CNN and GAN with GPU Acceleration - Final Report

Convolutional Neural Networks (CNNs) with GPU Acceleration

Overview:

CNNs are particularly well-suited for image processing tasks due to their ability to capture spatial hierarchies in data.

When trained using GPUs, CNNs benefit immensely from parallelization, especially in convolutional and matrix operations.

Experiment Setup:

- Dataset: Cats vs Dogs (Kaggle)

- Input: 150x150 grayscale images

- Architecture:

Conv2D -> ReLU -> MaxPooling -> Conv2D -> ReLU -> MaxPooling -> Flatten -> Dense -> Output

- Activation: ReLU, Sigmoid

- Loss: Binary Crossentropy

- Optimizer: Adam

GPU Utilization:

Convolution and matrix multiplication layers utilize the GPU cores in parallel. This leads to much faster training

compared to CPUs, particularly as batch sizes and model complexity increase.

Visualizing Filters:

Using Keras, we accessed and visualized the first layer's learned filters:

filters, _ = model.layers[0].get_weights()

This allowed us to inspect what the model was learning in early convolutional layers.

Results:

- Epochs: 10

- Accuracy: ~85% with minimal tuning

- Training Time: ~5x faster on GPU vs CPU

Conclusion:

CNNs are one of the best examples of GPU utilization in AI, due to highly parallelizable operations.

Generative Adversarial Networks (GANs) with GPU Acceleration

Overview:

GANs consist of two models: a Generator (G) and a Discriminator (D), trained in opposition.

The Generator learns to create realistic data while the Discriminator learns to distinguish fake from real.

Why GPU Acceleration?

- GANs train slowly due to multiple forward and backward passes for both G and D.

- Parallel GPU cores help speed up training of both networks simultaneously.

Experiment Setup:

- Dataset: MNIST (handwritten digits)

- Generator: Dense layers with ReLU/Tanh

- Discriminator: Dense layers with LeakyReLU/Sigmoid

- Loss: Binary Crossentropy

- Optimizer: Adam

Training Loop:

Each training step involves:

1. Generating fake images from random noise (G)

2. Training D on real and fake images

3. Training G to fool D

```
# Sample generator input
noise = np.random.normal(0, 1, (batch_size, latent_dim))
generated_images = generator.predict(noise)
```

Results:

- Trained for 10,000 steps

- Generated realistic MNIST digits

- Training Time: ~4x faster on GPU

Conclusion:

GANs demand significant compute, making them a perfect use-case for GPU acceleration.