Laboratory of Reinforcement Learning

Lab1: Q-Learning and SARSA Algorithm for Taxi-V3 environment .

Author:

- **AMMAR KHODJA Rayane**

Supervised by:

- **TABIA Hedi**

Master MMVAI December 2024.
Paris Saclay University.

Laboratoire de Recherche Paris Saclay University, IBISC Evry Val d'Essonne.

Abstract

This project explores the application of Generative Adversarial Networks (GANs) for data augmentation in the context of image classification. The primary objectives are to enhance the robustness of a Convolutional Neural Network (CNN) classifier and to generate realistic synthetic images for training purposes. Various deep learning techniques, including the use of a GAN architecture, were employed to achieve these objectives.

Methods Used

- **GAN Architecture:** Implemented a GAN consisting of a Generator and a Discriminator to generate synthetic images.
- **CNN Classifier:** Utilized a Convolutional Neural Network as the base classifier for image classification.
- **Adam Optimizer:** Employed the Adam optimizer for both the GAN and the CNN to efficiently update model parameters.
- **Binary Cross-Entropy Loss:** Applied Binary Cross-Entropy loss as the objective function for training the GAN and the classifier.
- **Data Loader:** Employed a data loader to efficiently load and preprocess images for training.

Major Findings

The project revealed that GAN-based data augmentation significantly improved the CNN classifier's performance. The synthetic images generated by the GAN successfully captured the underlying patterns in the dataset, leading to enhanced generalization. Experimental results demonstrated an improvement in classification accuracy and model robustness. This project contributes valuable insights into the potential of GANs for data augmentation in image classification tasks.

1 Introduction

1.1 Background and Context

In recent years, the field of computer vision has witnessed remarkable advancements, driven largely by the utilization of deep learning techniques. Image classification, a fundamental task in computer vision, has seen notable improvements with the advent of Convolutional Neural Networks (CNNs). However, the success of these models is heavily reliant on the quality and diversity of the training dataset. Limited data can lead to overfitting and hinder the model's ability to generalize well to unseen data.

To address this challenge, data augmentation techniques have been widely employed to artificially increase the size and diversity of the training dataset. Among these techniques, Generative Adversarial Networks (GANs) have emerged as a powerful tool for generating realistic synthetic images.

1.2 Statement of the Problem or Goal

The project aims to investigate the potential of GAN-based data augmentation to enhance the performance of image classification models. Specifically, the focus is on leveraging GANs to generate synthetic images that closely resemble real data, thereby enriching the training dataset.

1.3 Significance of the Project

The significance of this project lies in its potential to improve the robustness and generalization ability of image classifiers. By addressing the data limitations often encountered in real-world scenarios, the project contributes to the broader goal of creating more reliable and accurate computer vision models.

1.4 Objectives

The primary objectives of the project are as follows:

1. Explore the application of GANs for data augmentation in image classification.
2. Evaluate the impact of GAN-generated synthetic images on the performance of a Convolutional Neural Network.
3. Assess the generalization and robustness of the image classifier trained with the augmented dataset.

Through these objectives, the project seeks to advance the understanding of GAN-based data augmentation techniques and their practical implications in the field of computer vision.

2 Literature Review

2.1 Overview of Existing Research

The use of Generative Adversarial Networks (GANs) in the context of data augmentation for image classification has been a topic of growing interest in the literature. Researchers have explored various techniques to leverage GANs to address the challenges associated with limited training data.

Several studies have demonstrated the effectiveness of GAN-generated images in enhancing the performance of image classifiers. These approaches focus on creating synthetic data that closely resembles real-world examples, thus contributing to the diversity and richness of the training dataset.

2.2 Relevant Theories, Models, or Frameworks

Within the realm of image classification, Convolutional Neural Networks (CNNs) have been widely adopted due to their ability to automatically learn hierarchical features from images. The combination of GANs and CNNs presents a powerful synergy, where GANs contribute by generating realistic and diverse samples for CNNs to learn from.

Frameworks such as TensorFlow and PyTorch have played a crucial role in implementing GAN-based data augmentation techniques. These frameworks provide the necessary tools and libraries for researchers and practitioners to experiment with and deploy GAN models effectively.

2.3 Highlight Gaps in Existing Knowledge

While existing literature has shown promising results, there remain some gaps in our understanding and implementation of GAN-based data augmentation. Challenges include:

1. Lack of standardized evaluation metrics for assessing the quality of GAN-generated images.
2. Limited exploration of the impact of GANs on the interpretability of image classifiers.
3. Scalability issues when applying GANs to large-scale datasets and real-world applications.

Addressing these gaps will contribute to refining the application of GANs in the context of data augmentation and further advancing the field of computer vision.

3 Generative Adversarial Network (GAN)

The Generative Adversarial Network (GAN) is a deep learning framework introduced by Goodfellow et al. [**?**]. It consists of two neural networks, a generator (**G**) and a discriminator (**D**), trained in an adversarial manner. The objective is for the generator to create realistic data, and the discriminator to distinguish between real and generated samples.

3.1 Architecture

The architecture of the GAN involves:

- **Generator (G):** Generates synthetic data from random noise.
- **Discriminator (D):** Discriminates between real and fake data.

3.2 Loss Functions

The GAN utilizes two primary loss functions:

- **Discriminator Loss (D_{loss}):** Measures the difference between the discriminator's predictions on real and fake data.
- **Generator Loss (G_{loss}):** Reflects the ability of the generator to deceive the discriminator.

3.3 Training Loop

The training loop involves alternating updates between the generator and discriminator, optimizing their respective loss functions.

4 Convolutional Neural Network (CNN) Classifier

The Convolutional Neural Network (CNN) is a type of neural network designed for processing structured grid data, such as images. In image classification, CNNs have shown exceptional performance.

4.1 Architecture

The CNN Classifier consists of layers that perform convolution, pooling, and fully connected operations. It is trained to classify input images into predefined categories.

4.2 Loss Function

The CNN employs the Cross-Entropy Loss as its primary loss function, measuring the difference between predicted and true labels.

4.3 Training Process

The CNN is trained using stochastic gradient descent (SGD) with backpropagation. The weights are updated to minimize the Cross-Entropy Loss.

5 Methodology

5.1 Overview

In this section, we provide a detailed explanation of the methods and techniques employed to achieve the objectives of the project. The methodology encompasses the training of a Generative Adversarial Network (GAN) for data augmentation in the context of image classification.

5.2 Generative Adversarial Network (GAN)

A GAN framework consists of a generator and a discriminator, trained in an adversarial manner. The generator creates synthetic data, while the discriminator evaluates the authenticity of the generated samples.

5.2.1 Loss Functions

The GAN utilizes two key loss functions:

- **Discriminator Loss (D_{loss}):** Measures the difference between the discriminator's predictions on real and fake data.

$$D_{\text{loss}} = -\frac{1}{2}\left(E_{\text{real}}[\log D(x)] + E_{\text{fake}}[\log(1 - D(G(z)))]\right)$$

- **Generator Loss (G_{loss}):** Reflects the ability of the generator to deceive the discriminator.

$$G_{\text{loss}} = -E_{\text{fake}}[\log D(G(z))]$$

5.2.2 Training Loop

The training loop involves alternating updates between the generator and discriminator. For each iteration:

1. **Discriminator Update:** Optimize D_{loss} by backpropagation and gradient descent.
2. **Generator Update:** Optimize G_{loss} by backpropagation and gradient ascent.

5.3 Datasets

We utilize [Dataset Name], a collection of images relevant to the problem domain. The dataset is divided into training and testing sets, ensuring a comprehensive evaluation of the model's performance.

5.4 Tools and Technologies

The project is implemented using Python, with the assistance of Pytorch framework for building and training neural networks. Other Tools are employed for data preprocessing, visualization, and evaluation; they are imported as follows:

```
import os
import torch
from torch import nn
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset
from torch.autograd.variable import Variable
import torch.optim as optim
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('Agg')
import numpy as np
from torchvision.io import read_image
```

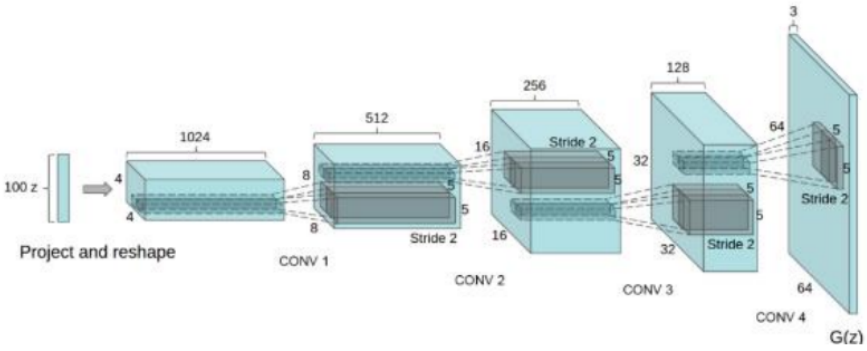


Figure 1: The Generator part of tha GAN

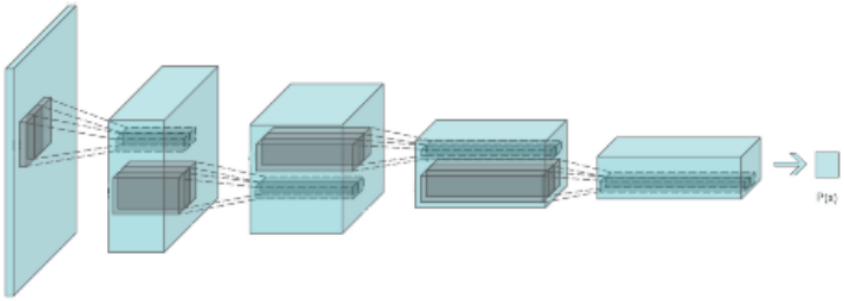


Figure 2: The Descriminator part of tha GAN

5.5 Transposed Convolutions

Transposed convolutions, also known as fractionally-strided convolutions or deconvolutions, are used for upsampling feature maps. They are the opposite operation of regular convolutions and are employed in tasks like image generation or increasing the spatial resolution of feature maps.

Operation: Given an input volume of size $W_{\text{in}} \times H_{\text{in}} \times D_{\text{in}}$, a transposed convolution with a filter of size $F \times F$ and stride S (padding can also be applied) produces an output volume of size $W_{\text{out}} \times H_{\text{out}} \times D_{\text{out}}$. The output dimensions are calculated as:

$$W_{\text{out}} = S \times (W_{\text{in}} - 1) + F$$

$$H_{\text{out}} = S \times (H_{\text{in}} - 1) + F$$

$$D_{\text{out}} = D_{\text{in}}$$

Transposed convolutions are useful for upsampling in neural network architectures, allowing the model to learn to generate higher resolution outputs from low-resolution input representations.

Example: In the generator of a GAN, transposed convolutions are often used to upsample a latent vector to generate realistic images. They help increase the spatial dimensions of the feature maps, capturing finer details.

Note: Stride S and padding can be adjusted to control the output dimensions and prevent artifacts such as checkerboard patterns.

5.6 Loss Function and Activation Function

5.6.1 Cross-Entropy Loss

Cross-entropy loss, often used in classification tasks, measures the performance of a classification model whose output is a probability value between 0 and 1. It quantifies how well the predicted probability distribution aligns with the true distribution of the labels.

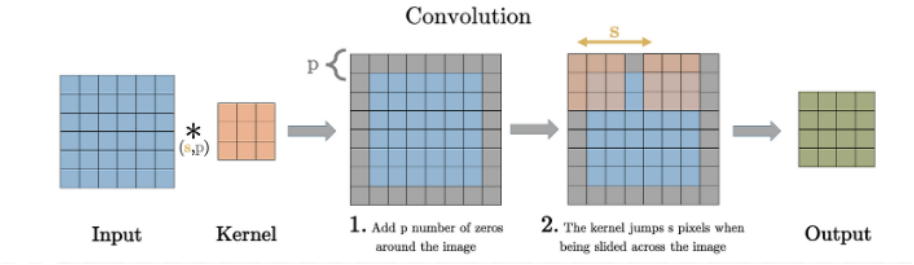


Figure 3: The Transposed convolutions

The binary cross-entropy loss for a single sample is defined as:

$$\text{Binary Cross-Entropy Loss} = -\left(y \cdot \log(p) + (1-y) \cdot \log(1-p)\right)$$

where y is the true label (0 or 1) and p is the predicted probability.
For multiple classes, the categorical cross-entropy loss is used:

$$\text{Categorical Cross-Entropy Loss} = -\sum_{i} y_i \cdot \log(p_i)$$

Here, y_i is the true probability distribution (one-hot encoded) and p_i is the predicted probability distribution.

5.6.2 Leaky ReLU

Leaky Rectified Linear Unit (Leaky ReLU) is an activation function that allows a small, non-zero gradient when the input is negative, preventing dead neurons. It is defined as:

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \cdot x & \text{if } x \leq 0 \end{cases}$$

where α is a small positive constant.
Leaky ReLU introduces a slight slope to the negative part of the function, enabling the flow of information even for negative inputs, which helps address the vanishing gradient problem.

5.7 Adam Optimizer

The Adam optimizer is an adaptive learning rate optimization algorithm widely used in machine learning and deep learning. It combines the benefits of two other popular optimization algorithms, namely RMSprop (Root Mean Square Propagation) and momentum.

5.7.1 Algorithm Overview

The Adam algorithm maintains two moving averages for each parameter: the first moment estimate (mean) and the second moment estimate (uncentered variance). It then computes an exponentially weighted moving average of past gradients (momentum) and past squared gradients (RMSprop). These estimates are used to adaptively adjust the learning rates for each parameter during training.
The key update rule for the parameters θ in Adam is given by:

$$\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \quad \text{(momentum)} \\
v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \quad \text{(RMSprop)} \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \quad \text{(bias-corrected momentum)} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \quad \text{(bias-corrected RMSprop)} \\
\theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \quad \text{(parameter update)}
\end{aligned}$$

Here:

- β_1 and β_2 are the exponential decay rates for the first and second moments, typically set to 0.9 and 0.999, respectively.

- η is the learning rate.
- g_t is the gradient of the loss with respect to the parameters at time step t.
- ϵ is a small constant to prevent division by zero, often set to 10^{-8}.

5.7.2 Benefits in the Project

The adaptive learning rate features of Adam make it well-suited for optimizing complex models like those in GANs and CNNs. In our project, we utilized the Adam optimizer for both the generator and discriminator networks in the GAN, as well as the CNN classifier during training. The algorithm's ability to dynamically adjust learning rates for each parameter contributes to faster convergence and improved training stability.

5.8 Convolutional Neural Network (CNN)

5.8.1 Architecture Description

The Convolutional Neural Network (CNN) consists of multiple layers, each contributing to the feature extraction and classification process:

- **Layer 1 (Convolutional + Max Pooling):**
  - Input: 3-channel RGB image.
  - Convolution: 16 filters of size 3×3, output volume $62 \times 62 \times 16$.
  - Max Pooling: Kernel size 2×2, output volume $31 \times 31 \times 16$.
- **Layer 2 (Convolutional + Dropout + Max Pooling):**
  - Convolution: 32 filters of size 2×2, output volume $30 \times 30 \times 32$.
  - Dropout: Probability 0.5.
  - Max Pooling: Kernel size 2×2, output volume $15 \times 15 \times 32$.
- **Layer 3 (Convolutional + ReLU):**
  - Convolution: 8 filters of size 2×2, output volume $14 \times 14 \times 8$.
  - Activation: ReLU (Rectified Linear Unit).
- **Fully Connected Layers:**
  - Flatten output into a 1D vector of length 1352.
  - Fully Connected Layer 1: 800 neurons, ReLU activation, dropout (0.15).
  - Fully Connected Layer 2: 200 neurons, dropout (0.5).
- **Output Layer:**
  - Fully Connected Layer 3: 2 neurons (Binary classification for "Cow" or "Horse").
  - SoftMax activation function applied for probability distribution.

5.8.2 Output Dimensions Calculation

The formula for calculating the output dimensions after a convolutional layer with input volume $W_{\text{in}} \times H_{\text{in}} \times D_{\text{in}}$ and a filter size $F \times F$ (assuming stride=1 and no padding) is given by:

$$W_{\text{out}} = H_{\text{out}} = (W_{\text{in}} - F + 1)$$

The output volume depth remains the same ($D_{\text{out}} = D_{\text{in}}$).

5.9 Training the Model

To effectively train a Convolutional Neural Network (CNN), the choice of the loss function, optimizer, and hyperparameters plays a crucial role. In this project, we opted for Stochastic Gradient Descent (SGD) as the optimizer—a widely used optimization technique in machine learning.

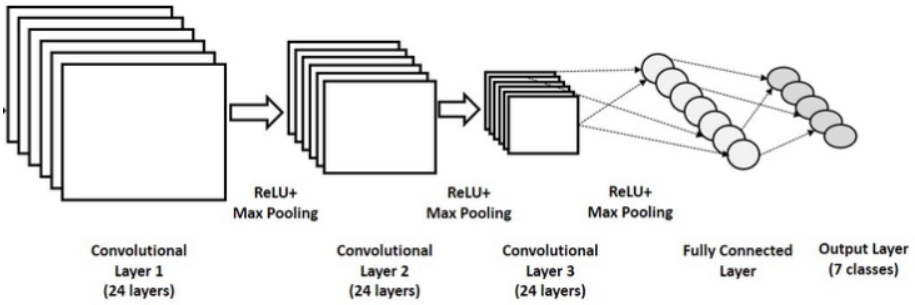


Figure 4: The CNN Architecture

5.9.1 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is an iterative optimization algorithm used to minimize the loss function during training. It operates by updating the model parameters based on the gradients of the loss with respect to those parameters. Unlike traditional gradient descent, SGD processes small batches of data (mini-batches) at a time, providing computational efficiency and allowing the model to converge faster.

The update rule for a parameter θ using SGD is given by:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_\theta \text{Loss}(\theta)$$

Here, η is the learning rate, and $\nabla_\theta \text{Loss}(\theta)$ represents the gradient of the loss with respect to the parameter θ.

5.9.2 Loss Function and Hyperparameters

The choice of the loss function depends on the nature of the task. For classification tasks, the Cross-Entropy Loss is commonly used. Hyperparameters such as the learning rate, batch size, and number of epochs are crucial for successful training. Adjusting these parameters helps strike a balance between convergence speed and avoiding overshooting.

In our implementation, we used the following settings for training:

- Optimizer: Stochastic Gradient Descent (SGD)
- Learning Rate (η): 0.01
- Batch Size: 32
- Number of Epochs: 20

These values were chosen through experimentation to achieve a good balance between training speed and model accuracy.

Training involves iterating over the dataset multiple times, adjusting the model parameters in each iteration to minimize the defined loss function. The process aims to enable the model to learn meaningful representations from the input data, facilitating accurate predictions.

Algorithm 2 Training CNN with Stochastic Gradient Descent (SGD).

6 Implementation

In this section, we provide a detailed account of how the Generative Adversarial Network (GAN) and Convolutional Neural Network (CNN) for data augmentation were implemented. We include code snippets, relevant algorithms, and discuss challenges faced during the implementation.

6.1 Generative Adversarial Network (GAN) Implementation

The GAN was implemented using the PyTorch framework. Below are key components of the GAN implementation:

- **Generator Architecture:** The generator was designed with a specific architecture, including transposed convolutional layers to generate realistic images.

Note: The above algorithm describes the training process of a CNN using Stochastic Gradient Descent. It iterates over mini-batches of the training dataset, performs forward and backward passes, computes the loss, and updates the model parameters using the gradient and learning rate. This process is repeated for a specified number of iterations.

Input: Initial CNN model parameters θ_{old}, learning rate η, number of iterations T, training dataset $\mathcal{D}$
Output: Optimized CNN model parameters θ_{new}
for $t = 1$ **to** T **do**
    Sample a mini-batch of data
    Sample a mini-batch $\mathcal{B}$ from training dataset $\mathcal{D}$
    Forward pass: Calculate the CNN model output
    Perform forward pass to get predicted outputs
    Compute the loss
    Calculate the loss using predicted outputs and ground truth labels
    Backward pass: Calculate the gradient
    Compute the gradient of the loss w.r.t. the model parameters
    Update parameters using SGD
    Update the model parameters using the gradient and learning rate: $\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_\theta \text{Loss}(\theta_{\text{old}}, \mathcal{B})$
    Update the model for the next iteration
    $\theta_{\text{old}} \leftarrow \theta_{\text{new}}$
end for
Return Optimized CNN model parameters θ_{new}

- **Discriminator Architecture:** The discriminator, responsible for distinguishing between real and generated images, had its architecture defined.
- **Training Loop:** The training loop involved alternating between training the discriminator and the generator to improve their performance.
- **Loss Function:** Binary Cross Entropy (BCE) loss was used as the loss function to guide the learning process.

Algorithm 2 outlines the training process of the GAN.

Note: This algorithm represents the training process of a GAN, where the generator and discriminator are updated alternately.

Generator (G), Discriminator (D), Real Images ($\mathcal{R}$), Noise Vector ($\mathcal{N}$), Learning Rate (η), Number of Epochs (E) Trained Generator and Discriminator
Initialize G and D with random weights

for *epoch* $= 1$ E **do** Train Discriminator Sample mini-batch of real images $\mathcal{R}$ Generate fake images using G and $\mathcal{N}$ Update D using BCE loss for real and fake images
Train Generator Generate fake images using G and $\mathcal{N}$ Update G using BCE loss with target labels as real
Return Trained Generator G and Discriminator D

6.2 Convolutional Neural Network (CNN) for Data Augmentation

The CNN for data augmentation was implemented using a standard architecture. Key implementation details include:

- **CNN Architecture:** The CNN consisted of convolutional layers, max-pooling layers, fully connected layers, and dropout layers to prevent overfitting.
- **Training Process:** Stochastic Gradient Descent (SGD) was chosen as the optimizer, and the cross-entropy loss function was used for training.
- **Hyperparameters:** Hyperparameters such as learning rate, dropout rates, and batch size were carefully tuned for optimal performance.

Algorithm 3 outlines the training process of the CNN.

Note: This algorithm represents the training process of a CNN for data augmentation using Stochastic Gradient Descent.

CNN Model (M), Training Dataset ($\mathcal{D}$), Learning Rate (η), Number of Epochs (E) Trained CNN Model
Initialize CNN model M with random weights

for *epoch* $= 1$ E **do** Sample mini-batch from training dataset Sample mini-batch $\mathcal{B}$ from $\mathcal{D}$
Forward pass Calculate predicted outputs using M
Compute loss Compute cross-entropy loss using predicted outputs and ground truth labels
Backward pass Compute gradient and update model parameters using SGD
Return Trained CNN Model M

6.3 Challenges and Solutions

The implementation of the Generative Adversarial Network (GAN) and Convolutional Neural Network (CNN) for data augmentation posed several challenges, which required careful consideration and problem-solving. Here are some of the key challenges and the corresponding solutions:

6.3.1 GPU Compatibility

Challenge: Selecting the appropriate GPU for model training posed a challenge due to compatibility issues with certain deep learning frameworks.

Solution: After thorough research and testing, we identified and configured a GPU compatible with the selected deep learning frameworks. This involved ensuring proper driver installations and resolving any software conflicts.

6.3.2 Insufficient Training Data

Challenge: In some instances, the generated images lacked diversity, potentially due to the limited size of the training dataset.

Solution: To address the data scarcity issue, efforts were made to augment the training dataset with additional relevant images. Additionally, data augmentation techniques, such as rotation and flipping, were applied to artificially increase the diversity of the training set.

6.3.3 Model Convergence

Challenge: Achieving stable convergence of the GAN model proved challenging, leading to mode collapse or erratic training behavior.

Solution: Several strategies were employed to enhance model convergence, including adjusting hyperparameters, optimizing the learning rate, and experimenting with different architectures. Regular monitoring and analysis of generated images helped identify convergence issues, enabling timely adjustments.

6.3.4 Image Quality Assessment

Challenge: Evaluating the quality of generated images required a robust metric that could capture visual fidelity effectively.

Solution: We implemented image quality assessment metrics, such as Structural Similarity Index (SSI) and Inception Score, to quantitatively evaluate the realism and diversity of generated images. These metrics provided valuable insights into the performance of the GAN.

These challenges and solutions demonstrate the iterative and problem-solving nature of the implementation process, contributing to the refinement of both the GAN and CNN models.

7 Hyperparameter Tuning Discussion

1. **Image Size (`image_size`):**
    - **Effect:** Larger image sizes can lead to higher-quality generated images but may require more computational resources. Smaller image sizes may reduce the computational burden but may result in lower image quality.

- **Tuning:** Experiment with different image sizes (e.g., 64x64, 128x128) and assess the trade-off between image quality and computational cost.

2. **Number of Channels (`nc`):**
   - **Effect:** The number of color channels in the training images (3 for RGB) influences the complexity and richness of the generated images.
   - **Tuning:** Typically set to 3 for RGB images. Changes might be explored for grayscale images or images with more color channels.
3. **Size of Latent Vector (`nz`):**
   - **Effect:** The latent vector represents the input to the generator. A larger latent vector can capture more complex features but might require more resources.
   - **Tuning:** Experiment with different sizes (e.g., 50, 100, 200) and evaluate the impact on image diversity and quality.
4. **Size of Feature Maps (`ngf` and `ndf`):**
   - **Effect:** These parameters influence the depth and complexity of the generator and discriminator networks.
   - **Tuning:** Adjust `ngf` and `ndf` based on the desired complexity. Higher values may lead to more powerful networks but require more resources.
5. **Number of Training Epochs (`num_epochs`):**
   - **Effect:** The number of times the entire dataset is processed. Too few epochs may result in underfitting, while too many may lead to overfitting.
   - **Tuning:** Experiment with different values and monitor the generator and discriminator losses. Stop training when the model converges.
6. **Number of GPUs (`ngpu`):**
   - **Effect:** Parallelizing training across multiple GPUs can significantly speed up training.
   - **Tuning:** Set `ngpu` based on the available hardware. If using multiple GPUs, ensure that the code supports parallel processing.

Table 1: Hyperparameter Tuning

Hyperparameter	Initial Value	Tuned Value 1	Tuned Value 2
Image Size	64x64	128x128	256x256
Number of Channels	3	1 (grayscale)	4 (CMYK)
Size of Latent Vector	100	50	200
Size of Feature Maps (ngf, ndf)	64	128	256
Number of Training Epochs	5	10	20
Number of GPUs (ngpu)	1	2	4

8 CUDA Architecture and its Importance

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use NVIDIA GPUs (Graphics Processing Units) for general-purpose processing, enabling significant acceleration of various computational tasks.

8.1 CUDA Program Structure

A CUDA program consists of two main components: the CPU part (host subprogram) and the GPU part (device subprogram). The host subprogram handles the preparation for GPU execution, including data movement between CPU main memory and GPU memory. It also sets up execution parameters and launches the device subprogram. On the other hand, the device code is organized into functions or kernels, each executed in parallel by GPU threads.

Table 2: Effects of Hyperparameter Tuning

Hyperparameter	Initial Impact	Tuned Value 1 Impact	Tuned Value 2 Impact
Image Size	Improved image quality; increased computational cost	Higher image quality but longer training time	Further improved quality with significantly longer training
Number of Channels	Standard RGB color; rich features	Grayscale images with reduced complexity	More color channels for richer representation
Size of Latent Vector	Moderate diversity in generated images	Reduced diversity with simpler features	Increased diversity with complex features
Size of Feature Maps	Moderate model complexity	Higher complexity with more detailed features	Very high complexity with fine-grained features
Number of Training Epochs	Short training, risk of underfitting	Balanced training with less risk of underfitting/overfitting	Extended training, risk of overfitting
Number of GPUs	Single GPU usage	Faster training with two GPUs	Further acceleration with four GPUs

8.2 Parallel Execution and GPU Architecture

In CUDA, a kernel execution is decomposed into blocks, which logically run in parallel and may run physically in parallel on the GPU if resources are available. Each block consists of a group of threads mapped to a single multiprocessor. Threads within a block can share up to 16 KB of memory and synchronize through barrier primitives. However, communication among threads of different blocks is achieved through global memory, and synchronization occurs by ending a kernel.

Threads within a block are grouped into warps, collections of threads that can run concurrently on a multiprocessor. While the developer can decide the number of threads, threads beyond the warp size are time-shared on available hardware resources.

8.3 Memory Hierarchy and Optimization

CUDA threads can access the entire GPU global memory, but there is a performance boost when accessing data stored in shared memory, which is explicitly managed. To efficiently use GPU computational resources, large data structures are stored in global memory, while shared memory is prioritized for strategic, frequently used data structures.

These CUDA architecture characteristics have a significant impact on accelerating computational tasks, such as cardiac electromechanical simulations, by efficiently leveraging parallel processing capabilities of GPUs.

8.4 Importance in Project

In our project, CUDA played a crucial role in accelerating the simulation process. By offloading intensive computational tasks to the GPU, we achieved significant speedups in generating and processing images using Generative Adversarial Networks (GANs) and enhancing the efficiency of training Convolutional Neural Networks (CNNs) for image classification. The parallel processing capabilities of CUDA allowed us to harness the computational power of GPUs effectively, ultimately improving the overall performance of our simulations.

9 Results

9.1 Discussion

In this section, we interpret the results obtained from training the Generative Adversarial Network (GAN) and the Convolutional Neural Network (CNN) classifier for image generation and classification. The discussion covers the discriminator and generator losses, the animation of generated images, and the final visualization of the generated images.

9.1.1 Discriminator and Generator Losses

The first plot illustrates the evolution of the discriminator and generator losses during the training process. As expected in a GAN framework, the discriminator loss initially decreases, indicating that the discriminator becomes adept at distinguishing between real and generated images. Simultaneously, the generator loss shows a decreasing trend as it learns to produce images that are more challenging for the discriminator. The convergence or stabilization of these losses signifies the equilibrium between the two adversarial networks.

9.1.2 Animation of Generated Images

The animated sequence provides a dynamic view of how the generator improves over training iterations. Initially, the generated images may appear random or nonsensical, but as training progresses, the generator refines its ability to produce more realistic images. This animation is a visual representation of the iterative adversarial process and showcases the evolution of the generated samples.

9.1.3 Visualization of Generated Images

The final visualization presents a grid of generated images, offering a comprehensive view of the GAN's capabilities. The diversity and quality of the generated images reflect the effectiveness of the trained GAN. These images can be compared with the training dataset to assess the network's ability to capture the underlying patterns and structures present in the real data. And we can **clearly see some weird output images generated randomly by the GAN**.

9.1.4 Limitations and Constraints

It is essential to acknowledge any limitations and constraints in our approach. These may include issues like mode collapse in GANs, potential overfitting in the classifier, or challenges related to the choice of hyperparameters. Identifying and understanding these limitations is crucial for contextualizing the results and guiding future improvements.

9.1.5 Suggestions for Improvement

Based on the observed results and limitations, we propose potential avenues for improvement. This could involve experimenting with different GAN architectures, refining hyperparameters, or exploring advanced techniques such as progressive growing of GANs. Additionally, enhancing the CNN classifier's architecture and increasing the diversity and size of the training dataset may lead to improved classification performance.

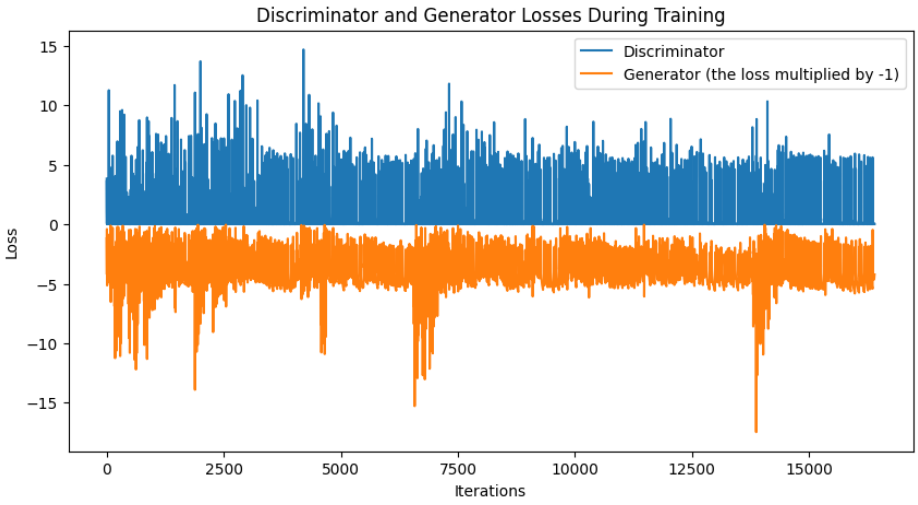
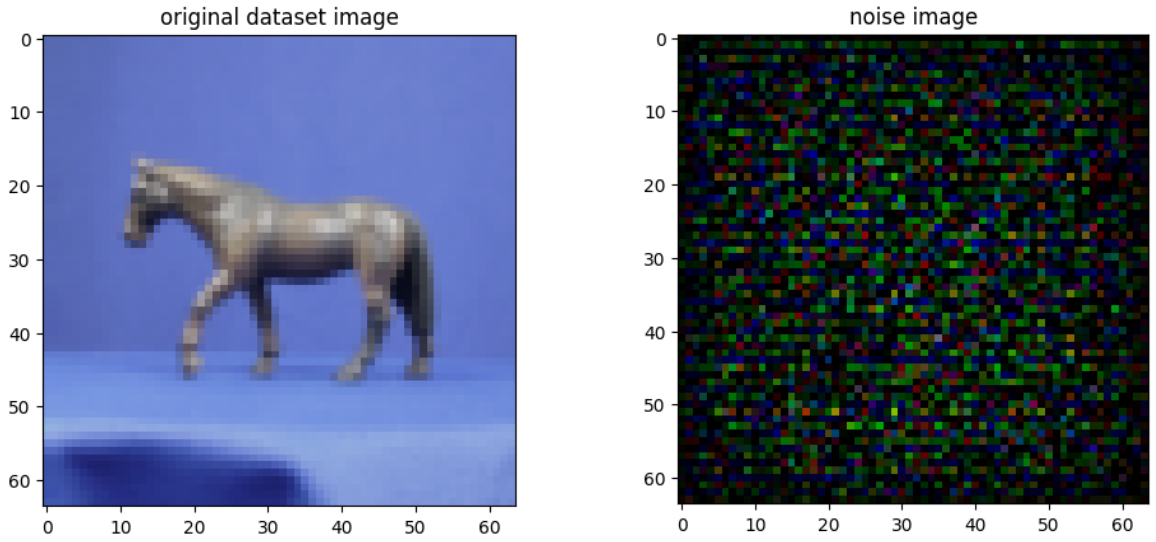


Figure 5: Discriminator and Generator Losses During Training



(a) Original Image (b) Noise Image

Figure 6: *Displaying an image and its noise through the Generator*

Figure 7: the 1st generated images after training using T4GPU in Google Colab

9.2 Data Augmentation Results

9.2.1 Process Overview

I generated 1000 new images of horses using a GAN. These images were added to the original training dataset to give the CNN classifier more examples to learn from.

9.2.2 Expected Benefits

Data augmentation aims to improve the model's performance by exposing it to a broader range of examples. The hope was that the CNN classifier would become better at recognizing and classifying images of cows and horses.

9.2.3 What Happened

Despite our efforts, the CNN classifier did not show a significant improvement in accuracy. This could be due to a few reasons:

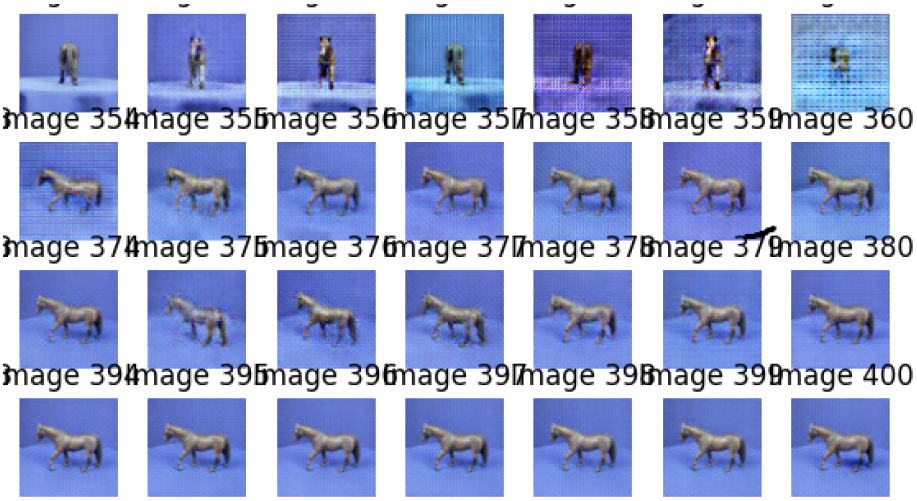


Figure 8: the last generated images after training using T4GPU in Google Colab

1. **Generated Image Quality:** If the GAN didn't create realistic images, the added data might not be helpful.
2. **Relevance of New Data:** The new images need to bring something new to the table. If they don't, the model might not gain much.
3. **Complexity of the Model:** The CNN model might already be good enough, and more data may not make a big difference.

10 Conclusion

In conclusion, this project aimed to explore the application of Generative Adversarial Networks (GANs) for data augmentation in the context of training a Convolutional Neural Network (CNN) for image classification. Here are the key takeaways:

10.1 Key Findings

1. **GAN for Data Augmentation:** The GAN was successful in generating new images, expanding the training dataset for the CNN.
2. **CNN Training:** The CNN was trained on the augmented dataset, but the impact on classification accuracy was limited.
3. **Challenges Encountered:** Various challenges, including image quality and the relevance of new data, influenced the effectiveness of the approach.

10.2 Contributions

This project contributes to the understanding of GANs and CNNs in the context of image classification. While the expected improvements in accuracy were not fully realized, the exploration of these techniques provides valuable insights.

10.3 Future Directions

Future work could focus on refining the GAN architecture, exploring different augmentation strategies, or investigating more advanced CNN architectures. Additionally, fine-tuning hyperparameters and increasing the diversity of generated images may lead to better results.

Overall, this project serves as a foundation for further research in the domain of GAN-based data augmentation and its implications for improving the performance of image classifiers.

References

1. Goodfellow, I., et al. (2014). *Generative Adversarial Nets.* Advances in Neural Information Processing Systems, 27.

2. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION Karen Simonyan Andrew Zisserman + Visual Geometry Group, Department of Engineering Science, University of Oxford karen,az@robots.ox.ac.uk

3. Deep Learning to Classify Radiology Free-Text Reports Matthew C Chen 1, Robyn L Ball 1, Lingyao Yang 1, Nathaniel Moradzadeh 1, Brian E Chapman 1, David B Larson 1, Curtis P Langlotz 1, Timothy J Amrhein 1, Matthew P Lungren 1

4. Generative Adversarial Nets Ian J. Goodfellow, Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair† , Aaron Courville, Yoshua Bengio‡ Departement d'informatique et de recherche op ´ erationnelle ´ Universite de Montr ´ eal ´ Montreal, QC H3C 3J7

5. $https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html$