REPORT : ASSIGNMENT 4 CONDITIONAL GANs DEEP LEARNING (CSL7590)

Dipan Mandal (M23CSA009)
Divyaansh Mertia (M23CSE013)
Amresh Kumar (M23CSA004)

Colab link: Assignment 4 .ipynb

Introduction:

The objective of this assignment was to **generate realistic images from paired/unpaired sketches** using **Conditional Generative Adversarial Networks(cGANs)**. The Generator of the cGAN should be able to provide images based on the sketch and label provided. For our assignment we used the paired sketches to train the network and generate outputs.

The dataset provided in this case was the **ISIC skin cancer dataset** which contains **9016 images** in the training set along with the same number of paired sketches. The sketches were mainly the outline of the segmentation mask of the cancerous part.
Similarly the test set contains **1000** pairs of images and their corresponding masks.

Data Processing and Dataloader:

Data processing:

Here are the steps that we followed while processing the data:

- **Resizing :** we resized both the images and sketches to a size of 256 x 256. We experimented with different dimensions such as 64 x 64 and 128 x 128. Although the epochs were faster, they failed to produce convincing results. So we decided to move forward with the mentioned dimension.
- **Normalization :** we applied normalization on the train and test images for better training. We kept both mean and standard deviation as [0.5, 0.5, 0.5](for 3 channels).

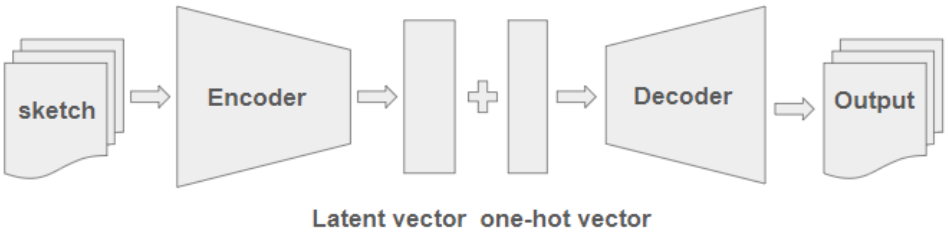
Dataloader:

For the dataloader we have used a batch size of 4. Each batch contains the image, the corresponding contour/sketch and the class label for the type of cancer present in the image.
At first we started with a batch size of 16, but the model was taking more time to converge and the generator loss was increasing whereas the discriminator loss was decreasing. So we decreased the batch size to 4 and we got much better results.

cGAN Architecture:

Generator Architecture:

For the generator, we have used an encoder-decoder architecture to incorporate the class label information into sketch data. The generator takes a 3 x 256 x 256 sketch and the 7 dimensional one-hot vector as input and produces a 3 x 256 x 256 image as output. Here is an overall view of our architecture for the generator.



For the encoder we have used a pre-trained ResNet-18 model (trained on ImageNet dataset). This enabled us to generate better latent vectors. The dimension of our latent vector **(512,)** with which we concatenate the 7 dimensional one-hot label which makes our input to the decoder to have a dimension of **(519,).**

Now for the decoder, we have designed a custom architecture with 7 `ConvTranspose2d()` layers along with batch normalization and leaky ReLU after each layer. At the end of the decoder we have used a Tanh function(as proposed in the original paper).

The total number of parameters in the Generator architecture are : **14.002 millions**

Discriminator Architecture:

For the discriminator, we have used a custom architecture with 5 `Conv2d()` layers. And each layer is accompanied by batch normalization and leaky ReLU. We also added **dropout layers** after each convolution layer with a probability of 0.3 which significantly improved the performance of our model.

The discriminator takes the image and the label as input and gives 0 if the image is fake and 1 if the image is real according to the discriminator.

Experiments:

Here are some of the experiments that we performed to improve the performance of the cGAN:

- **Changing the loss function:** We incorporated MSE loss in the loss function associated with the generator. This improved the performance of the generator as MSE loss helps to preserve the details and structures present in the images.
- **Different training rate:** In our model, the discriminator was overpowering the generator as we observed that the generator loss was increasing while the discriminator loss was very low. So we experimented with different training rates for the generator and the discriminator. The Generator was being trained every epoch and the discriminator was being trained every alternate epoch. This improved image generation a bit but later we discarded this idea as the generator seemed to overfit.
- **Converting the labels to embedding:** We converted the labels into embedding vectors of 256 X 256 dimensions and then concatenated with the images rather than doing that in the latent dimension.

We also experimented with applying **regularization** and **attention** in the generator, but we discarded them later as the model didn't show any significant improvement.

[We conducted these experiments for 50 epochs each on a sample size of 1000]

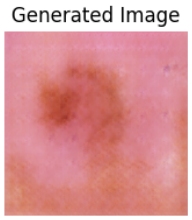
Training:

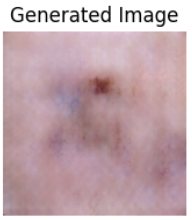
After conducting the above experiments, we finalized the architectures for our Discriminator and Generator and trained the cGAN architecture for **50 epochs on the full train dataset.**
Here are the hyperparameters that we used in training:

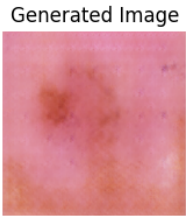
Batch size : 4
Learning rate : 0.0002
Beta values : (0.5, 0.999) (for Adam optimizer)

Here are some of the results that we obtained from the cGAN during the training:

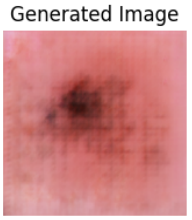
10 epochs:

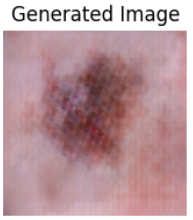




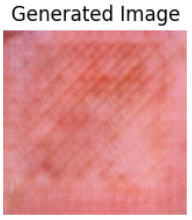


20 epochs:

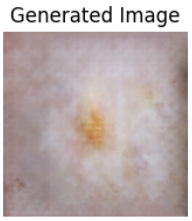




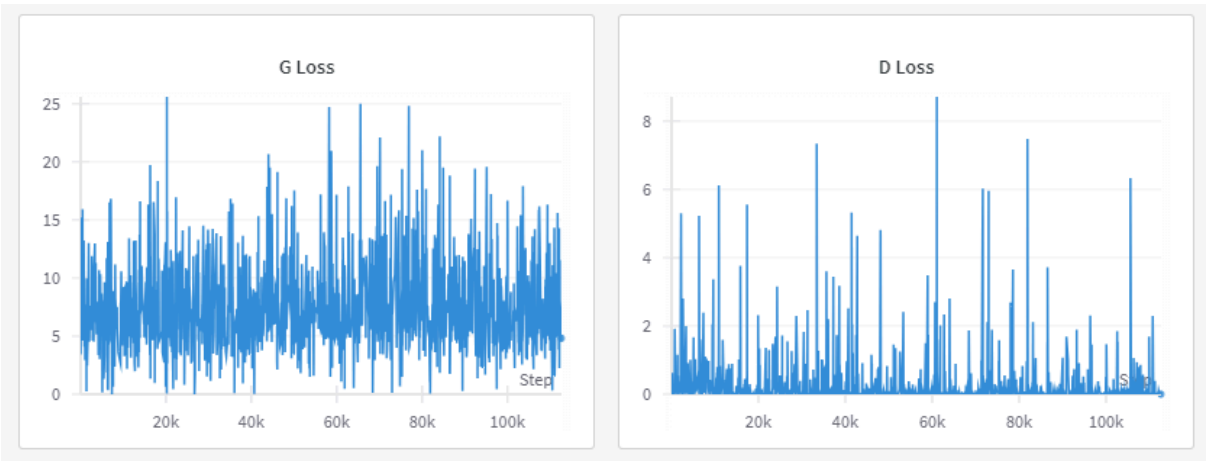
30 epochs:



40 epochs:



As we can see, the image quality improves a bit while the number of epochs increases. Of course we will not get any satisfactory results as GANs need to be trained for at least 100 epochs to generate good results.

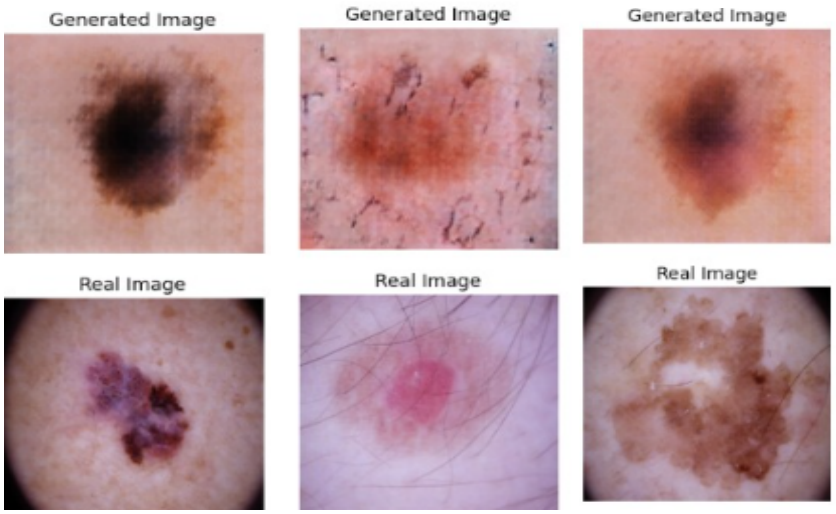


Link to WandB: https://wandb.ai/divyaansh/DL-Assignment_4/runs/9mii4ty4

These are the Generator loss and the discriminator loss that we got in 50 epochs. The loss fluctuates a lot and sometimes spikes to get to almost 25+ values for generator and 8+ for discriminator. The final loss that we got after 50 epochs was around 3.00 for the generator and 0.03 for the discriminator.

Testing:

After we were done with the training, we tested the performance of the generator on the test sketches. We loaded the saved weights of the generator for inference and then supplied the test sketches to the generator. Here are some of the results that we got while testing the model.



Here are some of the scores that we got from the test set:
Frechet Inception Distance(FID) : 4.31
Inception score : 1.99(+- 0.03)

Classifier:

For the classifier, we built a simple CNN architecture with 4 2D convolutional layers followed by a MLP classifier. For training we used the train images and their corresponding labels. (we trained on a sample size of 5000 due to time constraints)
Then we trained the model for **20 epochs** with the following parameters:
Learning rate : 0.001
Optimizer : Adam
Total number of parameters : 570,695

The classifier got an accuracy of **96.44%** on the training data.
When we tested the model on generated images we got an accuracy of **40.60%**
For the test images we got an accuracy of **56.30%**

Areas of Improvement:

The main area where we can improve the model performance is definitely training for more epochs. Due to multiple experiments, we weren't able to train the model for more than 50 epochs. Training for **at least 100 epochs** would definitely improve the model performance and image generation.

Also this architecture was fully developed from scratch. We can definitely use more sophisticated cGAN architectures like **WGAN, cycleGAN, Pix2Pix** to get more clear results. Using cycleGAN we can even get good results for unpaired sketches, which was the initial aim of the project.

Also in the current architecture, we can experiment with attention mechanisms in the encoder-decoder architecture for better results.
