Image generation with diffusion models

Important

Place the following into a file named ".env" to access trained models stored in Wandb:

WANDB_API_KEY=f6e9e0132f1f469b6d0592815318bd8b4f9e2105
WANDB_USERNAME=asztrikx-budapesti-m-szaki-s-gazdas-gtudom-nyi-egyetem
IN_LOCAL=0

Task

Implement and train unconditional diffusion models, such as DDPM (Denoising Diffusion Probabilistic Model) or DDIM (Denoising Diffusion Implicit Model) for generating realistic images. Evaluate the capabilities of the models on two different datasets, such as CelebA and Flowers102.

Introduction

Our chosen topic was image generation with DDPM diffusion model, where we chose the denoising diffusion method to generate the images from random (noisy) images. The aim was to generate images of 32x32 pixels, as we did not have a resource with a large computing capacity.

The implementation was based on the articles mentioned below and in the README file, and we used the available LLMs such as ChatGPT and Gemini, mainly for Wandb API usage as it has poor documentation. Our implementation can easily be inside Docker or on the host. We mainly use Numpy, Matplotlib, Lightning, Wandb Sweep and Gradio. Out implementation even supports toggling on or off the following features for faster runs:

- Evaluation only mode
- Handling only Flowers dataset
- Evaluating metrics
- Visualizing and analysing

Datasets

The two datasets we used were Flowers102 and CelebA, the latter of which we downloaded to have it available locally, as otherwise we could not guarantee that we would always be able to load it when we needed it.

For efficient data loading we used the LightningDataModule class with

- More than 1 number of workers
- Custom splitting the dataset for balanced train, test and validation sizes

- Cropping the images to get equal ration of width and height
- Resizing them to be 32 by 32 pi
- Normalization
- Reverse transformation methods

We also provided a wrapper DataModule class called NoTargetDataset to seamlessly drop classification target data, which we don't need for this task.

Data acquisition, analysing, cleansing and preparation

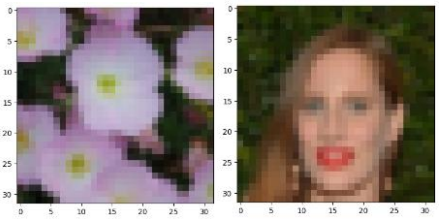
We started with the data processing. We analysed the size of the images, found that they came in several different sizes, so we standardised them using a centre crop based on analysing the minimum weight and height values (choosing the minimum of them).

```
Flowers102 min height 500
Flowers102 min width 500
```

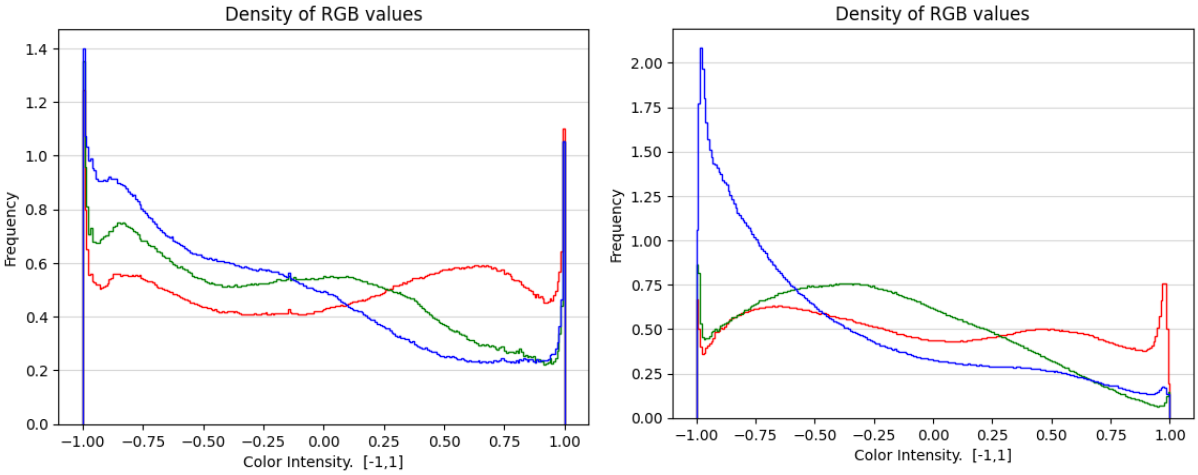
```
CelebA min height 218
CelebA min width 178
```

1. Figure: Size of images after data preparation

Applying the corresponding DataModule classes to the datasets we got the following images (after making them 32 by 32):



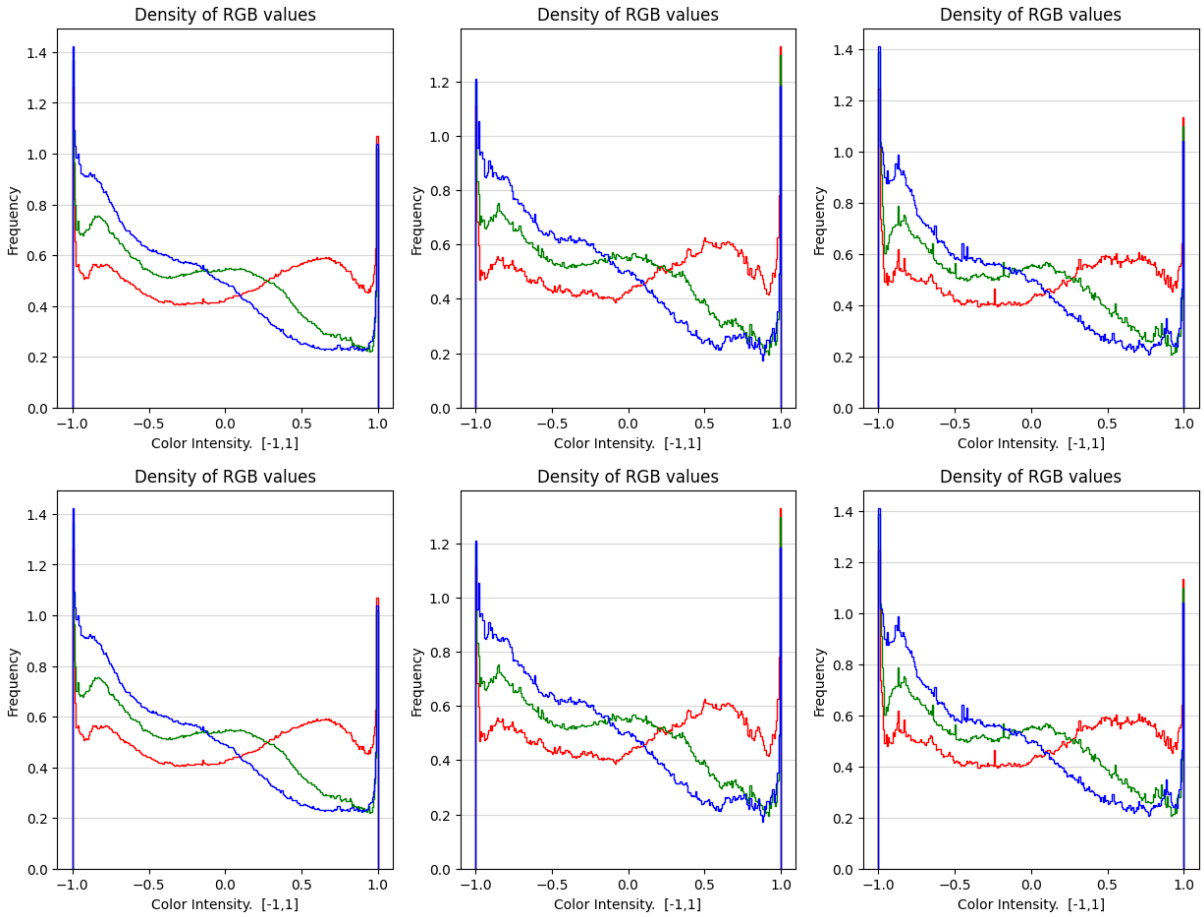
We then analysed the colour density of the images on the RGB scales. This was evaluated for each data set. The resulting plot can be seen in the following images.



2. Figure: Density of RGB values of an image from Flowers102 and from CelebA

Then we divided the datasets into train, test and validation parts, and we also evaluated the density of the colours on each of them, the corresponding plot is shown in

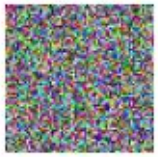
the next figure. The top row is for the Flowers102 dataset, and the bottom row is for the CelebA dataset. It can be clearly seen that the colours follow the same distribution on the different parts of data sets.



3. Figure: Density of RGB values on the different splits (train, test, val)

Baseline models

Our first baseline solution was to generate random noise image:



4. Figure: Generated noise image

The other baseline solution was the Variational Autoencoder (VAE). This was built using the version implemented in the lab of the Machine Learning course in the previous semester. It is a simpler model with few layers and small latent dimensions. We did not use more layers because we did not want the basic model to be too complex. On the one hand because of the training time and on the other hand because the goal is not to fine-tune the baseline. We used “16-mixed” mode for faster training and early stopping for

validation loss. The models are saved in “models/vae-flowers.ckpt” and “models/vae-celeba.ckpt”. The following figures show the images generated by VAE.

5. Figure: Flowers and faces generated by the baseline VAE model

Methods of training

For training we used diffusers library. We determined the model to use by looking at the official github page and choose the appropriate the size for our model, which resulted in google/ddpm-cifar10-32. Example images of this foundation model:

6. Figure: Example images of the chosen model

We finetuned this foundation model using this documentation. We determined batch_size by setting it to the largest value that an 8GB VRAM could hold.

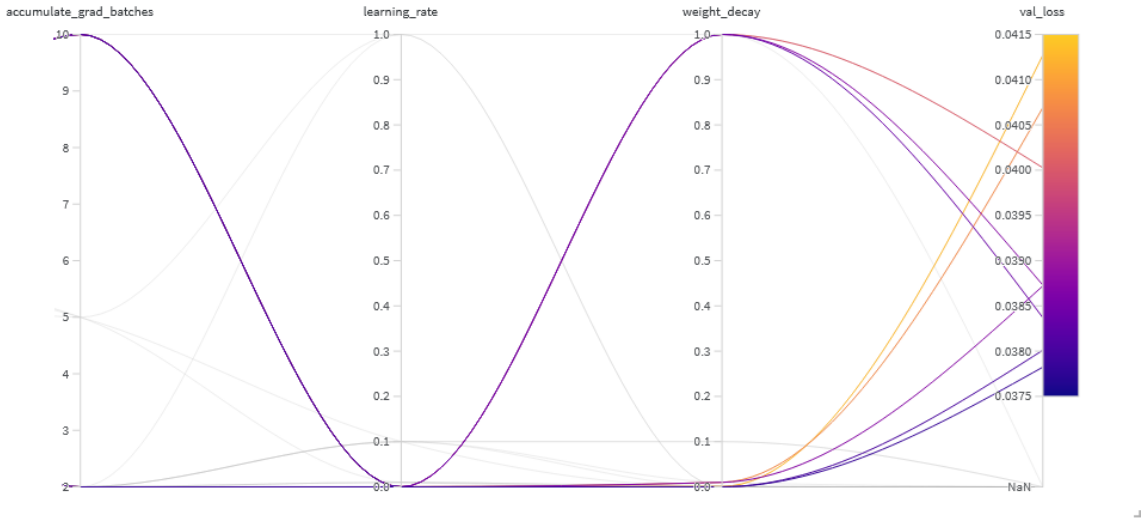
The training follows these steps:

- We generate a random noise
- We take a random timestep (max defined by built in scheduler)
- We add noise scaled according to the timestamp to images
- We predict the noise with U-Net giving the noisy images and timestamp as input
- We take the MSE loss between the original noise and the predicted noise

We also use “16-mixed” mode and early stopping just like in VAE. The image generation consists of the following steps:

- Generate random noise images
- For the number of timesteps defined in the scheduler
    - Predict the noise with U-Net for current timestamp
    - Denoise image by the amount for current timestamp with scheduler

We also hyperparameter optimized the model for learning rate, AdamW's weight decay and gradient accumulation. We run 20 trainings (each taking around 60 minutes) with bayesian method, unfortunately with a wrong class definition so these can't be loaded anymore. However, we could extract the best params and train the new class definition with the best params (we didn't rerun the 20 training because they take very long time). Therefore the currently references sweep only contains one run, while you can query the old sweep with the same "dl-nhf-diff-flowers-old" (we also only did the hyperparameter optimization for flowers dataset). The following plot contains the runs hyperparameters and val_loss on a parallel coordinates plot (with the failed ones, resulting in NaN val_loss grayed out):



7. Figure: Result of hyperparameter optimization

Our implementation currently retrains the model if „eval_only" is false, however when its true it queries the last sweep's best model through Wandb API, then downloads and loads it.

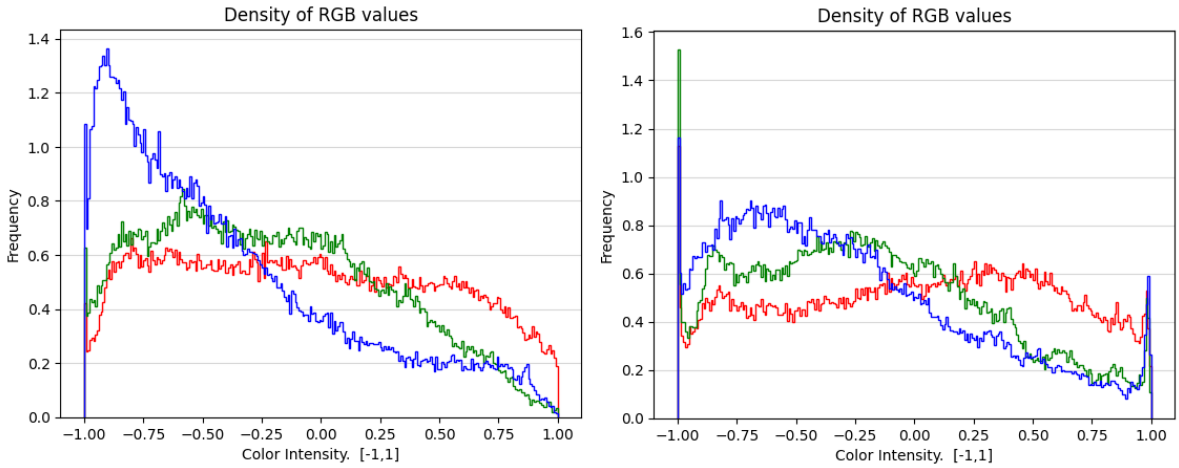
Model evaluation

The images below show some generated images from the finetuned models for the Flowers and Celeba dataset:

8. Figure: Generated images of the finetuned model on Flowers102

9. Figure: Generated images of the finetuned model on CelebA

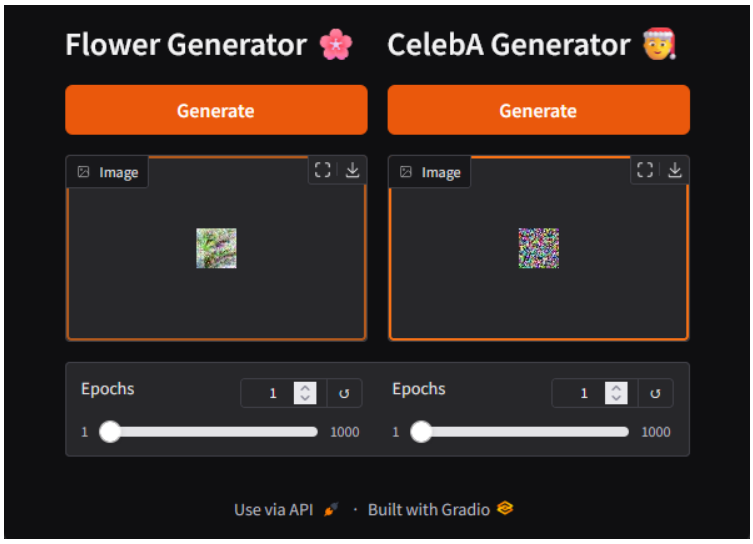
We also generated batch_size number of images to display the average RGB pixel distribution:



10. Figure: Generated Flowers and CelebA image pixel distribution

ML as service – UI

We have also created an interface using Gradio that allows you to try out the trained model. By clicking on the *Generate* button, starting from a random noise, the model removes the noise step by step, updating the image displayed on the interface after each step. Once the generation is complete, a slider allows you to view the state of each step. Both Flowers102 and CelebA can be tested on the interface at 127.0.0.1:8887.



11. Figure: Gradio interface

Conclusion

Approach		Inception score		FID Flowers102	FID CelebA
		mean	std		
Random noise baseline	1.0844	0.0582		50.4702	
VAE baseline	1.4691	0.1964		3.2527	2.4564
Diffusion model	1.6420	0.1707		2.0693	1.0349

12. Figure: Different metrics on for the different approaches

The table above summarises the metrics for the different approaches. One metric is a trained model called Fréchet inception distance (FID) which gives a score by comparing the original dataset with generated images, where a lower FID is better. The other trained model is Inception which gives a score only by looking at the generated dataset, where the higher mean score is better (1 is the lowest possible). This model is trained on multiple image classes therefore to achieve non 1 score we must give both CelebA and Flowers generated images as an input. We didn't give the whole test dataset to FID as for Diffusion model metric evaluation this would have taken a long time, so we consistently made the evaluation on fewer models across all models and metrics.

It can be clearly seen that VAE as a baseline gives a pretty good solution but is successfully outperformed by our fine-tuned diffusion model. In terms of metric values, this means that it achieved a higher inception score and a lower FID value.

Related works

- https://huggingface.co/blog/annotated-diffusion
- https://github.com/huggingface/diffusers
- https://keras.io/examples/generative/ddim/
- DDPM: https://huggingface.co/docs/diffusers/api/pipelines/ddpm

- UNET: https://huggingface.co/docs/diffusers/v0.31.0/en/api/models/unet2d#diffusers.UNet2DModel
- Schedulers: https://huggingface.co/docs/diffusers/using-diffusers/schedulers
- Their Usage: https://huggingface.co/learn/diffusion-course/en/unit2/2