CS597 REPORT 8

Yash Pradeep Gupte
MS Computer Science
A20472798

1] Transformer based generator architecture for our implementation – Text to Image synthesis using Transformer GANs

(A) Training data

- coco2014 – zebra class
- Image resolution = 64x64x3
- Train data = 1324 images
- Test data = 677 images

(B) Generator Architecture (Generator_transformer.py)

This generator architecture is inspired from TransGAN implementation. It uses only the transformer network to generator image from caption. To upsample features, we make use of Upsample Blocks. These upsample blocks are constructed using StageBlocks (Blocks of transformer encoders) and Pixel Upsample (Pixel shuffle) techniques. The Block module consists of Attention mechanism followed by a MLP (Multilinear Perceptron).

- Batch size = 32
- Input to generator is caption vectors of dimension (32x16x768) where, batch size is 32, caption vectors are 16 dimensional (16 words) and embedding dimension is 768.
- The generator first adds position embeddings to the input.
- Then we set Height and Width (H and W) values to 4. This is because our caption vector is a 4x4 or 16-dimensional vector.
- Then we go to a single block where we perform Attention mechanism and MLP.
- Inside the attention mechanism we get the query, key and value parameters which gives us the output of attention module, which is then passed to the MLP.
- We then iterate over Upsample blocks to generate the desired image.
    - The upsample block consists of four StageBlocks.
    - First, we upsample pixels using Pixel Shuffle technique.
    - Second, we add position embeddings to the resultant features.
    - Lastly, perform transformer mechanism inside the stage blocks.
- Once we have successfully iterated over all the stage blocks (one Upsample Block), we pass our output (shape 32x4096x3) to a deconvolutional layer which outputs image of size 32x3x64x64.

The feature map transition throughout the generator looks as follows

32x16x768 - **INPUT**
32x64x192
32x256x48
32x1024x12
32x4096x3
32x3x64x64 - **OUTPUT**

(C) Discriminator Architecture (Discriminator.py)

- The discriminator architecture has been kept the same.
- It takes input a 32x3x64x64 image and outputs whether the image is Real / Fake.

(D) Other parameters

- The rest parameters like loss functions, optimizers, learning rates, etc. have been kept the same.

(E) Code Implementation

- Go to Code folder
- Training: Execute the main_stage1_wgan.py file with the following arguments / command

python3 main_stage1_wgan.py /Users/yashgupte21/Desktop/CS597/week2/coco_data/images16/train -j0 -b32 --glr 0.0002 --dlr 0.0002 --netgarch generator_transformer --netdarch discriminator

- Change the dataset directory to fetch data from your local machine