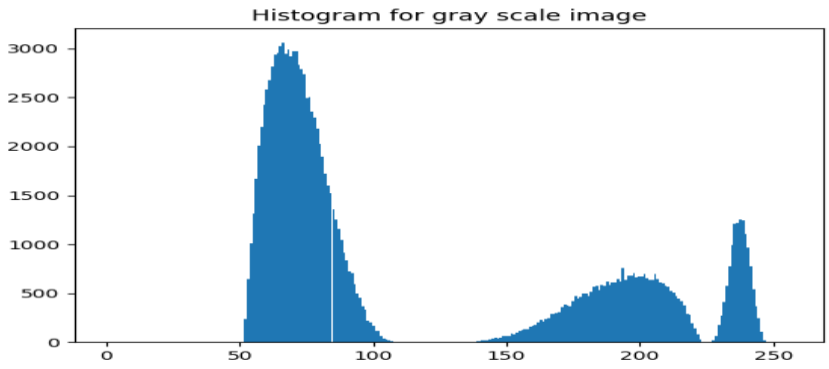
Deep Generative Modelling for Learning Medical Image Statistics (DGM-Image Challenge)

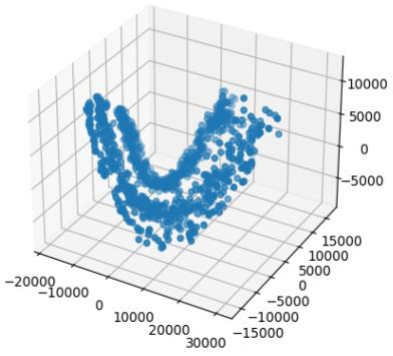
Strategies:

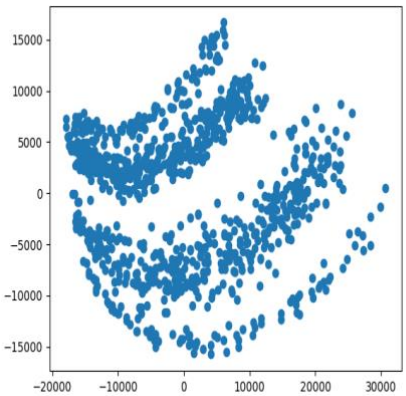
1. Data Downloading: Link
2. Data Pre-processing:
   - The folder contains 110 subfolders, therefore combine the folders.
   - After doing this upload the data to google drive to use in google colab.
   - Apply Image Transformations like **Resize, Centre Crop, Normalize** (transforming the mean and S.D in range [-1,1]) and finally convert the data to **tensors**.
   - Converting the data to Data Loader with **batch size of 128**.

3. Data Visualization:
   - Grid wise plotting of the data.
   - Plot distribution of pixels. (Probability distribution followed by the Train data).



4. For a Good GAN Model the output values should follow same/approximately same probability distribution as followed by real images. The Distribution followed by the real images is as follows:





5. Loading data and the model in GPU to improve the training speed.

6. Creating Discriminator (Deep Convolutional Generative Adversarial network):
    - Created 5 Convolutional Layers for binary **classification of Real and Fake Images.**
    - 4 Layers with dimensions of **[(3,64), (64,128), (128,256), (256,512)]** with **kernel size of 4, stride of 1 and padding of 1**.
    - Used Batch Normalization and **LeakyReLU** for intermediate activation function.
    - The last output layer contained dimension of (512,1) with kernel size of 4, stride of 1 and padding of 0 and **sigmoid** activation layer for binary classification.

7. Creating Generator (Deep Convolutional Generative Adversarial network):
    - Created 5 Convolutional Layers for generating batches of **images from random gaussian noise of latent size = 128.**
    - 4 Layers with dimensions of **[(128,512), (512,256), (256,128), (128,64)]** with **kernel size of 4, padding of 1 and stride of 1,2,2,2 respectively**.
    - Used Batch Normalization and **ReLU** for intermediate activation function.
    - The last output layer contained dimension of (64,3) with kernel size of 4, stride of 2 and padding of 1 and **Tanh** activation layer for generating the output between -1 and 1.

8. Training discriminator:
    - The discriminator is trained keeping the **generator untouched** (not training it).
    - First the discriminator is trained with **real images giving the label True or 1**.
    - Fake images are generated using generator.
    - Then the discriminator is trained using the **fake images keeping the labels False or 0.**
    - For an ideally trained discriminator, its output should come around 0.5.

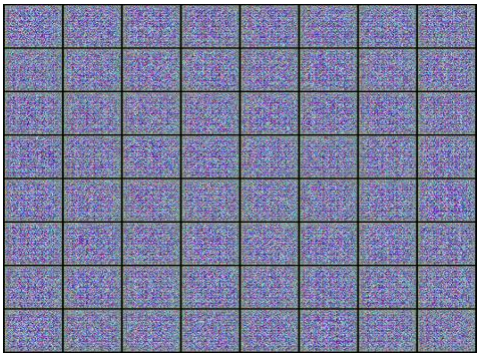
9. Training generator:
    - Discriminator is used to train the generator.
    - The discriminator predicts the fake images and the generator is trained with this prediction taking the label value as real image or 1.
    - Binary Cross Entropy Loss function is used.

10. Final Training/Optimization:
    - Adam optimizer is used with **betas=(0.5, 0.999**) and learning rate which is fine tuned while training.

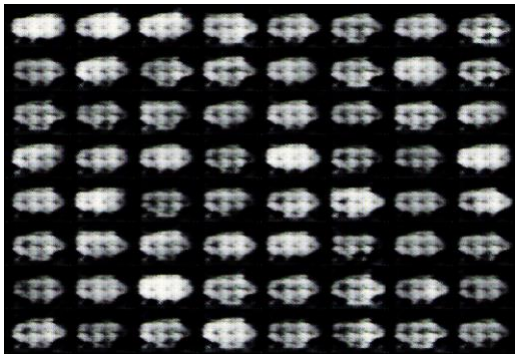
11. Evaluation:
    - Visual Inspection: The generated images after every epoch was saved.
    - We can evaluate GAN by verifying the fact that the Real and Generated images will follow the same probability distributions.
    - Real Data Comparison
    - Inception Score
    - Fréchet Inception Distance

My Work and Observations: (Video Link of the training process Link)

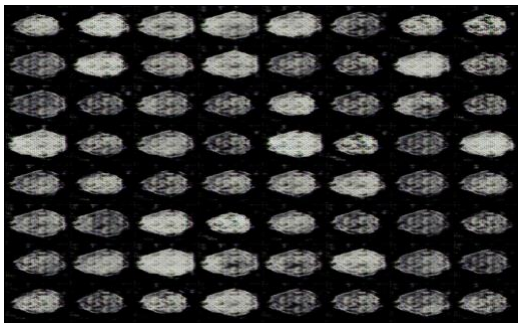
1. I have only used **1000 images for training the GAN** due to GPU constraints.
2. **GitHub Link of the entire project(Code)**: GitHub
3. Batch size and latent size of 128 was used.
4. While training with different learning rates [0.00002, 0.0002, 0.002, 0.02, 0.2], **0.002** was working best.
5. Epochs of **100** was used.
6. While Training the fake score should tend to 0 and the real score should tend to 1.



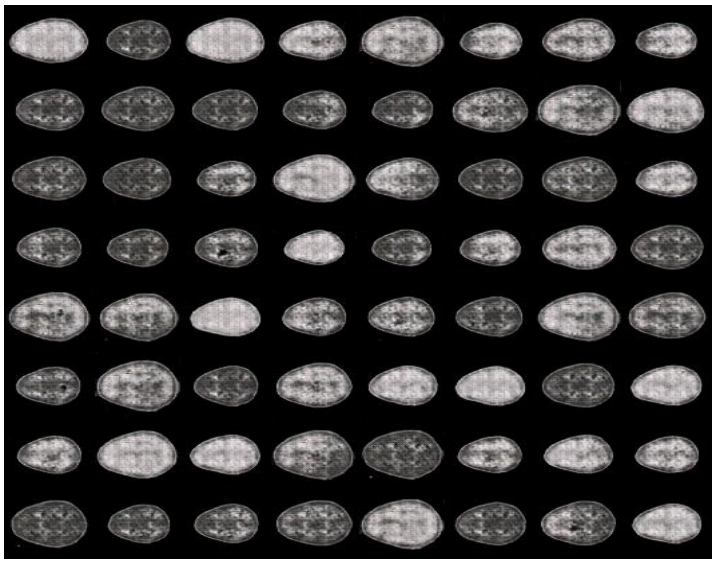
Epoch 0



Epoch 25



Epoch 50



Epoch 100