Image In Painting Analysis

By Jayanth Rajaram (js12891) Saaketh Koundinya(sg7729)

Executive Summary

Motivation:

- In-painting is a critical task in image processing that aims to fill in missing or corrupted regions in images.
- GAN’s which are used in inpainting are generally unstable,slow and require lots of data
- Goal is to improve energy efficiency and performance while using GAN’s .

Novelties and Technical Contributions:

- Bottleneck Analysis and Optimization
- Roofline Model Validation.
- Mixed Precision Training
- Data Augmentation

Technical Challenges

Technical difficulties and Limitations:

- Calculating the exact number of FLOPs per second during training is challenging because of our complicated loss functions.
- Solution : Approximate estimate of FLOPS using number of parameters

Implementation Details:

- **Hardware Accelerators:** A100 and V100
- **Libraries :** pytorch,pynvml and torch.cuda.amp
- **Platform**: NYU HPC and Google Colab
- **Dataset :** celebA dataset

Generative Adversarial Networks

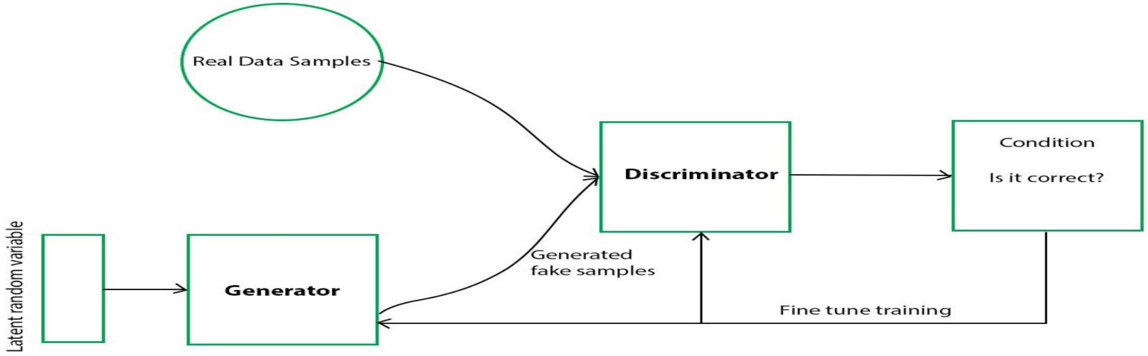
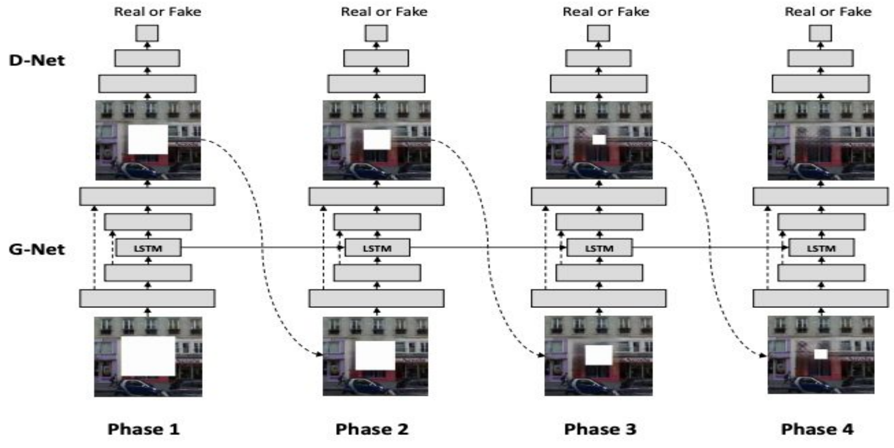


Fig 1: GAN architecture

- Consist of two components: generator and discriminator.
- Generator generates synthetic data.
- Discriminator distinguishes real from generated samples.
- Trained simultaneously in an adversarial fashion.
- Generator improves realism as discriminator gets better.

Progressive GAN Followed by a LSTM



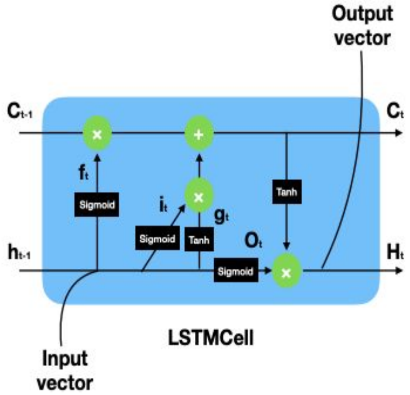


Fig 2: Progressive GAN Architecture

Probabilistic Diverse GAN

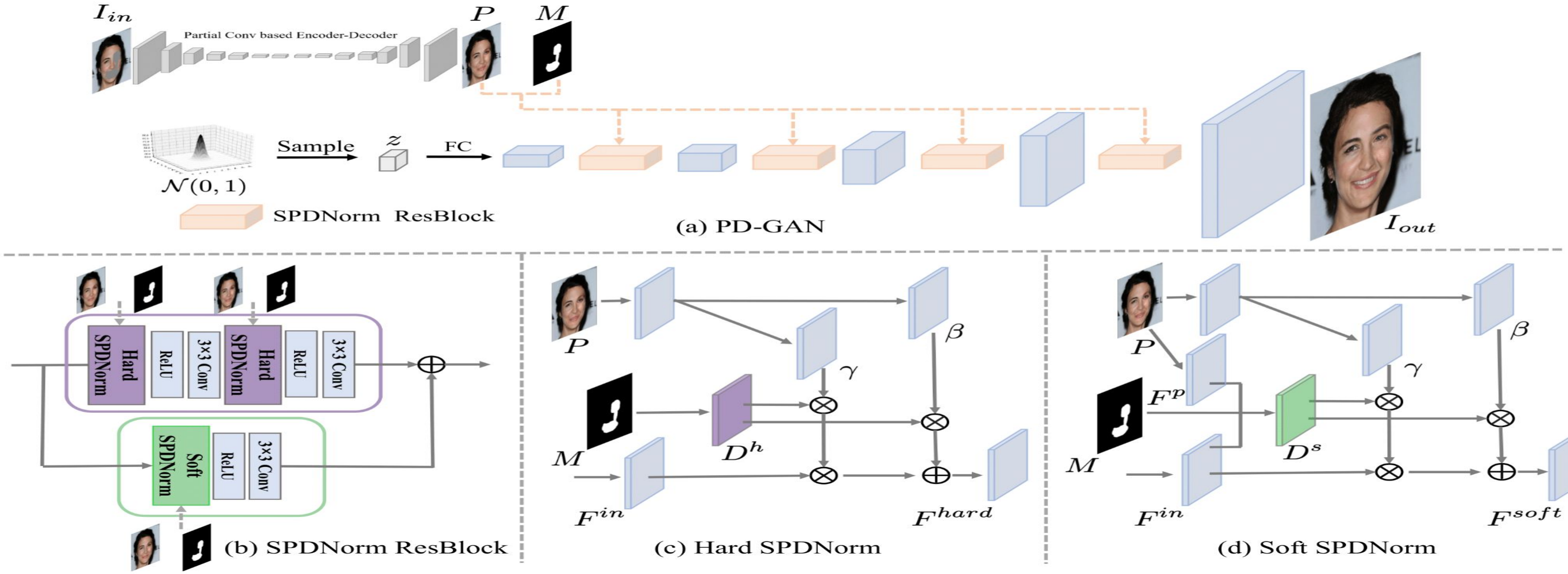


Fig 2: Probabilistic GAN Architecture

NYU

Solution Approach

Implementation Details:

- **Hardware Accelerators:** Nvidia A100 and Nvidia V100
- **Libraries :** pytorch,pynvml
- **Platform**: NYU HPC and Google Colab
- **Dataset :** celebA dataset
- **Models:** Progressive Generative Network and PD-GAN.

Solution Approach

Analysis

- Train both our inpainting models on V100 and A100
- Profiling during training with the help of pytorch profiler.
- Metrics evaluated during training : GPU Power Consumption, GPU Utilization, GPU Temperature, Time per Iteration, Loss, PSNR, and SSIM.
- Compared the evaluated metrics across the different hardware accelerators and the different inpainting techniques used.

Optimization

- Optimized bottlenecks using mixed precision training.
- Built roofline models for our inpainting models to validate our optimization.
- Mixed Precision Training: Enables lower precision data types which can speed up computations of the backward pass .

NYU

Observations and Results

- **Profiling results**: Bottleneck for both our inpainting models is a result of their convolutional layers backward pass.
- **Reason:** High number of gradient calculations during backpropagation.
- To improve the performance of of our backward pass we make use of mixed precision training.
- Mixed Precision Training results.

Model	Average Loss
Progressive GAN -SP-A100	0.452
Progressive GAN -MP-A100	0.511
Progressive GAN -SP-V100	0.4391
Progressive GAN -MP-V100	0.5213

Model	Average Loss
PDGAN -SP-A100	3.820019
PDGAN -MP-A100	4.6606
PDGAN -SP-V100	4.8655
PDGAN -MP-V100	4.9626

Table 1: Loss for different models.

NYU

Mixed Precision Training results

Model	GPU	%Bandwidth Increase	%Performance increase	Speedup	%Power decrease
Progressive-GAN	A100	484.559	484.559	3.668	-5.711
Progressive-GAN	V100	50.360	50.3601	1.368	-51.2106
PD-GAN	A100	134.593	155.434	1.329	-14.947
PD-GAN	V100	94.582	106.0308	2.244	-33.984

Table 2: Mixed precision training results

Results: Profiling Progressive GAN

A100

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	CPU Mem	Self CPU Mem	CUDA Mem	Self CUDA Mem	# of Calls
cudaLaunchKernel	58.37%	802.756ms	58.37%	802.756ms	252.518us	100.385ms	7.32%	101.684ms	31.986us	0 b	0 b	0 b	0 b	3179
autograd::engine::evaluate_function: ConvolutionBack...	0.15%	2.127ms	38.90%	534.982ms	2.346ms	0.000us	0.00%	860.423ms	3.774ms	0 b	0 b	2.55 Gb	-2.83 Gb	228
ConvolutionBackward0	0.12%	1.656ms	38.73%	532.584ms	2.336ms	0.000us	0.00%	860.170ms	3.773ms	0 b	0 b	5.38 Gb	0 b	228
aten::convolution_backward	1.37%	18.868ms	38.61%	530.928ms	2.329ms	816.299ms	59.52%	860.170ms	3.773ms	0 b	0 b	5.38 Gb	3.21 Gb	228
cudaMemcpyAsync	23.03%	316.683ms	23.03%	316.683ms	2.262ms	3.498ms	0.26%	3.498ms	24.986us	0 b	0 b	0 b	0 b	140
aten::masked_select	0.01%	175.000us	21.62%	297.303ms	74.326ms	16.000us	0.00%	53.000us	13.250us	0 b	0 b	2.00 Kb	-2.00 Kb	4
aten::nonzero	0.02%	241.000us	21.60%	297.012ms	74.253ms	37.000us	0.00%	37.000us	9.250us	0 b	0 b	2.00 Kb	0 b	4
aten::add_	0.49%	6.716ms	10.14%	139.488ms	171.361us	25.263ms	1.84%	43.985ms	54.036us	620 b	-296 b	0 b	0 b	814
autograd::engine::evaluate_function: CudnnBatchNormB...	0.13%	1.757ms	9.75%	134.042ms	657.069us	0.000us	0.00%	77.553ms	380.162us	0 b	0 b	373.04 Mb	-1.71 Gb	204
CudnnBatchNormBackward0	0.06%	777.000us	9.60%	132.011ms	647.113us	0.000us	0.00%	74.321ms	364.319us	0 b	0 b	2.07 Gb	-390.23 Mb	204

Self CPU time total: 1.375s
Self CUDA time total: 1.371s

V100

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %	CUDA total	CUDA time avg	CPU Mem	Self CPU Mem	CUDA Mem	Self CUDA Mem	# of Calls
cudaLaunchKernel	58.37%	802.756ms	58.37%	802.756ms	252.518us	100.385ms	7.32%	101.684ms	31.986us	0 b	0 b	0 b	0 b	3179
autograd::engine::evaluate_function: ConvolutionBack...	0.15%	2.127ms	38.90%	534.982ms	2.346ms	0.000us	0.00%	860.423ms	3.774ms	0 b	0 b	2.55 Gb	-2.83 Gb	228
ConvolutionBackward0	0.12%	1.656ms	38.73%	532.584ms	2.336ms	0.000us	0.00%	860.170ms	3.773ms	0 b	0 b	5.38 Gb	0 b	228
aten::convolution_backward	1.37%	18.868ms	38.61%	530.928ms	2.329ms	816.299ms	59.52%	860.170ms	3.773ms	0 b	0 b	5.38 Gb	3.21 Gb	228
cudaMemcpyAsync	23.03%	316.683ms	23.03%	316.683ms	2.262ms	3.498ms	0.26%	3.498ms	24.986us	0 b	0 b	0 b	0 b	140
aten::masked_select	0.01%	175.000us	21.62%	297.303ms	74.326ms	16.000us	0.00%	53.000us	13.250us	0 b	0 b	2.00 Kb	-2.00 Kb	4
aten::nonzero	0.02%	241.000us	21.60%	297.012ms	74.253ms	37.000us	0.00%	37.000us	9.250us	0 b	0 b	2.00 Kb	0 b	4
aten::add_	0.49%	6.716ms	10.14%	139.488ms	171.361us	25.263ms	1.84%	43.985ms	54.036us	620 b	-296 b	0 b	0 b	814
autograd::engine::evaluate_function: CudnnBatchNormB...	0.13%	1.757ms	9.75%	134.042ms	657.069us	0.000us	0.00%	77.553ms	380.162us	0 b	0 b	373.04 Mb	-1.71 Gb	204
CudnnBatchNormBackward0	0.06%	777.000us	9.60%	132.011ms	647.113us	0.000us	0.00%	74.321ms	364.319us	0 b	0 b	2.07 Gb	-390.23 Mb	204

Self CPU time total: 1.375s
Self CUDA time total: 1.371s

Results: Profiling PD-GAN

A100

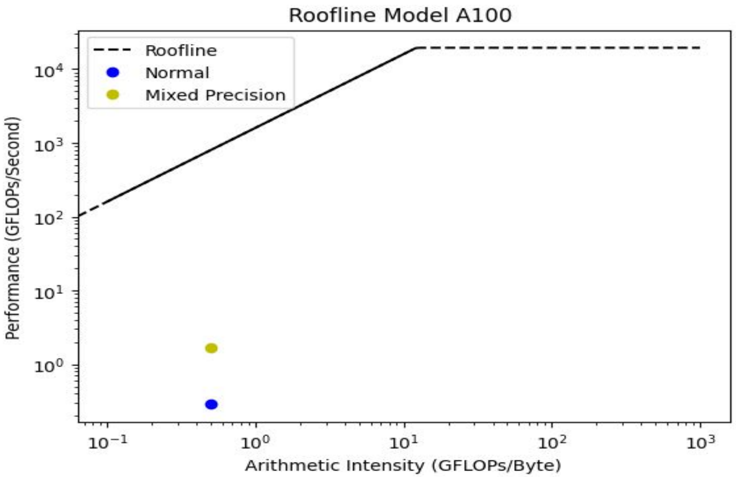
```
-------  ------------
                                                    Name    Self CPU %      Self CPU   CPU total %     CPU total  CP
U time avg     Self CUDA   Self CUDA %    CUDA total  CUDA time avg      CPU Mem  Self CPU Mem      CUDA Mem  Self
CUDA Mem    # of Calls
-------------------------------------------------------  ------------  ------------  ------------  ------------  --
----------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  -----
-------  ------------
enumerate(DataLoader)#_MultiProcessingDataLoaderIter...        39.91%     292.056ms        39.93%     292.220ms
292.220ms       0.000us         0.00%       0.000us       0.000us     24.00 Mb      24.00 Mb           0 b
0 b             1
                                 DataParallel.forward         3.97%      29.070ms        21.53%     157.581ms
22.512ms       0.000us         0.00%     167.689ms      23.956ms         -28 b      -1.81 Kb       8.17 Gb      -5.
32 Gb             7
                                     cudaLaunchKernel        11.64%      85.204ms        11.64%      85.204ms
11.657us      30.027ms         9.30%      30.030ms       4.109us           0 b           0 b           0 b
0 b          7309
autograd::engine::evaluate_function: ConvolutionBack...         0.52%       3.779ms        10.65%      77.955ms
445.457us       0.000us         0.00%     112.035ms     640.200us           0 b           0 b      -3.80 Gb      -1
0.66 Gb           175
                                          aten::copy_         2.18%      15.976ms        10.06%      73.602ms
166.144us       3.859ms         1.19%       5.991ms      13.524us           0 b           0 b           0 b
0 b           443
                                 ConvolutionBackward0         0.17%       1.228ms         9.91%      72.505ms
414.314us       0.000us         0.00%     109.447ms     625.411us           0 b           0 b       5.46 Gb
0 b           175
                             aten::convolution_backward         4.98%      36.441ms         9.74%      71.277ms
407.297us     105.328ms        32.61%     109.447ms     625.411us           0 b           0 b       5.46 Gb      -1
2.48 Gb           175
                                       aten::randperm         4.41%      32.265ms         8.82%      64.548ms
32.274ms       0.000us         0.00%       0.000us       0.000us       1.55 Mb          -8 b           0 b
0 b             2
                                             aten::to         0.30%       2.175ms         8.28%      60.588ms
142.225us       0.000us         0.00%       2.850ms       6.690us      12.00 Mb           8 b      20.02 Mb
512 b           426
                                       aten::_to_copy         0.24%       1.755ms         7.99%      58.464ms
221.455us       0.000us         0.00%       2.851ms      10.799us      12.00 Mb       3.00 Mb      20.02 Mb
0 b           264
-------------------------------------------------------  ------------  ------------  ------------  ------------  --
----------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  -----
-------  ------------
Self CPU time total: 731.784ms
Self CUDA time total: 322.977ms
```

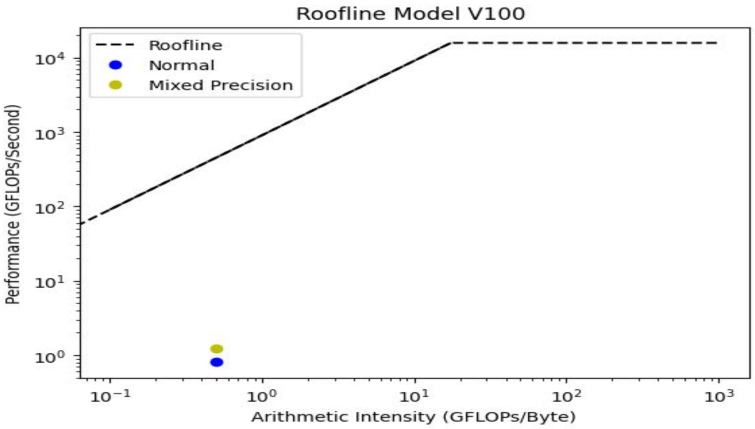
V100

```
                                                    Name    Self CPU %      Self CPU   CPU total %     CPU total  CP
U time avg     Self CUDA   Self CUDA %    CUDA total  CUDA time avg      CPU Mem  Self CPU Mem      CUDA Mem  Self
CUDA Mem    # of Calls
-------------------------------------------------------  ------------  ------------  ------------  ------------  --
----------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  -----
-------  ------------
                                     cudaLaunchKernel        35.67%     383.928ms        35.67%     383.928ms
27.653us     123.934ms        13.56%     123.936ms       8.927us           0 b           0 b        -512 b       -
512 b         13884
                                          aten::copy_         2.85%      30.645ms        21.74%     234.004ms
268.970us       5.399ms         0.59%      12.132ms      13.945us           0 b           0 b           0 b
0 b           870
                                 DataParallel.forward         3.66%      39.383ms        21.44%     230.755ms
16.483ms       0.000us         0.00%     368.725ms      26.337ms         -56 b      -2.02 Kb       8.27 Gb      -5.
61 Gb            14
                                             aten::to         0.65%       6.998ms        19.17%     206.292ms
253.430us       0.000us         0.00%       3.652ms       4.486us      12.00 Mb          12 b      20.28 Mb
3.00 Mb           814
                                       aten::_to_copy         0.20%       2.187ms        18.99%     204.371ms
399.162us       0.000us         0.00%       3.666ms       7.160us      12.00 Mb     768.08 Kb      20.28 Mb
0 b           512
                              cudaStreamSynchronize        18.00%     193.707ms        18.00%     193.707ms
1.699ms     862.000us         0.09%     862.000us       7.561us           0 b           0 b           0 b
0 b           114
                           Optimizer.step#Adam.step         1.85%      19.932ms        15.23%     163.916ms
40.979ms       0.000us         0.00%      46.974ms      11.743ms         -16 b         108 b           0 b      -2.
90 Gb             4
autograd::engine::evaluate_function: ConvolutionBack...         0.33%       3.571ms        13.99%     150.542ms
430.120us       0.000us         0.00%     417.096ms       1.192ms           0 b           0 b      -3.08 Gb      -1
0.69 Gb           350
                                 ConvolutionBackward0         0.16%       1.709ms        13.03%     140.197ms
400.563us       0.000us         0.00%     411.451ms       1.176ms           0 b           0 b       6.21 Gb
0 b           350
                             aten::convolution_backward         3.43%      36.922ms        12.87%     138.488ms
395.680us     392.522ms        42.96%     411.451ms       1.176ms           0 b           0 b       6.21 Gb      -3
8.00 Gb           350
-------------------------------------------------------  ------------  ------------  ------------  ------------  --
----------  ------------  ------------  ------------  ------------  ------------  ------------  ------------  -----
-------  ------------
Self CPU time total: 1.076s
Self CUDA time total: 913.718ms
```

NYU

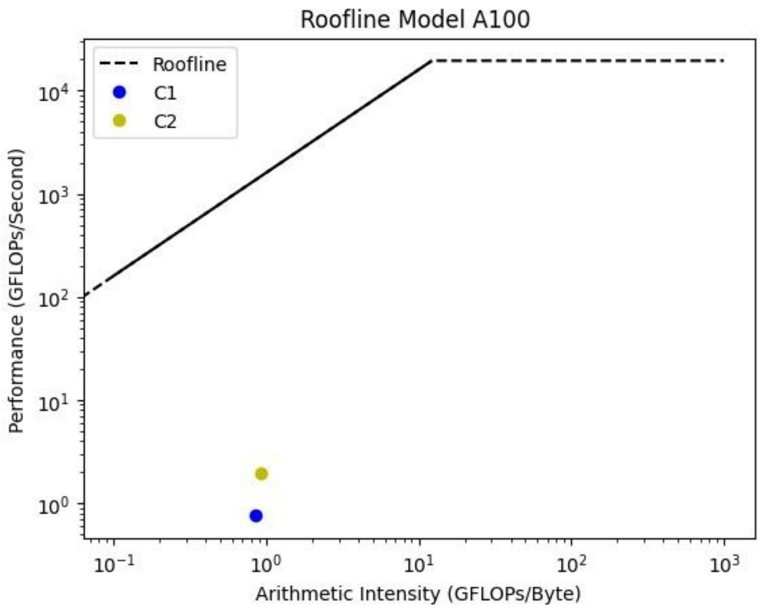
Roofline Models Progressive GAN

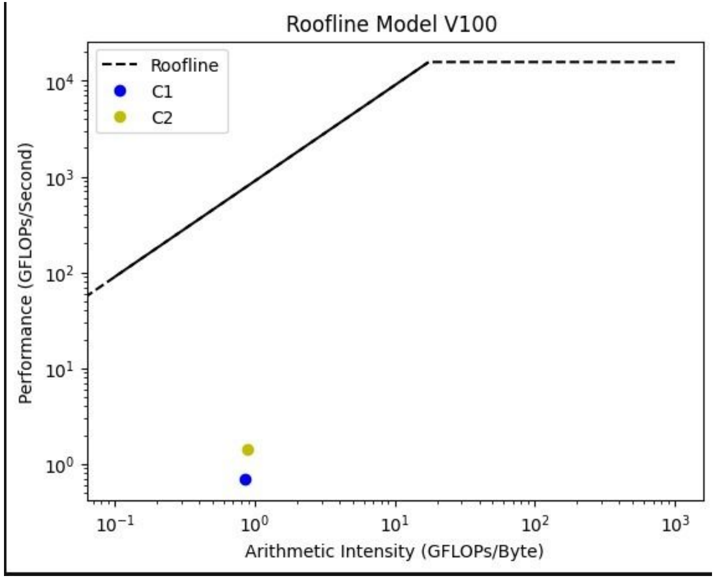




NYU

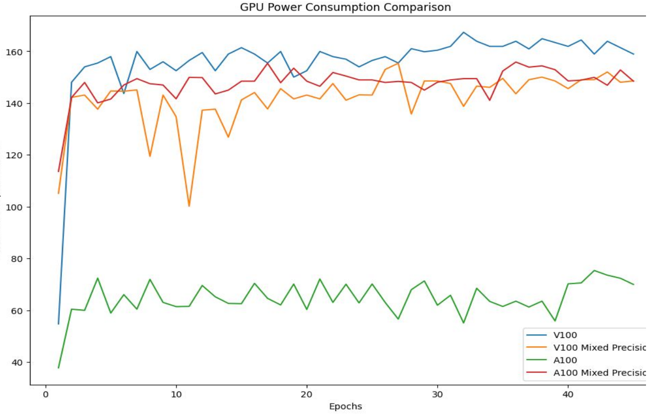
Roofline Models PD-GAN

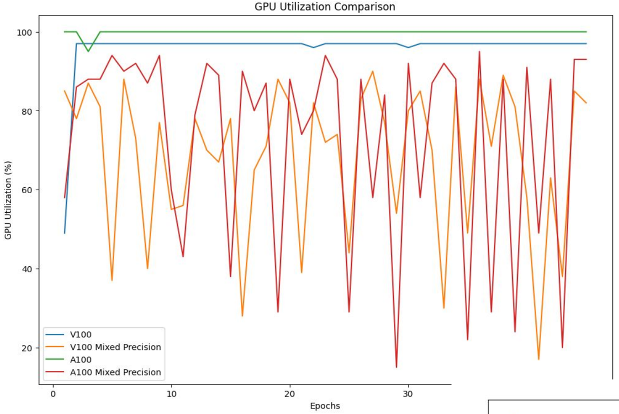


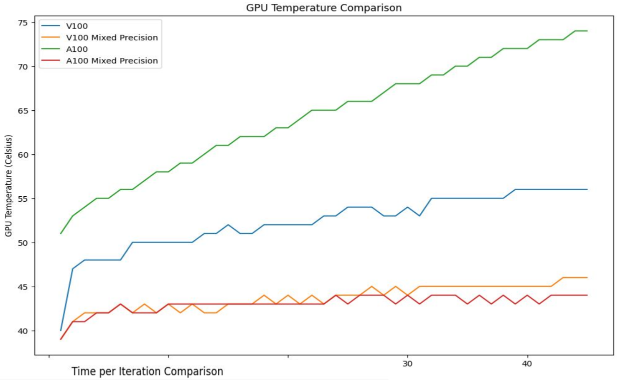


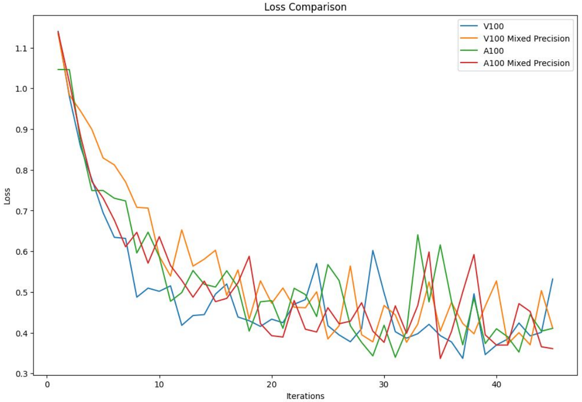
NYU

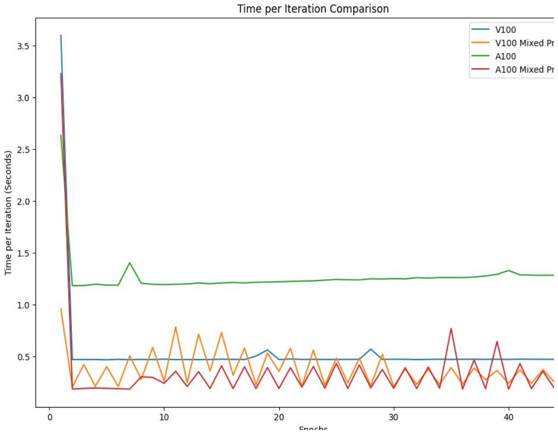
Other Metrics : Progressive GAN





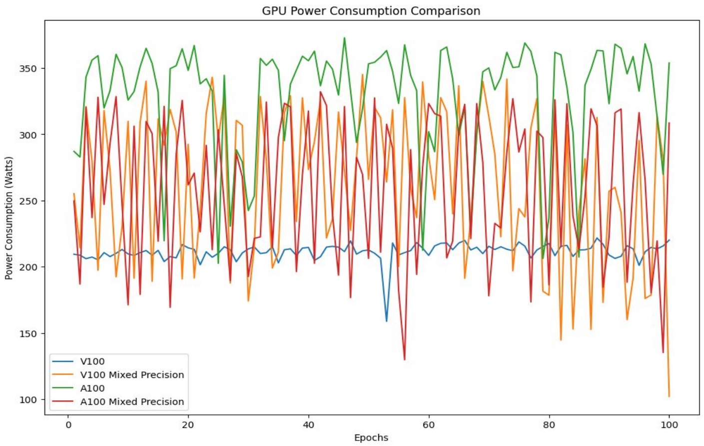


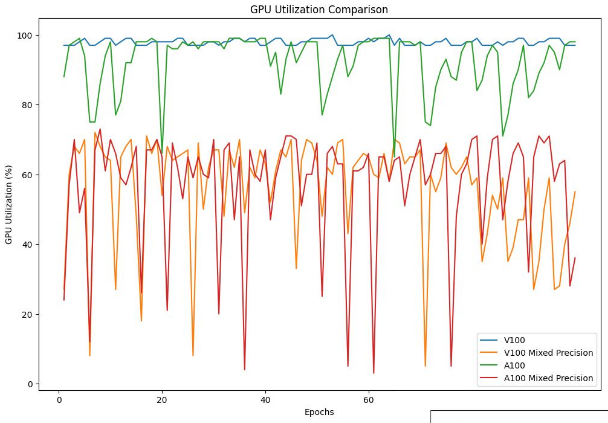


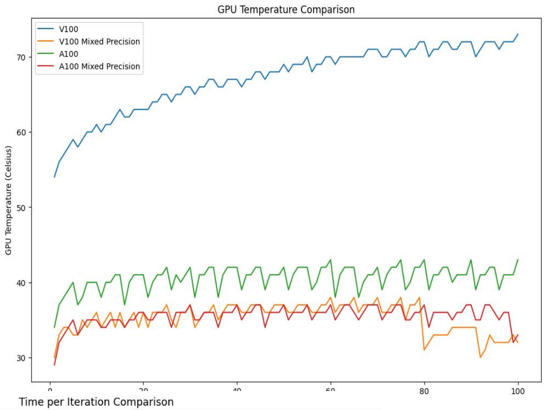


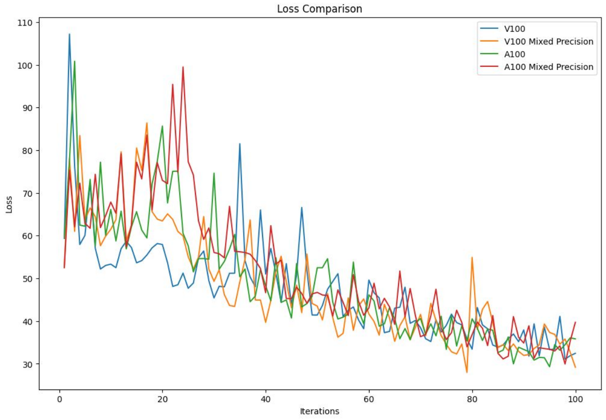
NYU

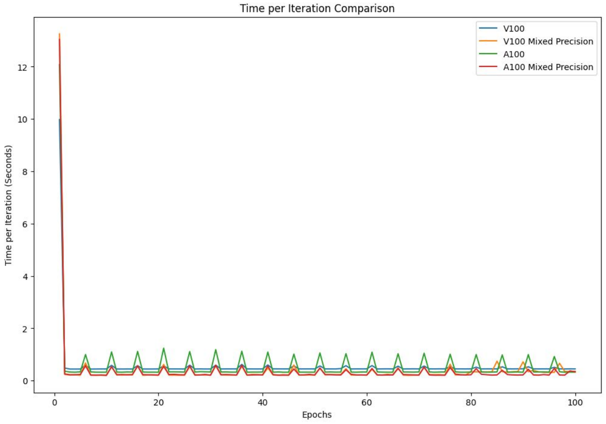
Other Metrics : PD GAN





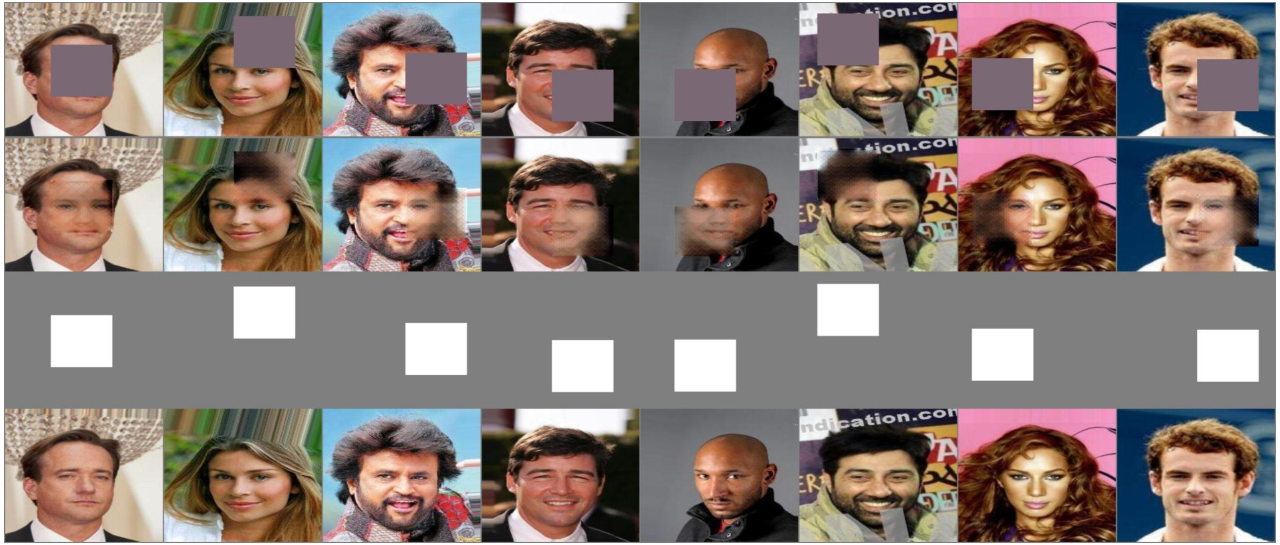






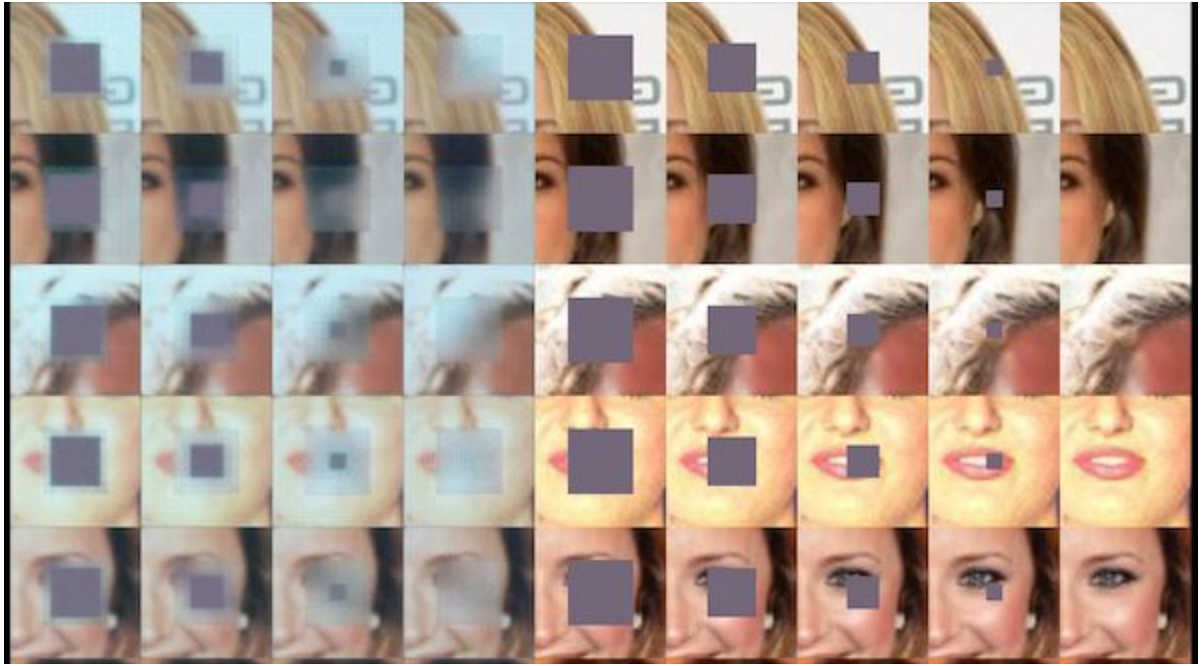
NYU

Demo - PDGAN



Output from PDGAN on celebA after 100 iterations. First row is the input to network, 2nd row is the predicted outputs, 3rd rows is the random mask generated on base images, and 4th row is the ground truth.

Demo - Progressive GAN



Output of a Progressive GAN after 100 Iterations. First 4 columns are the fake image created by the discriminator and the next 5 columns are the input images.

NYU

Conclusion

We found that the bottleneck in our inpainting models was a result of their convolutional layers backward pass. We addressed this issue by implementing mixed precision training, which enabled the use of lower precision data types to speed up computations of the backward pass. Profiling results showed that this optimization led to significant performance improvements across both our Progressive-GAN and PD-GAN models, with the A100 GPU demonstrating the best speedup of 3.67x.

Further Improvements

- **Architecture optimization:** In addition to mixed precision training, optimizing the architecture of the models may help reduce the number of calculations during backpropagation and improve performance.
- **Hyperparameter tuning:** Fine-tuning the hyperparameters of the models such as learning rate, batch size, and regularization can help optimize the training process.
- **Distributed training:** Implementing distributed training techniques such as data parallelism or model parallelism can improve training speed and efficiency on multiple GPUs or machines.

Github Code

https://github.com/tommarvoloriddle/Optimization-of-Image-inpainting-GANs/tree/main

References

- Semantic Image Inpainting with Progressive Generative Networks
- PD-GAN: Probabilistic Diverse GAN for Image Inpainting

NYU