1) **A link to your public google colab notebook.**

https://colab.research.google.com/drive/1Zi7hDP8T_y7vkyEzhAkGf0Aqy0hhvZ64?usp=sharing

2) **Explanation of your GAN model, design decisions, training-test dataset descriptions and what other factors were considered to improve your model.**

GAN model

- Discriminator:

The discriminator is responsible for distinguishing between real and fake images. It takes an image as input and passes it through a series of fully connected layers (fc1, fc2, fc3) with 0.2 negative slope parameter in leaky ReLU activation and 0.3 probability for dropout. Finally, after the fc4 fully connected layer, it outputs a probability value between 0 and 1 using the sigmoid activation function.

- Generator:

The generator takes random noise (latent vector) as input and generates synthetic images. It also consists of fully connected layers (fc1, fc2, fc3) with 0.2 negative slope parameter leaky ReLU activation. After it is sent through the fc4 fully connected layer and output layer uses the hyperbolic tangent (tanh) activation function to ensure the pixel values of the generated images are in the range [-1, 1] (Normalise form as our dataset since we get it using normalise transform).

- Hyperparameters:

The hyperparameters include the learning rate (lr), latent space dimension (z_dim), image dimension (img_dm), batch size, and number of epochs.

Design Decisions

I have experimented with different architectures for the discriminator and generator. Adding more fully connected layers, gave better results. Leaky ReLU is useful when the data has noise or outliers because it retains and utilizes potentially important information from negative input values, which can improve performance compared to ReLU. Number of epochs is decided after the Generator and Discriminator become more stable. Tuning the learning rate, number of epochs can impact the training process and the quality of generated images. Also adding dropout can reduce the overfitting of models. Adam optimizer is used for both the discriminator and generator models. The binary cross-entropy loss function (BCELoss) is employed to compute the adversarial losses for both the discriminator and generator.Training process involves alternating updates between the discriminator and generator. For each epoch, the code iterates through the batches of real images. The discriminator is trained to maximise the probability of correctly classifying real images and fake images generated by the generator. The generator is trained to generate images that can deceive the discriminator into classifying them as real.

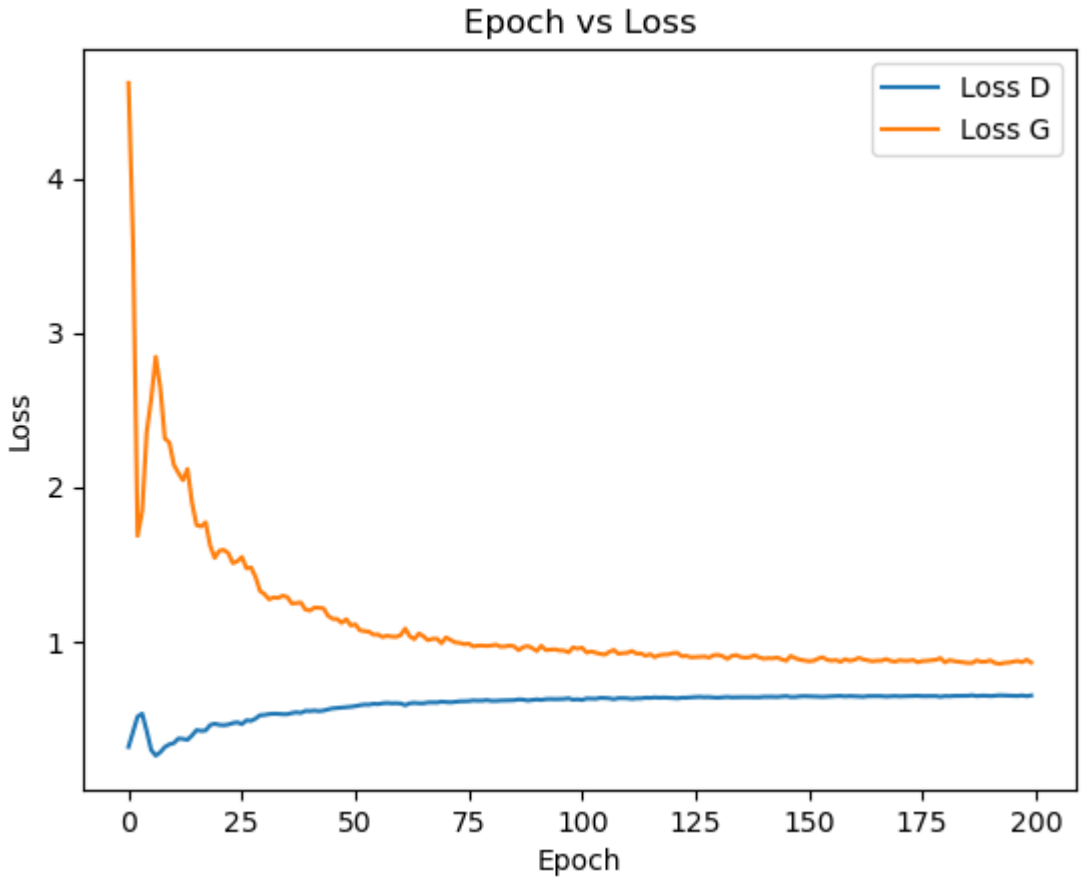
Training-test dataset descriptions

MNIST dataset, which is a collection of hand-drawn digits from 0 to 9. The dataset is loaded using PyTorch's MNIST class and transformed into tensors. The training data is split into batches using the DataLoader class. Also I used Normalise transformation on this dataset (mean = 0.5, sd=0.5).

Other factors were considered to improve your model

I have tried several convolutional layers in both Generator and Discriminator but it seems generator loss was not going to converge. After I implemented Generator and Discriminator with fully connected layers with 30% dropouts and 0.2 LeakyRelu activation function.

3) **Training accuracy of the GAN model at the end of each epoch.**




```
Epoch [0/200] Loss D: 0.3154, Loss G: 4.6198
Epoch [1/200] Loss D: 0.4153, Loss G: 3.5898
Epoch [2/200] Loss D: 0.5116, Loss G: 1.6837
Epoch [3/200] Loss D: 0.5307, Loss G: 1.8419
Epoch [4/200] Loss D: 0.4199, Loss G: 2.3545
Epoch [5/200] Loss D: 0.2956, Loss G: 2.5703
Epoch [6/200] Loss D: 0.2585, Loss G: 2.8448
Epoch [7/200] Loss D: 0.2807, Loss G: 2.6436
```

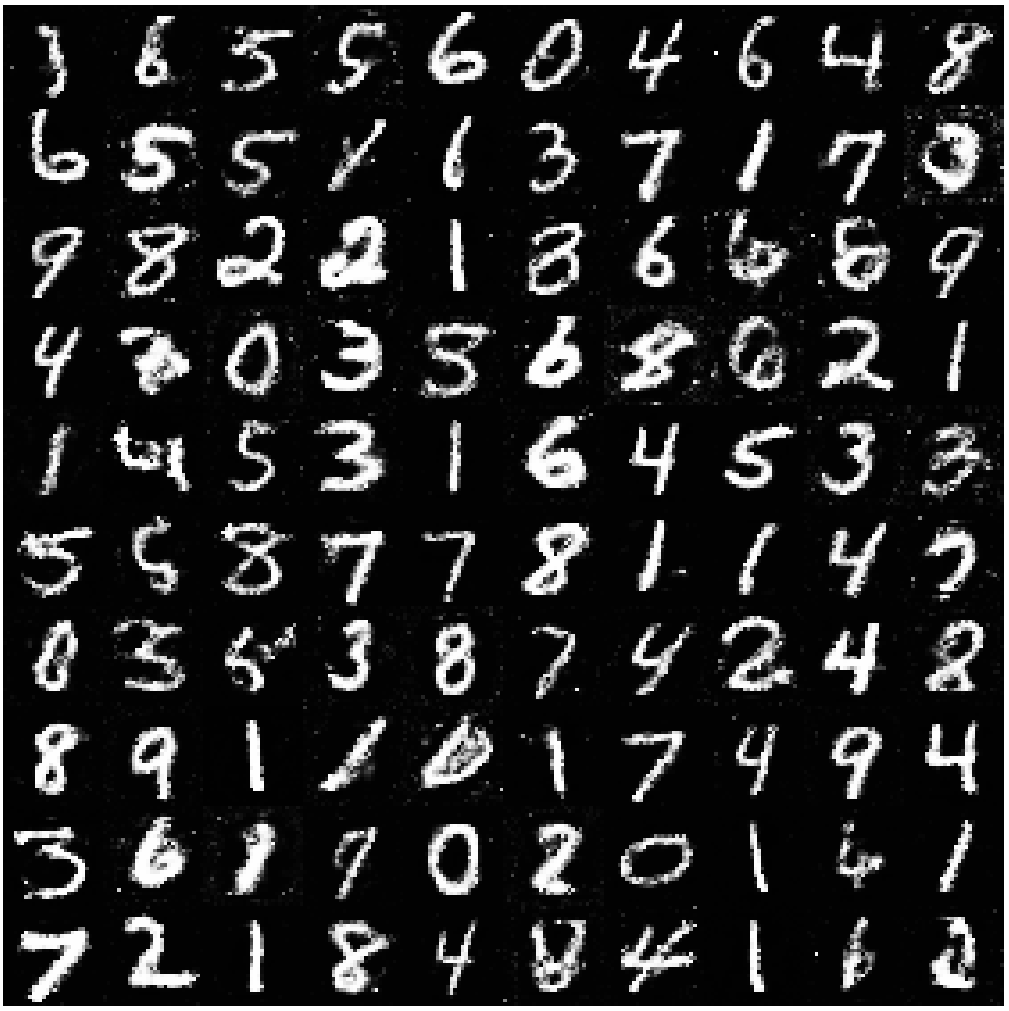
```
Epoch [8/200] Loss D: 0.3143, Loss G: 2.3152
Epoch [9/200] Loss D: 0.3315, Loss G: 2.2887
Epoch [10/200] Loss D: 0.3402, Loss G: 2.1441
Epoch [11/200] Loss D: 0.3700, Loss G: 2.0912
Epoch [12/200] Loss D: 0.3670, Loss G: 2.0441
Epoch [13/200] Loss D: 0.3616, Loss G: 2.1175
Epoch [14/200] Loss D: 0.3895, Loss G: 1.8968
Epoch [15/200] Loss D: 0.4244, Loss G: 1.7538
Epoch [16/200] Loss D: 0.4193, Loss G: 1.7446
Epoch [17/200] Loss D: 0.4215, Loss G: 1.7711
Epoch [18/200] Loss D: 0.4570, Loss G: 1.6194
Epoch [19/200] Loss D: 0.4657, Loss G: 1.5409
Epoch [20/200] Loss D: 0.4582, Loss G: 1.5862
Epoch [21/200] Loss D: 0.4560, Loss G: 1.5928
Epoch [22/200] Loss D: 0.4603, Loss G: 1.5710
Epoch [23/200] Loss D: 0.4697, Loss G: 1.5064
Epoch [24/200] Loss D: 0.4748, Loss G: 1.5192
Epoch [25/200] Loss D: 0.4630, Loss G: 1.5466
Epoch [26/200] Loss D: 0.4881, Loss G: 1.4734
Epoch [27/200] Loss D: 0.4843, Loss G: 1.4786
Epoch [28/200] Loss D: 0.4989, Loss G: 1.4177
Epoch [29/200] Loss D: 0.5206, Loss G: 1.3261
Epoch [30/200] Loss D: 0.5227, Loss G: 1.3049
Epoch [31/200] Loss D: 0.5283, Loss G: 1.2720
Epoch [32/200] Loss D: 0.5304, Loss G: 1.2847
Epoch [33/200] Loss D: 0.5297, Loss G: 1.2814
Epoch [34/200] Loss D: 0.5275, Loss G: 1.2952
Epoch [35/200] Loss D: 0.5272, Loss G: 1.2872
Epoch [36/200] Loss D: 0.5349, Loss G: 1.2450
Epoch [37/200] Loss D: 0.5397, Loss G: 1.2484
Epoch [38/200] Loss D: 0.5359, Loss G: 1.2510
Epoch [39/200] Loss D: 0.5490, Loss G: 1.2058
Epoch [40/200] Loss D: 0.5479, Loss G: 1.1993
Epoch [41/200] Loss D: 0.5509, Loss G: 1.2182
Epoch [42/200] Loss D: 0.5473, Loss G: 1.2172
Epoch [43/200] Loss D: 0.5508, Loss G: 1.2135
Epoch [44/200] Loss D: 0.5587, Loss G: 1.1650
Epoch [45/200] Loss D: 0.5649, Loss G: 1.1457
Epoch [46/200] Loss D: 0.5673, Loss G: 1.1445
Epoch [47/200] Loss D: 0.5702, Loss G: 1.1195
Epoch [48/200] Loss D: 0.5728, Loss G: 1.1433
Epoch [49/200] Loss D: 0.5754, Loss G: 1.1028
Epoch [50/200] Loss D: 0.5797, Loss G: 1.1101
Epoch [51/200] Loss D: 0.5856, Loss G: 1.0714
Epoch [52/200] Loss D: 0.5899, Loss G: 1.0661
Epoch [53/200] Loss D: 0.5891, Loss G: 1.0623
Epoch [54/200] Loss D: 0.5949, Loss G: 1.0433
Epoch [55/200] Loss D: 0.5929, Loss G: 1.0438
Epoch [56/200] Loss D: 0.5978, Loss G: 1.0281
Epoch [57/200] Loss D: 0.6004, Loss G: 1.0357
Epoch [58/200] Loss D: 0.5976, Loss G: 1.0307
```

```
Epoch [59/200] Loss D: 0.5987, Loss G: 1.0289
Epoch [60/200] Loss D: 0.5968, Loss G: 1.0404
Epoch [61/200] Loss D: 0.5857, Loss G: 1.0824
Epoch [62/200] Loss D: 0.5977, Loss G: 1.0330
Epoch [63/200] Loss D: 0.6009, Loss G: 1.0138
Epoch [64/200] Loss D: 0.5981, Loss G: 1.0503
Epoch [65/200] Loss D: 0.5978, Loss G: 1.0329
Epoch [66/200] Loss D: 0.6033, Loss G: 1.0075
Epoch [67/200] Loss D: 0.6041, Loss G: 1.0155
Epoch [68/200] Loss D: 0.6032, Loss G: 1.0173
Epoch [69/200] Loss D: 0.6085, Loss G: 0.9875
Epoch [70/200] Loss D: 0.6060, Loss G: 1.0251
Epoch [71/200] Loss D: 0.6042, Loss G: 1.0087
Epoch [72/200] Loss D: 0.6081, Loss G: 0.9928
Epoch [73/200] Loss D: 0.6107, Loss G: 0.9902
Epoch [74/200] Loss D: 0.6119, Loss G: 0.9810
Epoch [75/200] Loss D: 0.6124, Loss G: 0.9850
Epoch [76/200] Loss D: 0.6166, Loss G: 0.9670
Epoch [77/200] Loss D: 0.6150, Loss G: 0.9717
Epoch [78/200] Loss D: 0.6154, Loss G: 0.9721
Epoch [79/200] Loss D: 0.6189, Loss G: 0.9701
Epoch [80/200] Loss D: 0.6139, Loss G: 0.9726
Epoch [81/200] Loss D: 0.6139, Loss G: 0.9778
Epoch [82/200] Loss D: 0.6161, Loss G: 0.9652
Epoch [83/200] Loss D: 0.6164, Loss G: 0.9668
Epoch [84/200] Loss D: 0.6175, Loss G: 0.9720
Epoch [85/200] Loss D: 0.6183, Loss G: 0.9673
Epoch [86/200] Loss D: 0.6230, Loss G: 0.9427
Epoch [87/200] Loss D: 0.6217, Loss G: 0.9621
Epoch [88/200] Loss D: 0.6160, Loss G: 0.9692
Epoch [89/200] Loss D: 0.6190, Loss G: 0.9559
Epoch [90/200] Loss D: 0.6232, Loss G: 0.9370
Epoch [91/200] Loss D: 0.6199, Loss G: 0.9707
Epoch [92/200] Loss D: 0.6248, Loss G: 0.9431
Epoch [93/200] Loss D: 0.6259, Loss G: 0.9452
Epoch [94/200] Loss D: 0.6246, Loss G: 0.9479
Epoch [95/200] Loss D: 0.6258, Loss G: 0.9408
Epoch [96/200] Loss D: 0.6263, Loss G: 0.9400
Epoch [97/200] Loss D: 0.6301, Loss G: 0.9286
Epoch [98/200] Loss D: 0.6213, Loss G: 0.9601
Epoch [99/200] Loss D: 0.6238, Loss G: 0.9523
Epoch [100/200] Loss D: 0.6200, Loss G: 0.9583
Epoch [101/200] Loss D: 0.6292, Loss G: 0.9273
Epoch [102/200] Loss D: 0.6280, Loss G: 0.9341
Epoch [103/200] Loss D: 0.6274, Loss G: 0.9260
Epoch [104/200] Loss D: 0.6330, Loss G: 0.9184
Epoch [105/200] Loss D: 0.6325, Loss G: 0.9138
Epoch [106/200] Loss D: 0.6302, Loss G: 0.9313
Epoch [107/200] Loss D: 0.6247, Loss G: 0.9437
Epoch [108/200] Loss D: 0.6315, Loss G: 0.9191
Epoch [109/200] Loss D: 0.6317, Loss G: 0.9233
```

```
Epoch [110/200] Loss D: 0.6303, Loss G: 0.9253
Epoch [111/200] Loss D: 0.6267, Loss G: 0.9370
Epoch [112/200] Loss D: 0.6300, Loss G: 0.9203
Epoch [113/200] Loss D: 0.6303, Loss G: 0.9207
Epoch [114/200] Loss D: 0.6344, Loss G: 0.9064
Epoch [115/200] Loss D: 0.6329, Loss G: 0.9163
Epoch [116/200] Loss D: 0.6362, Loss G: 0.8978
Epoch [117/200] Loss D: 0.6340, Loss G: 0.9100
Epoch [118/200] Loss D: 0.6353, Loss G: 0.9136
Epoch [119/200] Loss D: 0.6342, Loss G: 0.9146
Epoch [120/200] Loss D: 0.6324, Loss G: 0.9230
Epoch [121/200] Loss D: 0.6304, Loss G: 0.9247
Epoch [122/200] Loss D: 0.6343, Loss G: 0.9030
Epoch [123/200] Loss D: 0.6344, Loss G: 0.9059
Epoch [124/200] Loss D: 0.6370, Loss G: 0.8961
Epoch [125/200] Loss D: 0.6384, Loss G: 0.8976
Epoch [126/200] Loss D: 0.6396, Loss G: 0.8980
Epoch [127/200] Loss D: 0.6375, Loss G: 0.9013
Epoch [128/200] Loss D: 0.6384, Loss G: 0.8944
Epoch [129/200] Loss D: 0.6357, Loss G: 0.9096
Epoch [130/200] Loss D: 0.6345, Loss G: 0.9119
Epoch [131/200] Loss D: 0.6361, Loss G: 0.9032
Epoch [132/200] Loss D: 0.6391, Loss G: 0.8857
Epoch [133/200] Loss D: 0.6367, Loss G: 0.9053
Epoch [134/200] Loss D: 0.6373, Loss G: 0.9106
Epoch [135/200] Loss D: 0.6383, Loss G: 0.8978
Epoch [136/200] Loss D: 0.6380, Loss G: 0.8942
Epoch [137/200] Loss D: 0.6375, Loss G: 0.8970
Epoch [138/200] Loss D: 0.6376, Loss G: 0.9096
Epoch [139/200] Loss D: 0.6369, Loss G: 0.8964
Epoch [140/200] Loss D: 0.6400, Loss G: 0.8876
Epoch [141/200] Loss D: 0.6405, Loss G: 0.8905
Epoch [142/200] Loss D: 0.6403, Loss G: 0.8898
Epoch [143/200] Loss D: 0.6392, Loss G: 0.8944
Epoch [144/200] Loss D: 0.6413, Loss G: 0.8832
Epoch [145/200] Loss D: 0.6447, Loss G: 0.8739
Epoch [146/200] Loss D: 0.6389, Loss G: 0.9052
Epoch [147/200] Loss D: 0.6394, Loss G: 0.8905
Epoch [148/200] Loss D: 0.6432, Loss G: 0.8810
Epoch [149/200] Loss D: 0.6445, Loss G: 0.8759
Epoch [150/200] Loss D: 0.6438, Loss G: 0.8710
Epoch [151/200] Loss D: 0.6442, Loss G: 0.8737
Epoch [152/200] Loss D: 0.6414, Loss G: 0.8871
Epoch [153/200] Loss D: 0.6400, Loss G: 0.8969
Epoch [154/200] Loss D: 0.6420, Loss G: 0.8802
Epoch [155/200] Loss D: 0.6437, Loss G: 0.8746
Epoch [156/200] Loss D: 0.6450, Loss G: 0.8783
Epoch [157/200] Loss D: 0.6458, Loss G: 0.8669
Epoch [158/200] Loss D: 0.6428, Loss G: 0.8844
Epoch [159/200] Loss D: 0.6452, Loss G: 0.8733
Epoch [160/200] Loss D: 0.6442, Loss G: 0.8800
```

```
Epoch [161/200] Loss D: 0.6420, Loss G: 0.8925
Epoch [162/200] Loss D: 0.6409, Loss G: 0.8827
Epoch [163/200] Loss D: 0.6447, Loss G: 0.8760
Epoch [164/200] Loss D: 0.6454, Loss G: 0.8706
Epoch [165/200] Loss D: 0.6451, Loss G: 0.8743
Epoch [166/200] Loss D: 0.6445, Loss G: 0.8744
Epoch [167/200] Loss D: 0.6420, Loss G: 0.8848
Epoch [168/200] Loss D: 0.6433, Loss G: 0.8733
Epoch [169/200] Loss D: 0.6452, Loss G: 0.8680
Epoch [170/200] Loss D: 0.6432, Loss G: 0.8774
Epoch [171/200] Loss D: 0.6439, Loss G: 0.8736
Epoch [172/200] Loss D: 0.6457, Loss G: 0.8745
Epoch [173/200] Loss D: 0.6447, Loss G: 0.8786
Epoch [174/200] Loss D: 0.6469, Loss G: 0.8640
Epoch [175/200] Loss D: 0.6446, Loss G: 0.8724
Epoch [176/200] Loss D: 0.6451, Loss G: 0.8736
Epoch [177/200] Loss D: 0.6456, Loss G: 0.8772
Epoch [178/200] Loss D: 0.6452, Loss G: 0.8801
Epoch [179/200] Loss D: 0.6404, Loss G: 0.8910
Epoch [180/200] Loss D: 0.6453, Loss G: 0.8633
Epoch [181/200] Loss D: 0.6439, Loss G: 0.8779
Epoch [182/200] Loss D: 0.6462, Loss G: 0.8714
Epoch [183/200] Loss D: 0.6466, Loss G: 0.8676
Epoch [184/200] Loss D: 0.6470, Loss G: 0.8629
Epoch [185/200] Loss D: 0.6469, Loss G: 0.8599
Epoch [186/200] Loss D: 0.6502, Loss G: 0.8594
Epoch [187/200] Loss D: 0.6450, Loss G: 0.8767
Epoch [188/200] Loss D: 0.6475, Loss G: 0.8671
Epoch [189/200] Loss D: 0.6475, Loss G: 0.8682
Epoch [190/200] Loss D: 0.6450, Loss G: 0.8767
Epoch [191/200] Loss D: 0.6478, Loss G: 0.8595
Epoch [192/200] Loss D: 0.6499, Loss G: 0.8538
Epoch [193/200] Loss D: 0.6492, Loss G: 0.8597
Epoch [194/200] Loss D: 0.6485, Loss G: 0.8620
Epoch [195/200] Loss D: 0.6469, Loss G: 0.8676
Epoch [196/200] Loss D: 0.6458, Loss G: 0.8731
Epoch [197/200] Loss D: 0.6491, Loss G: 0.8642
Epoch [198/200] Loss D: 0.6446, Loss G: 0.8806
Epoch [199/200] Loss D: 0.6489, Loss G: 0.8623
```

4) **In a 10 × 10 grid, show 100 digits generated by your generator.**



5) **Explanation of your Classifier model, design decisions, training-test dataset descriptions and what other factors were considered to improve your model.**

Classifier Model

The Classifier model consists of two convolutional layers (conv1 and conv2) both have kernel size=3, stride=1, padding=1 to keep size of output and with Relu activation followed by max-pooling layers (pool) for reduces the spatial dimensions (width and height) of the feature maps, effectively downsampling the input. The output from the convolutional layers is reshaped and passed through a fully connected layer (fc1) with ReLU activation. Finally, the output layer predicts the class probabilities for each image.

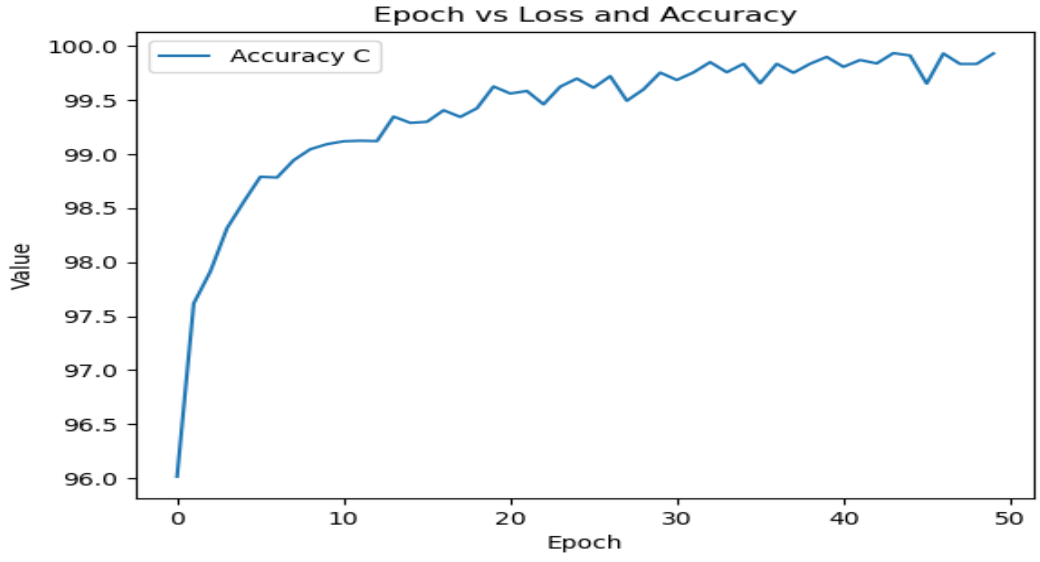
Design and Design decisions

First I used fully connected layers with Relu function but convolution layers with maxpool and Relu function gave more accuracy over training and test dataset. Experimenting with different learning rates, batch sizes, or number of epochs can give further better accuracy. Hyperparameters include the number of input channels (in_channels), the number of classes (num_classes), the learning rate, batch size, and number of epochs. For epochs I used 50 since accuracy on the training set and loss nearly the same after 50. Adam optimizer is used to optimise the CNN model's parameters, and the cross-entropy loss function (CrossEntropyLoss) is employed to compute the classification loss. Cross entropy loss is generally used for multi-class classification tasks, while (Binary Cross Entropy Loss) is used for binary classification tasks. Therefore I chose a cross-entropy loss function to compute the classification loss.The check_accuracy function calculates the classification accuracy of the model on a given data loader. It iterates through the batches, feeds the data through the model, and compares the predicted labels with the ground truth labels. Training process involves iterating through each epoch and mini-batches of data. For each batch, the data and targets are loaded onto the device (CPU or GPU). The forward pass is performed by passing the data through the CNN model, and the loss is calculated based on the predicted scores and target labels. The optimizer's gradients are then set to zero, and the backward pass is performed to compute the gradients of the model's parameters. Finally, the optimizer takes a step based on the computed gradients to update the model's parameters.

Training and Test Datasets

MNIST dataset, which contains grayscale images of hand-drawn digits from 0 to 9. The training and test datasets are loaded using PyTorch's MNIST class and transformed into tensors. The datasets are further split into batches using the DataLoader. The check_accuracy function calculates the classification accuracy of the model on a given data loader. It iterates through the batches, feeds the data through the model, and compares the predicted labels with the ground truth labels.class. Also I used Normalise transformation on this dataset (mean = 0.5, sd=0.5).

6) Training accuracy of the Classifier model at the end of each epoch.



```
Epoch [0/50] Loss C: 0.4243 Accuracy C: 96.0150%
Epoch [1/50] Loss C: 0.1103 Accuracy C: 97.6167%
Epoch [2/50] Loss C: 0.0764 Accuracy C: 97.9133%
Epoch [3/50] Loss C: 0.0645 Accuracy C: 98.3133%
Epoch [4/50] Loss C: 0.0552 Accuracy C: 98.5567%
Epoch [5/50] Loss C: 0.0499 Accuracy C: 98.7883%
Epoch [6/50] Loss C: 0.0446 Accuracy C: 98.7817%
Epoch [7/50] Loss C: 0.0403 Accuracy C: 98.9417%
Epoch [8/50] Loss C: 0.0379 Accuracy C: 99.0433%
Epoch [9/50] Loss C: 0.0348 Accuracy C: 99.0900%
Epoch [10/50] Loss C: 0.0323 Accuracy C: 99.1167%
Epoch [11/50] Loss C: 0.0297 Accuracy C: 99.1217%
Epoch [12/50] Loss C: 0.0278 Accuracy C: 99.1183%
Epoch [13/50] Loss C: 0.0260 Accuracy C: 99.3450%
Epoch [14/50] Loss C: 0.0244 Accuracy C: 99.2867%
Epoch [15/50] Loss C: 0.0233 Accuracy C: 99.2967%
Epoch [16/50] Loss C: 0.0211 Accuracy C: 99.4033%
Epoch [17/50] Loss C: 0.0199 Accuracy C: 99.3417%
Epoch [18/50] Loss C: 0.0186 Accuracy C: 99.4217%
Epoch [19/50] Loss C: 0.0183 Accuracy C: 99.6233%
Epoch [20/50] Loss C: 0.0165 Accuracy C: 99.5583%
Epoch [21/50] Loss C: 0.0161 Accuracy C: 99.5817%
Epoch [22/50] Loss C: 0.0159 Accuracy C: 99.4583%
Epoch [23/50] Loss C: 0.0146 Accuracy C: 99.6217%
Epoch [24/50] Loss C: 0.0134 Accuracy C: 99.6967%
Epoch [25/50] Loss C: 0.0123 Accuracy C: 99.6117%
Epoch [26/50] Loss C: 0.0122 Accuracy C: 99.7183%
Epoch [27/50] Loss C: 0.0121 Accuracy C: 99.4917%
Epoch [28/50] Loss C: 0.0112 Accuracy C: 99.5967%
Epoch [29/50] Loss C: 0.0106 Accuracy C: 99.7517%
Epoch [30/50] Loss C: 0.0105 Accuracy C: 99.6833%
Epoch [31/50] Loss C: 0.0096 Accuracy C: 99.7533%
Epoch [32/50] Loss C: 0.0086 Accuracy C: 99.8483%
Epoch [33/50] Loss C: 0.0088 Accuracy C: 99.7550%
Epoch [34/50] Loss C: 0.0088 Accuracy C: 99.8333%
Epoch [35/50] Loss C: 0.0076 Accuracy C: 99.6533%
Epoch [36/50] Loss C: 0.0076 Accuracy C: 99.8333%
Epoch [37/50] Loss C: 0.0071 Accuracy C: 99.7500%
Epoch [38/50] Loss C: 0.0081 Accuracy C: 99.8333%
Epoch [39/50] Loss C: 0.0070 Accuracy C: 99.8967%
Epoch [40/50] Loss C: 0.0060 Accuracy C: 99.8050%
Epoch [41/50] Loss C: 0.0055 Accuracy C: 99.8683%
Epoch [42/50] Loss C: 0.0062 Accuracy C: 99.8367%
Epoch [43/50] Loss C: 0.0048 Accuracy C: 99.9317%
Epoch [44/50] Loss C: 0.0055 Accuracy C: 99.9100%
Epoch [45/50] Loss C: 0.0057 Accuracy C: 99.6500%
Epoch [46/50] Loss C: 0.0052 Accuracy C: 99.9283%
Epoch [47/50] Loss C: 0.0043 Accuracy C: 99.8317%
Epoch [48/50] Loss C: 0.0046 Accuracy C: 99.8317%
Epoch [49/50] Loss C: 0.0048 Accuracy C: 99.9283%
```

7) **Testing accuracy of the Classifier model. (For the whole MNIST test data set)**

```
samples 9866/10000 with accuracy 98.66%
```

8) **Report the classification errors for test sets S0 and S1.**

```
S0 => Got correct samples 98/100 with accuracy 98.00%
S1 => Got correct samples 88/100 with accuracy 88.00%
```

9) **Discuss your observations.**

The generated fake MNIST numbers have a lower classification accuracy compared to the real MNIST batch. This suggests that the Generator is not effectively replicating the characteristics and details of the real digits. Potential reasons for this discrepancy include the Generator's limited learning capability, the Discriminator's superior ability to differentiate real and fake images, and the need for further training and convergence. Improving the performance of the Generator could involve optimising its architecture, adjusting training parameters, increasing the diversity of the training data, and exploring advanced GAN techniques.

10) **Instead of generating random digits, if I asked you to generate specific digits (e.g., generate an image of 0 when the integer 0 is given as input) how would you do it? You have to use the Generator that you have already created. Describe your architecture along with a diagram and explain how you train and test it. (What are the inputs? What are the outputs? etc)**

To generate specific digits using the existing Generator, a separate neural network can be created to output the corresponding latent representation (Z) for the desired digit. This can be achieved by training a classifier network that takes the digit (0, 1, 2…9) as input and learns to predict the associated latent representation.

- Architecture

The architecture consists of two parts: the Digit Classifier and the Generator.

Digit Classifier

The Digit Classifier is a neural network that takes the desired digit as input. The output layer of the classifier should have the same dimension as the latent representation Z .

Generator

It takes the latent representation Z as input and generates an image of the corresponding digit.

Training Process:

The training process involves training the Digit Classifier to predict the correct latent representation Z for each digit. The input (digit) is passed through the classifier network, and the output is compared to the ground truth latent representation Z. The loss is computed based on the difference between the predicted and true Z values. The classifier's parameters are updated using backpropagation and gradient descent to minimise the loss.

Overall, the architecture involves training a Digit Classifier to predict the latent representation Z for each digit, and then using the Generator to generate the desired digit image based on the predicted Z. This approach allows for the generation of specific digits by mapping them to the appropriate latent space representation.

Used sites and tutorials

1. https://www.youtube.com/playlist?list=PLhhyoLH6IjfxeoooqP9rhU3HJIAVAJ3Vz
2. https://www.youtube.com/playlist?list=PLZbbT5o_s2xrfNyHZsM6ufI0iZENK9xgG
3. https://stackoverflow.com/questions/23339315/read-image-grayscale-opencv-3-0-0-dev
4. https://stackoverflow.com/questions/11854847/how-can-i-display-an-image-from-a-file-in-jupyter-notebook
5. https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
6. https://stackoverflow.com/questions/51503851/calculate-the-accuracy-every-epoch-in-pytorch
7. https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/