0, -1):
                print(f'sampling timestep {i}\n')
                #current_t = torch.tensor([i]).to(device)
                current_t = torch.tensor([i-1],device=device)  #i-1 for one-hot encoding
                t_batch = current_t.repeat(batch_size) #(batch_size)

                # duplicate the current img for unconditioned sampling
                x_doubled = last_batch.repeat(2,1,1,1) #(2*batch_size, channel, img_size, img_size)
                t_doubled = t_batch.repeat(2) #(2*batch_size)

                # predict the diffusion noise, and interpolate
                diff_noise = self.model(x_doubled, labels, t_doubled)
                diff_con = diff_noise[:batch_size]
                diff_uncon = diff_noise[batch_size:]
                diff_noise_final = (1+guidence_w)*diff_con - guidence_w*diff_uncon
                # predict x_t-1
                #observation_noise = torch.randn(batch_size, 1, image_size, image_size).to(device) if i > 1 e
                observation_noise = torch.randn(batch_size, 1, image_size, image_size).to(device) if i > 1 el
                x_predict = (1/torch.sqrt(self.alphas[i-1])) * (last_batch - diff_noise_final * (1 - self.alp

                last_batch = x_predict
            return last_batch

    def compute_loss(self, batch_input, batch_noise, batch_time, batch_lables):
        #a batch of diffused input (by the forward process), with its difuusion noise,  timesteps and labels
        # return the loss of this batch for training the model

        predicted_noise = self.model(batch_input.to(device), batch_lables.to(device), batch_time.to(device))
        return self.mse(batch_noise.to(device), predicted_noise)
```

Show that you can generate images before the model is trained

This should demonstrate that the backwards pass works. The generated images are expected to be noise.

In [8]:
```
model = ConditionedBackbone(hidden_channels = 128)
forward = Forward_pass()
betas,alphas,alpha_bars = forward.ret_coefs()
reverse = Reverse_pass(model, alphas, alpha_bars, betas)
return_batch = reverse.sample(n_samples=5)
show_images_custom(return_batch, "Images generated from the untrained model")
```

sampling timestep 500
sampling timestep 499
sampling timestep 498
sampling timestep 497
sampling timestep 496
sampling timestep 495
sampling timestep 494
sampling timestep 493
sampling timestep 492
sampling timestep 491
sampling timestep 490
sampling timestep 489
sampling timestep 488
sampling timestep 487
sampling timestep 486
sampling timestep 485
sampling timestep 484
sampling timestep 483
sampling timestep 482
sampling timestep 481
sampling timestep 480
sampling timestep 479
sampling timestep 478
sampling timestep 477
sampling timestep 476
sampling timestep 475
sampling timestep 474
sampling timestep 473
sampling timestep 472
sampling timestep 471
sampling timestep 470
sampling timestep 469
sampling timestep 468
sampling timestep 467
sampling timestep 466
sampling timestep 465
sampling timestep 464
sampling timestep 463
sampling timestep 462
sampling timestep 461
sampling timestep 460
sampling timestep 459
sampling timestep 458
sampling timestep 457
sampling timestep 456
sampling timestep 455
sampling timestep 454
sampling timestep 453
sampling timestep 452
sampling timestep 451
sampling timestep 450
sampling timestep 449
sampling timestep 448
sampling timestep 447
sampling timestep 446
sampling timestep 445
sampling timestep 444
sampling timestep 443
sampling timestep 442
sampling timestep 441
sampling timestep 440
sampling timestep 439
sampling timestep 438
sampling timestep 437
sampling timestep 436
sampling timestep 435
sampling timestep 434
sampling timestep 433
sampling timestep 432
sampling timestep 431
sampling timestep 430
sampling timestep 429
sampling timestep 428
sampling timestep 427
sampling timestep 426
sampling timestep 425
sampling timestep 424
sampling timestep 423
sampling timestep 422
sampling timestep 421
sampling timestep 420
sampling timestep 419
sampling timestep 418
sampling timestep 417
sampling timestep 416
sampling timestep 415
sampling timestep 414
sampling timestep 413
sampling timestep 412
sampling timestep 411
sampling timestep 410
sampling timestep 409
sampling timestep 408
sampling timestep 407
sampling timestep 406
sampling timestep 405
sampling timestep 404
sampling timestep 403
sampling timestep 402
sampling timestep 401
sampling timestep 400
sampling timestep 399
sampling timestep 398
sampling timestep 397
sampling timestep 396
sampling timestep 395
sampling timestep 394
sampling timestep 393
sampling timestep 392
sampling timestep 391
sampling timestep 390
sampling timestep 389
sampling timestep 388
sampling timestep 387
sampling timestep 386
sampling timestep 385
sampling timestep 384
sampling timestep 383
sampling timestep 382
sampling timestep 381
sampling timestep 380
sampling timestep 379
sampling timestep 378
sampling timestep 377
sampling timestep 376
sampling timestep 375
sampling timestep 374
sampling timestep 373
sampling timestep 372
sampling timestep 371
sampling timestep 370
sampling timestep 369
sampling timestep 368
sampling timestep 367
sampling timestep 366
sampling timestep 365
sampling timestep 364
sampling timestep 363
sampling timestep 362
sampling timestep 361
sampling timestep 360
sampling timestep 359
sampling timestep 358
sampling timestep 357
sampling timestep 356
sampling timestep 355
sampling timestep 354
sampling timestep 353
sampling timestep 352
sampling timestep 351
sampling timestep 350
sampling timestep 349
sampling timestep 348
sampling timestep 347
sampling timestep 346
sampling timestep 345
sampling timestep 344
sampling timestep 343
sampling timestep 342
sampling timestep 341
sampling timestep 340
sampling timestep 339
sampling timestep 338
sampling timestep 337
sampling timestep 336
sampling timestep 335
sampling timestep 334
sampling timestep 333
sampling timestep 332
sampling timestep 331
sampling timestep 330
sampling timestep 329
sampling timestep 328
sampling timestep 327
sampling timestep 326
sampling timestep 325
sampling timestep 324
sampling timestep 323
sampling timestep 322
sampling timestep 321
sampling timestep 320
sampling timestep 319
sampling timestep 318
sampling timestep 317
sampling timestep 316
sampling timestep 315
sampling timestep 314
sampling timestep 313
sampling timestep 312
sampling timestep 311
sampling timestep 310
sampling timestep 309
sampling timestep 308
sampling timestep 307
sampling timestep 306
sampling timestep 305
sampling timestep 304
sampling timestep 303
sampling timestep 302
sampling timestep 301
sampling timestep 300
sampling timestep 299
sampling timestep 298
sampling timestep 297
sampling timestep 296
sampling timestep 295
sampling timestep 294
sampling timestep 293
sampling timestep 292
sampling timestep 291
sampling timestep 290
sampling timestep 289
sampling timestep 288
sampling timestep 287
sampling timestep 286
sampling timestep 285
sampling timestep 284
sampling timestep 283
sampling timestep 282
sampling timestep 281
sampling timestep 280
sampling timestep 279
sampling timestep 278
sampling timestep 277
sampling timestep 276
sampling timestep 275
sampling timestep 274
sampling timestep 273
sampling timestep 272
sampling timestep 271
sampling timestep 270
sampling timestep 269
sampling timestep 268
sampling timestep 267
sampling timestep 266
sampling timestep 265
sampling timestep 264
sampling timestep 263
sampling timestep 262
sampling timestep 261
sampling timestep 260
sampling timestep 259
sampling timestep 258
sampling timestep 257
sampling timestep 256
sampling timestep 255
sampling timestep 254
sampling timestep 253
sampling timestep 252
sampling timestep 251
sampling timestep 250
sampling timestep 249
sampling timestep 248
sampling timestep 247
sampling timestep 246
sampling timestep 245
sampling timestep 244
sampling timestep 243
sampling timestep 242
sampling timestep 241
sampling timestep 240
sampling timestep 239
sampling timestep 238
sampling timestep 237
sampling timestep 236
sampling timestep 235
sampling timestep 234
sampling timestep 233
sampling timestep 232
sampling timestep 231
sampling timestep 230
sampling timestep 229
sampling timestep 228
sampling timestep 227
sampling timestep 226
sampling timestep 225
sampling timestep 224
sampling timestep 223
sampling timestep 222
sampling timestep 221
sampling timestep 220
sampling timestep 219
sampling timestep 218
sampling timestep 217
sampling timestep 216
sampling timestep 215
sampling timestep 214
sampling timestep 213
sampling timestep 212
sampling timestep 211
sampling timestep 210
sampling timestep 209
sampling timestep 208
sampling timestep 207
sampling timestep 206
sampling timestep 205
sampling timestep 204
sampling timestep 203
sampling timestep 202
sampling timestep 201
sampling timestep 200
sampling timestep 199
sampling timestep 198
sampling timestep 197
sampling timestep 196
sampling timestep 195
sampling timestep 194
sampling timestep 193
sampling timestep 192
sampling timestep 191
sampling timestep 190
sampling timestep 189
sampling timestep 188
sampling timestep 187
sampling timestep 186
sampling timestep 185
sampling timestep 184
sampling timestep 183
sampling timestep 182
sampling timestep 181
sampling timestep 180
sampling timestep 179
sampling timestep 178
sampling timestep 177
sampling timestep 176
sampling timestep 175
sampling timestep 174
sampling timestep 173
sampling timestep 172
sampling timestep 171
sampling timestep 170
sampling timestep 169
sampling timestep 168
sampling timestep 167
sampling timestep 166
sampling timestep 165
sampling timestep 164
sampling timestep 163
sampling timestep 162
sampling timestep 161
sampling timestep 160
sampling timestep 159
sampling timestep 158
sampling timestep 157
sampling timestep 156
sampling timestep 155
sampling timestep 154
sampling timestep 153
sampling timestep 152
sampling timestep 151
sampling timestep 150
sampling timestep 149
sampling timestep 148
sampling timestep 147
sampling timestep 146
sampling timestep 145
sampling timestep 144
sampling timestep 143
sampling timestep 142
sampling timestep 141
sampling timestep 140
sampling timestep 139
sampling timestep 138
sampling timestep 137
sampling timestep 136
sampling timestep 135
sampling timestep 134
sampling timestep 133
sampling timestep 132
sampling timestep 131
sampling timestep 130
sampling timestep 129
sampling timestep 128
sampling timestep 127
sampling timestep 126
sampling timestep 125
sampling timestep 124
sampling timestep 123
sampling timestep 122
sampling timestep 121
sampling timestep 120
sampling timestep 119
sampling timestep 118
sampling timestep 117
sampling timestep 116
sampling timestep 115
sampling timestep 114
sampling timestep 113
sampling timestep 112
sampling timestep 111
sampling timestep 110
sampling timestep 109
sampling timestep 108
sampling timestep 107
sampling timestep 106
sampling timestep 105
sampling timestep 104
sampling timestep 103
sampling timestep 102
sampling timestep 101
sampling timestep 100
sampling timestep 99
sampling timestep 98
sampling timestep 97
sampling timestep 96
sampling timestep 95
sampling timestep 94
sampling timestep 93
sampling timestep 92
sampling timestep 91
sampling timestep 90
sampling timestep 89
sampling timestep 88
sampling timestep 87
sampling timestep 86
sampling timestep 85
sampling timestep 84
sampling timestep 83
sampling timestep 82
sampling timestep 81
sampling timestep 80

```
sampling timestep 79
sampling timestep 78
sampling timestep 77
sampling timestep 76
sampling timestep 75
sampling timestep 74
sampling timestep 73
sampling timestep 72
sampling timestep 71
sampling timestep 70
sampling timestep 69
sampling timestep 68
sampling timestep 67
sampling timestep 66
sampling timestep 65
sampling timestep 64
sampling timestep 63
sampling timestep 62
sampling timestep 61
sampling timestep 60
sampling timestep 59
sampling timestep 58
sampling timestep 57
sampling timestep 56
sampling timestep 55
sampling timestep 54
sampling timestep 53
sampling timestep 52
sampling timestep 51
sampling timestep 50
sampling timestep 49
sampling timestep 48
sampling timestep 47
sampling timestep 46
sampling timestep 45
sampling timestep 44
sampling timestep 43
sampling timestep 42
sampling timestep 41
sampling timestep 40
sampling timestep 39
sampling timestep 38
sampling timestep 37
sampling timestep 36
sampling timestep 35
sampling timestep 34
sampling timestep 33
sampling timestep 32
sampling timestep 31
sampling timestep 30
sampling timestep 29
sampling timestep 28
sampling timestep 27
sampling timestep 26
sampling timestep 25
sampling timestep 24
sampling timestep 23
sampling timestep 22
sampling timestep 21
sampling timestep 20
sampling timestep 19
sampling timestep 18
sampling timestep 17
sampling timestep 16
sampling timestep 15
sampling timestep 14
sampling timestep 13
sampling timestep 12
sampling timestep 11
sampling timestep 10
sampling timestep 9
sampling timestep 8
sampling timestep 7
sampling timestep 6
sampling timestep 5
sampling timestep 4
sampling timestep 3
sampling timestep 2
sampling timestep 1
```

Images generated from the untrained model

Implement training loop

Train the model here. There should be some indication of how long the model took to train, both total and per epoch. For good results you will want to train the model for several epochs, but with a good implementation you should expect to see something that looks like digits after only a single epoch.

In [9]:

```python
def train_conditionedDDPM(n_epochs = 10, learning_rate = 1e-4):
    print(f"{sum(p.numel() for p in reverse.parameters() if p.requires_grad)} parameters to train")
    optimizer = torch.optim.Adam(reverse.parameters(), lr=learning_rate)
    for i in range(n_epochs):
        print(f"Training epoch: {i+1}")
        progress_loader = tqdm(loader)
        total_loss = 0
        for batch_feats,batch_labels in progress_loader:
            optimizer.zero_grad()
            #sample with forward pass
            #z_out, epsilon_target, times, labels= forward.q_sample_random(batch_feats, batch_labels)
            z_out, epsilon_target, times, labels= forward.q_sample_random2(batch_feats, batch_labels)
            loss = reverse.compute_loss(z_out, epsilon_target, times, labels)
            total_loss += loss
            loss.backward()
            optimizer.step()

        print(f'Loss for the current epoch is {total_loss}')
```

Train and visualize the model

We want to see several generated examples of each digit.

In [10]:

```python
model = ConditionedBackbone(hidden_channels = 128)
forward = Forward_pass()
betas,alphas,alpha_bars = forward.ret_coefs()
reverse = Reverse_pass(model, alphas, alpha_bars, betas)
train_conditionedDDPM(n_epochs = 10)
```

```
12973697 parameters to train
Training epoch: 1
100%|██████████| 469/469 [00:27<00:00, 17.12it/s]
Loss for the current epoch is 43.41958236694336
Training epoch: 2
100%|██████████| 469/469 [00:27<00:00, 16.85it/s]
Loss for the current epoch is 24.419374465942383
Training epoch: 3
100%|██████████| 469/469 [00:28<00:00, 16.44it/s]
Loss for the current epoch is 21.434226989746094
Training epoch: 4
100%|██████████| 469/469 [00:30<00:00, 15.54it/s]
Loss for the current epoch is 19.79697608947754
Training epoch: 5
100%|██████████| 469/469 [00:31<00:00, 15.00it/s]
Loss for the current epoch is 18.852949142456055
Training epoch: 6
100%|██████████| 469/469 [00:30<00:00, 15.20it/s]
Loss for the current epoch is 18.14109992980957
Training epoch: 7
100%|██████████| 469/469 [00:28<00:00, 16.37it/s]
Loss for the current epoch is 17.489013671875
Training epoch: 8
100%|██████████| 469/469 [00:28<00:00, 16.67it/s]
Loss for the current epoch is 17.0726318359375
Training epoch: 9
100%|██████████| 469/469 [00:28<00:00, 16.75it/s]
Loss for the current epoch is 16.6922550201416
Training epoch: 10
100%|██████████| 469/469 [00:27<00:00, 16.78it/s]
Loss for the current epoch is 16.3807373046875
```

In [12]:

```python
return_batch1 = reverse.sample( n_samples = 5, guidence_w = 0.5)
show_images_custom(return_batch1, "Imgs from 0-9, the last row is the unconditioned sampling")
```

```
sampling timestep 500
sampling timestep 499
sampling timestep 498
sampling timestep 497
sampling timestep 496
sampling timestep 495
sampling timestep 494
sampling timestep 493
sampling timestep 492
sampling timestep 491
sampling timestep 490
sampling timestep 489
sampling timestep 488
sampling timestep 487
sampling timestep 486
sampling timestep 485
sampling timestep 484
sampling timestep 483
sampling timestep 482
sampling timestep 481
sampling timestep 480
sampling timestep 479
sampling timestep 478
sampling timestep 477
sampling timestep 476
sampling timestep 475
sampling timestep 474
sampling timestep 473
sampling timestep 472
sampling timestep 471
sampling timestep 470
sampling timestep 469
sampling timestep 468
sampling timestep 467
sampling timestep 466
sampling timestep 465
sampling timestep 464
sampling timestep 463
sampling timestep 462
sampling timestep 461
sampling timestep 460
sampling timestep 459
sampling timestep 458
sampling timestep 457
sampling timestep 456
sampling timestep 455
sampling timestep 454
sampling timestep 453
sampling timestep 452
sampling timestep 451
sampling timestep 450
sampling timestep 449
sampling timestep 448
sampling timestep 447
sampling timestep 446
sampling timestep 445
sampling timestep 444
sampling timestep 443
sampling timestep 442
sampling timestep 441
sampling timestep 440
sampling timestep 439
sampling timestep 438
sampling timestep 437
sampling timestep 436
sampling timestep 435
sampling timestep 434
sampling timestep 433
sampling timestep 432
sampling timestep 431
sampling timestep 430
sampling timestep 429
sampling timestep 428
sampling timestep 427
sampling timestep 426
sampling timestep 425
sampling timestep 424
sampling timestep 423
sampling timestep 422
sampling timestep 421
sampling timestep 420
sampling timestep 419
sampling timestep 418
sampling timestep 417
sampling timestep 416
sampling timestep 415
sampling timestep 414
sampling timestep 413
sampling timestep 412
sampling timestep 411
sampling timestep 410
sampling timestep 409
sampling timestep 408
sampling timestep 407
sampling timestep 406
sampling timestep 405
sampling timestep 404
sampling timestep 403
sampling timestep 402
sampling timestep 401
sampling timestep 400
sampling timestep 399
sampling timestep 398
sampling timestep 397
sampling timestep 396
sampling timestep 395
sampling timestep 394
sampling timestep 393
sampling timestep 392
sampling timestep 391
sampling timestep 390
sampling timestep 389
sampling timestep 388
sampling timestep 387
sampling timestep 386
sampling timestep 385
sampling timestep 384
sampling timestep 383
sampling timestep 382
sampling timestep 381
sampling timestep 380
sampling timestep 379
sampling timestep 378
sampling timestep 377
sampling timestep 376
sampling timestep 375
sampling timestep 374
sampling timestep 373
sampling timestep 372
sampling timestep 371
sampling timestep 370
sampling timestep 369
sampling timestep 368
sampling timestep 367
sampling timestep 366
sampling timestep 365
sampling timestep 364
sampling timestep 363
sampling timestep 362
sampling timestep 361
sampling timestep 360
sampling timestep 359
sampling timestep 358
sampling timestep 357
sampling timestep 356
sampling timestep 355
sampling timestep 354
sampling timestep 353
sampling timestep 352
sampling timestep 351
sampling timestep 350
sampling timestep 349
sampling timestep 348
sampling timestep 347
sampling timestep 346
sampling timestep 345
sampling timestep 344
sampling timestep 343
sampling timestep 342
sampling timestep 341
sampling timestep 340
sampling timestep 339
sampling timestep 338
sampling timestep 337
sampling timestep 336
sampling timestep 335
sampling timestep 334
sampling timestep 333
sampling timestep 332
sampling timestep 331
sampling timestep 330
sampling timestep 329
sampling timestep 328
sampling timestep 327
sampling timestep 326
sampling timestep 325
sampling timestep 324
sampling timestep 323
sampling timestep 322
sampling timestep 321
sampling timestep 320
sampling timestep 319
sampling timestep 318
sampling timestep 317
sampling timestep 316
sampling timestep 315
sampling timestep 314
sampling timestep 313
sampling timestep 312
sampling timestep 311
sampling timestep 310
sampling timestep 309
sampling timestep 308
sampling timestep 307
sampling timestep 306
sampling timestep 305
sampling timestep 304
sampling timestep 303
sampling timestep 302
sampling timestep 301
sampling timestep 300
sampling timestep 299
sampling timestep 298
sampling timestep 297
sampling timestep 296
sampling timestep 295
sampling timestep 294
sampling timestep 293
sampling timestep 292
sampling timestep 291
sampling timestep 290
sampling timestep 289
sampling timestep 288
sampling timestep 287
sampling timestep 286
sampling timestep 285
sampling timestep 284
sampling timestep 283
sampling timestep 282
sampling timestep 281
sampling timestep 280
sampling timestep 279
sampling timestep 278
sampling timestep 277
sampling timestep 276
sampling timestep 275
sampling timestep 274
sampling timestep 273
sampling timestep 272
sampling timestep 271
sampling timestep 270
sampling timestep 269
sampling timestep 268
sampling timestep 267
sampling timestep 266
sampling timestep 265
sampling timestep 264
sampling timestep 263
sampling timestep 262
sampling timestep 261
sampling timestep 260
sampling timestep 259
sampling timestep 258
sampling timestep 257
sampling timestep 256
sampling timestep 255
sampling timestep 254
sampling timestep 253
sampling timestep 252
sampling timestep 251
sampling timestep 250
sampling timestep 249
sampling timestep 248
sampling timestep 247
sampling timestep 246
sampling timestep 245
sampling timestep 244
sampling timestep 243
sampling timestep 242
sampling timestep 241
sampling timestep 240
sampling timestep 239
sampling timestep 238
sampling timestep 237
sampling timestep 236
sampling timestep 235
sampling timestep 234
sampling timestep 233
sampling timestep 232
sampling timestep 231
sampling timestep 230
sampling timestep 229
sampling timestep 228
sampling timestep 227
sampling timestep 226
sampling timestep 225
sampling timestep 224
sampling timestep 223
sampling timestep 222
sampling timestep 221
sampling timestep 220
sampling timestep 219
sampling timestep 218
sampling timestep 217
sampling timestep 216
sampling timestep 215
sampling timestep 214
sampling timestep 213
sampling timestep 212
sampling timestep 211
sampling timestep 210
sampling timestep 209
sampling timestep 208
sampling timestep 207
sampling timestep 206
sampling timestep 205
sampling timestep 204
sampling timestep 203
sampling timestep 202
sampling timestep 201
sampling timestep 200
sampling timestep 199
sampling timestep 198
sampling timestep 197
sampling timestep 196
sampling timestep 195
sampling timestep 194
sampling timestep 193
sampling timestep 192
sampling timestep 191
sampling timestep 190
sampling timestep 189
sampling timestep 188
sampling timestep 187
sampling timestep 186
sampling timestep 185
sampling timestep 184
sampling timestep 183
sampling timestep 182
sampling timestep 181
sampling timestep 180
sampling timestep 179
sampling timestep 178
sampling timestep 177
sampling timestep 176
sampling timestep 175
sampling timestep 174
sampling timestep 173
sampling timestep 172
sampling timestep 171
sampling timestep 170
sampling timestep 169
sampling timestep 168
sampling timestep 167
sampling timestep 166
sampling timestep 165
sampling timestep 164
sampling timestep 163
sampling timestep 162
sampling timestep 161
sampling timestep 160
sampling timestep 159
sampling timestep 158
sampling timestep 157
sampling timestep 156
sampling timestep 155
sampling timestep 154
sampling timestep 153
sampling timestep 152
sampling timestep 151
sampling timestep 150
sampling timestep 149
sampling timestep 148
sampling timestep 147
sampling timestep 146
sampling timestep 145
sampling timestep 144
sampling timestep 143
sampling timestep 142
sampling timestep 141
sampling timestep 140
sampling timestep 139
sampling timestep 138
sampling timestep 137
sampling timestep 136
sampling timestep 135
sampling timestep 134
sampling timestep 133
sampling timestep 132
sampling timestep 131
sampling timestep 130
sampling timestep 129
sampling timestep 128
sampling timestep 127
sampling timestep 126
sampling timestep 125
sampling timestep 124
sampling timestep 123
sampling timestep 122
sampling timestep 121
sampling timestep 120
sampling timestep 119
sampling timestep 118
sampling timestep 117
sampling timestep 116
sampling timestep 115
sampling timestep 114
sampling timestep 113
sampling timestep 112
sampling timestep 111
sampling timestep 110
sampling timestep 109
sampling timestep 108
sampling timestep 107
sampling timestep 106
sampling timestep 105
sampling timestep 104
sampling timestep 103
sampling timestep 102
sampling timestep 101
sampling timestep 100
sampling timestep 99
sampling timestep 98
sampling timestep 97
sampling timestep 96
sampling timestep 95
sampling timestep 94
sampling timestep 93
sampling timestep 92
sampling timestep 91
sampling timestep 90
sampling timestep 89
sampling timestep 88
sampling timestep 87
sampling timestep 86
sampling timestep 85
sampling timestep 84
sampling timestep 83
sampling timestep 82
sampling timestep 81
sampling timestep 80
```

```
sampling timestep 79
sampling timestep 78
sampling timestep 77
sampling timestep 76
sampling timestep 75
sampling timestep 74
sampling timestep 73
sampling timestep 72
sampling timestep 71
sampling timestep 70
sampling timestep 69
sampling timestep 68
sampling timestep 67
sampling timestep 66
sampling timestep 65
sampling timestep 64
sampling timestep 63
sampling timestep 62
sampling timestep 61
sampling timestep 60
sampling timestep 59
sampling timestep 58
sampling timestep 57
sampling timestep 56
sampling timestep 55
sampling timestep 54
sampling timestep 53
sampling timestep 52
sampling timestep 51
sampling timestep 50
sampling timestep 49
sampling timestep 48
sampling timestep 47
sampling timestep 46
sampling timestep 45
sampling timestep 44
sampling timestep 43
sampling timestep 42
sampling timestep 41
sampling timestep 40
sampling timestep 39
sampling timestep 38
sampling timestep 37
sampling timestep 36
sampling timestep 35
sampling timestep 34
sampling timestep 33
sampling timestep 32
sampling timestep 31
sampling timestep 30
sampling timestep 29
sampling timestep 28
sampling timestep 27
sampling timestep 26
sampling timestep 25
sampling timestep 24
sampling timestep 23
sampling timestep 22
sampling timestep 21
sampling timestep 20
sampling timestep 19
sampling timestep 18
sampling timestep 17
sampling timestep 16
sampling timestep 15
sampling timestep 14
sampling timestep 13
sampling timestep 12
sampling timestep 11
sampling timestep 10
sampling timestep 9
sampling timestep 8
sampling timestep 7
sampling timestep 6
sampling timestep 5
sampling timestep 4
sampling timestep 3
sampling timestep 2
sampling timestep 1
```

Imgs from 0-9, the last row is the unconditioned sampling