**Deep Learning Home Work Assignment**
**Deep Diffusion Implicit Models**

# Abstract

For our course work, we decided to dive deep into the world of machine learning based image generation. In recent times, a new family of image generation techniques emerged, called diffusion networks. These diffusion networks proved to be on par of established state-of-the-art solutions like Generative Adversarial Network, even surpassing them under some circumstances. To explore the potential of the diffusion networks, we implemented a simplified version of the technique and conducted experiments to evaluate its performance on generating high-fidelity images. Unlike many commercially available solutions, ours is only a simplified model, trained on one of two datasets and it does not take any text prompts as input. As such our solution can be considered an unconditioned model. We opted for the Deep Diffusion Implicit Model as it has superior inference times. In this documentation we cover the design decisions of our solution, its key characteristics and performance.

# Kivonat

A házifeladatunkhoz úgy döntöttünk, hogy mélyebben elmerülünk a gépi tanuláson alapuló képgenerálás világában. Az utóbbi időben megjelent a képgenerálási technikák egy új családja, az úgynevezett diffúziós hálózatok. Ezek a diffúziós hálózatok hasonló eredményekket tudtak elérni, mint a már olyan bevált, korszerű megoldások, mint a Generative Adversarial Network-ök, sőt bizonyos körülmények között felülmúlták is azokat. A diffúziós hálózatokban rejlő lehetőségek feltárása érdekében a módszer egy egyszerűsített változatát implementáltuk, majd aztán kísérleteket végeztünk hogy megállapítsuk mennyire tud hatékonyan magas minőségű képeket generálni. Sok kereskedelmi forgalomban megtalálható megoldással ellentétben a miénk csak egy egyszerűsített modell, amelyet két adathalmaz egyikén képeztünk ki, és nem fogad el bemenetként szöveges utasításokat. Ezáltal, a mi megoldásunk “feltétel nélküli” modellnek tekinthető. A Deep Diffusion Implicit Model mellett döntöttünk, mivel az “inference” ideje jobb, mint a probabilisztikus rokonáé. Ebben a dokumentációban a megoldásunk tervezési döntéseivel, főbb jellemzőivel és teljesítményével foglalkozunk.

# Introduction of Diffusion Models

In the most basic sense, Diffusion Models are nothing more than a denoising machine learning solution. They employ UNET-s, which are essentially autoencoders with connections between convolution layers of the same size. During training, these models learn to remove noise from an image. As learning how to remove all the noise at once would be extremely difficult, the authors of the DDPM paper decided to use sinusoidal embedding to indicate the current time, or level of the noise that needs removing. This way, the model can learn more iteratively, learning how to remove a given amount of noise.

During training, a timestep is basically chosen and the corresponding amount of noise is applied. Then the model tries to remove it.
The DDIM and DDPM are essentially the same besides their sampling techniques. For the DDIM, the model iteratively removes the noise applied during inference using a closed form solution.

# Model architecture and system design

As we stated earlier, we decided to implement a DDIM machine learning network. As such, we needed to implement some of the basic building blocks of U-NETs.
Our implementation contains the following:

- Components:
    - Down block
    - Up block
    - Residual block
    - Sinusoidal block
- Scheduler:
    - DDIM scheduler
- U-NET

We used PyTorch and PyTorch Lightning for our implementation. PyTorch lightning allowed us to speed up the model building process, as we did not need to implement some boiler plate code.

We tried to make our network as configurable as possible. The network depth, the corresponding block width, the embedding frequencies etc. are all configurable. The user can also easily set the learning rate, the number of epochs and other hyperparameters using the provided command line arguments.

Initially, we tried to do early stopping, using our validation loss and PSNR, but it proved to be pointless. While these metrics often did not improve, they were not indicative of the model's performance. Using KID for the evaluation dataset was also discussed, but it significantly slowed down our training and as such we left it out.

We opted to omit the Exponential Moving Average model often found in the literature. For us, it yielded inferior results, while significantly increasing our models complexity.

# Data acquisition and preparation

- **CelebA**: ~200k images with diverse facial features (e.g., glasses, hairstyles). Balanced RGB distribution, high feature variance, and overlapping clusters make it complex and diverse.
- **Flowers**: ~8k images focused on flowers. Dominated by red and green colors, with compact clusters and lower diversity. Easier to model but risks overfitting due to smaller size.

**Key Insights for Diffusion Models:**

- **CelebA**: Great for generating varied human faces due to its size and diversity, but challenging due to overlapping clusters.
- **Flowers**: Focused and compact, ideal for learning specific patterns, but requires augmentation to address size and color biases.

**Data preparation:**

Both datasets were treated the same way, to ensure that the results are comparable across the board. We applied a plethora of transformations to better facilitate the learning potential of our network:
Firstly, we resized all the images to the same resolution and then applied a random crop to them to ensure that the model sees a greater variety of the training data, and not just certain parts of the images. Furthermore, we also applied a random horizontal flip with the hopes of better generalization. Finally, we converted the images to tensors and normalized them, with a mean and standard deviation of 0.5

The datasets were divided into three separate subsets. 80% of the images were used for training, while the remaining 10-10% were respectively used for evaluation and testing.

Unfortunately, the Flowers102 dataset is greatly unbalanced, so we needed to redistribute the images to achieve the aforementioned 80-10-10 split of the dataset.

During our extensive data analysis, we established that there are a few characteristics of the datasets, which we expect to find in our results as well, such as the prominence of the color red in the Flowers102 dataset.

**Note**: The analysis is available in `celeba_data_analyse.ipynb` and `flowers_data_analyse.ipynb`.

# Training

The training process for the diffusion models focused on comparing the **reference model** using **DDPM** (Denoising Diffusion Probabilistic Models) and our custom model using **DDIM** (Denoising Diffusion Implicit Models). Both models shared the same hyperparameter configurations, except for the underlying sampling scheduler.

**Training Machines:**
We used three separate machines for training, two local machines and a Google Colab instance. The local machines were equipped with RTX 2060 and RTX 3090 GPUs. The models were also trained on A100 GPUs using Google Colab.

---

## Reference Model (DDPM):

- **Scheduler**: DDPM relies on a probabilistic approach with fixed variance, making it computationally slower during both training and sampling phases.
- **Architecture**: The model is based on the **UNet2DModel** architecture from the `diffusers` library.
- **Epochs**:
    - **CelebA**: Trained for **1 epoch**.
    - **Flowers**: Trained for **4 epochs**.
- **Challenges**: The training process for the reference model was very slow, limiting the number of epochs and impacting overall performance.
- **Objective**: Used as a baseline to compare against our custom model.

---

## Custom Model (DDIM):

- **Scheduler**: DDIM introduces deterministic sampling, significantly speeding up training and improving the quality of generated images.
- **Architecture**: A simple UNet architecture with up and down residual blocks and sinusoidal embedding. As opposed to the reference architecture, we did not use attention blocks. We used the sampling technique as outlined in the original DDIM paper and the Keras blogpost.
- **Epochs**:
    - **CelebA**: Trained for **101 epochs**.
    - **Flowers**: Trained for **171 epochs**.
- **Hyperparameter Optimization**: Conducted manual tuning and testing to achieve the best performance, focusing on key parameters such as learning rate, batch size, and noise scheduling. This approach ensured that the model was tailored for high-quality image generation.
- **Outcome**: The custom model outperformed the reference in both FID, KID, and IS metrics. As the model was significantly simpler than our reference model, we could achieve greater training speeds and as such, it benefitted from the longer training period made possible by its optimized architecture.

---

## Additional Insights

1. **Hyperparameter Tuning**:
    - Due to the nature of image data, hyperparameter optimization was performed manually, testing configurations iteratively for both models. This hands-on approach allowed us to fine-tune the models to handle complex visual features effectively.
2. **Efficiency**:
    - The custom model's use of DDIM significantly reduced the computational cost per epoch, enabling training for a larger number of epochs (101 and 171) compared to the reference model's 1 and 4 epochs. We wish to emphasize that while the number of epochs differ greatly between the two models, the training times were comparable.

3. **Potential Improvements for the Reference Model**:
    - The reference model would likely show much better performance if trained for more epochs. However, the slower training speed due to DDPM scheduling and the lack of architectural optimization limited its practical scalability.
4. **Potential Improvements for the Custom Model**:
    - The custom model did not use attention blocks. Attention blocks can be beneficial, especially at greater depths. Implementing these blocktypes could potentially improve its performance.
    - The architecture of the model was quite simple, and while it proved to be an advantage because of its superior training time, increasing the models complexity with more learnable parameters could lead to better results.
    - This model did not use EMA (Exponential Moving Average). While we found that our model performed worse with our basic initial EMA implementation, it is possible that in the long run, it could have proved to be useful.

# Evaluation

This evaluation assesses the quality of a trained diffusion model using **Frechet Inception Distance (FID)**, **Inception Score (IS)**, and **Kernel Inception Distance (KID)**. The evaluation compares generated images against real images from the Flowers102/CelebA dataset.

## Metrics and Their Purpose

### 1. Frechet Inception Distance (FID)

- **Purpose**: Measures similarity between generated and real images in feature space.
- **Details**: FID computes the Wasserstein-2 distance between feature distributions extracted from an Inception model.
- **Interpretation**: Lower FID indicates closer similarity to real images, capturing overall quality.

### 2. Inception Score (IS)

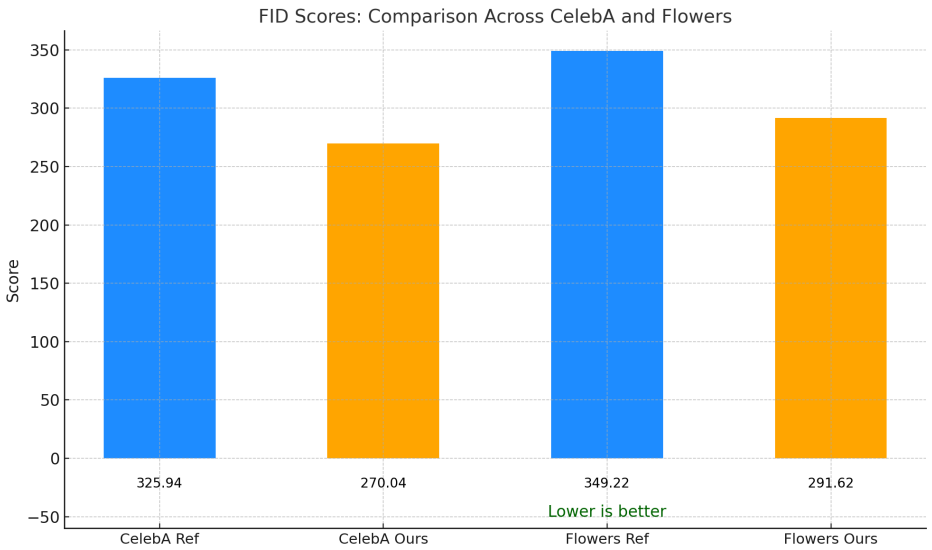
- **Purpose**: Assesses image quality and diversity.
- **Details**: IS evaluates the confidence and spread of class predictions from an Inception model:
    - High-quality images lead to confident predictions (low entropy).
    - Diverse images spread predictions across multiple classes.
- **Interpretation**: Higher IS scores indicate better quality and diversity.

### 3. Kernel Inception Distance (KID)

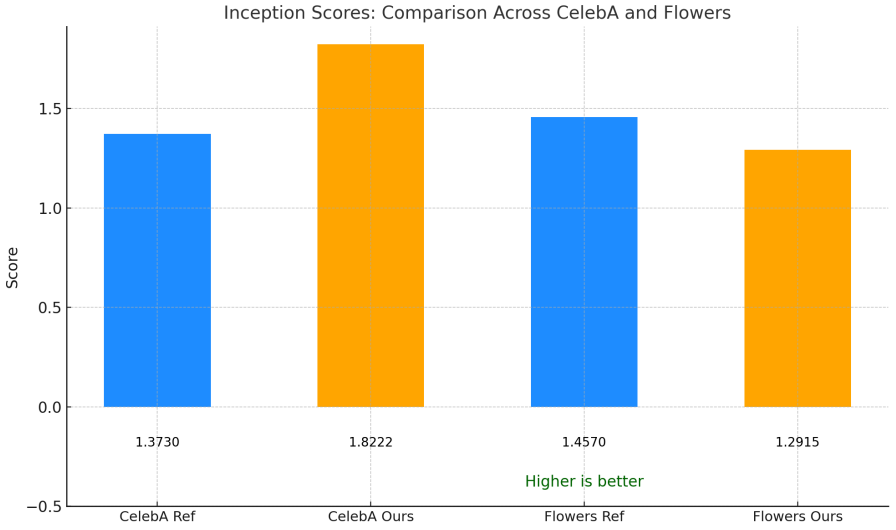
- **Purpose**: Measures similarity using kernel-based methods.
- **Details**: KID computes the squared Maximum Mean Discrepancy (MMD) between real and generated features with polynomial kernels.

- **Interpretation**: Lower KID values indicate closer alignment between real and generated image distributions. Unlike FID, KID is unbiased and better suited for smaller datasets.

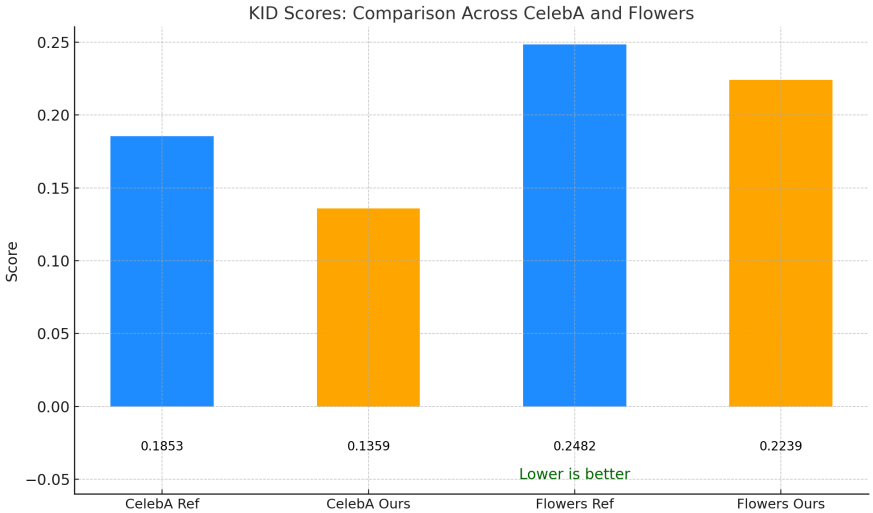
### FID score of models:




### IS score of modells:




### KID Score of models:




Despite training for far fewer epochs (4 vs. 171), the reference model performed slightly better in IS. However, our Flowers model excelled in both FID and KID, indicating higher overall quality and closer alignment with the real data.

With 101 epochs of training compared to just 1 epoch for the reference model, our model demonstrates superior performance across all metrics. The results highlight its ability to produce higher-quality, more diverse, and realistic images on the CelebA dataset. However, it is likely that the reference model would achieve significantly better performance if trained for more epochs, potentially narrowing the gap or even surpassing our model in certain areas.

# Future works

If possible, we wish to further optimize our model so it manages to create lifelike images. As of now, unfortunately the model seemingly stops learning after a handful of epochs. Increasing its complexity and implementing some additional state-of-the-art solutions would hopefully allow it to better generalize and create images that are comparable to other third-party solutions.
Furthermore, we presented a simplistic web interface for generating images on the two datasets. This is not a final working product but only a proof of concept. Improving on this to achieve a working ML as a service solution is another possibility for future works.

# Related works

[1] Ho, J., Jain, A., & Abbeel, P.: Denoising Diffusion Probabilistic Models, https://arxiv.org/abs/2006.11239 (2024. Dec. 09.) [FSZR]

[2] Song, J., Meng, C., & Ermon, S.: Denoising Diffusion Implicit Models, https://arxiv.org/abs/2010.02502 (2024. Dec. 09.) [FSZR]

[3] Keras Documentation: Denoising Diffusion Implicit Models (DDIM) Example, https://keras.io/examples/generative/ddim/ (2024. Dec. 09.) [MSI]

[4] GitHub: Denoising Diffusion Implicit Models by ermongroup, https://github.com/ermongroup/ddim (2024. Dec. 09.) [FSZR]

[5] scikit-learn Documentation: t-SNE: sklearn.manifold.TSNE, https://scikit-learn.org/1.5/modules/generated/sklearn.manifold.TSNE.html (2024. Dec. 09.) [MSI]

[6] DataCamp: Introduction to t-SNE, https://www.datacamp.com/tutorial/introduction-t-sne (2024. Dec. 09.) [MSI]

[7] Lightning AI: Train a Diffusion Model with PyTorch Lightning, https://lightning.ai/lightning-ai/studios/train-a-diffusion-model-with-pytorch-lightning (2024. Dec. 09.) [MSI]

Note: In this project, LLMs (e.g., GitHub Copilot, ChatGPT) were used as a search tool, for drafting comments, brainstorming ideas, translation and providing guidance on HTML and CSS, with all outputs carefully reviewed and validated.