ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Machine Learning for Computer Vision

TD4: Data augmentation and Generative Models

Département d'informatique

Ruiwen HE

Course Review

1. Introduction to Data Augmentation

Importance of Data Augmentation

- Solves the **limited data problem**: Many datasets are small or imbalanced, e.g., medical imaging.
- Prevents **overfitting**: Adds diversity, enabling models to generalize better.
- Mimics real-world variations, improving model robustness.

Challenges in Image Datasets

- **Class imbalance**: Uneven distribution of classes (e.g., 80 % dogs, 20 % cats).
- **Lack of variability**: Limited perspectives, lighting, or scale in images.
- **Small datasets**: Insufficient data for deep neural network training.

2. Basic Augmentation Techniques

Geometric Transformations

- Spatial manipulations: Rotation, scaling, flipping, cropping.
- Purpose: Simulates object appearance variations to enhance model generalization.

Color Transformations

- Adjustments to brightness, contrast, saturation, and hue.

Noise Injection

- Adds noise (Gaussian, salt-and-pepper, etc.) to improve robustness.

Limitations of Basic Techniques

- Information loss (e.g., cropping), distortion, or unnatural artifacts.
- Performance and applicability depend on the task.

3. Deep Learning-Based Generative Models

Generative Adversarial Networks (GANs)

- Proposed by Ian Goodfellow (2014), inspired by game theory.
- Comprised of a **generator** (to synthesize data) and a **discriminator** (to classify data as real or fake).
- Applications: Image synthesis, resolution enhancement, text-to-image translation.

Diffusion Models

- Generate data by reversing a noise-adding process.
- Advantages: Stable training, high diversity, and realistic outputs.
- Examples: DDPM, Stable Diffusion, DALL·E 2.

4. Advanced GAN Architectures

DCGAN (Deep Convolutional GAN)

- Replaces fully connected layers with convolutional layers for generative tasks.

StyleGAN Series

- **StyleGAN/StyleGAN2**: Improves control over visual features, reduces artifacts.
- **StyleGAN3**: Achieves rotation and translation invariance.

5. Challenges and Metrics

Challenges in GANs

- **Mode collapse**: Generator fails to produce diverse samples.
- **Non-convergence**: Training may fail to reach a stable equilibrium.

Challenges in Diffusion Models

- High computational cost for training and inference.
- Slow sampling times due to iterative denoising steps.
- Requires expertise in handling noise distributions and latent spaces.

Evaluation Metric for Generative Models

- **Fréchet Inception Distance (FID)**: Measures the similarity between real and generated images, based on deep feature distributions.

Objective

In this practical session, we will explore key concepts and implementations of generative models, focusing on **data augmentation**, **Generative Adversarial Networks (GANs)**, and **diffusion models**. By completing the exercises, you will:

- Understand and implement common data augmentation techniques.
- Develop and train a basic GAN for image generation.
- Implement a simplified diffusion model and compare it with GANs.

Part I: Data Augmentation

Data augmentation is an essential technique to enhance dataset diversity and improve the generalization of deep learning models. This section includes practical exercises focusing on basic and advanced augmentation techniques, their implementation, and performance evaluation.

Exercise 1: Implement Basic Data Augmentation

Objective: Apply fundamental data augmentation techniques to CIFAR-10 images and visualize the transformations. **Steps:**

1. Implement the following transformations:
    - Random horizontal flip
    - Random crop with padding
    - Brightness, contrast, and saturation adjustment
    - Gaussian noise addition
2. Use the augmented dataset to visualize the results.

Deliverables:

- Augmented images displayed alongside the original images.
- Code implementation.

Exercise 2: Evaluate the Impact of Data Augmentation on Model Performance

Objective: Train a CNN with and without data augmentation and compare the results. **Steps:**

1. Define a simple CNN architecture.
2. Train the CNN twice:
    - Using the original CIFAR-10 dataset.
    - Using the augmented CIFAR-10 dataset.
3. Evaluate test accuracy and compare.

Deliverables:

- Accuracy comparison between augmented and non-augmented datasets.
- Training and evaluation code.

Part II: Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a powerful class of generative models that learn to create data samples by training two competing neural networks: a generator and a discriminator. This section includes practical exercises to implement, evaluate, and enhance GAN performance.

Exercise 3: Implement a Basic GAN

Objective: Train a GAN to generate MNIST handwritten digits. **Steps:**

1. Implement the generator and discriminator:
    - **Generator input:** 100-dimensional noise vector.
    - **Generator output:** 28x28 grayscale images.
    - **Discriminator input:** 28x28 images (real or fake).
2. Train the GAN for at least 10 epochs.
3. Visualize generated samples at different training stages.

Deliverables:

- Code for GAN implementation and training.
- Generated images at different epochs.

Exercise 4: Control Generator Input

Objective: Experiment with the generator's input noise to observe its effect on the generated images. **Steps:**

1. Train the GAN with the generator's noise drawn from:

- Uniform distribution.
- Gaussian distribution.

2. Compare the diversity and quality of generated images for each distribution.

Deliverables:

- Visual comparison of generated images.
- Observations on input noise effects.

Exercise 5: Implement a DCGAN to Generate Face Images

Objective: Implement a Deep Convolutional GAN (DCGAN) to generate realistic face images using a dataset like CelebA.
Steps:

1. **Prepare the dataset:**
   - Load the CelebA dataset.
   - Resize and normalize the images to a uniform size of 64×64.
2. **Build the DCGAN model:**
   - Define the generator and discriminator networks using convolutional layers.
3. **Train the DCGAN:**
   - Train the DCGAN for several epochs, alternating between the generator and discriminator updates.
   - Use a binary cross-entropy loss function for both networks.
4. **Generate and visualize images:**
   - Generate face images using the trained generator.
   - Visualize the generated samples at different training epochs.
5. **Evaluate the results:**
   - Analyze the quality of generated images through visual inspection.

Deliverables:

1. The implementation code for the DCGAN (including model definition and training).
2. Generated face images at various training stages.
3. Observations on the quality and diversity of generated face images.

Part 3: Diffusion Models

Diffusion models are a class of generative models that learn to create new data samples by reversing a noise-adding process. This section includes exercises to implement, compare, and optimize diffusion models for image generation tasks.

Exercise 8: Implementing and Training Denoising Diffusion Probabilistic Models (DDPM)

Objective: Simulate forward noise addition and reverse denoising to generate CIFAR-10 images. **Steps:**

1. Implement the forward diffusion process:
   - Add Gaussian noise to an image iteratively for a defined number of steps.
2. Implement the reverse diffusion process:
   - Train a neural network to remove noise step-by-step.
3. Visualize noisy images from the forward process and the reconstructed images from the reverse process.

Deliverables:

- Code for forward and reverse diffusion processes.
- Visualizations of noisy and reconstructed images.

1. **Import Necessary Libraries:** First, import the required libraries for building and training the model.
2. **Define the Forward Diffusion Process:** In the forward diffusion process, noise is added to the input data step by step.

   a) **Forward Diffusion Formula:**

   $$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

   b) **Beta Schedule:** A schedule controls the variance (β_t) of the noise added at each timestep t. A linear schedule is commonly used:

   $$\beta_t = \text{linear\_interpolation}(\text{start} = 10^{-4}, \text{end} = 2 \times 10^{-2}, \text{steps} = T)$$

   c) **Sampling at Time t:** Using the reparameterization formula:

   $$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

3. **Define the UNet Model:** The UNet model is used as the denoising network to predict the noise at each timestep.

   a) **UNet Architecture:** The network contains convolutional layers to extract spatial features and progressively downsample the input image. It then reconstructs the image through upsampling layers. The architecture can be summarized as:

   $$\text{UNet}(x) = \text{Conv2D}(\text{ReLU}(\text{Conv2D}(x)))$$

4. **Train the DDPM:** The goal of training is to predict the noise (ϵ) added to the image at each timestep.

   a) **Loss Function:**

   $$\mathcal{L}_{\text{simple}} = \|\epsilon - \epsilon_\theta(x_t, t)\|_2^2$$

   b) **Training Loop:** For each batch:
   - Sample a random timestep t uniformly from $\{1, \ldots, T\}$.
   - Add noise to the input data x_0 using the forward process to obtain x_t.
   - Use the model $\epsilon_\theta(x_t, t)$ to predict the noise.
   - Compute the mean squared error between the predicted and true noise.

5. **Define the Reverse Diffusion Process:** During sampling, we iteratively denoise the image starting from pure noise.

   a) **Reverse Diffusion Formula:**

   $$p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$$

   The parameters for the Gaussian distribution are computed as:

   $$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

   $$\sigma_t^2 = \beta_t$$

   b) **Sampling Process:**
   - Start with pure Gaussian noise x_T.
   - For $t = T, T-1, \ldots, 1$:
     - Predict the noise using the model $\epsilon_\theta(x_t, t)$.
     - Sample x_{t-1} from $p(x_{t-1}|x_t)$.

6. **Complete Workflow:**

   a) **Forward Diffusion:** Use the forward process to gradually add noise to the input data. Ensure that the added noise follows a Beta schedule.

   b) **Training:** Train the UNet model to minimize the MSE loss by predicting the noise.

   c) **Reverse Diffusion:** Start from random Gaussian noise and iteratively apply the reverse process to generate images.