Weekly Progress

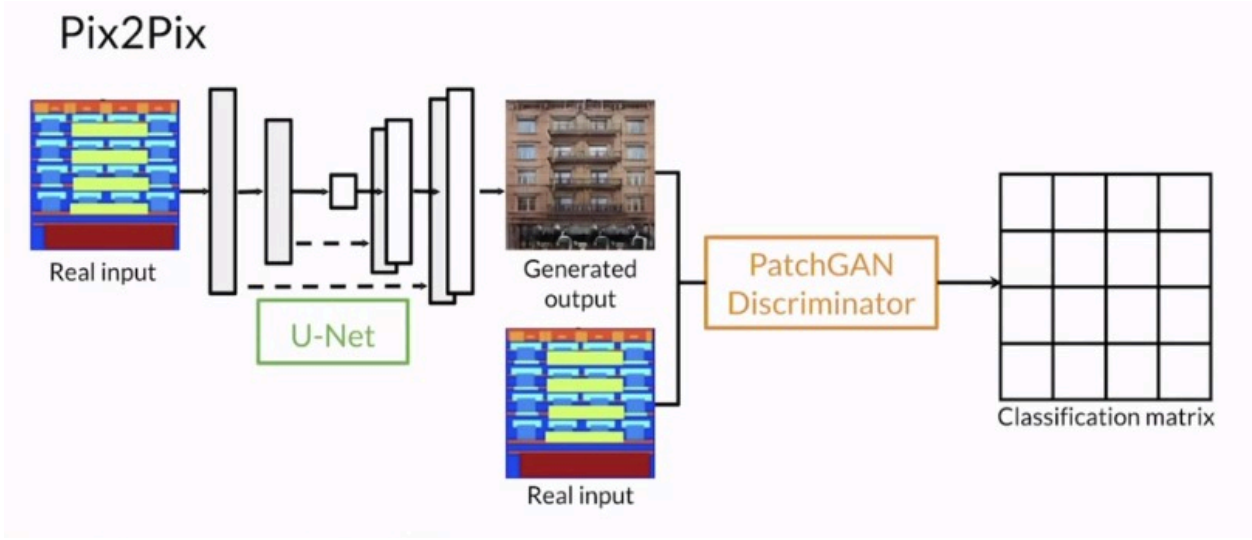
Project Title : 17. High-Resolution Image Synthesis with GANs

Week 1 :

Dataset : https://www.kaggle.com/datasets/jessicali9530/celeba-dataset

Architecture

We're gonna be using Pix2Pix Architecture

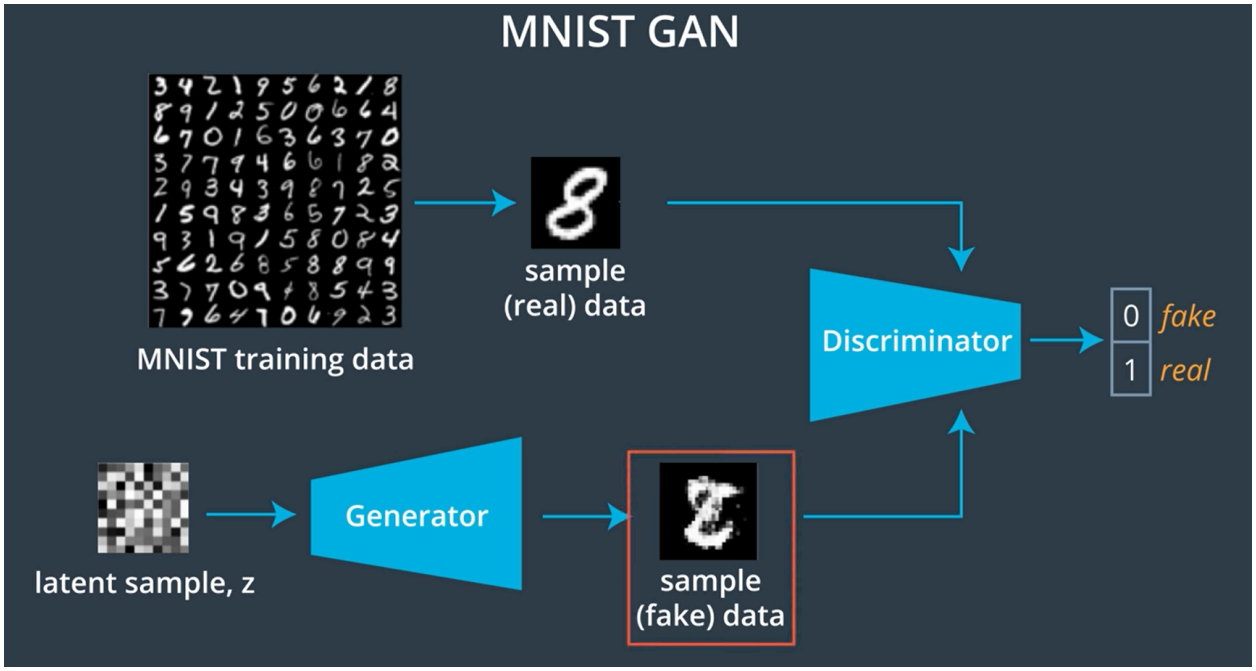


Model Architecture

Loss Function Definitions

1. **Generator Loss**:
    - L1 Loss: Measures the absolute difference between the generated image and the target image.
    - Adversarial Loss: Uses binary cross-entropy to evaluate how well the discriminator classifies the generated image as real.
2. **Discriminator Loss**:
    - This will use binary cross-entropy to determine how well the discriminator can distinguish between real and fake images.
3. **Combined GAN Loss**:
    - This combines the generator loss components, specifically defined as L1 loss plus a weighted binary cross-entropy loss.

For Starters we decided to train a model for generating MNIST images using GAN



Objective: The goal is to generate high-quality handwritten digit images based on the MNIST dataset, which consists of 28x28 pixel grayscale images of digits from 0 to 9.

Methodology:

- Using a GAN architecture, which consists of a Generator and a Discriminator.
- The Generator will learn to create new images that resemble the MNIST digits, while the Discriminator will evaluate the authenticity of the generated images compared to real samples from the dataset.
- Considering various architectures (like DCGAN or Progressive Growing GAN) to optimize image quality.

Next Steps:

- Setting up the GAN architecture, training the model, and evaluating the generated images.
- Exploring hyperparameter tuning for improved performance.

Architecture of the Generator:

- The generator consists of four layers: One input and output and two hidden layers.
- All the layers except the final layer is a combination of three steps: the transposed convolution layer for upscaling the image, the batch normalization for normalizing the

output for a layer and ReLU activation for introducing non linearity so that the models can handle complex inputs.

- In the final layer: instead of ReLU we use the Tanh activation function and the batch normalization is not applied. The benefit of using Tanh activation is that it gives a normalized distribution in [-1,1] which is the same as the distribution of the input image for the MNIST dataset. The batch normalization is not applied in the last layer because it distorts the output distribution making it difficult to predict.

Architecture of the Discriminator:

- The discriminator uses three layers: out of which two are input and the output layer and one is a hidden layer.
- In each of the layers except the final layer we perform three operations: Convolution for **downsampling** the image, batch **normalization** for normalizing the output for a layer and **Leaky ReLU** activation function for intr**oducing non-linearity** into the model.
- Unlike ReLU, Leaky ReLU is preferred because it prevents the condition of dead neurons: as for negative inputs the graph does not have slope zero and so some gradient is provided for the negative inputs also: and so learning is not stopped.
- In the final layer only **convolution** is used which produces a final score that indicates whether the input image is real or fake. Including a non-linear activation function (like Leaky ReLU) in the output layer is not necessary for binary classification tasks.

Loss Function:

- We used the Binary Cross Entropy loss function because it is useful when it comes to binary classification tasks. It measures the difference between the predicted probabilities and the actual binary labels (0 or 1).

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right]$$

Where, N is the number of samples

y_i is the true label

And **p_i** is the predicted probability.

- For good predictions the BCE is close to 0 and for bad predictions the value of BCE is large.

Steps used in Training:

- We begin with training the discriminator: a batch of fake images is generated from a noise sample which is then fed into the discriminator with label = 0 i.e. fake and then a real set of images is fed into the discriminator with label=1 i.e. real: the loss for both of them is calculated and average value of them is taken to train the weights of the discriminator.

- After that in the same iteration we produce another batch of fake images using the generator and calculate the loss for the generator by giving these images label=1 i.e. real as the goal of the generator is to produce real images. Once the loss is calculated we update the parameters of the generator.
- We repeat this process iteratively until our training is complete. The important thing is to not to train the generator and the discriminator simultaneously.

Adam Optimizer:

- The Adam Optimization provides us with adaptive learning rate and thus helps in dealing with sparse gradients and improves convergence.

References: The DCGAN Paper

DCGAN vs. Pix2Pix: Outline the differences and why you chose Pix2Pix for this stage of your project. DCGAN could be useful as a baseline model, while Pix2Pix allows for conditional image generation, which may improve the model's control over complex patterns.

Progress beyond MNIST :

With MNIST as our solid base, the next steps for the project will be to expand into more intricate and diverse images. We will be testing and trying out CIFAR-10, SVHN, or Fashion MNIST datasets. These are now larger and more complex datasets, yet they're still manageable in terms of the size and complexity, but they bring new challenges of color channels and much more complex patterns that we can test and adapt our model further. For instance, CIFAR-10 has ten classes of 32x32 color images, adding color and diversity with shapes of objects. The other dataset is Fashion MNIST, which provides us with grayscale images of clothing items. This means the model needs to understand different textures and patterns. Using these datasets, we hope to understand how well our GAN can generalize to more detailed and variable data. Once the appropriate dataset is selected, we will proceed to refine our architecture and optimize hyperparameters to handle these new data characteristics.

This would indicate that the processing of our new data in GAN does not require simple architectural changes. One such example may be adding a few extra convolutional layers for both the generator and discriminator to express higher resolutions. For the generator, we shall require more filters, maybe even add some more for finer detailing in the images produced in this scenario. We look into the progressive growth of techniques that have the GAN generated at a low resolution and then increase resolutions progressively. It may help to make the process more stable and feasible for producing realistic images at high detail levels. We could also look to balance the adversaries with content loss by further investigating batch size, learning rates, and weights applied to the loss function. We believe that these modifications to the architecture will prove critical in the production of good quality images throughout the datasets.

Reference for high quality image to image translation