6CCS3PRJ Final Year: Advancements & optimisations into UNet to produce high-quality images in time-saving run-times for text-to-image generative artificial intelligence models.

Final Project Report

Author: Abdurrahman Lleshi

Supervisor: Atsushi Suzuki

Student ID: K20014224

August 19, 2023

Abstract

In recent years, the field of artificial intelligence (AI) has advanced rapidly, with generative AI and machine learning (ML) at the forefront of these breakthroughs. In this research study, we intended to challenge the boundaries of these developments with a particular emphasis on text-to-image models. Our main goal is to optimise industry standard UNet model. The UNet is a key element in the framework of the diffusion model which sits inside the text-to-image model. This paper presented a thorough overview of the most recent research on UNet optimisation. We aimed to achieve a lower Fréchet inception distance (FID) as well as a lower mean absolute error (l1) loss compared to other presented state-of-the-art in text-to-image generation models. We accomplished this by architectural changes to the UNet. However, we went with a more focus goal of a faster convergence indicated by a lower loss score during training of our UNet model. Ultimately, this study will contribute to readers understanding of the diffusion model within generative text-to-image AI, with regards to the UNet advancements and optimisations and its role in the commercial, consumer, and economic spheres.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Abdurrahman Lleshi

August 19, 2023

Acknowledgements

I'd like to thank my supervisor, DR. Atsushi Suzuki. He was a massive help during the projects overall structure and helped direct my ideas into a more narrow, focus leading to a larger gain of knowledge in the topic of machine learning specifically neural networks. I enjoyed this project as it allowed me to strength my understand of artificial intelligence, which the non-technical users think will take over the world such as ChatGPT :0.

Contents

Chapter 1

Introduction

How can we push the boundaries in the latest advancements in generative artificial intelligence (AI) to optimise machine learning (ML) models in text-to-image? The field of generative AI has advanced rapidly in recent years due to the developments in ML models at the forefront.

At its core, text-to-image processing contains a diffusion model. It is used to add random noise to images and reverse the process using mathematical equations. This process involves applying a series of noise vectors and gradually refining them in the reverse to generate high-quality images from noise. However, one of the main components of the reverse process for the diffusion model relies heavily on the use of UNet. The UNet component is used to reverse the image back to its original form. This has been highly effective in segmentation tasks as it effectively reversing the diffusion process.

In this research paper, the aim is to achieve an optimised UNet model. This will be accomplish by exploring the unique advantages and limitations of UNet approach and providing an overview of the cutting-edge techniques that can be applied to the model. One main goal will be to generate a faster convergence with a lower loss score than other state-of-the-art (SOTA) text-to-image papers, such as Imagen, DallE2. Measurements of time taken, space complexity, and processing power will be taken into account when producing such goals of lowering the loss.

Ultimately, the result will be advancing the state-of-the-art text-to-image generating which will unlock new applications and use cases. We will present a comprehensive analysis for consumers interested in UNet. Additionally, we will be highlighting the key advancements and changes to our model over models such as Google, Imagen or CompVis, Stable Diffusion and/or Latent Diffusion. The diffusion model has the capacity to transform fields like computer vision processing, art and content creation. Consequently, it will pave the way for new and exciting

applications within a broader field than just AI technology.

1.1 What are the goals of the project?

The goals of the project will be to contribute solutions to improve UNet specifically on the diffusion model process. This section will outline the improvements that will be applied to UNet and it will illustrates how these changes will be evaluated.

1. This project will first require an implementation of a Frechet inception distance score calculation written in python. This will be achieved by using the following equation:

$$d^2 = ||\mu_1 - \mu_2||^2 + Tr(\Sigma_1 + \Sigma_2 - 2\sqrt{(\Sigma_1 \times \Sigma_2)})$$

   This equation is detailed in the specification chapter.

2. We will be utilising a loss function score within PyTorch library. A loss score is an indication of how bad our UNet model's prediction was on a single generated reverse image. It will be used throughout the project to keep track of changes to the score for images produced during training of the UNet model.

3. We will use an optimisation technique called pruning the UNet model. This will help to optimise our neural network model.

4. Our goal will be to use optimisation methods such as Adam or AdaMax to train our UNet model. This will contribute to lower loss scores during the training process.

5. A development will be to explore the scaling of skip connections to preserve the loss of the previous layer. Consequently, this will help with a faster convergence speed.

6. Since the release of the original UNet paper in 2015, there has been a significant advancement in research. UNet architecture has been modified to be applied to text-to-image models. Therefore, these new architectural changes may be applied to our model.

7. As time progresses, technological discoveries such as quad-core and octa-core processing have become vastly available. This has led to working on splitting work load balances on graphics processing units (GPUs) and concurrently run a larger dataset on the UNet architecture. We will test out concurrency to see if we find a decrease in waiting times for generating an image.

Once the models are improved, we will compare performance across other state-of-the-art text-to-image models, by an open source COCO FID score. This will act as a clear benchmark to compare with the work we produce. We will add the lose functionality as another benchmark for our development of optimising the UNet.

This project will also engage with a broader community to share the improvements and assess the benefits and drawbacks in the field. Our solutions will have the potential to be used and implement into future open sourced projects.

In summary, we will understand and explain UNet, provide advancements in the field of text-to-image models via UNet modifications and contribute to other open source projects.

Chapter 2

Review & Background

2.1 Background

Generative Artificial Intelligence (AI) models are enabling computers to create new materials (images, videos, audios, predicting text) from existing content. In this report we will be exploring one of the latest state-of-the-art generative AI models, text-to-image models - the ability of models to generate images from a text prompt. Considering previous research, we will explore the main component of the diffusion models - the UNet. We will examine current and previous improvements to UNet, algorithmic changes that can be applied and optimisation techniques that have been developed and deployed to machine learning models. We will attempt to add these techniques to our UNet model. This will in turn maintain a low loss score during training of our UNet model and attempt to lower Fréchet inception distance (FID) score (lower score correlates with higher-quality images).

Diffusion models in text-to-image work on the image generating. Their aim is to apply Gaussian noise to images then track-back and reverse the process as seen in the below figure 2.1. This process begins by the forward diffusion model where an image is taken. Then a set of mathematical operations is applied with the use of Markov chains. This can help define the forward diffusion for each successive step by gradually adding noise, using a scheduler from t(time)=0 to t=T(time), e.g., $t_0, t_1, t_2, ..., t_T$. Then the reverse diffusion process inverts this mathematical operation to get back to the original state using the UNet. Overall, diffusion models are a powerful set of tools for image processing. They are applied in text-to-image models to be able to pick along the latent diffusion and add noise at certain steps or reverse at a step. The process includes the need for UNet to be able to achieve the reverse in the

backward diffusion of the noise applied. The UNet helps generate a new images in the diffusion model.

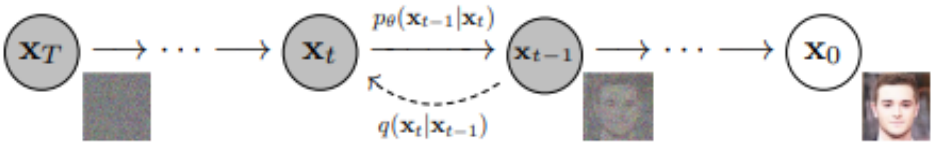


Figure 2.1: Directed graphical model used in Denoising Diffusion Probabilistic Models (DDPM) [7]. Side node: $q(x_t|x_{t-1})$ is a Gaussian noise distribution for the forward diffusion. The $p_\theta(x_{t-1}|x_t)$ is the trajectory of the reverse diffusion of the distribution in which we approximate $q(x_{t-1}|x_t)$ using our UNet model.

The project has originated from a growing interest in the potential of generative AI models to revolutionise the way we create new, unique forms of visual art and content.

The specific focus will be on the improvement of UNet, which is within the reverse process of the diffusion model. This is due to their popularity and effectiveness as a component of text-to-image processing. By exploring the capabilities and limitations of UNet, the project aims to identify potential improvements and modifications that can be made to lower computational power (CPU, GPU, RAM), enhance the image quality and achieve optimisation of the UNet neural network. Consequently, a lower FID score will be achieved. UNet is a convolutional neural network architecture that was first introduced by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in their paper "UNet: Convolutional Networks for Biomedical Image Segmentation" [17] in 2015. The architecture is known for its ability to perform well on image segmentation tasks, which involve predicting the class or label for each pixel in an image.

The UNet architecture is based on the concept of "contraction(encoder) - expansion(decoder)". The contraction part of the network is responsible for extracting features from the input image where it tries to "increase the "what" and reduce the "where"" [16], while the expansion part is responsible for using those features to generate a segmentation mask. The UNet architecture is unique in that it uses skip connections, which allow the expansion part of the network to access the feature maps from the contraction part directly. This reduces the need to learn them from scratch and makes the network more efficient. The contraction part can be shown on the figure 2.2 via the red max pooling arrows and the expansion, which is shown by the green up convolutional arrows. We will expand on the UNet architecture in the design section of this research paper.

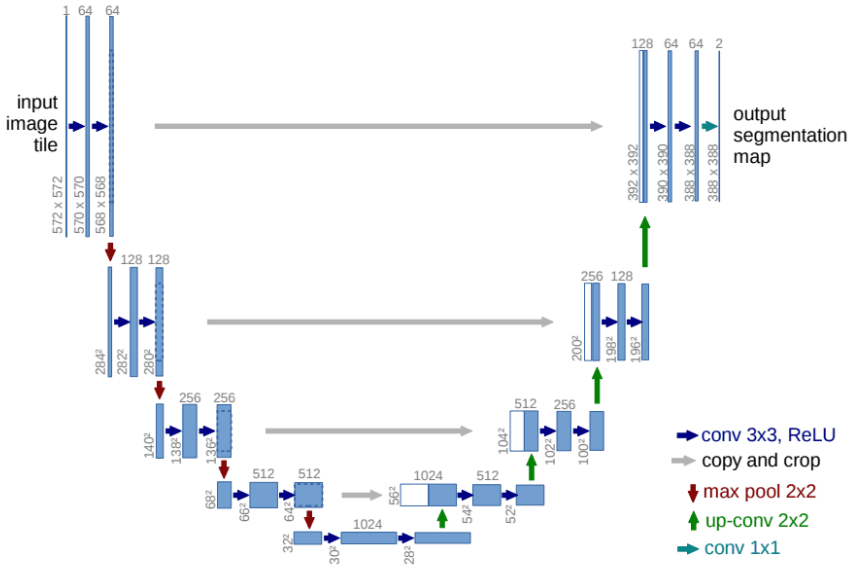


Figure 2.2: UNet architecture.[17]

2.2 Text to image and UNet: A Review of the Literature

Text-to-image in artificial intelligence is a rapidly growing field that has received significant attention from both researchers and industry practitioners. In 2022, this gained massive news and social media coverage from OpenAI with their release of Dall-E 2[14]. This shone a light on what can be achieved when piecing together multiple techniques from machine learning to be able to accurately and effectively generate high-quality images from text descriptions. To demonstrate the success in this field, due to the research conducted into UNet models. This literature review will examine other researchers' and AI enthusiasts' improvements and evaluations of UNet models for diffusion process.

UNet is a popular deep learning architecture that was originally developed by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in 2015 in their paper, "UNet: Convolutional networks for biomedical image segmentation" [17]. This paper demonstrated the benefits of this architecture in its ability to provide precise segmentation of biomedical images. However, a drawback to the findings is that it focuses on biomedical images, which can be useful for a range of applications such as skin disease diagnosis. Another drawback relates to the fact that the scope of investigations did not expand on how these UNet could be produced for other applications, such as text to image. Overall, the method introduced for UNet architecture in this study has been a valuable tool for becoming a main component for text to image AI application.

A number of questions remained on how UNet architecture implementation could be added to images and used for text to image. This is where researchers such as Google, introduced their own research into text to image diffusion model, which they named as Imagen.

Imagen was created by Google, "new state-of-the-art Fréchet inception distance (FID) score of 7.27 on the COCO dataset"[18]. This study builds on text to image models, where they were able to produce one of the lowest COCO FID scores in 2022. The main strength of this paper is that it used an open source dataset called COCO which is "large-scale object detection, segmentation, and captioning dataset"[10]. The advantage of this is that FID scores of the text-to-image can be benchmark across other models in the industry along with comparing with other researchers. We will conduct our benchmark results with this dataset to test out what can be achieved with these models. Another advantage is the contribution in the new "Efficient UNet"[19] architecture which led to faster convergence and had greater memory efficiency[19]. Moreover, the discussion within the paper led to the idea of a better text encoder model, which then allowed for greater results combined with the efficient UNet. However, a multinational corporation like Google who have access to large investment funds and enough processing power of cloud computing (dedicated Graphics Processing Units (GPUs)) can run vast amount of training data with little to no limit on computational power. This question arises whether the model achieved such accuracy due to its large amount of training data and time to run. Furthermore, the article raises the issue of safety of these applications' datasets. For example, one of the multiple datasets used within this paper, LAION-400M ("CLIP-Filtered 400 Million Image-Text Pairs") [20] contained "inappropriate content including pornographic imagery, racist slurs, and harmful social stereotypes" [19]. Overall, Google's Imagen opened the doors to a new look of UNet model and what improvments can be taken and applied to improve their efficency.

Exploring further into diffusion models, research has investigated removing the Markovian diffusion task, which can be replaced with a non-Markovian diffusion process instead. It is possible to hypothesise that to produce Denoising Diffusion Implicit Models (DDIMs) which Jiaming Song, Chenlin Meng and Stefano Ermon tackled in their paper [24]. This paper goes into detail with the drawbacks of Denoising Diffusion Probabilistic Models (DDPMs) [7], such that the slow speeds of running DDPMs on a sample of 50k images; where it takes 20 hours for the size 32 x 32 pixels. This is compared to Generative Adversarial Networks (GANs) [5] which tried to generate new images from random noise taking "less than a minute" [24] against DDPMs. To decrease DDPM's time taken, researchers for DDIM proposed to

focus on the *forward diffusion process*. This was a successful optimising as it demonstrated multiple benefits, such as "superior sample generation quality", "semantically meaningful image interpolation" [24]. These findings suggest the implementation of deterministic sampling for DDIM vs stochastic sampling found in DDPM. Nonetheless, a massive drawback to this paper is that the parameters used for the success on the model is not clear. The author(s) offers little to no explanation what parameters and limitations where used within the research of DDIM against DDPM to calculate of acceleration in sampling by 10x to 100x. Additionally, the paper is heavily algorithmic and mathematical equation based, which can cause non-technical background readers to become confused of how these changes added advantage to DDIM over DDPM. Overall, this paper provided an eye opener into what can be achieved deterministically with the use of non-Markovian diffusion process.

This investigation focuses mainly on optimisation methods. One such method is the Adam optimiser published by Diederik P. Kingma and Jimmy Lei Ba [8]. This paper uses a stochastic optimisation approach, which is a variable process where the outcome involves some randomness and results in some uncertainty. The authors' use a stochastic gradient descent which iteratively updates network weights during training. This paper will benefit our project as it additionally discusses the use of AdaMax, another optimiser method. The result of this research indicates that the Adam optimiser is generally better than other optimisation algorithms with regards to optimising deep learning models. This is due to Adam having a faster computation time, minimal memory requirements which are well suited to larger dataset. Consequently, requires fewer parameters when tuning our model. The designers of Adam model achieved this by combining two other stochastic gradient descents: adaptive gradient algorithm and root mean square propagation. In summary, Adam optimiser efficiently demonstrated how to solve practical deep learning problems across multiple models and datasets [8].

In conclusion, text to image in artificial intelligence is ever changing field with new state-of-the-art models coming out frequently. UNet models, are a popular deep learning architecture that has proven effective in text-to-image AI as a whole. However, there is always changes in machine learning to make UNet models more effective. For example, Google's Imagen model, with its low COCO FID score and contributions to the "Efficient UNet" architecture, has demonstrated success in the text-to-image field. However, the use of large amounts of datasets and computational power in the development of text to image raises questions about model's accuracy and safety of its datasets. An alternative approach to text to image generation is the use of non-Markovian diffusion processes, as demonstrated in Jiaming Song, Chenlin Meng, and

Stefano Ermon's research on Denoising Diffusion Implicit Models (DDIMs). DDIMs improve upon the limitations of traditional diffusion models, but further research is needed to fully understand the potential of non-Markovian diffusion processes in text-to-image generation. The addition in machine learning optimisers also help optimise UNet models. These can be applied to the models and adjust accordingly to achieve an optimised UNet. In this review, we addressed the current progress and challenges in text-to-image models, and the expansion of UNet. Further study into this topic can help improve text-to-image generation and compare their performance such as image quality.

Chapter 3

Design, Specification & Prerequisites

This project will be deployed on King's College London's (KCL) computational research, engineering and technology environment (CREATE) high performance computing (HPC) servers. Originally, the plan with Google Colab was removed due to computational drawbacks and time limits within Google's cloud services. Upon approval for the usage of KCL HPC servers, we will be running our Python Jupyter notebooks on these servers to help narrow down the specific investigation within the research space of UNet, testing the research on the same software and hardware components. The implementation of the diffusion model will be using the paper Denoising Diffusion Probabilistic Models [7] which will be created on a Python Jupyter Notebook to not only add markdown to explain what each code cell does but to run on the KCL's HPC server. This will allow for faster, more precise results from the improvements and optimisations added to the UNet.

3.1 Prerequisites & Requirements

3.1.1 Software

The software requirements of this project will require the following:

- Access to a terminal
- Terminal able to execute Secure Shell (SSH)
- Knowledge on terminal-based text editors such as NANO or VIM

- VSCode
- VSCode Jupyter plugins installed
- GitHub CLI
- Operating system Ubuntu 20.04.5 LTS (x86_64) - preinstalled on HPC Server
- Python version: 3.8.12

For the integrated development environment (IDE), we have selected VSCode. This is due to the ability to use plugins such as Jupyter Notebook plugins. The project will be developed in a Python Jupyter Notebook. Jupyter notebooks are interactive development environments that allow for code to be executed in modular cells with connection to external server to utilise the server's RAM, CPU and GPU. Each cell contains Python code which are executed sequentially. This allows for flexibility in our machine learning research project as we can generate more cells to show types of changes we have added to our project without the need to remove previous module changes.

To access KCL's HPC servers, it is important to have access to a terminal (Unix, Linux, or Windows's cmd) which can execute SSH commands. This is due to the fact that to connect to these servers it will require to generate SSH RSA key pair. KCL's HPC servers allow for batch jobs to run, which are scheduled programs executed on the server when ran. To create these files it is important we know how to use a terminal-based text editor such as NANO or VIM. Once the batch job is running we can connect our Jupyter notebook and run our code. The setup as well as these steps will be outline in the user guide for our end-users.

A software to keep track of version control for the project will be the use of GitHub. The knowledge of GitHub's Git version control system will enable to keep track of changes and help revert to previous versions. This will also help with issues of bugs or errors occurring because of packages being constantly updated in PyTorch for Python. It is important to have GitHub CLI installed as this will allow for terminal based commands to be executed to push, checkout or revert to a branch.

3.1.2 Hardware

For each approved project on the CREATE HPC server, King's College London allocates with the following shared hardware:

- NVIDIA A100-SXM4-40GB Graphic Processing Unit (GPU)

- 6912 CUDA cores for the A100 GPU
- Max 350 concurrent cores per user (10% of CREATE total capacity)
- Max 8 concurrent A100 cards per user (20% of CREATE total capacity)
- 1GB limit of RAM per batch job.

We will be running the development and testing of project on KCL's CREATE server, a high performance computing (HPC) service provided by King's College London (KCL) for "e-Research which is a group that is focused on providing King's College London's researchers with infrastructure and support for conducting research using software development, computational, and data analysis methods" [3]. This project originally would be on Google Colab. However, was limited to Tesla T4 GPUs which were equipped with 16 GB of video random access memory (vRAM). Upon approval of the project to access the KCL's HPC CREATE servers, the drawbacks of the use of Google Colab where eliminated and we did not have to use Google Colab IDE browser. The university's CREATE servers are equipped with A100 NVIDIA 40GB GPUs. These powerful GPUs allow for faster training and generation of AI models, enabling researchers to experiment with larger and more complex datasets. This is due to their large 6912 Compute Unified Device Architecture (CUDA) cores, these cores are equivalent to nvidia's own CPU cores except for GPU purpose. However, it is important to note that these resources come with a running time limit of 48 hours and a batch job size of 1 GB. Within this project, we need to optimise our training processes and consider the size and complexity of the models to ensure that they can make the most of the available resources within the time and batch constraints. With careful planning and optimisation, the use of the university's CREATE servers with A100 NVIDIA 40GB GPUs can significantly accelerate the development of optimising the UNet model for generative AI models and lead to faster results in the context of this project.

3.1.3 The type and size of images or text as the input and output

The input of this project will use celebA to train our model. This is an open source dataset which was used in the original DDPM paper [7]. The use of an open source dataset as our input allows us to have a variety of options to apply to out input images. The dataset will be implemented using torchvision, which is a PyTorch package containing popular and common datasets. This can allow for other datasets to be loaded during testing.

This project will focus on trying to run different batch of images (around 64, 128, 246 images). This will allow us to calculate the steps needed before hand for each epoch (one passing of data through out UNet model) during training our model.

Images sizes would be relative small of 64 x 64. However, we will increase and decrease during training the UNet model to test for any changes and improvements in our model.

These parameters are taken into consideration to maximise the ability of training our model with the goal of lowering the loss score as well as minimising the amount of errors cause by running out of RAM (Random Access Memory) or vRAM. The output of the images generated via the model can then be used to try and score with FID to the original images in the dataset.

3.1.4 Whether the data generation is random or deterministic

This research project will focus on generating deterministic images. This means that the generated images will be based on a specific, predetermined dataset, and will be consistent and predictable in their output.

There are several reasons for focusing on deterministic generation. First, deterministic models are easier to evaluate and compare, as the input and output are fixed and consistent. This allows for more reliable and accurate assessments of the performance of the model when calculating FID score.

Secondly, deterministic models are more transparent and easier to interpret, as they can be traced back to the specific input that led to the generated output. This can provide valuable insights into the workings and capabilities of the model.

Finally, deterministic models are more useful in practical applications, as they can be used to generate specific images on demand, rather than random and potentially irrelevant images. Overall, the focus on deterministic text-to-image generation in this research project will provide a more reliable and useful basis for evaluating and advancing the state-of-the-art in generative AI models.

3.1.5 The number of parameters, which we are motivated to limit in the project & performances evaluation

The number of parameters in a diffusion model can significantly impact its performance, were are therefore motivated to limit the amount of parameters by the following as well as solutions to over come them.

1. Image sizes of around (32x32 $\rightarrow$ 64x64) pixels.

2. Text prompts pre-defined and trained on and used throughout the project.

3. Ability to change aspect ratio for images.

4. Limit details of the image generation, for example, generate only 30 steps (30% completion).

5. Consider more lightweight model architecture such as SD-UNet [4].

6. Decrease the upsampling layers and convolutional layer to a single layer than multiple.

7. Use of regularisation techniques to help reduce over fitting of new data.

For the performance evaluations of the quality generation of images, it is largely dependent on the computational power available. However, even with limited computational resources, it is possible to generate relatively high quality images by training the model using appropriate optimisation techniques. We will also explore techniques such as super-resolution to mitigate loss of detail and resolution during the up-scaling process, this is due to loss of information up-scaling an image from low resolution.

For performance evaluations of computational complexity we can evaluate two ways as follows:

1. Time complexity of a machine learning algorithm, referring to the amount of time to execute the algorithm.

2. Space complexity, referring to the amount of memory required to execute the algorithm.

All in all, lower computational complexity are preferred, creating more efficient algorithms or using PyTorch optimisations to be able to train and run faster. However, it is important to balance between computational complexity and quality of image. The trade-off between these to achieve better results of images or decrease execution times.

3.2 Specification

This section we will discuss the deep learning equations and functions that will be required in the design of our code base and evaluation models.

3.2.1 Fréchet Inception Distance (FID)

The Fréchet Inception Distance (FID) is a measure of the distance between the distributions of real images and generated images. It is calculated using the Inception network, a pre-trained convolutional neural network that has been trained on the ImageNet dataset. The FID is denoted by:

$$FID :: d^2 = ||\mu_1 - \mu_2||^2 + Tr(\Sigma_1 + \Sigma_2 - 2\sqrt{(\Sigma_1 \times \Sigma_2)}) \qquad [6]$$

where μ_1 the mean for the original images and μ_2 are the mean for the generated images, σ_1 covariance matrix over the activation for the real images and σ_2 covariance matrix over the activation for the generated images. Tr is the sum of the elements along the main diagonal of a square matrix $(\Sigma_1 \times \Sigma_2)$. The FID is a lower-bound distance between the two distributions, with a lower score indicating a closer match between the real and generated images. It is commonly used as a metric for evaluating the performance of generative models in tasks such as text to image generation.

3.2.2 Forward Diffusion equation

The following equations define the forward diffusion process. This follows:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \qquad [24]$$

State variable at time t (denoted as x_t) given the state variable at time t-1 (denoted as x_{t-1}) e.g. $x_0, x_1, x_2, ..., x_{t-1}$. In this equation, the conditional Gaussian distribution of x_t given x_{t-1} is represented by the normal distribution $\mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$, where β_t is a "variance schedule" [24] that controls the variance of the distribution, can be linear, quadratic, cosine, etc.

3.2.3 Reverse Diffusion equation

The following equation defines the reverse diffusion equation:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \qquad [24]$$

This is the reverse of the forward diffusion equation. However, we can not estimate $q(x_{t-1}|x_t)$ simply as it would require knowledge of the whole dataset we therefore approximate with p_θ. p_θ is the parameterised model for approximating $q(x_{t-1}|x_t)$ for our model this will be the UNet neural network. The use of timestep t helps the model to better predict the Gaussian noise parameters. Given p_θ for the previous time step is equal to the current timestep with a mean of $\boldsymbol{\mu}_\theta(x_t, t)$ and a covariance matrix of $\boldsymbol{\Sigma}_\theta(x_t, t)$. The variables of μ and Σ are functions for the current time step.

3.2.4 L1 loss equation

Within the PyTorch library we have access to multiple loss functions. Each loss function has its own unique properties and applied to different machine learning problems. A loss is the penalty for a bad prediction. As the ML model is trained we expect a lower score. In the scope of this project we went with a L1 loss also known as the mean absolute error (MAE). The L1 loss computes the average of the sum of absolute differences between and actual value and predicted value. In our project this will apply to the noise added to a diffusion image. The L1 loss is a commonly used for a regression problems. This applies to the scope of the project well due to predicting a continuous numerical value when we reverse the forward diffusion equation. The following equation defines the L1 loss of a singular two points (x, y):

$$loss(x, y) = |x - y|$$

Where x will be the actual value of the noisy image and y will be the predicted value of the noise added to the image.

3.3 Design

The design of this research project will involve utilising open-source projects and research papers that have created successfully functioning diffusion models. We can later add our optimisations to the UNet modules of these diffusion models and then using other open-source projects such as imagen_pytorch and use CompVis/latent-diffusion as a benchmark for our changes. These projects provide a framework for implementing fully working text-to-image models with CreativeML Open RAIL-M licenses in python code.

As part of the design, we will make improvements to the UNet component of these open source projects specifically using a library called PyTorch (open source machine learning library for python), based on our research and findings. This will involve implementing algorithmic changes and modifications to enhance the performance of the UNet in generating high-quality images. Our design of the UNet explained below will be a base for the optimisation we will implement to our UNet model.

Moreover, in this section we describe the training and sampling algorithms from the DDPM paper [7]. This is important part of the model as it will allow for forward diffusion of adding noise vectors to datasets, as well as reversing the diffusion in the backwards diffusion. These two algorithms are taken directly from the paper which we will adjust accordingly when developing in python.

In addition, we will also add the FID score implementation in Python. This will allow for the evaluation of the generated images using the COCO dataset as a reference for any changes within the score to indicate the worst or best solution (the higher the score, the worse the generated images). The implementation of the FID score will provide a quantitative measure of the performance of the UNet and the overall text-to-image generative AI model.

In the long term, the design of this research project will involve leveraging open-source projects as the primary source code and adding our own improvements and additions to enhance the performance and capabilities of the generative AI models.

3.3.1 UNet Design

When developing a well established diffusion model the design of a UNet model slightly differs from its original goal of image segmentation. In this section we will outline a more in dept design of the UNet model that will allow for implementing our UNet. We will describe a simplistic model design as well as improved designs to optimise the UNet to advance the diffusion model.

As described in the background a UNet is a convolutional neural network (CNN). A neural

network is a technique in deep learning which is inspired by the human brain. Nodes (neurons) are interconnected with one another. These nodes then have an activation function. An activation function is comparison between an input value to the node compared to the threshold value of the next node, if the input value is greater than the threshold the node is activated, else it is disabled. For our simplistic UNet design we will use a rectified linear activation unit (ReLU).

A ReLU is a piecewise linear function that output the input if positive else zero as shown in the diagram below 3.1. The ReLU activation forces parts of the model to deactivating certain nodes, which helps the model better generalise, preventing overfitting.

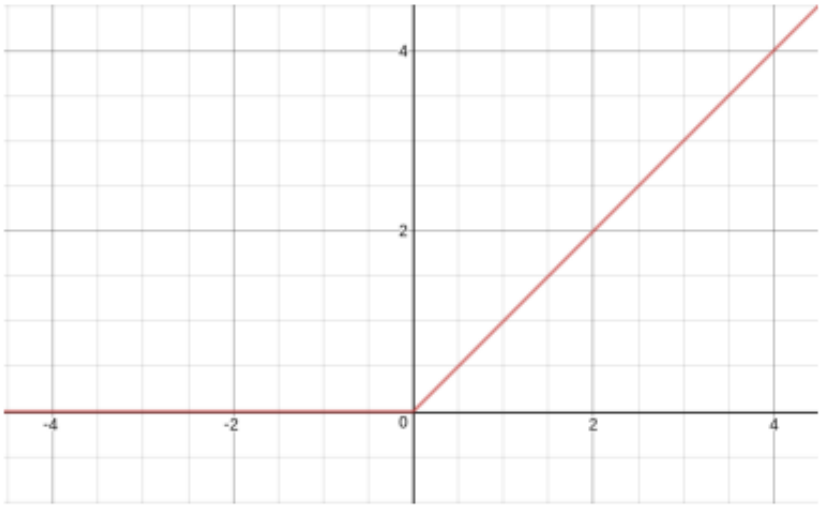


Figure 3.1: The Rectified Linear Unit (ReLU) activation function produces 0 as an output when $x < 0$, and then produces a linear with slope of 1 when $x > 0$ [1].

UNet - Encoder

The encoder is a convolutional block part of the network. Each one of these convolutional block consists of two 3×3 convolutions with a ReLU activation function coming after each convolution. This will help us to extra relevant features and information from an image of the dataset and feed them into following series of encoder blocks (nodes).

The matching decoder block input will be from the ReLU's output as a skip connection. Then the features extracted are cut in half via the 2×2 max-pooling block each time one block moves into another encoder block. This lowers the amount of trainable parameters, thereby lowering the computational cost. In the following figure 3.2 we have enlarged the UNet encoder. We can see the input of image is passed though two convolutional blocks with each containing a Conv2D, batch norm (normalisation method between neural network node) and the ReLU

function. Batch norm is applied to re-centre and re-scale the input. The last ReLU would be a skip connection, connecting to its decoder node counterpart.

Finally, feeding into the max pool which is later inputted into another encoder until it reaches the bridge into the decoder.

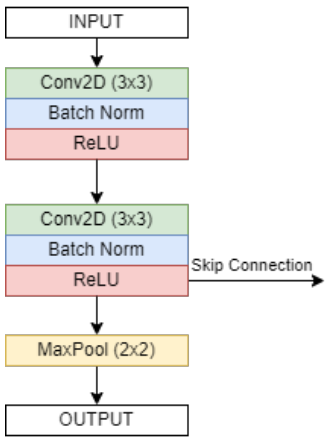


Figure 3.2: Zoom in of a single UNet encoder block (node). Two convolutional blocks (Conv2D, Batch Norm, ReLU), a MaxPool and skip connection.

UNet - Skip Connections

Within the UNet model contains skip connections. These connections skip (bypass) some of the layers (nodes) in the neural network. Skip connections have a variety of usages. In the UNet model it helps build images of different sizes. This is due to upscaling and downscaling the images in the UNet as we would want to extract specific features at various resolutions and timesteps. This allows during the encoder-decoder to extract the most important details in the previous node.

Researchers have tackled the skip connection to try visual the problem. In a paper called visualizing the loss landscape of neural nets [9], they have released variation of the loss function without skip connections and with skip connections. The loss function used was a high-dimensional non-convex function which was minimised in the paper. As presented in the figure 3.3 without skip connections the loss function was all over the place, creating jigged mapping which in turn shows the difficultly for a ML model trying to converge compared to with a skip connection (much smoother).

This is an important design of the UNet to make sure our model converges close to the output goal faster, and close towards the global minima.

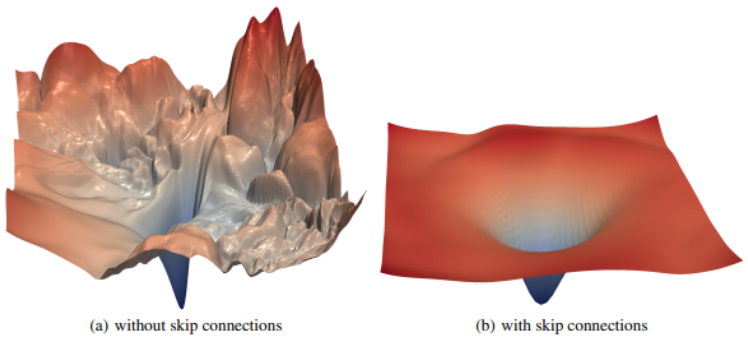


Figure 3.3: The loss surfaces of ResNet-56 with/without skip connections [9].

With skip connections the architecture needs to concatenate each neural network layer. This is shown via the grey (copy and crop) arrows in the original UNet architecture as we saw in the background of this project figure 2.2.

UNet - Bridge

The bridge is a simple connection to pass the features and information from the encoder into the decoder. It also consists of two 3×3 convolutional block with a ReLU activation function as shown in figure 3.4.

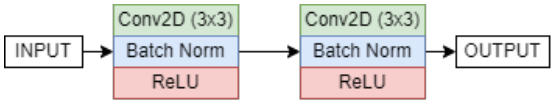


Figure 3.4: Zoom in of a single UNet bridge block (node). Two convolutional blocks.

UNet - Decoder

During the decode component of the UNet, it is important to upscale the map features via a transpose convolution. Transpose causes the input to switch dimensions with the output. We then have a concatenation from the corresponding encoder block acting as a skip connection. These are then passed through again two convolutional blocks the same as the encoder. The final decoder passes though a 1×1 convolutional block and outputs a segmentation map.

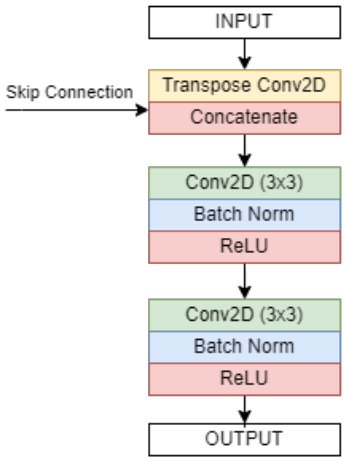


Figure 3.5: Zoom in of a single UNet decoder block (node). One transpose Conv2D with a concatenate for the skip connection, two convolutional blocks.

This design is the basic outline of the plan for our UNet model. We will create the UNet in the implementation chapter using PyTorch, a machine learning framework written in python.

3.3.2 Training & Sampling Algorithms

The training and sampling algorithms have both been taken from the original DDPM paper [7].

Training of the diffusion model

This section will clearly layout how we trained our diffusion model in accordance with the equations defined. The goal of training would be able to reverse the process of an image with noise vectors added. This is where we train the following: $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Are goal is to train the model in a manner which lowers the loss score over time. This will be achieved by implementing a training algorithm. The following is the pseudocode presented in the DDPM paper. During the process of the training it is important for us that we also include an optimiser to converge the algorithm faster. For this we went with the Adam optimiser.

The training algorithm works as follows:

- Random noise sample from a real data distribution $x_0 \sim q(x_0)$
- Uniformly sample the noise t between 1 and T (i.e, timestep)
- Define a Gaussian distributions and corrupt the input image at level t

Algorithm 1 Training [7]

1: **repeat**
2: $x_0 \sim q(x_0)$
3: $t \sim \text{Uniform}(\{1, ..., T\})$
4: $\epsilon \sim \mathcal{N}(0, I)$
5: Take gradient descent step on
6: $\bigtriangledown_0 ||\epsilon - \epsilon_0(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)||^2$
7: **until** converged

- Neural network is trained to predict this noise.
- Repeat the algorithm until convergence (loss settles within a range close to final value)

Sampling of the diffusion model

Sampling helps define generating new images from the diffusion model. This happens due to the Langevin dynamics which is a concept from physics. Langevin dynamics help statistically modelling molecular systems. The Langevin uses a stochastic gradient to apply Gaussian noise into the parameter updates which avoids collapsing into the local minima. Our goal for the UNet is to minimise and converge towards the global minima. However, this algorithm requires a great amount of knowledge in physics, maths and computer science to explain the equations. Therefore, we only present the sampling pseudocode in the design as open sourced projects have freely to use implementations of the sampling algorithm.

Algorithm 2 Sampling [7]

$x_T \sim \mathcal{N}(0, I)$
2: **for** t=T,...,1 **do**
$z \sim \mathcal{N}(0, I)$ if t > 1, else z = 0
4: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_0(x_t, t)) + \sigma_t z$
end for
6: **return** x_0

3.3.3 Libraries, Classes & functions to be used

- PyTorch (Python machine learning framework)
- pytorch.nn (Used to create UNet contains convolution, pooling layers)
- imagen_pytorch (Open source library with the implementation of imagen in pytorch)
- latent-diffusion (Open source code base for latent diffusion)
- numpy (Used for FID score calculation)

Chapter 4

Implementation

4.1 Implementation Body: A structure of the implementation

Over the course of the project's development, the structure followed a chronological order, with the development taking a waterfall approach. This was the most appropriate approach for a machine-learning research and development project. This was due to the need to correctly plan the project. We initially gather the requirements, design and analysis, which we then mapped out in Chapter 3. We started developing a diffusion model because the main component of the UNet model is within the diffusion model. Once a basic model was created, the use of the PyTorch library allowed us access to loss functions such as l1 and mean squared error loss. This also allowed for the implementation for an FID score. As well as the use of PyTorch pruning and optimisation methods that where readily available to use.

The main project took an abstract approach to the development of text-to-image models, basing our analysis on a simpler diffusion model and removing whole clusters of models such as encoder, decoder, and text transformer models. The decision to employ this strategy was centred on the extensive optimisation requirements of the UNet model that this research focuses on, which we saw as a potential bottleneck within text-to-image models. We have been able to save significant time and memory by using the diffusion model as its own separate model, along with lengthening the training periods for the UNet model beyond what was part of the original plan. This has enabled the results and evaluation of the project to be based on isolating the UNet model. As a valuable extension to existing diffusion models, our proposed model has the potential to be integrated into open-source projects.

This chapter reflects a step-by-step solution to achieving this. In the later chaptesr, we discuss the results of this optimisation and pruning of our deep learning UNet model. The results where generated using matplotlib. This allowed for the generation of graphs to better understand our results and improvements, as well as make it easier for the reader to grasp the concepts of the changes applied.

4.2 Diffusion Model: An implementation of the models architecture

The focus of this section will be to present a diffusion model, which will be coded in a Jupyter notebook, allowing for easy execution by future readers of this report.

4.2.1 Diffusion Model - Dataset

During development, we went with torchvision dataset. Torchvision is a package within PyTorch containing well established datasets. Tho torchvision allows the use of COCO dataset, we have went for celebA dataset. CelebA dataset is a popular and widely used data of over 202,599 celebrity images. These images contain a wide range of attributes such as various ethnicity, age, gender, different hair styles and a verity of facial expressions. This is a diverse set of data which has been applied in numerous research papers including DDPM [7], which inspired the code for the diffusion model.

4.2.2 Diffusion Model - Forward Diffusion

Forward diffusion is where the model is least relevant. This is due to its only usage of adding noise to any image or dataset without any training needed. This was implemented using a linear noise scheduler approach. This was because these are time-dependent functions. Linear approach has also been picked as it performs better than other noise scheduler. However, having multiple different approaches such as cosine, quadratic, and sigmoid is beneficial in the long run as it would allow for future work to approach with a different noise schedule. In the forward diffusion, we have added all four functions. However, only the linear noise scheduler was used. Once we define the beta scheduler, we then need to define the alphas, calculate the diffusion, and finally define the posterior. This is how we achieved this.

Defining the linear beta schedule

From the forward diffusion equation, where we had β_t as a variance schedule, we went with a linearly beta scheduler implementation. We used the PyTorch linspace functionality, which allows the creation of a one-dimensional tensor of time steps from start to end. In the denoising diffusion probabilistic models' paper[7], β_1 was set to 10^{-1} (0.0001) for the initial beta with an end of $\beta_T = 0.02$. In our implementation we also went with these two beta points. We also added a timesteps of 300, this is due to the timesteps being a parameter in which allows the linspace to add the steps of the tensor from 0.0001 to 0.02. We also thought this was an appropriate approach due to the linear out performing coise within fewer diffusion step research conducted by Alex, N and Prafulla D on improving DDPMs[11] with the graph 4.1 below showing the linear performance over cosine.

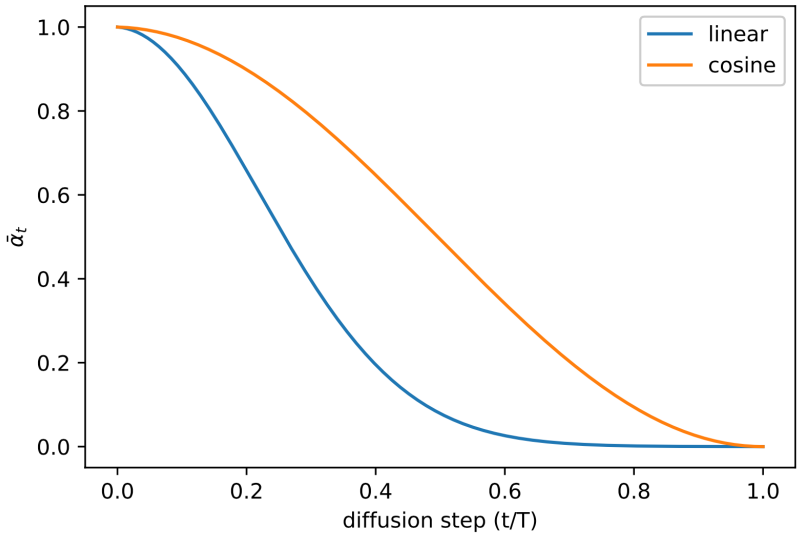


Figure 4.1: Linear vs cosine based scheduling of β_t during training.[11]

The function can be used as follows:

```
>>> linear_beta_schedule(300)
tensor([1.0000e-04, ... , 2.0000e-02])
```

Where 300 is the timestep of the tensors array.

Defining extra beta schedulers

Additionally, to the linear schedule, we have included cosine, quadratic, and sigmoid beta schedules. These where also included in the DDPM paper. However, we have implemented

these beta schedules in PyTorch for completeness of our diffusion model. These are redundant functions for our project. Though, these functions can future proof our project in terms of maintaining the software in case a newly improved modification to help speed up the process of the forward diffusion model is discovered due to the change in the beta schedule.

Forward diffusion sampling

The main functionality of the forward diffusion is the sampling; in the design of the project, we defined an algorithm used for sampling. The function get_index_from_list(), takes in three parameters. The first parameter is sqrt_alphas_cumprod, which comes from the precalculated variables seen in the section below. The second parameter is t, where t is the timestep which is another predefined variable at T = 300. The third parameter is x_shape. The calculation is done by x_0 is the image and shape is a PyTorch function returning an array of tensors for the batch images. This function returns the index t and batch dimension in a device format (the device can either be a CPU or GPU).

```
def get_index_from_list(vals, t, x_shape):
    batch_size = t.shape[0]
    out = vals.gather(-1, t.cpu())

    return out.reshape(batch_size, *((1,) * (len(x_shape) - 1))).to(t.device)
```

Returns an index t of vals while considering batch size.

```
def forward_diffusion_sample(x_0, t, device="cpu"):
    noise = torch.randn_like(x_0)
    sqrt_alphas_cumprod_t = get_index_from_list(sqrt_alphas_cumprod, t,
    ↪  x_0.shape)
    sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
        sqrt_one_minus_alphas_cumprod, t, x_0.shape
    )
    # mean + variance
    return sqrt_alphas_cumprod_t.to(device) * x_0.to(device) \
    + sqrt_one_minus_alphas_cumprod_t.to(device) * noise.to(device),
    ↪  noise.to(device)
```

Takes a timestep and returns the noisy version of the image with that timestep.

Defining precalculated variables

These variables are all predefined. We have taken them from the paper as well as the forward diffusion equation. With the addition of the comments in the following code to present where these variables tie in with the equation. This implementations only works within PyTorch due to its vast mathematical equations available to us to create these predefined variables.

Firstly, we defined the beta schedule with a timestep T. The variable betas can now be swapped out with another scheduler with minor changes in the function called. However, for the project we went with the linear beta schedule and 300 for timesteps.

```
T = 300
betas = linear_beta_schedule(timesteps=T)
...
```

Secondly, we had to define the alphas. The alphas calculate the cumulative product of the 1 minus beta. Then we pad the tensors. This adds padding to the alphas_cumprod variable. This adds a value of 0 around empty tensors to ensure the tensors are equivalent.

```
...
alphas = 1. - betas
alphas_cumprod = torch.cumprod(alphas, axis=0)
alphas_cumprod_prev = F.pad(alphas_cumprod[:-1], (1, 0), value=1.0)
sqrt_recip_alphas = torch.sqrt(1.0 / alphas)
...
```

Thirdly, we calculated the diffusion of $q(x_t|x_{t-1})$. Where we took the cumulative product of the alpha, then we square rooted the alphas in accordance with the forward diffusion equation of $\sqrt{1-\beta_t}$, where $1-\beta_t$ is the alphas.

```
...
sqrt_alphas_cumprod = torch.sqrt(alphas_cumprod)
sqrt_one_minus_alphas_cumprod = torch.sqrt(1. - alphas_cumprod)
...
```

Finally, from the forward diffusion equation we have a posterior variance which we calculate in Python the following:

```
...
```

```
posterior_variance = betas * (1. - alphas_cumprod_prev) / (1. -
↪ alphas_cumprod)
```

4.2.3 Diffusion Model - Image Helpers

Before beginning our investigation, the images from the data had to be loaded into Tensor arrays according to the DDPM paper as well as test if the forward diffusion is working correctly. In any deep learning project, it is important that the dataset is to be split on two variables one train for training and one test for testing. We split the celebA into the following:

```
def load_transformed_dataset():
    ...
    train = datasets.CelebA(root='', split="train", download=True,
    ↪ transform=data_transform)
    test = datasets.CelebA(root='', split="test", download=True,
    ↪ transform=data_transform)
    ...
```

The function load_transformed_dataset() allows for the celebA dataset to follow the DDPMs scale of: "We assume that image data consists of integers in {0,1,...,255} scaled linearly to $[-1, 1]$. This ensured that the neural network reverse process operates on consistently scaled inputs starting from the standard normal prior $P(x_t)$". This sums up the transformations needed to convert the dataset into the tensor array. We achieved this with the following code:

```
def load_transformed_dataset():
    transform = transforms.Compose([
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Lambda(lambda t: t * 2 - 1)])
    data_transform = transform
    ...
```

It is also important to be able to reproduce these images back to their original forms we have applied a reverse the tensor transforms. We defined a function in our project reverse_tensor_img() which allows to reverse the tensor transform.

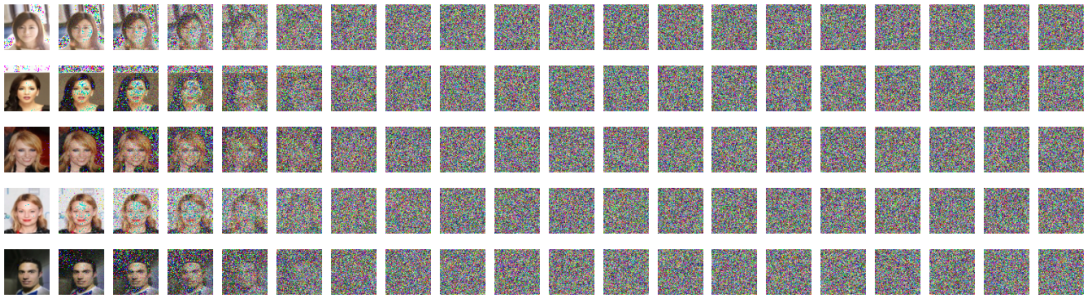


Figure 4.2: Five forward diffusion celebA images produced

From the figure 4.2 we have successfully been able to apply the forward diffusion to the celeA dataset.

4.2.4 Diffusion Model - Backwards Diffusion

Backwards diffusion was our main focus. Within the backwards diffusion is where the UNet is embedded. Following the design of the UNet in chapter 4 and rather than redesigning the wheel attempting to code our own unique backwards diffusion, we implemented an open source version of the backward diffusion taken from the DDPM paper as well as the "Annotated Diffusion Model" presented by Kashif Rasul and Niels Rogge [15]. This minimised the time spent trying to figure out the mathematical translation of the equation in the specification into code. This is due to the scope of our project not being on re-coding and programming from scratch, but rather using a model that has been well established to allow optimisation of the UNet model. This became massively advantageous to us as it gave us more time to focus on pruning, optimising, training, and changing the UNet model rather than being caught up with errors or bugs from re-coding a "custom" backwards diffusion model.

4.2.5 Diffusion Model - UNet in backwards diffusion

The UNet contains different building blocks as explained in the design of this project. To start with we first defined constants that will be the inputs of each up channels and down channels blocks, defining the image channels (RBG - Red, Blue, Green), time embedding dimensions and the final output dimension block of 1×1.

```
    ...
image_channels = 3 #RGB: 3 channels for RED, GREEN, BLUE
down_channels = (64, 128, 256, 512, 1024) #N.o. of channels in each downsample
    layer
up_channels = (1024, 512, 256, 128, 64) #N.o. of channels in each upsample
    layer
out_dim = 1 #1x1 final of output channels
time_emb_dim = 32 #Dimension of time embedding
    ...
```

Next we defined a time embedding with PyTorch sequential function which treats inputs as one module we call our SinusoidalPositionEmbeddings(), Linear and ReLU functions. These three functions become one module in which we use to build our UNet time embeddings. This is explained in detail in later paragraphs 4.2.5.

```
    ...
```

```
self.time_mlp = nn.Sequential(
    SinusoidalPositionEmbeddings(time_emb_dim),
    nn.Linear(time_emb_dim, time_emb_dim),
    nn.ReLU())
    ...
```

We built our convolutional blocks in our UNet, a detailed explanation of how each block is generated is explained in later paragraphs 4.2.5.

We first initialise a single convolutional (Conv2d) block called conv0. In PyTorch the Conv2d took in three main parameters, in channel, out channel and kernel size. Our initial block took in our image_channels of 3, an out channel of 64 and a kernel size of 3 with an optional parameter padding of 1.

Next we generated our down sample (encoder) and up sample (decoder). Both where designed in a for loop iteration manner and assigned accordingly to downs variable and ups variable. The use of the ModuleList provided by PyTorch allowed to store sub-modules (each convolutional block scale from 64→1024 and back to 1024→64) in a list.

Finally, we defined the last output convolutional 1×1 block called output, which was a simple Conv2d with the in channel 64, out channel 3 and kernel size of 1.

```
    ...
self.conv0 = nn.Conv2d(image_channels, down_channels[0], 3, padding=1)

self.downs = nn.ModuleList([Block(down_channels[i], down_channels[i+1],
↪  time_emb_dim)
            for i in range(len(down_channels)-1)])
self.ups = nn.ModuleList([Block(up_channels[i], up_channels[i+1],
↪  time_emb_dim, up=True)
            for i in range(len(up_channels)-1)])

self.output = nn.Conv2d(up_channels[-1], 3, out_dim)
    ...
```

To wrap up the UNet architecture in PyTorch we created a final function foward() which took in two parameters x and timestep. This generated all the channels (x) for the UNet and appended the timestep using the time_mlp function at each block.

UNet - Time or Position Embedding

It is important that we keep track of the time embeddings of where a particular time step the neural network is at for each image batch. This is so the noise levels can be computed. The SinusoidalPositionEmbeddings() function takes a tensor shape of (BATCH_SIZE, 1) of a set of noisy images in a batch and returns a tensor shape of (BATCH_SIZE, time_emb_dim). This is used to add the block to keep track of each timestep of the diffusion process. The sinusoidal position embeddings where taken from the "attention is all you need" paper [26]. The scope of investigation was beyond time embeddings and therefore, we did not include the equation in the specification.

UNet - Convolutional Block

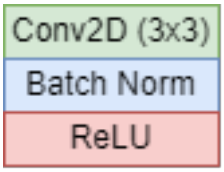


Figure 4.3: A simplified single convolutional neural network (CNN) block for our UNet model.

We defined a Block class. Our design for this convolutional block (node) class took in three parameters similar to the Conv2d defined previously. The first is the in_channel, this is where a corresponding down_channels variable we defined previously will start in the down sampling of the image. Here we started at $64 \rightarrow 1024$. Next we had an out_channel parameter, similar to the previous channel this is responsible for all the up_channels in our model. Then we had our time embedding this is to make sure each block knows what time t it is at. Finally, we had a boolean (default false) parameter called up. This was used to explain if we where in the decoder (expansion) part of the UNet where we had to apply a transpose of the convolution.

We then built two convolution blocks called Cov2d using PyTorch as well as a batch norm for both these blocks and attach a ReLU activation at the end.

In the foward() function of the block class is where the block is fully concatenated with the time embeddings, 2 convolutional blocks with the ReLU activation function to generate the encoder or decoder convolutional block for our UNet model.

4.2.6 Diffusion Model - Training

Loss function

We defined a simple get_loss() function. This function used the noise from the forward diffusion sample to the previous noise and returns a loss. In our project we went for the l1 loss. However, our code allowed for further expansion of other loss functionality.

```
def get_loss(model, x_0, t, type="l1"):
    if type == "l1":
        x_noisy, noise = forward_diffusion_sample(x_0, t, device)
        noise_pred = model(x_noisy, t)
        return F.l1_loss(noise, noise_pred)
    ...
    else:
        raise NotImplementedError()
```

As discussed in our specification, by default we used the L1 loss. However, we added the ability to expand on these functions and included extra loss function depending on future development.

Training

When training our UNet model we went with PyTorch optim library. This library contains an vast amounts of optimiser in deep learning such as Adam. The following explanation can be change to other optimisers creating a universal function the following will describe how we implemented Adam to optimise our UNet model and train the model. In the chapter section 4.3 we will go into more details of Adam optimiser.

We first had to import the Adam optimiser. Then we had to feed our model into a cuda available device, in our case was the A100 GPU. We initialised the Adam optimiser with a learning rate of 0.001 as follows:

```
...
optimiser = Adam(model.parameters(), lr=0.001)
...
```

Next, we applied two for loops. The outer for loop applies the max amount of epochs on a run. The inner for loop enumerates over the dataloader (contains the dataset). In the inner

loop for each step and batch we apply zero_grad() to the optimiser. This simple function in PyTorch allowed us to save memory on our device with setting all tensors to zero enhancing the performance of the UNet model. Then we generated a time t which is used for the get_loss function as well as keeping track of the timestep. After the time t is generated we get a loss function and apply a .backward() call. This call allowed us to accumulate (add) to the gradient of the UNet diffusion model when training our data. Finally, we applied a .step() function to the optimiser that updates the parameters of the gradient.

These steps are crucial in making sure our UNet model converges, with the gradient splitting of the data accordingly without overfitting the model.

```
...
for epoch in range(epochs):
    for step, batch in enumerate(dataloader):
        optimiser.zero_grad()

        t = torch.randint(0, T, (BATCH_SIZE,), device=device).long()
        loss = get_loss(model, batch[0], t, "l1")
        loss.backward()
        optimiser.step()
...
```

4.3 Diffusion Model - UNet Optimisations

The main section of this research is the UNet optimisation. Most optimisations for our UNet required experimental trial and errors to accomplish this. In the following section we aim to understand how we implemented certain changes for the experiments. However, we explain our reasoning and rational for these experimentations in depth in Chapter 5.

We will explain each implementation of our optimisation applied as follows:

- Optimisation - Optimsers

  Our main focus was the optimiser used. In the training section we spoke about how we implemented Adam optimiser when training our UNet model. In our work we implemented two other optimisers called AdaMax and Lion (Evolved Sign Momentum) [2]. In chapter 5 we will explain what learning rates, parameters and other modifications we made to each optimiser to try and fit our UNet model as well as possible with a goal of converging the model at a faster rate. Additionally, a simple explanation of AdaMax and Lion in our results.

- Optimisation - Batch size

  When programming our UNet model we have to keep in mind the amount of data we input into our neural network. We have implemented a variable called BATCH_SIZE in which are dataset is split into batches. We have gone with one main batch size of 128, with two experiential of 256 and a third of 512.

- Optimisation - Image Resizing

  Another optimisation is the image up scaling and down scaling in our UNet model. We can tackle this but editing our up and down channels. We experimented by changing from the original 64↔1024 channels to 32↔1024, 32↔512 as well as 64↔512. Due to the implementation of our UNet being a loop for the up convolutions and down convolutions we had to only change the list of down_channels and up_channels.

- Optimisation - Time Step

  During our investigation into UNet it is important to factor in time steps of the model. To reduce the time step during training it is important to increase the batch size. This was implemented via chaining the batch size variable as discussed previously above.

- Epoch size

  We implemented a variable epoch which we where able to change on experimentation

attempts to optimise our UNet model.

4.4 Diffusion Model - UNet Pruning

Pruning means the trimming away the excess. In machine learning, this means to remove redundant or insignificant elements of the model. In our prerequisites and requirements section 3.1.5 we discussed how regularisation techniques can prevent overfitting as well as help reduce the computational complexity of the model. Maintaining a compressed, optimised model with minimal effects in the model performance. This is where pruning can be used in our UNet.

Overfitting is where the data is forced to statically fit the the test data against the training data. This leds to issues of the model unable to generalise well when new data is added, this could be a change in a new dataset added as input to the UNet model.

Pruning has two different structures, unstructured and structured pruning. In our development we have gone with both. This is so we can compare our results against two prune methods against our original UNet model.

Pruning 1 - Unstructured

Unstructured pruning usually consists of individual low weights being set to zero. This is without any consideration for the structure of the convolutional neural network (CNN) as well as focusing on individual parameters of the convolutional neural network. In the results we expand on the change in weight and bias of our UNet neural network. To implement an unstructured pruning in PyTorch we require to first load our UNet model. Once we accomplished this we where apply to apply the pruning to our model.

In our experimentation we explain the attempts to prune our initial convolutional (conv0), down samples (downs), up samples (ups) and our final output convolutional layer.

Upon selecting a convolutional layer, we select if we would want to prune the bias or the weights. Once both selections have happened, we then pick an amount in which we prune our model.

We implemented two unstructured PyTorch pruning methods l1 and random, in the example below we attempt to prune the initial convolution covn0 however this is universal for other nodes in the UNet model:

```
module = model.conv0
prune.random_unstructured(module, name="weight", amount=0.5)
```

```
prune.l1_unstructured(module, name="bias", amount=0.5)
```

A random unstructured prune does not pay attention to the module, it will attempt to prune the module with the amount specified with no other details impacting the pruned neural network.

A l1 unstructured prune prunes the module based on if the value of a convolution connection is lower than l1-norm. Similar to the l1 loss function the l1 norm is the absolute value of two nodes. This then prunes the neural network accordingly.

Pruning 2 - Structured

Compared to unstructured pruning, structured pruning focuses on removing entire CNN parameters. In our UNet model the original model has around 62,438,883 parameters. In theory unstructured can achieve a higher pruning ratio. However, it will not speed up the UNet as you will have to compute all CNN parameters. Whereas compared to structured pruning it can prune the entire CNN and therefore lower computation calculations in each channel. In structured pruning we can calculate the sparsity of the trade off of the performance optimisation of the model and accuracy. Similarly to unstructured, we implemented one structured PyTorch pruning methods ln structured. ln-structed is similar to l1-norm however the n stands for a number for the type of norm (l1,l2). Where l2 is the square root of the sum of absolute values. This is where we can experiment and change the pruning parameters. We implemented the structured prune for the initial convolution covn0 the following way with l1 norm:

```
module = model.conv0
prune.ln_structured(module, name="weight", amount=0.5, n=1, dim=0)
prune.ln_structured(module, name="bias", amount=0.5, n=2, dim=0)
```

As previously mentioned we can also implement a sparsity percentage checker taken from PyTorch. Sparsity in the amount of elements in a tenors which value is zero, where a tensor is considered sparse if most of the attributed values are zero.

```
print(
    "Sparsity in conv0.weight: {:.2f}%".format(
        100. * float(torch.sum(model.conv0.weight == 0))
        / float(model.conv0.weight.nelement())))
```

This shows a simple implementation of the sparsity calculated weight of the conv0 node to two decimal places.

4.5 Implementing the diffusion model into open source project

Our main approach to this section was to lay out a plan on how we can implement our optimised UNet methods in our diffusion model back into an open source project. The open sourced project we focused on was the original latent diffusion produced by CompVis. This can be accessed at the forked GitHub repository a-lleshi/latent-diffusion as well as in the latent_scr folder of the projects code base. The latent diffusion model is a well established text-to-image model. The project is versatile in that it allows for contributors like ourselves to change the UNet model and parameters of the model accordingly. The following changes presented here today can be applied to the latent diffusion UNet model that was implemented in our own diffusion model:

- Activation function

  In the latent diffusion model the activation function used is a SiLU. This is a similar activation functionality to ReLU. However, SiLU does not have a vanishing gradient problem. In terms of our UNet model we stuck with ReLU and a change in activation to ReLU in the latent diffusion could be experimented with.

- Change of the diffusion model type

  In the script file txt2img.py, CompVis implemented the Denoising Diffusion Implicit Models (DDIM) [24]. However, they also have a redundant Denoising Diffusion Probabilistic Models (DDPM) [7]. We can first implement changes to the script to use the DDPM model. Upon inspection of the DDPM for the latent diffusion it also uses linear beta schedule with the same linear start and end as explained in the original DDPM paper. However, the loss functionality used in CompVis is the l2 function rather than our l1 loss functionality. This can also be implemented into the latent diffusion.

- Change of time embeddings

  In our project we went with a time embedding of 32 in our UNet model.

- Down and Up channels

  In our down and up channels of our UNet we implemented the of options of 64↔1024 channels to 32↔1024, 32↔512 as well as 64↔512. The latent diffusion has an in_channels which we can allow for (32, 64, 128, 512, 1024) and out_channels which can also be (1024, 512, 128, 64, 32)

This plan allows our implementations to be implemented within an open source project and add our own optimisations to these UNet model.

4.6 Implementation of FID Score & scoring with COCO dataset

The implementation of the FID score was in accordance with the equation presented in section 3.2.1. We first calculate the μ_1 and μ_2 as well as Σ_1 and Σ_2 for image one (an image from the dataset) and image two (a generated reversed diffusion image). Then added the implementation of an FID calculator function. In the expremintation we expand on the reasoning why we did not focus on scoring with the COCO dataset as well as not use the FID in our results.

4.7 Testing, verification & validation techniques

It is important to grasp the keywords of testing, verification and validation meaning in the use of machine learning as well as keep in mind the software engineering terminologies.

- Testing:

  In machine learning testing evaluates the performance of a already trained model on a new set of data. As well as testing the software assertion of a correctly generated UNet model.

- Verification:

  The machine learning model meet all requirements and a compelling verification unit test the model will not misbehave under different inputs or changes.

- Validation:

  A technique specifically used in machine learning to test the model during the training phase.

4.7.1 Testing

Within the realm of machine learning, creating a comprehensive test-unit comes with challenges. Testing, specifically in software engineering requires a set of assertions to be true or false with the data input expected to give an specific output depending on the assertion of the test. In software engineering majority of the test-unit are pre-defined such as testing for a login form on

a website. However, in machine learning the issue arises as you would not know what the output of a neural network could produce as a neural networks are mostly a ”black-box”. Although these challenges we can implement simple test units especially for our UNet model testing the input of an image is correct dimensions and other PyTorch tensor prone tests. We can also produce visual tests for a UNet project. When an image is diffused correctly it will produce an output image of a diffused image as well as when reversing the process that image can be seen reversed. These challenge can be over come by the following testing methodologies tailored towards machine learning.

We have created a test unit notebook. This notebook contains code that we implemented for our unit tests. We also test our model with different parameters.

1. For our unit testing we implemented checks as follows:
    - UNet colour channels are correct. This due to the three RGB (red, green, blue) colours.
    - UNet testing for the convolution block was correctly generated.
    - UNet test for the initial, up channels, down channels and output convolutions are all setup.
    - UNet time embeddings.
    - Generating of forward diffused images.
    - Generating of reverse diffused images.
2. Performance testing of our UNet model such as different batch sizes, to make sure our model can handle different dataset workloads.

4.7.2 Verification

Verification is another key concept when it comes to verifying that the software does what it is suppose to do. In machine learning this is important to make sure checks are carried out for the correctness of the model. By implementation of testing different scenarios and test cases for our model. With ML models the neurons are a black-box to the developer, they are not straightforwardly understandable on how certain behaviours arise. Verification methods help to reduce the risk of unexpected behaviour from ML models and ensures their reliability and trustworthiness for production use. This also prevents any future legal issues. However, our UNet model takes in only a dataset of images we have specified. This is due to the abstract

implementation of our project. This means our model will only diffuse and reverse the process of the UNet model without any different conditions and inputs unless we changed the model our self such as epoch runs. Therefore, this means that in text-to-image model verification would be important in the text encoding and image generating rather than the UNet model.

Verification techniques that can be implemented in text-to-image models as a whole are as follows:

1. Text input - it is important to make sure that the model is not used for malicious use and verification for different text prompts are required. Such as illegal characters (,/.?[]) within the text prompt, words that are disturbing, racist, threatening or banned by communities or governing bodies.

2. Image generated - another important verification is to check that the model is not generating any illegal content. Verification for the models output to be run through an AI filter.

A well planed verification coverage testing can help with the model be tested against any misuse or generating illegal content. These verification checks can be implemented by an AI filter which help limit the manual use of developers check and rather have an AI in place to run these checks in place of a developer.

4.7.3 Validation

Our main machine learning methodology to assess our model was validation. This is due to determining whether the model is learning from the training dataset (celebA) and making the correct predictions. To test validation during training we used the L1 loss score. This also helps illustrate if the model is overfitting or underfitting. Without validation testing for our UNet model, it would be difficult to see if the model will perform well on a new dataset.

We implemented the get_loss() functionality as explained in the training of the UNet model to keep track of the L1 loss. We also split the dataset into train and test and concatenate both with PyTorch ConcatDataset to return a tensor transformation of our dataset.

4.8 Software Maintenance

The importance maintaining software is a key part of the software development life cycle (SDLC). This cycle outlines the steps taken to produce software, from planning, analysis,

design, implementation, testing and finally, maintenance. This project involved an AI research based with minimal development and more of a focus on optimisation of UNet model. The maintenance of a software research project needs to include a well laid out plan on keeping up with latest optimisation algorithms.

During development of this project Lion optimiser presented better scores than Adam and AdaMax which was recently released. Therefore, shows a clear indications as researchers and developers in this field it is important to stay in communications with others in the community to keep up to date with the latest state-of-the-art (SOTA) methods and optimiser to able to our models.

Firstly, as developers of the project should keep up to data with deep learning neural networks optimiser. This will allow programmers to widen their knowledge and skills on machine learning as well as be apply to make sure the models do not fall behind in performance.

Secondly, the maintain a machine learning neural network like UNet, is important to constantly train the model. This coupled up with the latest SOTA optimiser will allow the model to keep up with industry models in turn allowing end-users to have access to these model with up to date optimiser and greater training time.

Thirdly, the model should also be maintained by not just training the model but training on a much larger dataset, as training time progresses and the larger dataset would allow the model to learn more precisely and quicker each time as well as allow for the model to be pruned further if required after training too.

In conclusion, software maintenance is an essential aspect of the SDLC, especially for AI research projects too. Keeping up to date with the latest SOTA methods and optimiser is crucial for ensuring that our model remains preformative and competitive. Furthermore, the importance of constant training and using larger datasets to enhance precision and lower the loss over time. By following these practices, this allows us developers to produce high-preforming models that benefit end-users in various industries. Cutting down memory usage and optimising on the hardware available, saving money and in turn allowing for AI model to have a lower entry point economically and become widely available.

Chapter 5

Experimentation, Results & Evaluation

5.1 Experimentation

It is important in any machine learning research to conduct experimentations. This includes changing different weighting values, batch sizes, deep learning optimiser, and other steps and approaches to make sure our UNet model converges in its optimised changes faster without overfitting or underfitting the model. Another experiment that was not in the original plan is the batch jobs time limit, upon investigating a batch job on the server had a limit for 24 hours compared to 48 hours stated on the KCL CREATE HPC website.

In this section we outline different steps and approaches we have taken to tackle optimising and advancing our UNet model.

The following is the approaches we taken during our experiments. We will talk about and how they effected our UNet model.

- Pruning
  - Unstructured
  - Structured
  - Weightings
- Optimisers
  - Adam

  - AdaMax
  - Lion
- Batch sizes
- Epoch
- FID scores

Pruning

Pruning has two main types, structured and unstructured. We have described both in the implementation of this paper. Pruning can be applied to per layer which is local or we can apply over multiple layers which is global. In our experiments we prune the whole model therefore we went for a global option. During pruning using the PyTorch library we had to experiment through different weights and biases for our neural network. Using unstructured we had two path ways, first was random and second was l1. As for structured we also have options of which type such as ln, as per the explanation in our implementation. We also had to experiment which convolutional module we wanted tamper the weighting or bias for. As well as experimenting with both pruning types and selecting the module we also had the options to define our own floating amount in which to adjust the module we selected. This created a major issue of limitless possibles. In our research to keep things simple and understandable we set out a simple experiment of the following:

- Modules to prune:
  1. conv0
  2. ups
  3. downs
  4. output
- Weight float amount of 0.5 for each module.
- Bias float amount of 0.5 for each module.
- Type of pruning:
  1. Structured (ln)
  2. Unstructured (l1, random)

This creates multiple combinations and therefore we narrowed it down to apply all modules pruned to the same amount of weight and bias with one attempt on each type of pruning.

Optimiser(s)

When it came to experimenting with optimisers there was not much leeway with what type of weighting we apply during training of the UNet model. With our Adam and AdaMax optimisers we went with a learning rate of 0.001. In comparison to Lion which was a recently released optimiser, there is and still are many issues with what sort of learning rate or weight decay to use. During reading of Phil Wang implementation of Lion, the example presented was a learning rate of 0.0001 and a weight decay of 0.01.

Batch sizes

We experimented with three types of batch sizes. One was 128 and another was 256 and finally 512. The batch size is important as it helps calculate the amount of steps for each epoch round. The dataloader variable contains exactly 182,732 images, this is due to part of the dataset being loaded as test data. The step sizes are calculated by the dividing the number of images by the batch size. We removed the batch size of 512 and 256 after experimenting with all three, we explained in following results sections for this change.

Epoch

When experimenting with epoch runs, the greater the better. This therefore means we just have to increase the amount of epoch runs during our optimising of our UNet model at training. We used four epoch runs for bench-marking. Additionally, we generated graphs with one epoch run to clearly output our results.

FID Scoring & COCO dataset

One of our experimentation was to score our model using the FID. However, this became a redundant factor for scoring our model. This is due to the model requiring a vast amount of time to run in training. The model with no optimisations would take around one hour and twenty minutes to execute for one epoch run. This was also due to the removal of the COCO dataset replacing with celebA. Also when calculating FID we would of required an original image from the dataset and an output sampled image as a second image. To produce a fully reversed sampled image would of taken the model over 48 hours per run. Therefore,

being infeasible to compare are results off the FID score. The score produced from the FID was massive in compassion to previous FID scores. Scores reached as high as 200 FID during experimentation. Therefore, we removed the benchmarks of focusing on the FID and moved our focus more towards the loss score and convergences of the UNet model.

5.2 Results

When tackling this section we have applied four epoch runs during training to gather our results. This is due to access of 1GB limit of RAM as well as the A100 GPU shared workload with other batch jobs. When we analysed the optimisers we ran for a full four epoch. For other optimisation changes we only run for one epoch. This is due to the convergence decreasing more rapidly and faster towards the final value during the first epoch. We went with this to reduce the wait time of running thought the dataset. This is due to taking 1427 steps with a batch size of 128 alone which is around three hours for four epoch runs.

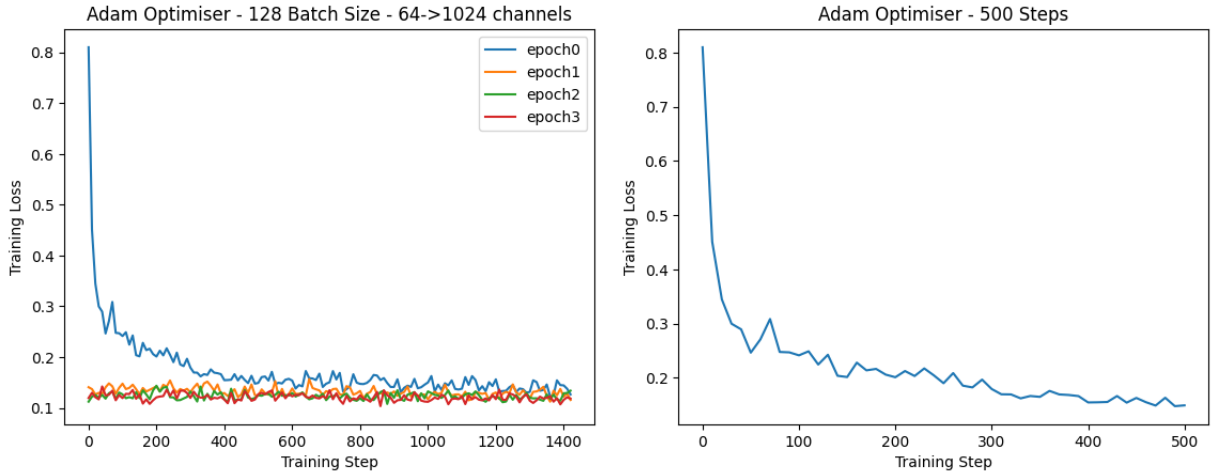
5.2.1 Optimisers

We approached the optimisation problem of which algorithm to use in our project to the results of three algorithms, Adam, AdaMax and Lion. The results produced where ran on four epoch runs with the same dataset and UNet model. We look at the run times between each algorithm and their convergence rates on the first epoch (epoch0) with 500 steps. The UNet model has around 62,438,883 parameters in a 64 to 1024 convolutional UNet neural network with batch size in these results at 128. Our L1 loss results will be explained to 5 decimal places.

Adam

We started our training on the Adam optimiser. The Adam optimiser took around 133 minutes to run four epoch rounds. From our results we gathered that the Adam optimiser converged the best out of the other two algorithms at four epoch runs. This is because in around 50 steps in the first epoch the model dropped from a L1 loss of 0.81008 to 0.24645. Almost a 69.5% decrease in the noise of the image. The more epoch runs we gave Adam optimiser the better it converged towards a loss of 0 as shown in graph b of figure 5.1. Similar AdaMax had the same results, however AdaMax struggled to converge at the end of each epoch compared to Adam. The background research conducted in our literature review already set the benchmark to use Adam optimiser in our training optimisation for our UNet model, with the results solidifying

the use of Adam optimiser.



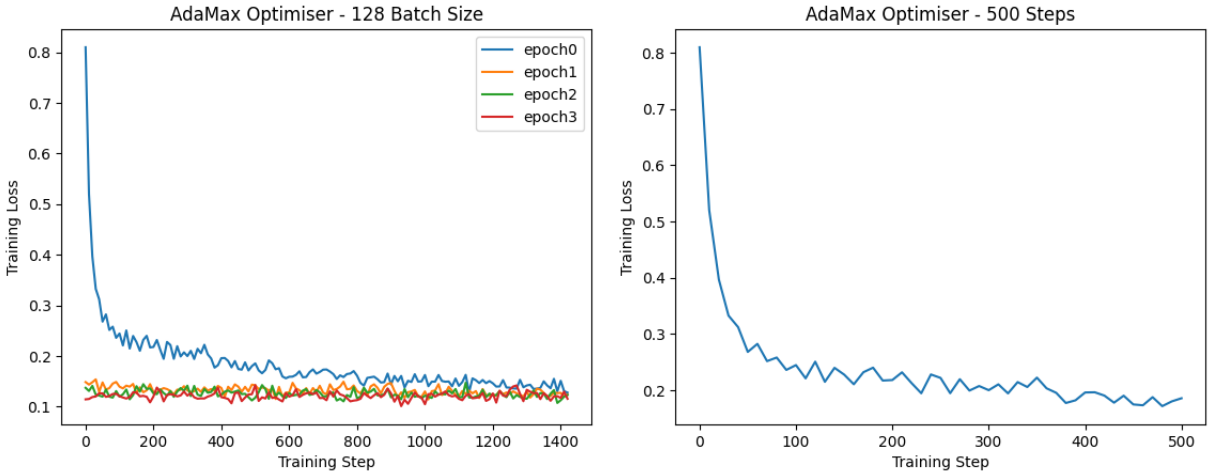
(a) Adam 4 epoch runs on 128 Batch

(b) Adam 500 steps on one epoch run

Figure 5.1: Adam optimiser results

AdaMax

When implementing the Adam optimiser into our work it was also important to add other comparisons. AdaMax was one algorithm that stood out due to the mention in the Adam paper [8]. AdaMax is a variant of Adam in which has been suited for time-variant process such as the UNet. Tho AdaMax had similar diminishing effect with the L1 loss at the start 0.80978 and at 50 steps 0.26778 with a percentage change of 66.9%. Adam had a roughly 3% difference over AdaMax. However, AdaMax did achieve a faster run time of around 131 minutes compared to 133 minutes to Adam, but a 2 minutes difference did not play a major improvement of AdaMax over Adam. Therefore, we stuck with Adam optimiser in our training. The faster run time was also due to running the optimiser late in the evening when the HPC server GPUs had less workload and batch jobs.



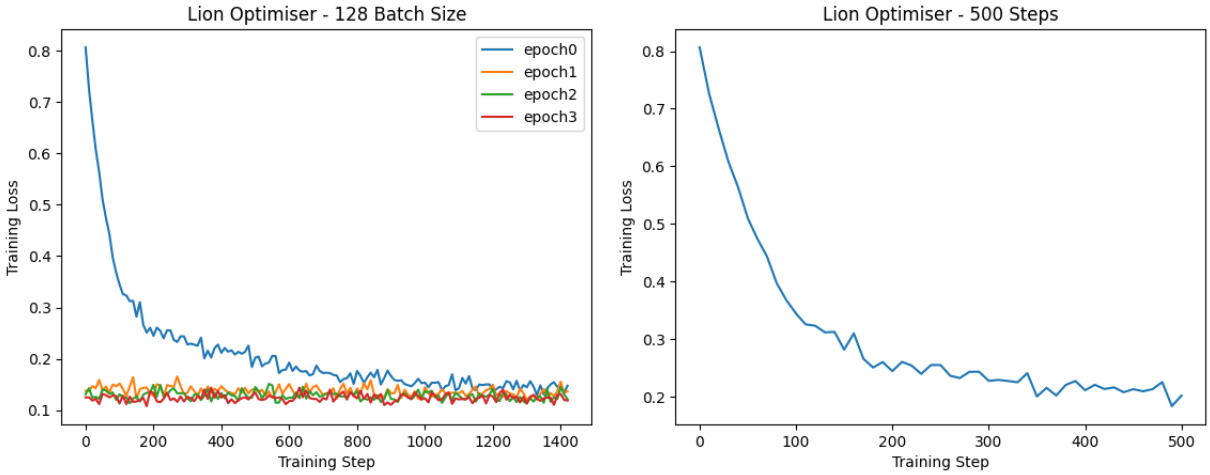
(a) AdaMax 4 epoch runs on 128 Batch

(b) AdaMax 500 steps on one epoch run

Figure 5.2: AdaMax optimiser results

Lion

During our implementation stage, a newly released optimiser was released. The lion optimiser boasts a more ”memory-efficient than Adam as it only keeps track of the momentum” which ”outperforms Adam by achieving a better FID score and reducing the training compute by up to 2.3x” [2]. These results presented by the authors where greatly advantageous and we experimented with applying to our training optimiser. However, we did not achieve the expected results as presented in the paper. In our experimentation the results produced was a worst convergence than both Adam and AdaMax. The main issue is that these new algorithms need a lot of fine tuning of the learning rate and weight decay in which would of taken time to generate and experiment with. As seen in figure 5.3 the model struggles to converge around 350 steps. The L1 loss percentage change was around a 36.7% convergence from the L1 loss of 0.80667 at the start to step 50 of around 0.51025. This shows how the UNet model struggled to fit during training with the lion optimiser. Therefore, we did not investigate the optimisation of the UNet based on the Lion optimiser.



(a) Lion 4 epoch runs on 128 Batch

(b) Lion 500 steps on one epoch run

Figure 5.3: Lion optimiser results

5.2.2 Epoch

In machine learning, the term ”epoch” describes the number of times the full training dataset is fed into the neural network during training. For the neural network to progressively learn and enhance its performance. In our results this proved that the greater the epoch cycle of training the UNet model the better results we got of sampled images generated from the reverse diffusion. At the start of an epoch run the L1 loss is extremely high with a very noisy image, as the training algorithm runs and the epoch increases the more defined the images. The result of this allowed us to see faces being generated from our celebA dataset. This shows how our experiment allowed for us to explore greater epoch runs with the UNet model gradually improved and its ability to identify and segment objects in images, leading to better accuracy and precision. As presented in our figure 5.4 for our results in the success of increased epoch.

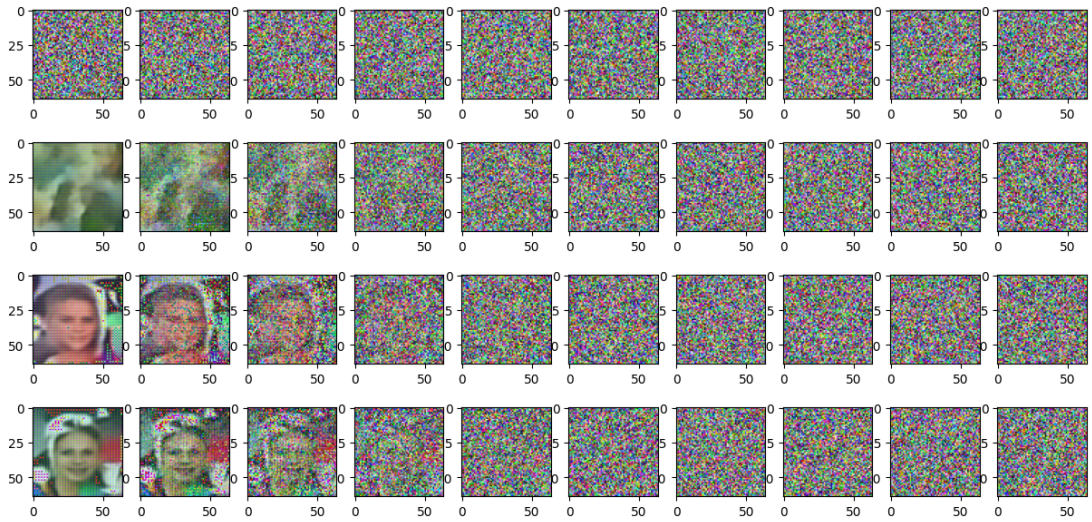


Figure 5.4: The UNet model working in the diffusion model during training of Adam optimiser to generate new faces from CelebA dataset. Row 1 to row 4 equate to epoch 0 to 3. Images produced in epoch round 2 to 3 showing humanly features (face - eyes, nose, hair) and less noisy with more smoother outputs.

5.2.3 Batch Size & Step size

The batch sizes affected the steps for our four epoch runs during training. The original process was to try and increase the batch size to lower the step size and in turn decrease time taken to generate a reversed diffused image. However, results proved this was an incorrect theory. As the batch size was increased so did the time taken for the model to converge. This shows how increasing the batch size could cause issues with a decreased step size of not enough time embeddings and information for the convolutional channels in the UNet. In the table below we see how the time taken has gradually increase over time with 512 a batch size being the largest increase. Upon investigation of these results during pruning we went with a batch size of 128. Another issue could have depended on the workload of the GPU on the server during our runs. To create fair results it is crucial to take into account that the server GPUs are shared across other batch jobs which could of also effected the results. However, due to student only user permission we could not allocate a dedicated private GPU on runs.

Batch size	Step Size	Time Taken (minutes)
128	1427	133
256	713	136
512	356	168

5.2.4 Pruning

Overall, pruning was the most successful optimisation technique applied to our UNet model. This is due to removing unnecessary calculations, matrix multiplications and matrix transpose. As outlined in out experimentation we tested out three types of pruning, two unstructured and one structured. In our results we tested the pruned neural networks on one epoch run. The sparsity for the UNet model convolutional modules where all step to 50% in each run. This implementation helped maintain a fair test for our results. We created two tables to present our result one for unstructured and the other structured. Our metrics for our results was the time taken, L1 loss percentage change and the image produced by each pruning type.

In the unstructured pruned neural network, random outperformed l1-norm. The random unstructured completed one epoch run in about 28 minutes. This alone cut down around 5 minutes of per epoch run compared to a non pruned optimised UNet model trained on Adam. The random unstructured pruned network results started with a L1 loss of 0.80381 and a final L1 loss function of 0.11763, the result of a percentage change of around 85.4% on the first epoch run. The l1 unstructured pruned network did not save much time compared to unpruned models. However, it did generate better quality reversed diffused images in one epoch run as well as results started with a L1 loss of 0.80864 and a final L1 loss function of 0.15123, the result of a percentage change of around 81.3% on the first epoch run.

Pruning - Unstructured —	Sparsity —	Time Taken (minutes) —	% change in L1 loss (%)
Random	50%	28	85.4
L1	50%	33	81.3

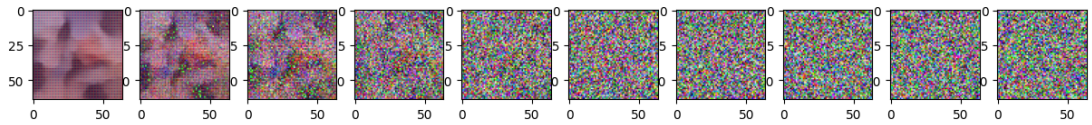


Figure 5.5: Final output of a random unstructured pruned UNet model. Fast convergence, less noisy image. Greater quality.

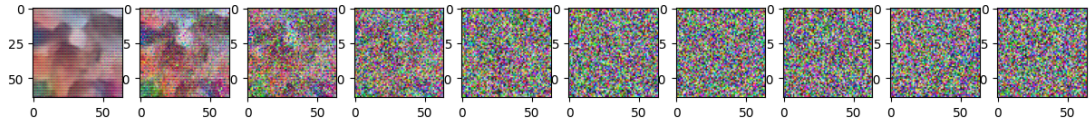


Figure 5.6: Final output of a l1 unstructured pruned UNet model. Similar to random unstructed. Fast convergence, less noisy image. Improved quality.

In our results, structured pruning achieved the worst result. This is because structured

pruning only works on the weight of convolutional blocks. The model in one epoch run struggled to converge to a L1 loss of under 0.5. This results shows one of the worst optimisation technique we added to our UNet model. However, we have to keep in mind we only created a 50% sparsity using the l1 norm of n=1. Future investigation into structured pruning would be need. As per results the structured pruning model started with an L1 loss of 0.80217 and a final L1 loss at the end of the epoch of 0.57778 showing only a 28% change. Almost 3 times as worst as our random unstructured.

Pruning - Structured — Ln	Sparsity — 50%	Time Taken (minutes) — 32	% change in L1 loss (%) 28

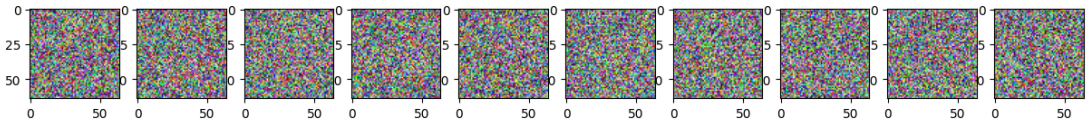


Figure 5.7: Final output of a structured pruned UNet model. Extremely noisy due to the struggle to converge.

5.3 Evaluation & Summary

The main evaluation metrics was the L1 loss score. This helped with evaluating our improvements and optimisations to our UNet model. We where then able to produce percentage changes from the L1 loss score. The higher the percentage the better the indication of the convergence of the model with respect to the optimisation algorithm fitting towards a lower score to 0.

Overall, we have a successful optimised our machine learning UNet model. The presented material evaluation of these results shows the importance of experimenting in machine learning to achieve optimal convergence. We have shown the importance of steps in our training in regards to batch size, the use of optimisations algorithms with regards to the benefits or drawbacks of Adam, AdaMax and Lion, the increase of epoch runs and pruning methodologies. These optimisations to neural networks are not a straight plug and play with the PyTorch library but a constant experiment of trail and error to for us to have achieved our desirable optimisation results. Some results in our UNet optimisations did not always go as planned and unexpected results arise. Therefore, it is key to understand what went wrong and attempt a new test case with different parameters such as a different learning rate. This has helped achieve our goal of optimising and advancing the UNet model.

Chapter 6

Legal, Social, Ethical and Professional Issues

The growth in generative AI models, particularly in the field of text-to-image, has attracted a lot of attention in the last year. Although these models have a wide range of applications, such Dall-E 2, they also pose a number of important legal, social, ethical, and professional issues. This section of the report will encompass the drawbacks of these generative AI models. They have been established by governing bodies such as the British Computer Society (BCS), who outline the Code of Conduct and Code of Good Practice to ensure their members total compliance. Another established governing body that helps with legal, social, and ethical issues is the Information Commissioner's Office (ICO) whose mission is "to help explain the processes, services, and decisions delivered or assisted by AI to the individuals affected by them (consumer)" [12]. These issues affect AI systems as a whole, as they are "becoming a part of everyday life. The key is to ensure that these machines are aligned with human intentions and values." (CTO at OpenAI, Mira Murati) [13].

6.1 Professional Issues

When conducting a machine learning project within the scope of an academic paper, members of the BCS need to take into consideration the BCS code of conduct and good practice. In the next section we outline the code of conduct and how this helps members maintain professional and well-evaluated practices.

6.2 British Computer Society (BCS)

The British Computer Society (BCS) is a charity with a royal charter. Their agenda is to lead the IT industry through its ethical challenges, support people within the IT industry and make IT good for society[23]. They are a professional body with the responsibility to set out rules and professional standards for their members. This section will outline their code of conduct and good practice, thus demonstrating how such governance bodies affect developers working on text-to-image models and the end-user.

6.2.1 BCS - Code of Conduct

The BCS code of conduct that contains four principles:

1. Public Interest
2. Professional Competence and Integrity
3. Duty to Relevant Authority
4. Duty to the Profession

These four principles are ”observed by every BCS member”, which shows the commitment to working within the public interest and accepting the developer's professional duties. In this section, the code of conduct will be linked to the scope of text-to-image and generative AI models.[23]

Public Interest

As society strives to become less reliant on manual labour and more reliant on technology, it is important to make sure everyone has access to IT. The following points outline the adherence to public interest:

One point is having due regard for the public health, privacy, security, and wellbeing of others and the environment. Text-to-image models can generate detailed and precise images from written descriptions, but they can also be abused and cause harm to their users. For instance, they might be used to fabricate unrealistic pictures that are deceptive or negatively impact the health and safety of the general population. For example, they may include images objectifying females, child pornography and gore images. Hence, it is crucial to make sure text-to-image models are created and applied responsibly, with limitations to generating images, taking into account issues with public health, privacy, security, and the environment. Another

solution is applying appropriate safety precautions, such as AI filters. They should be applied to the models to provide information and warnings about dangerous material or situations.

Another objective is to have due regard for the legitimate rights of third parties. Images of specific people or groups of people may be produced using text-to-image models. It is important to respect their legitimate rights, such as their right to privacy and dignity. This includes obtaining consent before using any personal information or images for training and ensuring that the images are not used in a defamatory or harmful manner toward the people depicted. Furthermore, models should be trained on diverse datasets representing various groups to avoid biases or unfair portrayals of specific individuals or groups.

As members of the BCS, all members of the BCS must conduct their professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age, disability, or any other condition or requirement. Regardless of the personal qualities or traits of the individuals involved, text-to-image models should be created and applied in a fair and impartial manner. This includes ensuring that the models are not biased towards any particular gender, race, religion, or other characteristic. Developers should also be aware of potential biases in the data used to train the models and take steps to correct them. Solutions to such issues can also be guided by open source communities that produce datasets to remove any possible bias within their datasets due to the fact that the majority of projects tend to lean toward readily available, free-to-use data.

Finally, to promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society, text-to-image model techniques have the potential to help people and groups in a number of ways. They vary from assisting those with disabilities to producing more readable information. This will guarantee that everyone has access to the advantages of modern technology. Among other things, this entails making sure that the models are created to be used by everyone, independent of their abilities or other traits. Also, initiatives should be taken to make sure that technology is applied in a way that fosters diversity and inclusion. This could be done by allowing access to a free version or open source version of our model.

Professional Competence and Integrity

The BCS makes sure members follow a level of integrity and competence when attempting a computer science based project. They take into account that humans do not know everything. Therefore, to complete tasks successfully, a member of the BCS should be constantly learning and growing their skills.

Firstly, a developer should only undertake to do work, or provide a service that is within their professional competence. Since text-to-image models demand a high level of technical skill and knowledge. Therefore, it is crucial to make sure that developers only take on work that is within their area of expertise. This makes it easier to guarantee that the models are created correctly and can be used in a safe and efficient manner. Also, developers should not assert any level of expertise that they do not have, as this could result in inaccurate or mistaken information being included in the final product. This asserted rule does not allow BCS members to claim any level of competence that they do not possess. This is crucial for the project as it allows the expansion of skills and knowledge within the realm of machine learning. As technology is advancing, each member of the BCS should develop their professional knowledge, skills, and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to their field. Considering text-to-image technology is mostly new in development, developers must update themselves on a regular basis. Maintaining awareness of new technologies, standards, and guidelines that are relevant to text-to-image models is part of the project. For developers to be able to build and maintain reliable and secure models, they must constantly acquire knowledge, which in turn will help them grow as developers.

Secondly, these members need to ensure that they have the knowledge of legislation and must comply with it in carrying out their professional responsibilities. Text-to-image model developers are required to grasp the law, abide by it when doing their professional duties. This involves making sure the models follow all applicable legal requirements on data protection, integrity, and ethics.

Thirdly, all members respect and value alternative viewpoints and accept criticism of their work. Within the realm of the text-to-image model, developers must appreciate and value opposing opinions as well as look for, accept, and provide sincere critique of their work. This could include new coding practices such as a specific way to implement an algorithm or keeping outdated implementations up to date and adjusting models accordingly. Also, they must refrain from acting dishonestly, maliciously, or carelessly in a way that endangers others, their property, their reputation, or their employment. This involves making sure the models are not applied to disparage or hurt specific people or groups.

Finally, one must reject and not offer any bribery or unethical inducement. Any such offer must be rejected by those who create text-to-image models. As a result, it is possible to guarantee that the models are developed and used in a way that is morally and justly beneficial

to society as a whole.

Duty To Relevant Authority

Acting members of the BCS need to respect the organisations or individuals that they work for. This is a key element for members to make sure they act in the client's or company's best interests, as well as taking personal or collective responsibility for actions while upholding ethical standards. Members need to maintain a duty to relevant authorities.

A member should carry out professional responsibilities with due care and diligence in accordance with the relevant authority's requirements, making professional judgements at all times during the project. During individual projects, it is important that one make professional judgements when conducting research into text-to-image models. Supervisors have clear specifications on the produced final work and these should be kept in mind. This is due to these projects having a set goal to achieve during development, i.e., optimising of UNet. A developer should stick to the requirements and specifications as well as add inputs into specific issues that could arise during programming and make a correct judgement of how to solve them, such as computing power or memory resource limitations to complete a task. Programmers should make sure they are not using any unauthorised code they do not have permission for, avoiding conflict of interest between themselves and their supervisor or authority bodies. None adherence to these regulations could result in a personal gain or third-party benefits that were not originally authorised. Therefore, members should act within their professional responsibilities and seek to get authorisations, if individuals want to capitalise on projects and sell services related to these text-to-image models to users.

Members should accept the authority of the work of colleagues they supervised. This applies to group projects within the BCS framework, which is not something an individual computer science based project encompasses. However, supervisors also need to comply with the BCS members' code of conduct. Therefore, it is important that the supervisor(s) make sure their supervisee(s) are meeting legal and ethical standards for projects to be safe and effective in their tasks for the end users.

Finally, it is crucial that a member does not withhold information on the performance of the product, system, or service (unless lawfully bound by a duty of confidentiality). This is essential in research-based projects into generative AI, as results may not go as anticipated in the background and goals of the project. In effect, this may lead to the member(s) taking advantage of the lack of knowledge or inexperienced readers and end users because the materials

were not published in accordance with the BCS duty to professionalism and good practices.

Duty To The Profession

It is also important that members of the BCS uphold responsibility and a high reputation of the information technology (IT) industry. This is to make sure that it does not jeopardise the profession and industry. In text-to-image models this is crucial as it is a new and evolving technology with media outlets readily to pounce on any critic of generative AI. This is important for members to make sure they are professional and conduct themselves at a high standard through their developments. It is essential to ensue that members' reputation is not smeared and the reputation and regulations of the BCS standards are respected. Members should seek to support other fellow members in a professional environment without hindering the development of these text-to-image models, and informatics as a whole.

6.3 Legal Issues

Legal issues, which have emerged as a major concern as the use of generative AI models are becoming more common. Copyright infringement, data protection, privacy, and ownership, in particular, are critical legal issues that have been identified. Legal teams must carefully create terms and conditions in order to prevent legal ramifications and protect individuals' rights within accordance to the law and regulation of AI.

6.3.1 Legal Issues - Copyright & Ownership

Copyright - protects work and stops others from using it without the permission of developers or end-user generating images [21]. Copyright regulations are at the forefront of owners claiming their work on unique images. Although text-to-image models are AI generated, this has led to significant copyright-related legal problems. In recent times, the US Copyright Office ruled that "human authorship" and "any creative contribution from a human actor" [25] during a lawsuit, Thaler v. Perlmutter, are prerequisites for copyright protection. Therefore, the US Copyright Office refuses to grant any copyright protection for AI generated images. In spite of this, corporations are still getting into legal problems on the grounds of infringement on intellectual property (IP) encompassing copyright.

The primary issue with IP and copyright claims is the dataset on which these AI models were trained. These text-to-image models use a vast amount of data from all parts of internet

without developers proper checks for the licensing of images used within their dataset to train their model. As developers, it is important not to claim work that is not ours. This also means that in order to properly check licensing and usage, the collected data must be allowed to be used by the teams, developers, or corporations developing text-to-image AI models. To reduce legal problems, it is important that training data is filtered without any IP infringing content. A solution would be to have open source communities to combat any copyright images within the datasets be removed. This would reduce the risk of lawsuits against companies that heavily rely on open-source datasets like COCO.

Another problem in tackling copyright is the minimal control that developers and companies have over what text is typed into the prompt of the generator by end-users. Such cases have caused controversy at corporations with regard to who takes responsibility for images produced with IP and copyright protections. For example, the Nike Air Force 1 shoe silhouette and design are trademarked and as IP of Nike LLC. If an end-user generates a text prompt of "red Nike Air Force 1s", this could cause a lawsuit against the AI model as well as the consumers of these AI generation products. Companies that deflect responsibility have stated in their Terms of Service (TOS) that they leave it to the users to exercise "responsibility" in using or distributing the images they generate. This is a short-term solution for companies. However, it is an unethical TOS to put consumers through. A solution would be to prompt warnings using another AI text model to filter any possibilities of copyright infringement claims and have users confirm if they would like to continue.

6.3.2 Legal Issues - Privacy & Data Protection

Privacy and data protection enable companies to store, update, change and delete any identifiable information on a user. When training AI models such text-to-image datasets are required. These datasets may contain current personal images, outdated images and images that have been taken down from the internet. These models are trained to retain their inputted data, in spite of any changes. This raises concern regarding Data Protection Act[22] which ensures that the user's personal data and privacy are protected.

With regards to image datasets, they contain from millions up to billions of images including captions such as a name tag. This makes it impractical for a human to keep track of law binding requirements. One of the elements of the Data Protection Act is to keep "accurate and, where necessary, up to date" data. This data could include a change of name for someone within the dataset. One solution to combat changes within the dataset is to create applications such as

a database which can be frequently queried, updated and deleted the dataset. In regards to privacy, this allows end-users to be able to remove or update any of their information.

Another point is that these text-to-image models can be misused. The Data Protection Act states that "protection against unlawful or unauthorised processing, access, loss, destruction or damage"[22] to data should be enforced. This is due to issues such as deepfakes. Deepfakes occur where personal data is used to create alterations which are false and used for malicious purposes or to spread misinformation. For example, recently text-to-image model, MidJourney 5, was used to create deep-fakes of Donald Trump being arrested 6.1. This raises concerns regarding how companies and developers can minimise abuse of their systems with users' personal data. A solution would be to require a sign up page which will allow corporations to identify any misuse of their model. Another method applied by developers of MidJourney 5 was to blacklist words such as "arrested" which resulted in removal of any further image generated from these blacklisted keywords. However, a common ground between developers and users would be required to help genuine end-users and customers. This would allow actual genuine image generation to be created without users being prevented by a massive blacklist of words. This will ensure making the model usable.

Figure 6.1: MidJourney 5 - Donald Trump arrest text-to-image generated by Twitter user (@EliotHiggins)

Overall, the UK Data Protection Act ensures that a consumers privacy is kept well regulated and their information is not being misused. These governing bodies help lay foundations in which end-users and developers can agree on helping one another to improve the AI text-to-

image models in a safe and legal manner.

6.4 Social Issues

Although generative AI technology progresses and becomes more widely used, social issues concerning its application have emerged. Social issues relate to informed consent, objectification and public perception that text-to-image models impact on society. AI developers must carefully consider minimising these issues. This will ensure that they are not causing future social, cultural, economic damage and harm, which could lead to legal action taken against developers or corporations.

6.4.1 Social Issues - Informed Consent

End-users should be informed how their data will be used. User data consists of datasets, their personal images and text inputs to these models to be trained. This should be clearly laid out for users allowing them to be able to know how the data will be used, who will have access to it and the potential consequences of participating within training generative AI models. Informed consent is particularly important with vulnerable groups, such as children, which may be too young to understand. Therefore, additional safeguard measurements must be put into place to ensure their rights are protected and their guardians are well informed.

A solution to this would be to have clear guidelines for developers to follow and to employ legal teams when creating terms and conditions for customers. This helps society as a whole as it would help the general public or end-users know how this technology would be used. This in turn helps build crucial trust between AI developers with the public, creating a foundation of legitimacy. The trust of the general public (users) in these models would allow developers to train them on more creative text prompts and images for future use.

6.4.2 Social Issues - Objectification

The use of text-to-image generative AI technology has lead to synthetic, realistic images of humans, raising concerns about objectification. Objectification is the reduction of an individual to their physical appearance, treating them as an object rather than as a person. Objectification has become a massive concern over the years. This is due to increase of social media users. Who edit their photos in applications such as photoshop, now with even more powerful tools such as text-to-image models.

Objectification in recent times has had a significant negative impact on individuals and society as a whole, changing social ideologies and shaping what a ”perfect” body image should be. This has impact on an individual's self-esteem, and led to a range of mental health problems. This creates a knock on effect in society. These affected individuals would require psychological support which can impact health services in nations. Therefore, it is critical models are inclusive. Having a wide variety of different body shapes, races, age, genders. At OpenAI to tackle this in DallE-2, they successfully reduced bias and improved safety within DallE-2. They achieved this by improving their AI content filters to filter images used in the dataset that violate content policy. After mitigation of these adjustments it allowed for broader diversity such as age, race and gender without impacting creative expression.

6.4.3 Social Issues - Public Perception

AI developers should engage in transparency with their ethical practices towards their users. This due to generative AI in recently news having a negative public perception. This perception of the public having their personal images used to train text-to-image models and create new AI images could be an invasion of their privacy as discussed in legal issues. This also applies to exploitation by larger corporations to capitalise on charging for the usage of these models. This could lead to lasting consequences for the industry and the public as a whole.

Addressing concerns of the public should be at the forefront of every text-to-image communities, developers and corporations. To make sure society is not taken massively back by developments in such technology, it is key for governing bodies to address public concerns with industry leaders. In addition, taking such steps with stakeholders in the industry will allow AI developers to continue building and optimising text-to-image models with the public's trust. This is to tackle changing the perception of text-to-image models to a more fruitful conversation and positive impacts on society.

6.5 Ethical Issues

Ethical concerns about the application of text-to-image have become more pressing. Fairness, responsibility, accountability, and transparency are critical ethical considerations that AI developers must closely address to ensure that their technology is applied in an ethical and responsible manner.

6.5.1 Ethical Issues - Fairness & Transparency

Fairness within ethics is a quality of making a moral judgement free from discrimination. In text-to-image models developers, corporations and end-users should all feel equally important and their opinions should be fairly treated. Programmers should be allowed to explore new developments within these models and be transparent in their development in these organisations. Programmers would benefit from being equally treated and their opinions valued producing an environment which is ethical and harmonious. End-users should also benefit from a transparent company where channels of communications between all involved parties are clear and ethically driven. This ensures text-to-image models' ethics.

6.5.2 Ethical Issues - Accountability & Responsibility

When developing text-to-image models, it is important that involved shareholders, programmers and users are responsible for their actions. This ensures that the correct parties are held accountable if difficulties occur. Companies should maintain a level of ethical standards when dealing with programmers and users. These standards ensure that programmers are responsible for any issues faced with these models. At the same time companies should provide support and guidance for development in these changes to models. Such ethical standards should also help users understand they are responsible for the text prompt typed into these models. This ensures that all parties involved are cooperating in an ethical manner.

Ultimately, all legal, social, ethical and professional issues should be well considered when trying to optimise text-to-image models' components such as the UNet, as we have presented in this paper. This is important to make sure developers like ourselves maintain ethical standards, abide by governing bodies to remain correct with the legal issues and the law in turn allowing for society, stakeholders and companies to flourish in the field of generative AI.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This research paper has explores the optimisations of one component of text-to-image models, the diffusion model. We examined the use of the UNet within the diffusion model. Moreover, the optimisations and advancements we proposed in our work included architectural changes, batch sizes of the dataset, optimisation algorithms (Adam, AdaMax, Lion) for machine learning models, up and down sampling convolutional layer changes, and finally pruning the convolutional neural network layers. These changes helped us to generate a faster convergence slope for our loss results. This in turn demonstrated how our changes then could be planned to be added into the latent diffusion open source project. In our original project, we anticipated the achievement of a lower FID score. However, during optimisation training, it became evident that it was not possible to use the FID score. Therefore, we made a decision to use the L1 loss score for our evaluation metrics and results. These stepping stones in optimisation of the UNet have proved beneficial. They will help society, stakeholders, end users and developers to use text-to-image models with greater considerations of legal, social, professional and ethical issues. The UNet advancements and optimisations presented in this paper have helped to save time during batch runs. Consequently, the UNet model converged faster, thus significantly improving generative text-to-image AI models.

7.2 Future Work

Text-to-image models are constantly changing to a more optimised and efficient models. In our investigation, we presented optimisations for a key component the UNet.

In future work, we will be continuing with our investigation into text-to-image models by focusing on optimisation of other modules. These modules include encoders and decoders of these models. The encoder encompasses the text tokeniser, which can be used to generate tokens for the text prompt. This work can be carried out in a Masters degree and a more focused approach can be adopted in a PhD.

Apart from optimising new modules, it is important to focus on training our previous UNet model. We can achieve this by deploying our training algorithm for longer epoch runs than previously experimented with in our study. This will require greater computational time, and longer batch job runs. It can be done by applying for dedicated Graphic Processing Units (GPUs) on KCL's HPC servers. This will result in a better, well trained UNet model.

Additionally, a final proposed study can be applied to discover optimal learning rates and weight decay for our optimisation algorithms to converge our UNet model faster than previously achieved. This would require a well laid out experiment on different learning rates and weight decay for optimisation algorithms such as Adam.

Finally, the proposed future work can be carried out by ourselves to continuously engage with the open source community to benefit advancements into text-to-image models.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[2] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms, 2023.

[3] e Research KCL. King's e-research website. Accessed: 05-03-2023.

[4] Pius Kwao Gadosey, Yujian Li, Enock Adjei Agyekum, Ting Zhang, Zhaoying Liu, Peter T. Yamak, and Firdaous Essaf. Sd-unet: Stripping down u-net for segmentation of biomedical images on platforms with low computational budgets. *Diagnostics*, 10(2):110, 2020.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[6] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets, 2018.

[10] Tsung-Yi Lin, Matteo R. Ronchi Genevieve Patterson, Michael Maire Yin Cui, Lubomir Bourdev Serge Belongie, James Hays Ross Girshick, Deva Ramanan Pietro Perona, and Piotr Dollár Larry Zitnick. Coco dataset. Accessed on 02.12.2022.

[11] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[12] Information Commissioner's Office. Explaining decisions made with Ai, 2023. Accessed: 18-03-2023.

[13] OpenAI. Safety & Responsibility website. Accessed: 19-03-2023.

[14] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[15] Kashif Rasul and Niels Rogge. The annotated diffusion model. Accessed: 03-03-2023.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 5 minute teaser presentation of the u-net. Accessed on 05.12.2022.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[18] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Sara Mahdavi, Rapha Gontijo Lopes, and et al. Imagen. Accessed on 15.12.2022.

[19] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S Sara Mahdavi, Rapha Gontijo Lopes, et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

[20] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. LAION-400M: open dataset of clip-filtered 400 million image-text pairs. *CoRR*, abs/2111.02114, 2021.

[21] Government Digital Service. How copyright protects your work, Nov 2015. Accessed: 22-03-2023.

[22] Government Digital Service. Data protection, Sep 2015. Accessed: 23-03-2023.

[23] BRITISH COMPUTER SOCIETY. The Chartered Institute for IT - About Us. Accessed: 13-03-2023.

[24] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[25] the united states copyright office. Thaler v. perlmutter, Jan 2023. Case 1:22-cv-01564-BAH.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.