|  |  |  |
| --- | --- | --- |
|  | **MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE**  **Kodambakkam, Chennai-600024** |  |

**NM1009 –GENERATIVE AI FOR ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**TOPIC: Handwriting Synthesis using Pytorch (GAN Model)**

|  |  |
| --- | --- |
| **FACULTY MENTOR:** | **Dr. S. AARTHI** |
|  |  |
|  |  |

**Project submitted by,**

**G Harshavardhan (311521104016)**

***Project report format***

1. **ABSTRACT**
2. **INTRODUCTION**
   1. Project Overview
   2. Purpose
3. **IDEATION AND PROPOSED SOLUTION**
   1. Problem statement definition
   2. Ideation and Brainstorming
   3. Proposed Solution
4. **REQUIREMENTS ANALYSIS**
   1. Functional Requirements
   2. Non-Functional Requirements
5. **PROJECT DESIGN** 
   1. Briefing
   2. Solution
6. **SOLUTIONS**
   1. Development Part I
   2. Development Part II
7. **RESULTS**

7.1 Performance Metrics

1. **ADVANTAGES AND DISADVANTAGES**
2. **CONCLUSION**
3. **FUTURE SCOPE**

SOURCE CODE

**ABSTRACT**

This project explores the application of Generative Adversarial Networks (GANs) in synthesis high-resolution hand-written text images using the training dataset. GANs have gained prominence in recent years for their ability to generate realistic synthetic data by learning the underlying distribution of real data. In this implementation, a GAN architecture comprising a generator and a discriminator is employed. The generator network generates synthetic word images from random noise vectors, while the discriminator network evaluates the authenticity of these generated images. Through an adversarial training process, the generator learns to produce images that are increasingly indistinguishable from real word images, while the discriminator becomes more adept at discerning between real and fake images.

The training process involves optimizing the parameters of both networks using the Adam optimizer and backpropagation. The generator is trained to minimize the discrepancy between the distribution of generated images and that of real images, while the discriminator is trained to correctly classify real and fake images. This adversarial training dynamic leads to a Nash equilibrium, where the generator produces realistic images that can effectively fool the discriminator.

The unconventional aspect of involves leveraging recent advancements in generative modeling, such as variational autoencoders (VAEs) or generative adversarial networks (GANs), to capture the intricate patterns and nuances of human handwriting. By training on a diverse dataset of handwritten samples, our model learns to generate strokes that mimic the variability and natural flow of real handwriting. Using deep learning framework Pytorch we synthesise the words both conventional and non-conventional methods.

Overall, this project showcases the potential of GANs in generating synthetic data for various applications, including letter generation, data augmentation, and artistic expression. By leveraging the power of deep learning and adversarial training, GANs offer a promising approach to generating realistic word images and advancing the field of generative artificial intelligence.

**INTRODUCTION**

In the realm of artificial intelligence and machine learning, the ability to generate realistic data has been a longstanding challenge. Pytorch have emerged as a powerful solution to this challenge, offering a novel approach to generating synthetic data that closely resembles real-world samples.

This project focuses on harnessing the capabilities of Pytorch to generate hand-written word images, a task with significant implications for various domains such as computer vision, pattern recognition, and word classification systems.

The primary objective of this project is to demonstrate the effectiveness of GANs in generating high-quality word images that are visually indistinguishable from real ones. By training a GAN architecture on the training dataset, we aim to produce synthetized conventional and non-conventional images that exhibit characteristics similar to those found in the original dataset.

We examine the role of hyperparameters in shaping the performance and stability of the GAN model. Furthermore, we evaluate the quality of generated word images through visualizations and performance metrics, providing insights into the efficacy of the GAN framework for word image generation tasks.

***Project Overview:***

The objective of this project is to develop a handwriting synthesis system that can generate realistic handwritten text using PyTorch. The system aims to replicate the natural variability and fluidity of human handwriting while also ensuring coherence and legibility in the generated text.

***Purpose:***

The purpose of this project is to advance handwriting synthesis using PyTorch, contributing to research, innovation, and education. By developing a robust system capable of generating realistic handwritten text, the project aims to provide practical utility in wordal communication, font creation, and document analysis while fostering community collaboration.

**IDEATION AND PROPOSED SOLUTION**

***Problem Statement***

The problem statement involves generating lifelike hand-written word images using Generative Adversarial Networks (GANs). Despite the success of GANs in generating synthetic data, producing realistic word images poses challenges due to the intricate details and variability of hand-written characters. The system aims to generate high-quality handwritten text that closely resembles human handwriting, offering practical utility in various applications while advancing research and technology in the domain of handwriting synthesis.

***Ideation and Brainstorming:***

During the ideation and brainstorming phase, several key considerations were taken into account to formulate an effective approach for generating lifelike hand-written word images using Generative Adversarial Networks (GANs).

1. Understanding GAN Architecture: The first step involved gaining a thorough understanding of the GAN architecture, including the roles of the generator and discriminator networks, and the adversarial training process.

2. Exploring Training Dataset: The Training dataset, containing a large number of labeled hand-written word images, served as the primary dataset for training the GAN model. Exploring the dataset helped in understanding the characteristics and variability of hand-written words.

3. Reviewing Related Work: Researching existing literature and projects related to GAN-based image generation, particularly focusing on word image generation, provided valuable insights into various methodologies, techniques, and best practices.

4. Data Preprocessing Techniques: Exploring data preprocessing techniques such as normalization and reshaping of the TRAINING images to ensure compatibility with the GAN model architecture.

***Proposed Solution:***

To address the problem of generating hand-written word images using Generative Adversarial Networks (GANs), the proposed solution involves a systematic approach encompassing problem definition, design thinking, innovation, and development phases.

**Project Steps**

**Phase 1: Problem Definition and Design Thinking**

**Problem Definition:** In this phase, the problem of generating realistic hand-written word images is clearly defined. Design thinking methodologies are employed to gain a deeper understanding of user requirements, identify pain points, and define the desired outcomes of the project.

**Design Thinking:**

* **Empathize:** Understand the needs and preferences of users who interact with hand-written word images, such as word recognition systems and machine learning researchers.
* **Define**: Clearly articulate the problem statement, objectives, and success criteria for the project.
* **Ideate**: Brainstorm potential solutions and approaches for generating lifelike word images using GANs, considering factors such as dataset selection, network architecture, and evaluation metrics.
* **Prototype**: Develop prototypes or mockups to visualize and test different design concepts and methodologies.
* **Test**: Gather feedback from stakeholders and iterate on the proposed solutions to refine and improve their effectiveness.

**Phase 2: Innovation**

During this phase, innovative techniques and methodologies are explored to enhance the performance and quality of the GAN-based word image generation process. This may involve experimenting with novel network architectures, optimization algorithms, and data augmentation techniques to achieve superior results.

**Phase 3: Development Part 1**

In the first development phase, the foundational components of the project are implemented. This includes data preprocessing, GAN model construction (generator and discriminator networks), definition of loss functions, selection of optimization algorithms, and initial training of the GAN model using the TRAINING dataset.

**Phase 4: Development Part 2**

The second development phase focuses on fine-tuning and optimizing the GAN model for improved performance and stability. This may involve hyperparameter tuning, regularization techniques, and advanced training strategies to mitigate issues such as mode collapse and training instability. Additionally, the generated word images are evaluated and refined to ensure high-fidelity results.

**Phase 5: Project Documentation & Submission**

The project is finalized and submitted, along with any supplementary materials or artifacts generated during the development process.

**Documentation**

Comprehensive documentation covering all aspects of the project, including problem definition, design rationale, implementation details, experimental results, and future recommendations, is prepared. This documentation serves as a valuable resource for understanding the project's objectives, methodologies, and outcomes.

**Submission**

1. Share the GitHub repository link containing the project's code and files.
2. Write a detailed README file explaining the project.

**REQUIREMENT ANALYSIS**

***Functional Requirements***

|  |  |  |
| --- | --- | --- |
| **S.No** | **Requirement** | **Description** |
| FR1 | Load TRAINING dataset | The system should be able to load the TRAINING dataset, which contains a large collection of labeled hand-written word images, for training the Generative Adversarial Network (GAN) model. |
| FR2 | Preprocess dataset | The system should preprocess the TRAINING dataset by reshaping the images, normalizing pixel values, and splitting it into training and testing sets to prepare the data for training the GAN model. |
| FR3 | Build generator model | The system should construct the generator model architecture, comprising layers such as dense, convolutional, and activation layers, to generate synthetic word images from random noise vectors. |
| FR4 | Build discriminator model | The system should construct the discriminator model architecture, consisting of convolutional layers, activation functions, and dropout layers, to distinguish between real and fake word images. |

***Non-Functional Requirements***

|  |  |  |
| --- | --- | --- |
| **S.No** | **Requirements** | **Description** |
| NFR1 | Scalability | The system should be scalable to handle larger datasets and accommodate variations in dataset size, enabling seamless integration with other datasets for potential expansion and experimentation. |
| NFR2 | Security | The system should incorporate appropriate security measures to safeguard sensitive data, protect against unauthorized access or modifications, and ensure the integrity and confidentiality of the TRAINING dataset and generated word images throughout the training and evaluation processes. |
| NFR3 | Reliability | The system should be reliable, with minimal downtime and error handling mechanisms in place to mitigate potential failures or disruptions during training and evaluation procedures, ensuring continuous and uninterrupted operation for long-term experimentation and usage. |
| NFR4 | Performance | The system should be capable of training the GAN model efficiently, with reasonable training times and computational resources, to generate high-quality word images within a reasonable timeframe. |
| NFR5 | Usability | The system should be user-friendly and accessible to researchers and developers, with clear documentation, intuitive interfaces, and informative feedback mechanisms to facilitate ease of use and experimentation with the GAN model. |

**PROJECT DESIGN**

***Briefing:***

The project aims to implement a Generative Adversarial Network (GAN) to generate lifelike hand-written word images using the TRAINING dataset. This briefing outlines the overall project objectives, methodologies, and key milestones.

***Solution***

The solution involves the implementation of a Generative Adversarial Network (GAN) to generate lifelike hand-written word images using the TRAINING dataset with Pytorch***.***

**SOLUTIONS**

***Development: Part 1***

In the first phase of development, foundational components of the project will be implemented. This includes loading and preprocessing the TRAINING dataset, designing the GAN architecture with generator and discriminator networks, defining appropriate loss functions, selecting optimization algorithms, and initiating training of the GAN model.

***Development: Part 2***

The second phase of development focuses on fine-tuning and optimizing the GAN model for improved performance and stability. This involves tuning, regularization techniques, and advanced training strategies to mitigate issues such as mode collapse and training instability. Additionally, the generated word images are evaluated and refined to ensure high-fidelity results.

**RESULTS**

The results phase encompasses the evaluation and validation of the GAN model performance. This includes visualizing the generated word images, assessing their quality and resemblance to real words, and analyzing performance metrics such as image quality measures and discriminator accuracy. The results are documented and analyzed to draw conclusions and insights into the effectiveness of the GAN-based word image generation process.

***Performance Metrics***

|  |  |  |
| --- | --- | --- |
| ***S. No*** | ***Metrics*** | ***Description*** |
| PM1 | Discriminator Loss | Measures the effectiveness of the discriminator network in distinguishing between real and fake word images during training. |
| PM2 | Generator Loss | Indicates how well the generator network is fooling the discriminator by generating realistic word images during the adversarial training process. |
| PM3 | Discriminator Accuracy | Represents the accuracy of the discriminator in correctly classifying real and fake word images, providing insights into the discriminator's ability to differentiate between the two classes. |
| PM4 | Image Quality Measures | Various quantitative measures such as structural similarity index (SSIM), peak signal-to-noise ratio (PSNR), and mean squared error (MSE) can be used to assess the quality and fidelity of the generated word images compared to real words. |
| PM5 | Inception Score | Evaluates the quality and diversity of the generated images by computing the KL divergence between the conditional class distributions of the images and the marginal distribution of the class labels. Higher IS scores indicate better quality and diversity of the generated word images. |

**ADVANTAGES AND DISADVANTAGES:**

***Advantages:***

1. **Data Augmentation**: GANs can augment datasets by generating synthetic data, which is particularly useful when datasets are limited or difficult to obtain. This can improve model generalization and performance.
2. **High-Quality Generation**: GANs are capable of generating high-quality and realistic data, including images, text, and audio, which can be used for various applications such as image synthesis, image editing, and content creation.
3. **Unsupervised Learning**: GANs enable unsupervised learning by learning to represent and generate data without explicit labels or supervision. This allows for discovery of underlying data distributions and patterns.
4. **Creative Applications**: GANs have creative applications in art generation, style transfer, and image manipulation, allowing for the creation of novel and artistic content with diverse visual styles.
5. **Adversarial Training**: The adversarial training process in GANs encourages competition between the generator and discriminator networks, leading to improved model performance and convergence, resulting in better-quality generated data.

***Disadvantages:***

1. **Training Instability**: GANs are prone to training instability, including issues such as mode collapse, where the generator produces limited variations of output, and oscillations in training dynamics, which can hinder convergence.
2. **Mode Collapse**: Mode collapse occurs when the generator learns to produce limited variations of output, ignoring parts of the data distribution. This results in poor diversity and coverage in generated data.
3. **Evaluation Challenges**: Evaluating the performance and quality of GAN-generated data is challenging, as traditional metrics may not accurately capture aspects such as diversity, novelty, and semantic coherence. Developing effective evaluation metrics remains an active area of research.
4. **Computationally Intensive**: Training GANs can be computationally intensive and time-consuming, requiring powerful hardware such as GPUs and significant computational resources. This can limit scalability and accessibility for smaller research teams or organizations.

# **CONCLUSION**

In conclusion, Generative Adversarial Networks (GANs) offer a powerful framework for generating realistic hand-written word images, as demonstrated in this project. By leveraging the adversarial training process, GANs enable the generation of high-quality word images that closely resemble real words from the TRAINING dataset. Despite facing challenges such as training instability and mode collapse, GANs have shown remarkable capabilities in synthesizing diverse and realistic data, with applications ranging from word recognition systems to creative content generation. Through careful design, optimization, and evaluation, GANs hold promise for advancing the field of image generation and contributing to various domains such as artificial intelligence, computer vision, and data augmentation.

**FUTURE SCOPE**

1. **Integrating Improved Realism**: Further refinement of the handwriting synthesis system to enhance the realism and naturalness of generated text, potentially incorporating additional contextual information or fine-tuning the model architecture.
2. **Multimodal Synthesis**: Exploration of multimodal approaches to handwriting synthesis, incorporating additional modalities such as texture, color, or writing instrument variations to generate even more diverse and realistic outputs.
3. **Multilingual Support**: Expansion of the system to support multiple languages and writing systems, accommodating diverse linguistic contexts and enabling broader accessibility and applicability.
4. **Integration with OCR Systems**: Integration of the handwriting synthesis system with Optical Character Recognition (OCR) systems to enhance the recognition and synthesis of handwritten text in digitized documents or images.
5. **Collaborative Research**: Collaboration with experts in fields such as linguistics, psychology, and digital humanities to further understand the cognitive and perceptual aspects of handwriting and inform the development of more sophisticated synthesis techniques.
6. **SOURCE CODE:   
   main.py**

import matplotlib.pyplot as plt

%matplotlib inline

from pathlib import Path

import pickle

import sys

sys.path.append('../')

from utils import draw

from modules import HandwritingSynthesisNetwork, HandwritingPredictionNetwork

from dataset import HandwritingDataset

import torch

root **=** Path('../logs/retrain\_seqlen\_256')

args **=** pickle**.**load(open(root **/** "args.pkl", "rb"))

dataset **=** HandwritingDataset('../' **+** args**.**path, split**=**'test')

model **=** HandwritingSynthesisNetwork(

test\_dataset**.**vocab\_size,

args**.**dec\_hidden\_size, args**.**dec\_n\_layers,

args**.**n\_mixtures\_attention, args**.**n\_mixtures\_output

)**.**cuda()

model**.**load\_state\_dict(torch**.**load(root **/** 'model.pt'))

chars **=** torch**.**from\_numpy(

dataset**.**sent2idx(string)

)**.**long()[**None**]**.**cuda()

chars\_mask **=** torch**.**ones\_like(chars)**.**float()**.**cuda()

**with** torch**.**no\_grad():

out **=** model**.**sample(chars, chars\_mask, maxlen**=**2000)[0]**.**cpu()**.**numpy()

\_ **=** draw(out[0], save\_file**=None**)

root **=** Path('../logs/new\_code\_uncond\_256')

args **=** pickle**.**load(open(root **/** "args.pkl", "rb"))

model **=** HandwritingPredictionNetwork(

args**.**dec\_hidden\_size, args**.**dec\_n\_layers,

args**.**n\_mixtures\_output

)**.**cuda()

model**.**load\_state\_dict(torch**.**load(root **/** 'model.pt'))

**with** torch**.**no\_grad():

out **=** model**.**sample(batch\_size**=**1)**.**cpu()**.**numpy()

\_ **=** draw(out[0], save\_file**=None**)

**Modules.py**

import numpy as np

import torch

import torch.nn as nn

import torch.nn.functional as F

from utils import concatenate\_dict

def mixture\_of\_bivariate\_normal\_nll(

data, log\_pi, mu, log\_sigma, rho, eps=1e-6

):

x, y = data.unsqueeze(-2).unbind(-1)

mu\_1, mu\_2 = mu.unbind(-1)

log\_sigma\_1, log\_sigma\_2 = log\_sigma.unbind(-1)

sigma\_1 = log\_sigma\_1.exp() + eps

sigma\_2 = log\_sigma\_2.exp() + eps

# Compute log prob of bivariate normal distribution

Z = torch.pow((x - mu\_1) / sigma\_1, 2) + torch.pow((y - mu\_2) / sigma\_2, 2)

Z -= 2 \* rho \* ((x - mu\_1) \* (y - mu\_2)) / (sigma\_1 \* sigma\_2)

log\_N = -Z / (2 \* (1 - rho \*\* 2) + eps)

log\_N -= np.log(2 \* np.pi) + log\_sigma\_1 + log\_sigma\_2

log\_N -= .5 \* torch.log(1 - rho \*\* 2 + eps)

# Use log\_sum\_exp to accurately compute log prob of mixture distribution

nll = -torch.logsumexp(log\_pi + log\_N, dim=-1)

return nll

def mixture\_of\_bivariate\_normal\_sample(

log\_pi, mu, log\_sigma, rho, eps=1e-6, bias=0.

):

batch\_size = log\_pi.shape[0]

ndims = log\_pi.dim()

if ndims > 2:

# Collapse batch and seq\_len dimensions

log\_pi, mu, log\_sigma, rho = [

x.reshape(-1, \*x.shape[2:])

for x in [log\_pi, mu, log\_sigma, rho]

]

# Sample mixture index using mixture probabilities pi

pi = log\_pi.exp() \* (1 + bias)

mixture\_idx = pi.multinomial(1).squeeze(1)

# Index the correct mixture for mu, log\_sigma and rho

mu, log\_sigma, rho = [

x[torch.arange(mixture\_idx.shape[0]), mixture\_idx]

for x in [mu, log\_sigma, rho]

]

# Calculate biased variances

sigma = (log\_sigma - bias).exp()

# Sample from the bivariate normal distribution

mu\_1, mu\_2 = mu.unbind(-1)

sigma\_1, sigma\_2 = sigma.unbind(-1)

z\_1 = torch.randn\_like(mu\_1)

z\_2 = torch.randn\_like(mu\_2)

x = mu\_1 + sigma\_1 \* z\_1

y = mu\_2 + sigma\_2 \* (z\_2 \* ((1 - rho \*\* 2) \*\* .5) + z\_1 \* rho)

# Uncollapse the matrix to a tensor (if necessary)

sample = torch.stack([x, y], 1)

if ndims > 2:

sample = sample.view(batch\_size, -1, 2)

return sample

class OneHotEncoder(nn.Module):

def \_\_init\_\_(self, vocab\_size):

super().\_\_init\_\_()

self.vocab\_size = vocab\_size

def forward(self, arr, mask):

shp = arr.size() + (self.vocab\_size,)

one\_hot\_arr = torch.zeros(shp).float().cuda()

one\_hot\_arr.scatter\_(-1, arr.unsqueeze(-1), 1)

return one\_hot\_arr \* mask.unsqueeze(-1)

class GaussianAttention(nn.Module):

def \_\_init\_\_(self, hidden\_size, n\_mixtures, attention\_multiplier=.05):

super().\_\_init\_\_()

self.linear = nn.Linear(hidden\_size, 3 \* n\_mixtures)

self.n\_mixtures = n\_mixtures

self.attention\_multiplier = attention\_multiplier

def forward(self, h\_t, k\_tm1, ctx, ctx\_mask):

B, T, \_ = ctx.shape

device = ctx.device

alpha, beta, kappa = torch.exp(self.linear(h\_t))[:, None].chunk(3, dim=-1) # (B, 1, K) each

kappa = kappa \* self.attention\_multiplier + k\_tm1.unsqueeze(1)

u = torch.arange(T, dtype=torch.float32).to(device)

u = u[None, :, None].repeat(B, 1, 1) # (B, T, 1)

phi = alpha \* torch.exp(-beta \* torch.pow(kappa - u, 2)) # (B, T, K)

phi = phi.sum(-1) \* ctx\_mask

w = (phi.unsqueeze(-1) \* ctx).sum(1)

attention\_vars = {

'alpha': alpha.squeeze(1),

'beta': beta.squeeze(1),

'kappa': kappa.squeeze(1),

'phi': phi,

}

return w, attention\_vars

class HandwritingSynthesisNetwork(nn.Module):

def \_\_init\_\_(

self, vocab\_size, hidden\_size, n\_layers,

n\_mixtures\_attention, n\_mixtures\_output

):

super().\_\_init\_\_()

self.encoder = OneHotEncoder(vocab\_size)

self.lstm\_0 = nn.LSTMCell(3 + vocab\_size, hidden\_size)

self.lstm\_1 = nn.LSTM(3 + vocab\_size + hidden\_size, hidden\_size, batch\_first=True)

self.lstm\_2 = nn.LSTM(3 + vocab\_size + hidden\_size, hidden\_size, batch\_first=True)

self.attention = GaussianAttention(hidden\_size, n\_mixtures\_attention)

self.fc = nn.Linear(

hidden\_size \* 3, n\_mixtures\_output \* 6 + 1

)

self.hidden\_size = hidden\_size

self.vocab\_size = vocab\_size

self.n\_mixtures\_output = n\_mixtures\_output

def \_\_init\_\_hidden(self, bsz):

hid\_0 = torch.zeros(bsz, self.hidden\_size \* 2).float().cuda()

hid\_0 = hid\_0.chunk(2, dim=-1)

hid\_1, hid\_2 = None, None

w\_0 = torch.zeros(bsz, self.vocab\_size).float().cuda()

k\_0 = torch.zeros(bsz, 1).float().cuda()

return hid\_0, hid\_1, hid\_2, w\_0, k\_0

if \_\_name\_\_ == '\_\_main\_\_':

vocab\_size = 60

hidden\_size = 400

n\_layers = 3

K\_att = 6

K\_out = 20

model = HandwritingSynthesisNetwork(

vocab\_size, hidden\_size, n\_layers,

K\_att, K\_out

).cuda()

chars = torch.randint(0, vocab\_size, (4, 50)).cuda()

chars\_mask = torch.ones\_like(chars).float()

strokes = torch.randn(4, 300, 3).cuda()

strokes\_mask = torch.ones(4, 300).cuda()

loss = model.compute\_loss(chars, chars\_mask, strokes, strokes\_mask)

print(loss)

out = model.sample(chars, chars\_mask)

print(out[0].shape)

**APPENDIX:**

Source code @github: [github.com/Harshavardhan-Gopi/Generative-AI](http://github.com/Harshavardhan-Gopi/Generative-AI)