Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Artificial Intelligence

Evaluating Learned Representations of Denoising Diffusion Models for Predictive Tasks

Master's Thesis

Author
Márton Gergely Herkules

Advisor
Dr. Bálint Gyires-Tóth
András Kalapos

December 14, 2024

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Herkules Márton Gergely*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2024. december 14.

Herkules Márton Gergely
hallgató

Kivonat

A generatív modellezés a gépi tanulás egyre fontosabb ágává válik, és kulcsszerepet játszik a nagy nyelvi modellek működésében, amelyek mára igen széles körű alkalmazhatósággal rendelkeznek. Egy fontos dilemma, amely a terület eredetéig nyúlik vissza és ma is fennáll, a generatív és a prediktív modellek által tanult belső reprezentációk közötti különbség és ezek különböző feladatokra való alkalmazhatósága. A zajszűrő diffúziós valószínűségi modellek olyan új architektúraként jelentek meg, amelyek messze felülmúlják a korábbi generatív megközelítéseket részletesség, minőség és változatosság tekintetében. Képesek részletes és valósághű képeket generálni, ami arra utal, hogy a vizuális adatokról és a világ ábrázolásáról rendkívül részletes és pontos reprezentációval rendelkeznek.

A belső reprezentációk kihasználása prediktív feladatokhoz új lehetőséget nyújthat a gépi tanulási modellek vizuális adatokon történő előtanításához, ami felveti azt az érdekes kérdést, hogy ezek a modellek alkalmazhatók-e képgeneráláson túlmenő feladatokra is, például képosztályozásra. Ha ezek a modellek alkalmasak ilyen célokra, akkor vajon a teljesítményük elég jó-e ahhoz, hogy versenybe szálljanak a meglévő osztályozási modellekkel, amelyek kifejezetten erre a feladatra lettek létrehozva?

Munkám során egy Denoising Diffusion Probabilistic Model-t (DDPM) (Zajszűrő Diffúziós Valószínűségi Modell) implementálok, amelyet három különböző adathalmazon tanítok. Ezt az alap modellt kiértékelem a megfelelő metrikákkal, hogy teszteljem a generatív képességeit.

A modell a tanítás során látott képek jellemzőit tanulja meg, hogy felhasználhassa azokat későbbi képgenerálási feladatokra. A modell képességeit egy diszkriminatív, osztályozási feladatonvizsgálom, amihez szükséges ezeket a jellemzőket a modell különböző rétegei által generált látens terekből kinyernem, hogy később lineáris és k-legközelebbi szomszéd (KNN) osztályozás bemeneteként használhassam.

Abstract

Generative modeling is an increasingly important branch of machine learning, serving as a key component in training large language models, which now have a wide range of applications. An important dilemma that dates back to the origins of the field and still persists today is the difference of the internal representations learned by generative and predictive models and their applicability to different tasks. Denoising diffusion models emerged as a method far surpassing previous generative approaches in terms of detail, quality and variety. Their ability to generate detailed and realistic images suggests that they have highly detailed and accurate representations of the visual data used to train them and the world they depict.

Exploiting internal representations for predictive tasks could offer a new paradigm for pre-training machine learning models on visual data, raising the interesting question of whether these models can also be applied to tasks beyond image generation, like image classification. If these models are suitable for such purposes, the next consideration is whether their performance is good enough to compete with existing classification models, which are specifically designed for this task.

In my work, I implement a Denoising Diffusion Probabilistic Model (DDPM), which I train on three different datasets. I evaluate this base model using appropriate metrics to test its generative capabilities.

The model learns the features of the images it has seen during training so that it can use them for future image generation tasks. I examin the model's capabilities on a discriminative classification task, for which it is necessary to extract these features from the latent spaces generated by the different layers of the model, so that they can later be used as inputs for linear and K-Nearest Neighbor (KNN) classification.

Chapter 1

Introduction

Today's widespread image-generating models are available to almost anyone, with some models capable of creating realistic images in seconds. In the recent past denoising diffusion models emerged as a method far surpassing previous generative approaches in terms of detail, quality and variety. Their ability to generate detailed and realistic images suggests that they have highly detailed and accurate representations of the visual data used to train them and the world they depict.

Exploiting this internal representations for predictive tasks could offer a new paradigm for pre-training machine learning models on visual data, raising the interesting question of whether these models can also be applied to tasks beyond image generation, like image classification. If these models are suitable for such purposes, the next consideration is whether their performance is good enough to compete with existing classification models, which are specifically designed for this task.

If these models are capable of achieving similar performance, it means that there would no longer be a need to label thousands or even millions of images to train large classification models, saving a significant amount of money, time, and resources.

The goal is to create and train an unconditional denoising diffusion probability model on three different datasets and test its image generation capability. Additionally, I analyze the latent representations of input images computed by various internal layers of the model. I investigate whether these representations capture sufficient semantic information for discriminative tasks by using them as inputs to classification tasks.

During my work I train an unconditional denoising diffusion probability model on the CIFAR-10, Flower 102 and STL-10 datasets without using their labels. Training the model without labels on the denoising task can be considered as a self-supervised learning method. To evaluate the descriptiveness of the latent representations, I

test the ability of simple classifiers to distinguish between classes based solely on these representations. In practice, I connect a simple linear classifier or K-Nearest Neighbor (KNN) classifier to the outputs of specific model layers, train these using supervised learning, and then evaluate their accuracy. Using such classifiers is common in self-supervised learning literature as it tests the model's ability to extract meaningful features from input images. These classifiers are also referred to as linear and k-NN probes. In summary, I evaluate both the generative capabilities of the original model and the abilities of the newly created classification models.

The following chapters describe the technologies, tools and environments used. I briefly describe the datasets used, the structure of the models, and how they are trained. In chapter 7, I describe the model evaluation methods and then analyze the results and finally, I will summarise the results.

Chapter 2

Literature review

2.1 Neural network

A neural network is a type of machine learning model that simulates the structure and functions of the human brain. This artificial way is composed of interconnected nodes or artificial neurons that help learn and process data. The neural network consists of layers of nodes, including the input layer, one or more hidden layers, and the output layer as shown on the figure 2.1. Every node in the network is connected to the others and has a weight and threshold of its own. Each node makes a decision based on the received input values and passes it on to the corresponding nodes of the next layer. This process enables the network to learn from training data, identify patterns, make decisions, and solve complex problems over time. Neural networks have a wide range of applications, including image recognition, speech recognition, and natural language processing, and are an essential component of deep learning models in artificial intelligence.

2.2 Convolutional neural network

The convolutional layer is the core building block of a convolutional neural network (CNN) [4], [12]. It requires a few components: input data, a filter, and a feature map. Let's assume that the input is a color image made up of a matrix of pixels in 3D. This means the input has three dimensions—height, width, and depth—corresponding to an image's RGB. This solution has a feature detector, also known as a kernel or a filter, which moves across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

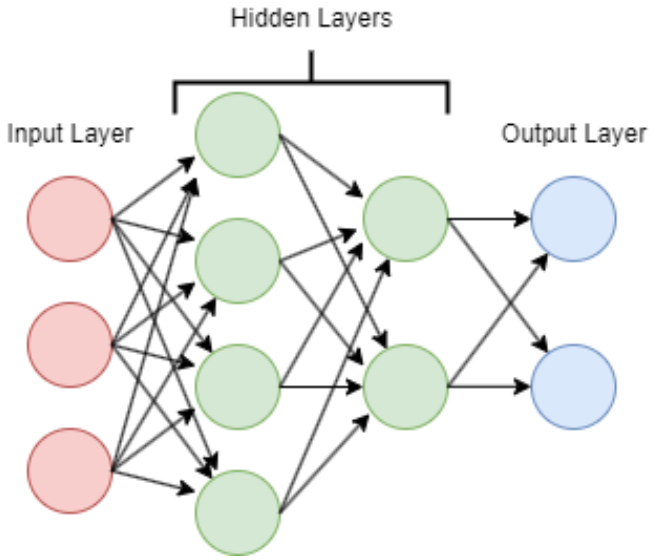

Figure 2.1: Basic neural network

The feature detector is a two-dimensional (2-D) array of weights representing part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the size of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. Afterward, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image as shown at figure 2.2. The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or convolved feature. The number of filters affects the depth of the output. For example, three distinct filters would yield three different feature maps, creating a depth of three.

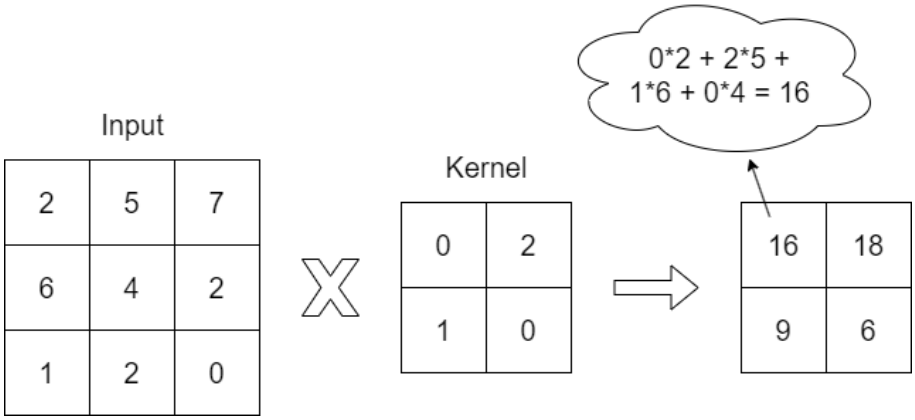

Figure 2.2: Convolution process

The weights in the feature detector remain fixed as it moves across the image, also known as parameter sharing. Through the process of backpropagation and gradient descent, certain parameters, such as weight values, are adjusted during training.

Another convolution layer can follow the initial convolution layers. When this happens, the structure of the CNN can become hierarchical as the later layers can see the pixels within the receptive fields of prior layers. Each layer of a given network can recognize different patterns in the images, creating a feature hierarchy within the CNN. Lower-level layers can recognize lines and simpler patterns. As we move up the hierarchy, the architecture recognizes more and more complex forms with the help of previously processed lower-level properties, until it is able to recognize the desired object. Ultimately, the convolutional layer converts the image into numerical values, allowing the neural network to interpret and extract relevant patterns.

2.3 U-net

U-Net [15] is a widely used deep learning architecture that was first introduced in the "U-Net: Convolutional Networks for Biomedical Image Segmentation" [14] paper. The primary purpose of this architecture was to address the challenge of limited annotated data in the medical field.

U-Net's architecture is unique in consisting of a contracting path (encoder) and an expansive path (decoder) as shown at figure 2.3. The contracting path contains encoder layers that capture contextual information and reduce the spatial resolution of the input. In contrast, the expansive path contains decoder layers that decode the encoded data and use the information from the contracting path via skip connections to generate a segmentation map.

The contracting path in U-Net is responsible for identifying the relevant features in the input image. The encoder layers perform convolutional operations that reduce the spatial resolution of the feature maps while increasing their depth, thereby capturing increasingly abstract representations of the input. This contracting path is similar to the feedforward layers in other convolutional neural networks. On the other hand, the expansive path works on decoding the encoded data and locating the features while maintaining the spatial resolution of the input. The decoder layers in the expansive path upsample the feature maps, while also performing convolutional operations. The skip connections from the contracting path help preserve the spatial information lost in the contracting path, which helps the decoder layers locate the features more accurately.

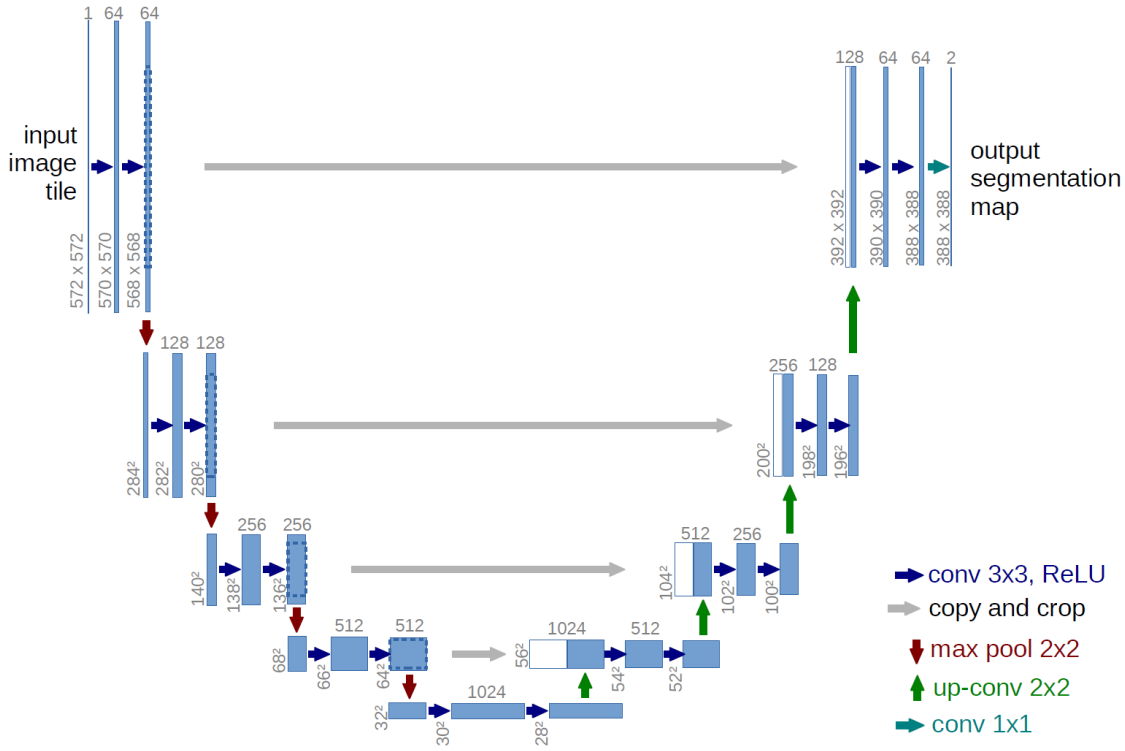

Figure 2.3: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [15]

2.4 Attention

An attention mechanism [18], [11] is a neural network architecture that allows the model to focus on specific sections of the input while executing a task. It dynamically assigns weights to different elements in the input, indicating their relative importance or relevance. By incorporating attention, the model can selectively attend to and process the most relevant information, capturing dependencies and relationships within the data. This mechanism is particularly valuable in tasks involving sequential or structured data, such as natural language processing or computer vision, as it enables the model to effectively handle long-range dependencies and improve performance by selectively attending to important features or contexts.

Soft attention works by weighing different parts of the image. Areas of high relevance are multiplied with a more significant weight, and areas of low relevance are tagged with smaller weights, as shown in figure 2.4. The model calculates these weights for each part of the images during training based on how much they contribute to

deciding the final result. As the model is trained, more focus is given to the regions with higher weights.

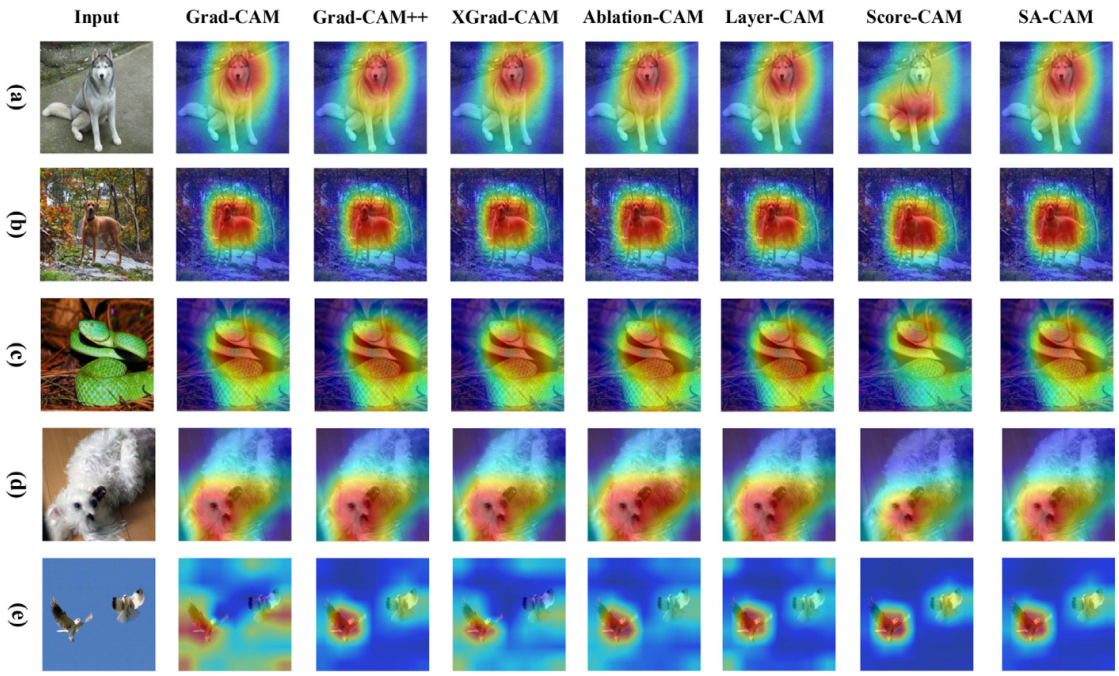

Figure 2.4: In the paper "Generating self-attention activation maps for visual interpretations of convolutional neural networks" [8], examples of how different models' attentions are on different images.

Due to the deterministic nature of soft attention, it remains differentiable and can be trained with standard backpropagation. As the model is trained, the weighting is also trained such that the model gets better at deciding which parts to pay attention to.

Soft attention plays a critical role in U-net networks. During upsampling in the expanding path, the recreated spatial information is often imprecise. To counteract this problem, the U-Net uses skip connections that combine spatial information from the downsampling path with the upsampling path. However, this brings across many redundant low-level feature extractions, as feature representation is poor in the initial layers. Soft attention implemented at the skip connections actively suppresses activations in irrelevant regions, reducing the number of redundant features brought across.

2.5 Denoising Diffusion Probabilistic Models

With the development of artificial intelligence, more tools are becoming available to address the challenges of image generation. Generative adversarial network (GAN) and Variational autoencoder (VAE) models provide quite a good solution, but they struggle with generating high-resolution or varied images. On the other hand, diffusion models [16], [5] are suitable for producing high-resolution, diverse, high-precision images. These are generative models whose operation is based on trying to "enhance" a basic noisy image until, by the end of the generation, the image becomes lifelike or suitable for the task.

2.5.1 Training diffusion models

The diffusion model is taught by taking a basic, untouched image and adding Gaussian noise with several repetitions so that it begins to distort, as illustrated in figure 2.5. During learning, the model receives the given images with different amounts of mixed noise and the value specifying the amount of noise for the given images. From this, the model can learn the transition between each noisy image and how to improve them during generation. The model learns by trying to recreate the original image from Gaussian noise, and the loss is calculated based on how different the final image is from the original image.

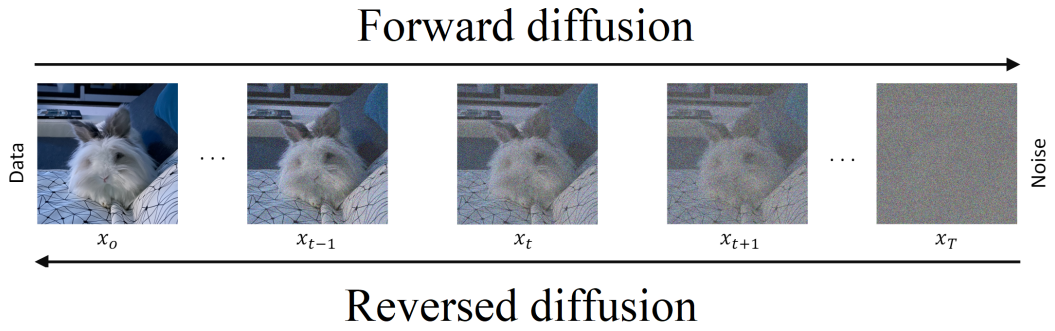

Figure 2.5: Diffusion process [3]

2.5.2 Image generation

When generating, the first step is to create a noise-only image. This is usually just an image with completely random pixels that we give as input to the model. The model assumes that this image is an average image mixed with a certain amount of noise, which it can filter out based on what it has learned. Since the model learned

to filter between levels of a given amount of noise during training, it iteratively goes through those levels and tries to filter out the remaining noise until it can finally return clean image as you can see at figure 2.5.

2.6 Evaluating Diffusion Models

Evaluating diffusion models, particularly those generated images, involves both qualitative and quantitative methods to provide a comprehensive assessment of their performance. This ensures that the selection of models for specific applications is informed by both objective metrics and human judgment.

2.6.1 Fréchet inception distance

The Fréchet Inception Distance (FID) [7] is a widely used quantitative metric for assessing the quality of generative models, including diffusion models. FID quantifies the similarity between two image datasets by computing the Fréchet distance between Gaussians fitted to feature representations from the Inception network. It is typically used to compare real and generated image distributions, providing an objective measure of similarity. The lower the FID score, the better the generated images are perceived to be similar to the real images. However, FID results can be influenced by several factors, including the number of images, randomness in the diffusion process, the number of inference steps, and the scheduler used.

2.6.2 Kernel inception distance

The Kernel Inception Distance (KID) [2] is a metric used for evaluating the quality of images generated by generative models, including diffusion models. KID is an enhanced version of the FID metric, which measures the similarity between two sets of images. KID uses a different kernel function, making it less sensitive to outliers and potentially more robust in assessing the similarity between produced and real picture distributions KID has been proposed as a replacement for FID. FID has no unbiased estimator, which leads to a higher expected value on smaller datasets. KID is suitable for smaller datasets since its expected value does not depend on the number of samples. It is also computationally lighter, more numerically stable, and simpler to implement.

2.7 Self-supervised learning

Self-supervised learning (SSL)[6], [10], [13] is a machine learning technique that relies on an unsupervised learning methodology to generate implicit labels based on unstructured data, thus avoiding the need for manually labeled data for training. It is particularly useful in fields such as computer vision and natural language processing, where obtaining large amounts of labeled data can be time-consuming and expensive.

SSL tasks are designed to infer "ground truth" from unlabeled data, making it a more efficient and cost-effective alternative to traditional supervised learning methods. The fundamental idea for self-supervised learning is to create supervisory signals by making sense of the unlabeled data provided to it in an unsupervised fashion on the first iteration. Then, the model uses the high-confidence data labels among those generated to train the model in subsequent iterations like the supervised learning model via backpropagation.

SSL can be categorized into two types of tasks: pretext tasks and downstream tasks. Pretext tasks involve training an AI system to learn meaningful representations of unstructured data, which can be used as input for downstream tasks, such as supervised learning or reinforcement learning tasks.

2.7.1 KNN with SSL pretrained backbone

To implement K-Nearest Neighbors (KNN) classification using self-supervised learning (SSL), it is necessary to train a backbone model with an SSL approach. This involves training the backbone model on a large set of unlabeled data by solving a pretext task, which enables the model to learn and extract meaningful features from the data. Once the SSL model is trained, it is able to extract feature representations from a given dataset. This process entails passing the labeled input data through the pretrained model up to a certain layer and collecting the output features. These extracted features are then used as inputs for KNN classifier training. It is then possible to run any new data through the feature extracting model, so that its output can be used by the trained KNN classification model to tell which class it belongs to.

2.8 Probing classifiers for evaluation

Probing classifiers [1] are techniques used to examine the internal representations learned by machine learning models. They aim to understand how a model processes and encodes various aspects of input data, such as syntax, semantics, and other features. Typically, probing classifiers are trained as separate models on top of a pretrained model's representations. This additional classifier is designed to predict specific properties or features within the latent space created by the pretrained model. To train a probing classifier, labeled input data is passed through the pretrained model up to a chosen layer, and the output features of that chosen layer are extracted to serve as inputs for training the classifier. By evaluating the probing classifier's performance, we can infer the extent to which the pretrained model has captured the data's important properties. The process involves fine-tuning the probing classifier while keeping the pretrained model's parameters unchanged.

2.8.1 KNN Probe

The K-Nearest Neighbors (KNN) classification [9] algorithm is a supervised machine learning technique. It operates on the principle of similarity, where the objective is to classify or predict the label of a new data point based on the labels of its K nearest neighbors in the training dataset. The distance between data points is measured using metrics such as Euclidean distance or Minkowski, facilitating the identification of the closest neighbors. This algorithm is particularly useful in scenarios where the data does not follow a specific distribution, making it a non-parametric method.

KNN is categorized as an instance-based or lazy learning algorithm because it defers the generalization of the training dataset until the evaluation is performed. This characteristic leads to faster training times but potentially slower and more resource-intensive testing phases. The choice of K, the number of neighbors to consider, plays a crucial role in determining the accuracy of predictions.

Using the KNN probe, the output of a specific layer in the backbone model serves directly as input for the KNN classifier. New data can then be processed through this feature-extracting model, enabling the KNN classifier to determine the appropriate class based on the extracted representations.

2.8.2 Linear Probe

A linear probe involves training a simple linear classifier on features extracted by a pretrained model. This method is frequently employed to evaluate the quality of the features learned by the self-supervised model, as it provides a straightforward way to assess how well these features can be used for classification tasks.

2.9 A Method for Evaluating Latent Space of Denoising Diffusion Models

In my work I have built upon the paper "Denoising Diffusion Autoencoders are Unified Self-supervised Learners" [17], the contents of which greatly helps to understand the importance of the topic:

The challenge in machine learning of managing vast amounts of data with limited human annotations has prompted a shift from supervised to self-supervised pre-training, especially in natural language processing (NLP). In computer vision, self-supervised learning has not matched the success seen in NLP. Generative adversarial networks (GANs) and autoregressive Transformers can produce high-quality images but do not significantly improve discriminative tasks. Contrastive methods and masked autoencoders have been used for feature extraction, yet they are not as effective for image generation. However, diffusion models, which are trained as multi-level denoising autoencoders (DAE, figure 2.6, show promise in bridging this gap. These models have demonstrated the potential to learn discriminative features through generative pre-training, as they can handle complex tasks that require a deep understanding of visual concepts. Recent studies have shown that diffusion models, like DDPM segmentation, can capture semantic information and improve feature quality for representation learning.

Regarding diffusion models for representation learning, various studies have used auxiliary encoders or modified diffusion frameworks to extract features, but these often focus on attribute manipulation or fail to outperform contrastive baselines significantly. Some research has incorporated classification objectives into diffusion training, but this generally harms generative performance and does not compete well with dedicated recognition models. This approach diverges by achieving comparable performance to self-supervised and supervised models without altering diffusion frameworks. Inspired by DDPM segmentation, which addresses segmentation on single super-class datasets, this work is pioneering in applying diffusion models for classification on complex multi-class datasets.

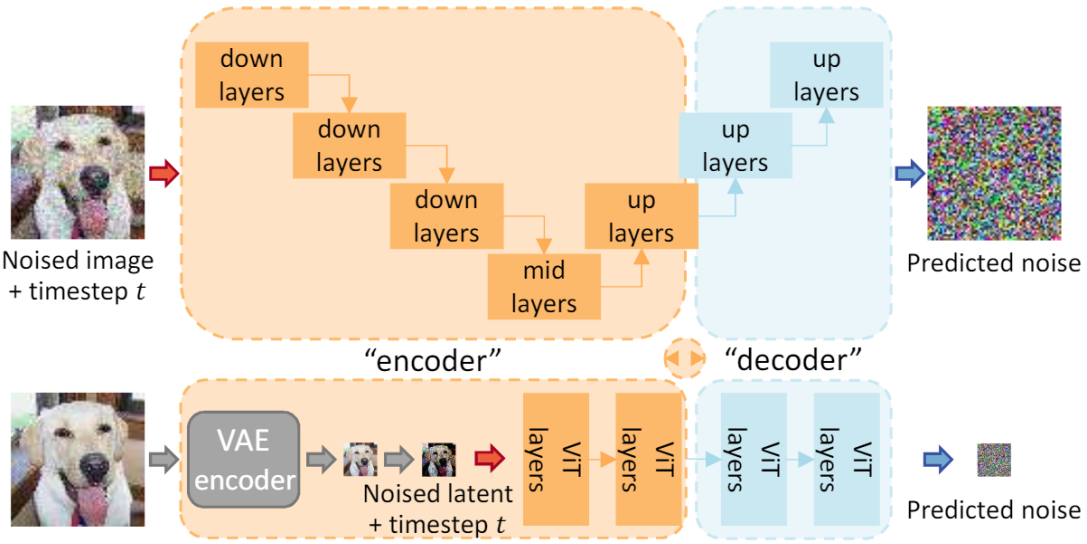

Figure 2.6: Denoising Diffusion Autoencoders (DDAE). Diffusion networks are essentially equivalent to level-conditional denoising autoencoders (DAE). The networks are named as DDAEs due to this similarity. The image is from the paper "Denoising Diffusion Autoencoders are Unified Self-supervised Learners" [17].

Evaluating discriminative learning capabilities in Denoising Diffusion Autoencoders (DDAEs) is challenging due to the complexity of extracting compact, classifiable information. While deterministic methods like DDIM can produce identifiable encodings, they are not compact enough for effective classification. Existing approaches often add separate encoders for classification, but this study proposes using intermediate activations from pre-trained DDAEs directly, inspired by methods used in iGPT and DDPM-seg. This approach is compatible with existing models and frameworks without modification. The research explores whether DDAEs can produce linearly separable representations at certain encoder-decoder interfaces, similar to denoising autoencoders (DAEs). Various network backbones were examined to identify the best layer for feature extraction, considering the impact of layer depth, backbone training epoch and noise scales on feature quality.

The experiments examine the impact of core design choices in diffusion models on both generative and discriminative performance through ablation studies. These studies were conducted using the CIFAR-10 and TinyImageNet datasets under various settings, such as linear probing, fine-tuning, and ImageNet transfer.

In the study's discussion and conclusion, the authors propose using diffusion pre-training as a unified method to enhance generative capabilities and deep visual understanding, aiming to develop comprehensive vision foundation models.

The findings suggest that diffusion models could serve as unified vision foundation models, capable of excelling in both generative and discriminative tasks. This dual capability may lead to advancements in vision pre-training and applications, potentially leveraging large-scale pre-trained models like Stable Diffusion for powerful discriminative knowledge transfer.

Findings suggest that the best features for classification are found midway through the up-sampling process rather than at the lowest resolution and that small perturbations of images with noise enhance linear probe performance, particularly in models trained with EDM. The optimal layer varies across different datasets and models. While linear probe accuracies reflect the highest-quality features found, fine-tuning further improves accuracy by using more layers.

Nevertheless, it was clearly shown in the results that the generative and classifier capabilities of a given backbone model do not evolve in parallel during its learning. There is a point during model training where its classifier ability starts to decrease, but the generative ability of the model still improves during training.

Chapter 3

Objectives

I aim to build an unconditional diffusion model on three different datasets that can also be utilized for classification. I intend to test the resulting model's image generation capability.

The primary goal is to test the representational ability of the latent space in individual model layers using probe methods, allowing me to identify which layer is best suited for classification tasks.

Building on previous research and hypotheses suggesting that the model's classification and generative abilities do not change monotonically as training progresses, I plan to examine the relationship between these abilities by testing checkpoints at different training stages to identify the most efficient scenario. In addition, I intend to assess how image size, the use of attention, noise, and probe methods impact performance.

The overall goal is to create a diffusion model using self-supervised learning that can also be used for classification. This self-supervised approach aims to save the need for labeled data, which is essential for traditional classification models, thereby saving a significant amount of time and resources in the labeling process.

Chapter 4

System design and methods

My research consist of three main steps as illustrated in figure 4.1.

1. Training a U-net using unlabelled images.
2. Generative evaluation of the neural network.
3. Freeze the weights of the trained U-net model and use the output of each block from it for classification in two ways.

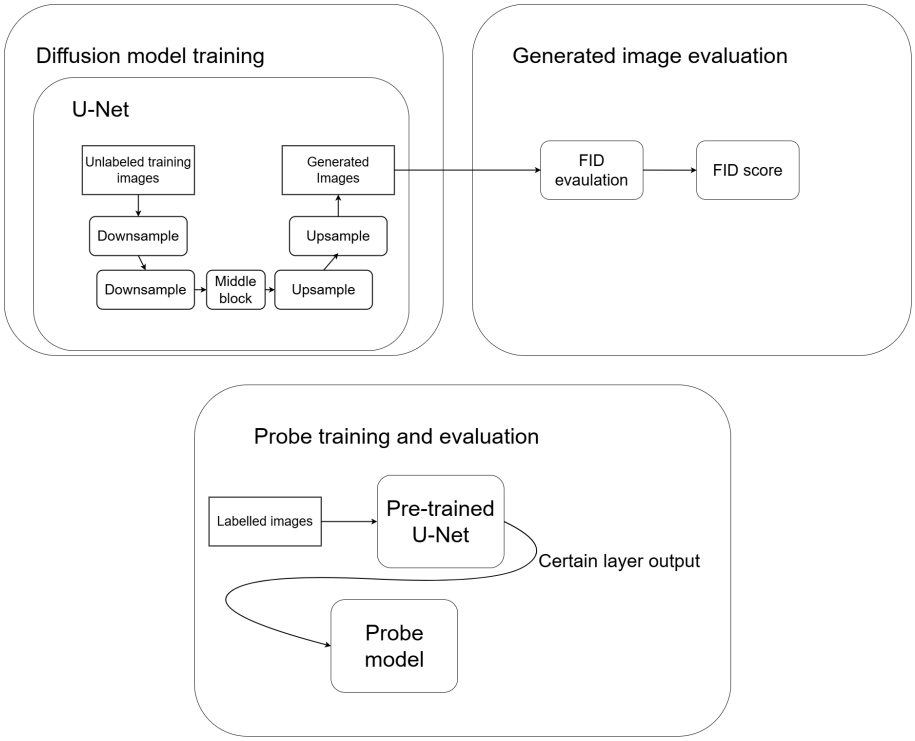

Figure 4.1: The three main steps of my work

4.1 Training U-net

As a first step, I train a U-net I built in a self-supervised way. As input it receives the given unlabeled images, processes them and by the end of the training it is able to generate similar images, as shown in figure 4.2. This neural network serves as the backbone for my classification tasks.

After training, I compare the generated neural network images with the test images using the FID metric as in section 2.6.1.

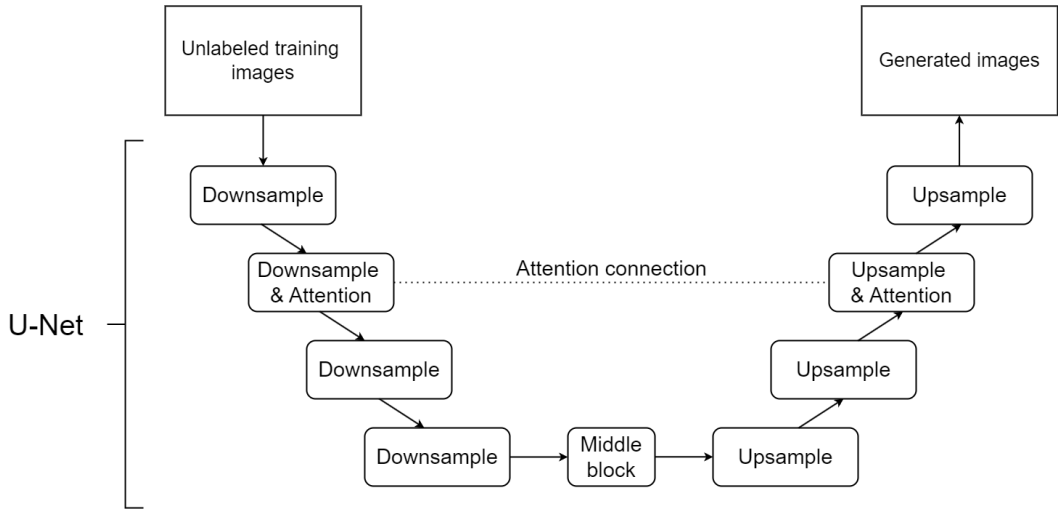

Figure 4.2: Training U-net on unlabeled data

4.2 Linear probe with U-net

I evaluate the self-supervised trained backbone in a supervised way. Before training the new model, I freeze all weights of the pre-trained U-net and select the output of the block I want to test. The selected output is connected to a fully connected neural layer, which serves as the final output of the classification as shown in picture 4.3.

During the final model training, the labeled data is processed by the backbone, the output of which is received by the fully connected neural layer, which is trained in this process. This layer serves to evaluate the U-net's latent space representation. This layer evaluates the U-net's latent space representation. By not using any complex classification method for the output, I can focus on testing only the strength of the backbone's latent space.

4.3 K-Nearest Neighbors classification

As in the Linear probe method, I do not modify the parameters of the already trained backbone. I give the output of the selected block generated from the labeled training images as input to the KNN classifier as shown in picture 4.3 .

Once the KNN has memorized the latent space representations for each class generated by the selected backbone block, the model can decide which class the new images belong to based on their representations provided by the backbone.

4.4 Software and Hardware

I do not have my own GPU to implement the necessary training; instead, I gained access to an NVIDIA A100 GPU provided by the DEEP4 system at BME TMIT. This allowed me to run the training processes in a Docker environment. The Docker setup was Python-based with PyTorch and "Diffusers - Hugging Face" packages, and I used Visual Studio Code to SSH into the container and execute the required runs. For training and testing the final five models, I needed around 350-400 GPU hours. However, throughout the entire project, I used at least 1,500 GPU hours.

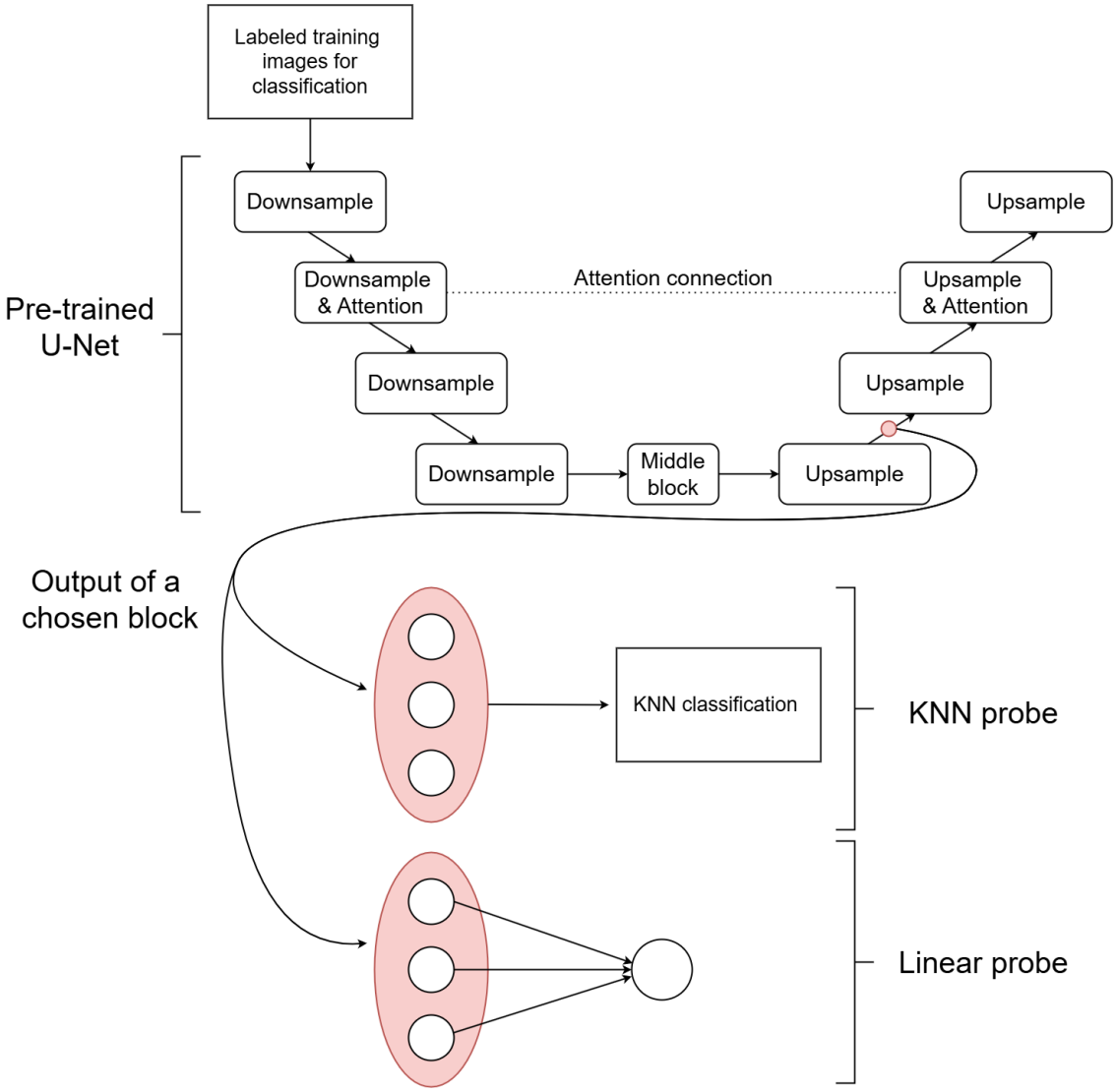

Figure 4.3: K-Nearest Neighbors and linear probe

Chapter 5

Datasets

I worked with three data sets, Flower102, CIFAR-10 and STL-10. Since I was training Unconditional Denoising Diffusion Probabilistic Models, I did not use the labels available for the dataset, but they were needed for the KNN classification and linear probe fitting.

5.1 Flowers 102

The Oxford 102 Flower dataset, also known as the 102 Category Flower Dataset, is a comprehensive collection of images used for various machine learning tasks, particularly in the field of image classification. The dataset consists of 102 flower categories, with each category containing between 40 and 258 images. These categories represent flowers commonly found in the United Kingdom as shown on figure 5.1. The images within the dataset exhibit significant variations in scale, pose, and lighting conditions.

Figure 5.1: Flowers 102

The dataset includes categories with large variations within the category and several very similar categories. The dataset is visualized using isomap with shape and color

features, providing a clear overview of the diversity and complexity of the flower categories. The dataset is divided into a training set, a validation set, and a test set. The training and validation sets each consist of 10 images per class (totaling 1020 images each), while the test set consists of the remaining images (minimum 20 per class).

5.2 CIFAR-10

The CIFAR-10 dataset is a foundational dataset in the field of machine learning and computer vision. It is widely used for training and benchmarking machine learning models, particularly in the context of image classification. The dataset consists of 60,000 32 × 32 color images divided into 10 different classes as shown in figure 5.2. These classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class has 6,000 images, with 5,000 images used for training and 1,000 for testing.

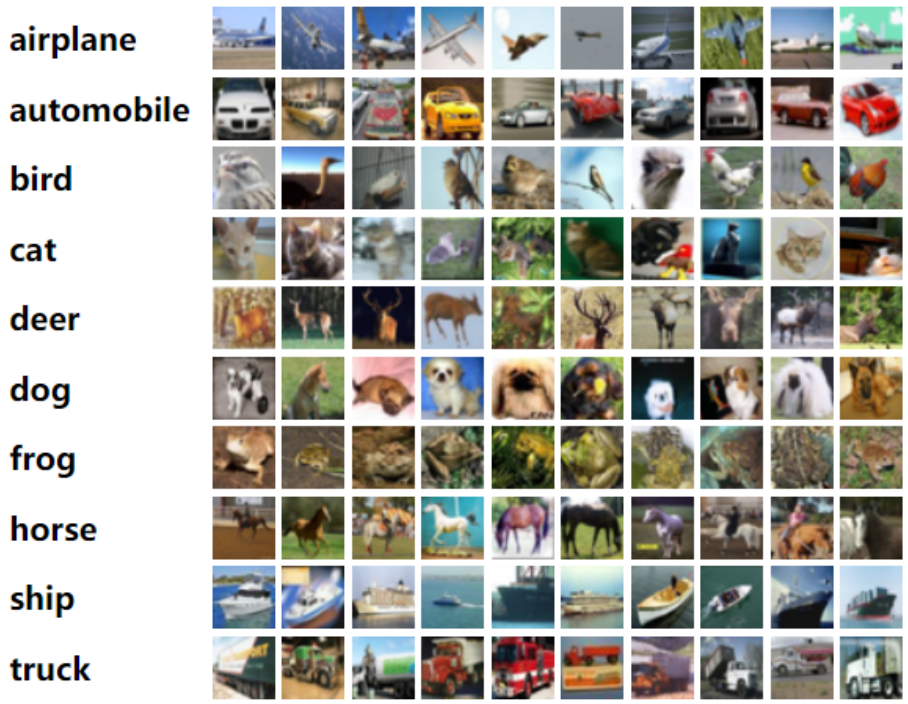

Figure 5.2: CIFAR-10

5.3 STL-10

The STL-10 dataset (shown in 5.3) is an image recognition dataset specifically designed for developing unsupervised learning models. It consists of 10 different classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. Unlike CIFAR-10, the dataset contains 96x96 colour images, with only 500 training images and 800 test images for each class. In addition, it has 100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in addition to the ones in the labeled set. This raises the question: since the test set contains only 10 classes, but the 100000 unlabeled training data include more than just those 10 classes, does this diversity help or hinder the model in learning the necessary properties of each class?

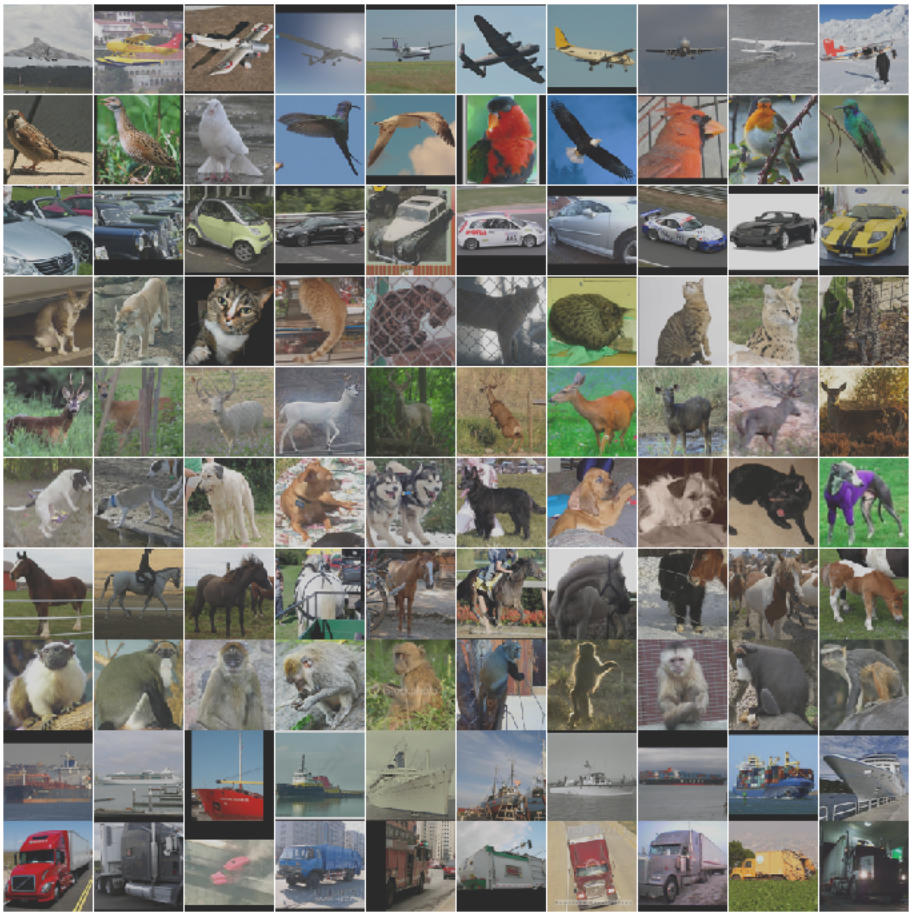

Figure 5.3: STL-10

Chapter 6

Implementation

The proposed DDPM model (shown at figure 6.1) consisted of 4 blocks and each block contained 2 ResNet layers. The layers in the second block of the network had attention, while the other layers were simple ResNet components. In some experiments, attention was excluded entirely during training. The outputs of the different layers contained 128, 256, 256 and 256 channels from top to bottom. The model had 3 color inputs and 3 outputs for both datasets, but the size of the images varied depending on the training.

6.1 DDPM model training

6.1.1 Data pre-processing and data loading

The CIFAR-10 dataset's training partition contained 50000 images, with 5000 32×32 pixel images per class. I made the necessary transformations for training: doing random horizontal flips, converting them to tensors to allow the model to process the images, and normalizing image values.

For the Flowers 102 dataset, the same pre-processing steps were required as for CIFAR-10, with the only difference being the different sizes of the images. Therefore, it was necessary to resize all the images during pre-processing. I used 64-pixel images for the training.

STL-10 had the highest image resolution, at 96 pixels. Due to limited computing capacity, the images had to be pre-processed in the same way as those in the Flowers 102 dataset. I used 32- and 64-pixel images for the training.

6.1.2 Training process

During the training, I used AdamW optimizer with a learning rate of 0.0001, same as in the paper [17] I built upon on. The process lasted between 1100 and 2000 epochs, while a given epoch was done using 1000 steps forward diffusion for both datasets. It would have been possible to shorten the steps of forward diffusion during model training, which would have sped up the process, but then I would have deviated from the architecture built in the paper [17] that inspired my work, which I assumed to be a good starting point, and which could have greatly degraded the quality of the model. During training, an image was generated every 20-50 epochs to show what output the model could produce at the current state of the pipeline. Additionally, every 50 epochs, I saved the weights of the model so later testing could be performed.

When using attention for training, I was able to use a batch size of 320 for 32×32 pixel images, but for 64×64 pixel images, the batch size was limited to 20. The images were indeed only 4 times the size, so in theory, they should have taken up 4 times the GPU memory due to the properties of convolutional layers, but due to the attention layers, they were 16 times the size. In practice, this meant that 20 32×32 images used 2.5 gigabytes of memory, while 64×64 images used 40. Approximately with both datasets, the process took 12 hours.

In cases where I didn't use attention, I was able to work with significantly larger batch sizes, so in some cases, I preferred this method. Here I was even able to achieve 1024 batch sizes on 32×32 images and 320 batch sizes on 64×64 images.

I trained a total of 5 models. The first one included an attention layer and was trained on the Flower 102 dataset with 64-pixel images. Two models were trained on the CIFAR-10 dataset with 32-pixel images, one with an attention layer and one without. Lastly, I trained two diffusion models on the STL-10 dataset without attention, one using 32-pixel images and the other using 64-pixel images.

6.2 Evaluation methods for the layers of the DDPM model

A simplified representation of the U-net network I have created and trained is shown at figure 6.1. In practice, it consists of 9 complex parts with 9 different outputs. These outputs give a latent space representation of the images in each phase of the model. For example, the fourth downsample block (before the middle block) outputs a (1,256,4,4) tensor from a 32×32 image. In each case, this block produces a 256-

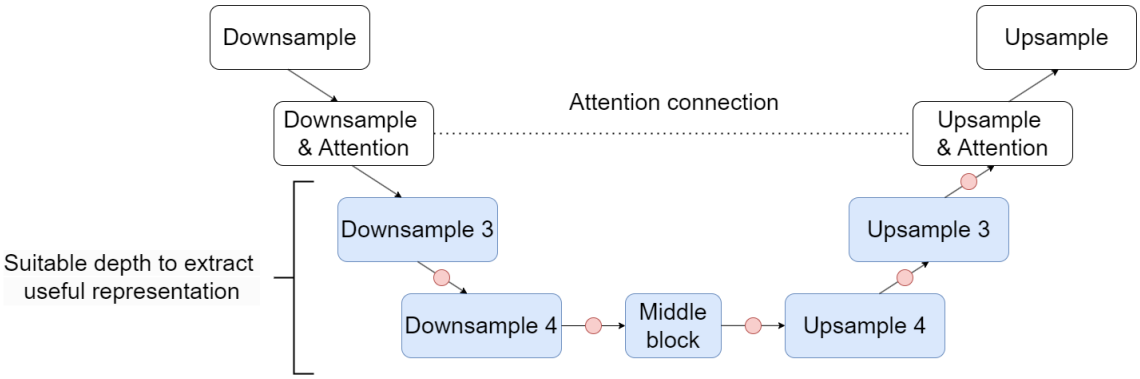

Figure 6.1: The simplified U-net model (the blocks tested are marked in blue and the values of the outputs marked in red are used for classification)

dimensional representation of the input, resulting in a tensor with shape (1, 256, 4, 4). The (4,4) part depends on the size of the processed image. For 64×64 images, the block returns tensors of shape (1,256,8,8). Due to the hierarchical convolutional neural networks, one possible effect is that the input is displayed in an increasingly smaller dimension. This is why the representation of images here is 4×4 instead of the original 32×32. The higher you are in the architecture of the network, the higher the size of the information being processed.

6.2.1 Evaluation with linear probe

In order to check the representational ability of the latent space of a given block, I need to connect a linear classification layer to its output. To do this, I freeze the parameters of the already trained U-net and train a fully connected neural layer on the output of the corresponding block using labeled training data. The classification layer is trained for 30 epochs using the Adam optimizer and a learning rate of 0.001. These parameters were chosen to avoid further optimization of the linear probe layer, focusing primarily on the representational strength of the latent space in each block. Adam typically offers a quick and effective solution for linear classification, and a learning rate of 0.001 has consistently proven reliable in my experience with simple single-layer linear classification.

6.2.2 Evaluation with KNN classifier

I also used supervised training in the case of KNN classification. Once again, freezing the weights of the given U-net, I used it to run through all the training samples to obtain the latent space representation of the images by the currently tested block.

On top of these multidimensional samples, I fitted KNN classifier so that by running the images in the test set through the system, I could predict which class they belonged to based on their and neighbours in the latent space.

To evaluate, I used top-k accuracy, which measures how often the true label of an instance is within the top k predicted labels by the classifier. In my case, "top-3" measures explicitly the accuracy of the KNN classifier when the true label is among the top 3 predicted labels.

The decision boundary is calculated based on the 50 nearest neighbours for CIFAR-10 and STL-10. The prediction on the Flowers 102 dataset is based on the five nearest neighbors, as it had far fewer data points per class.

Chapter 7

Results

7.1 DDPM image generation

I tested the images (figure 7.1) generated by my models in different training epochs on all datasets using the FID metric as explained in section 2.6.1. The aim was not to train high-resolution models but to analyze diffusion models in pixel space, so I did not use a higher resolution.

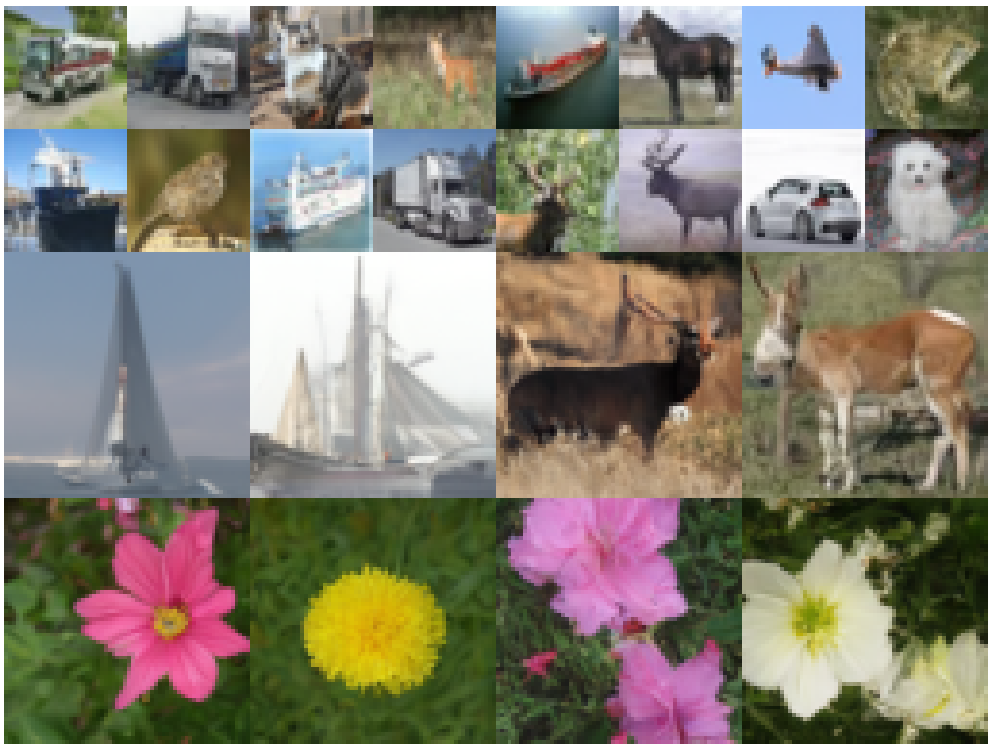

Figure 7.1: Top two rows are images generated from CIFAR-10/STL-10 (32 pixels) , middle row from STL-10 (64 pixels), bottom row from Flowers 102 (64 pixels) dataset.

7.1.1 CIFAR-10

I generated 2,000 images using models saved after every 50th epoch and compared them to the 10,000 images in the test set based on their FID scores. With and without attention, image quality appeared to improve throughout the entire training for about 2000 epochs, achieving FID scores of 28.9 and 30.7 as shown in figure 7.2.

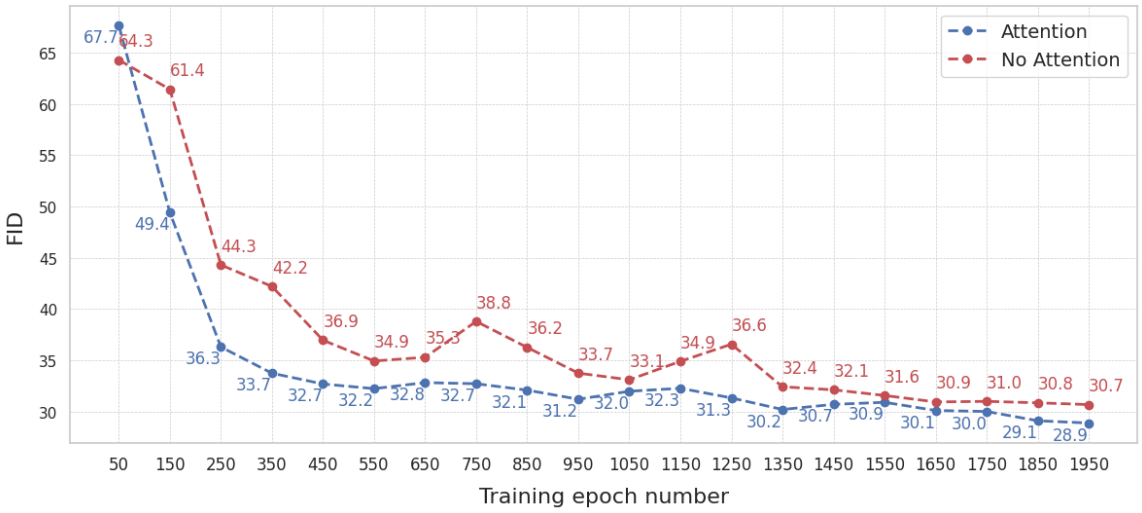

Figure 7.2: Diffusion models (with and without attention) FID results on the CIFAR-10 test dataset at different training epochs.

Visually, I did not notice any significant difference in the quality of the generated images. Figure 7.3 shows some of the images produced by the two models.

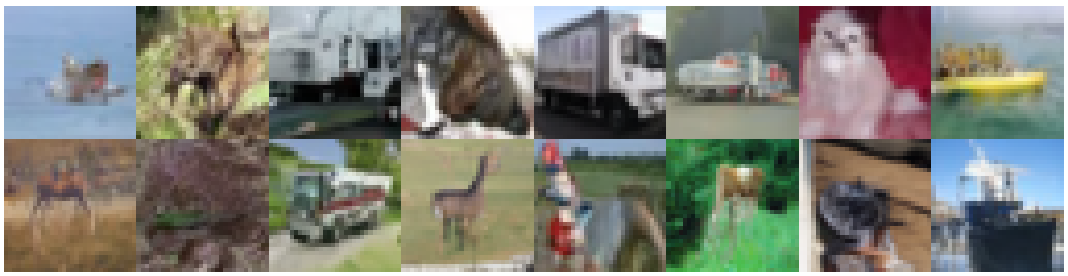

Figure 7.3: CIFAR-10 images generated without attention in the top row and with attention in the bottom row.

7.1.2 Flowers 102

I compared the Flowers 102 test dataset with the 2000 images generated by the model in different epochs. As shown in figure 7.4, the model exhibited its best generalization ability at epoch 450, achieving an FID score of 105, after which no further improvement was observed.

Figure 7.4: Diffusion model FID results on the Flowers 102 (64 pixel images) test dataset at different training epochs.

7.1.3 STL-10

For 32-pixel images, the model achieved its best performance at epoch 1050 with a FID score of 254.6, after which its generative capability began to fluctuate as shown in figure 7.6. Similarly, the model trained on 64x64 images reached its best performance at epoch 550 with a FID score of 214.8 (as also can be seen in picture 7.6), after which it also showed inconsistent FID evaluations, mirroring the behavior of the model trained on 32-pixel images. Figure 7.5 shows some of the images produced by the two models.

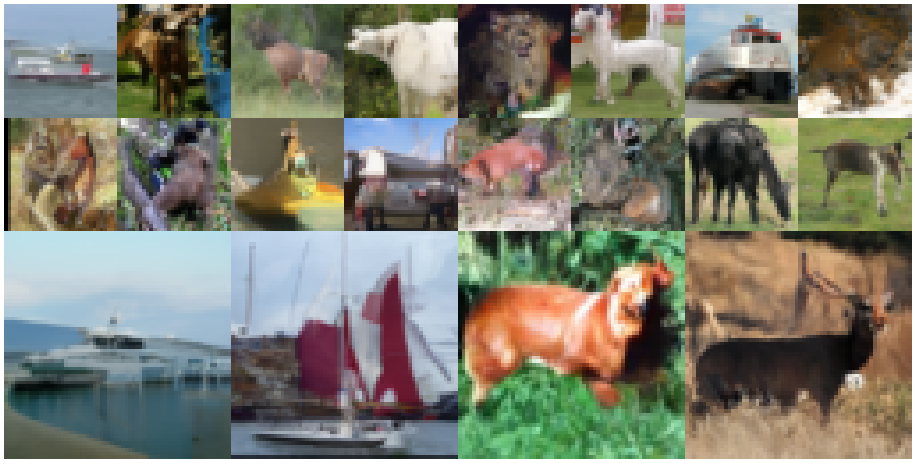

Figure 7.5: STL-10 images generated with 32 and 64 pixel size.

7.2 DDPM classification

I examined all 9 blocks of the model, but during initial experiments only 5 of them proved to perform considerably better in classification tasks compared to random

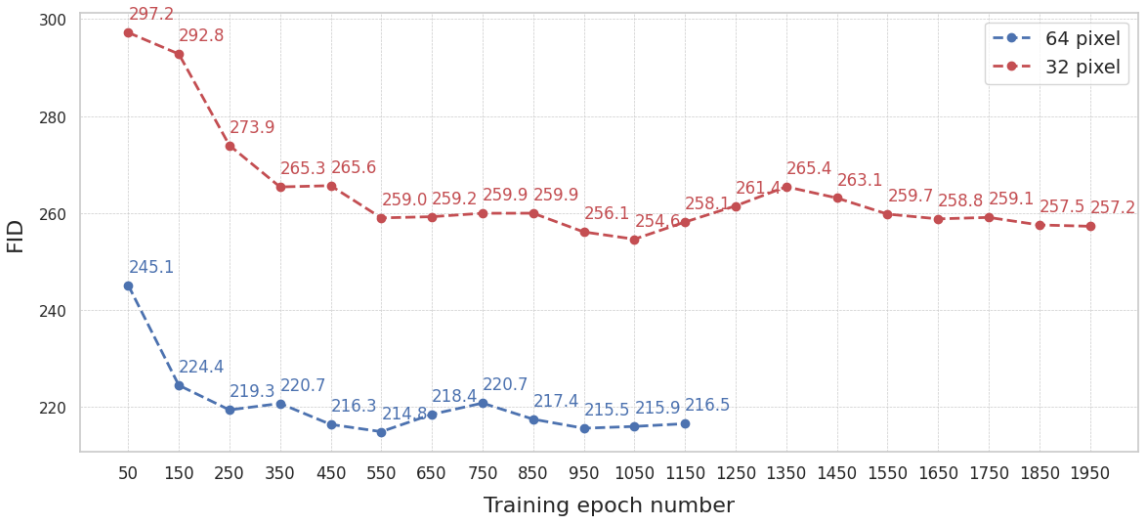

Figure 7.6: Diffusion model FID results on the STL-10 test dataset (32 and 64 pixel images) at different training epochs without attention layer. The 64-pixel model required significantly more computation due to the size of the images. Since the FID score did not improve, I decided not to continue training it. (Training for 1,150 epochs took five days.)

choise, so I will only discuss these. For ease of reference, I have marked the 5 blocks and their output tested in the U-net architecture, which is shown in figure 6.1.

To achieve a better classification result, I took several variables into account. The primary factor for the task is which block gives the best representation. Another important question was at which epoch of the training was the model most suitable for this task, as this was independent of the model's image generation capabilities previously tested. In addition, there was the question of whether noisy input images during classification would increase the performance, as the model works with noisy images for most of the training.

7.2.1 Best performing block

7.2.1.1 Linear and KNN probe results on CIFAR-10

Linear probe: The first column of figure 7.7 shows linear probe results on CIFAR 10. The top and bottom rows of the figure present results of pretrainings without and with attention layer. In both cases, the best-performing blocks were "Upsample 4" and "Upsample 3", with nearly identical validation accuracies of 0.818, while the worst-performing blocks were "Downsample 3" and "Downsample 4".

KNN probe: As shown in Figure 7.7, the "Downsample 3" and "Downsample 4" blocks performed the worst in both the attention and no-attention scenarios.

For the model with attention, the "Upsample 4" and "Upsample 3" blocks achieved the best results, each reaching a validation accuracy of 0.81. In contrast, for the model without attention, 'Upsample 4' was the top performer with a validation accuracy of 0.785.

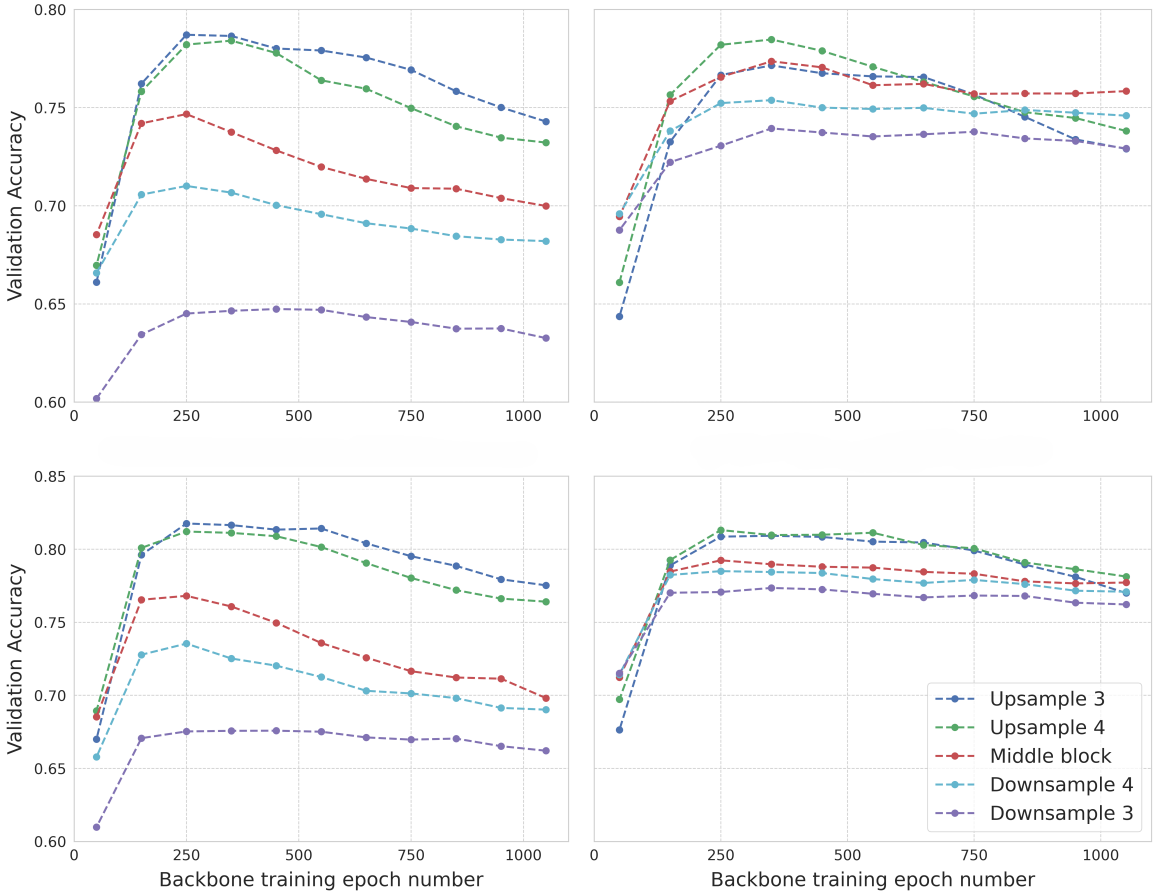

Figure 7.7: Performance of each block on CIFAR-10 dataset using linear and KNN probe on the backbone model. Top left: No Attention with linear probe. Top right: No Attention with KNN. Bottom left: Attention with linear probe. Bottom right: Attention with KNN. The Y-axis represents the validation accuracy of the linear and KNN probes, while the X-axis indicates the training epoch of the back bone model.

7.2.1.2 Linear and KNN probe with Flowers 102

Linear probe: For the Flower102 dataset, similar to previous results, "Upsample 3" performed the best with a validation accuracy of 0.44. The middle and downsample blocks showed nearly identical performance, but "Upsample 4" performed the worst, achieving only a 0.375 validation accuracy as you can see on picture 7.8.

KNN probe: "Upsample 4" performed the worst with a validation accuracy of 0.225, while "Upsample 3" achieved 0.276, just behind "Downsample 3", which performed the best with an accuracy of 0.284. "Downsample 4" and Middle block performed similar around 0.26 as shown on figure 7.8.

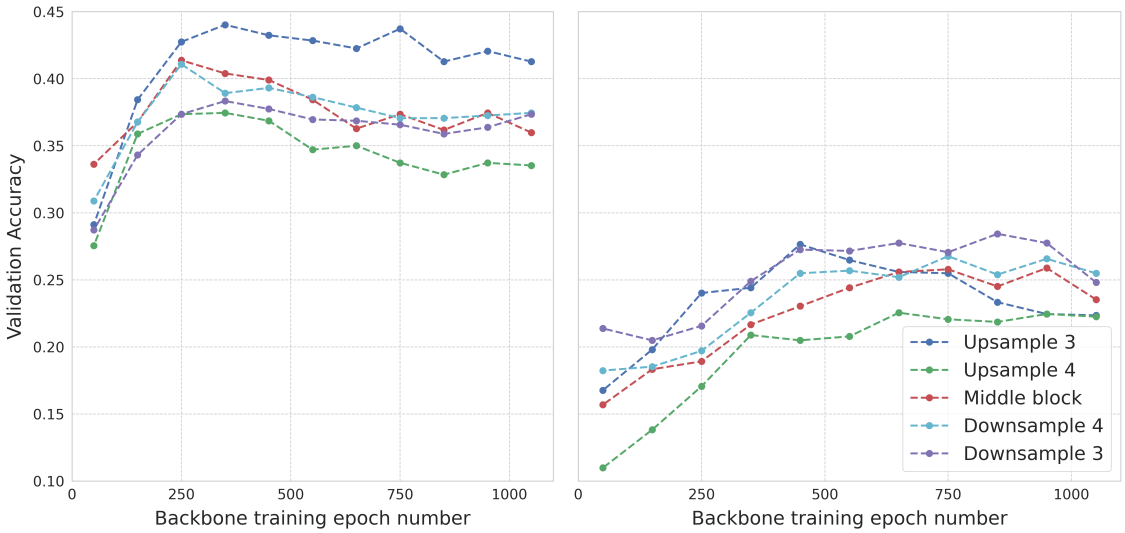

Figure 7.8: Performance of each block on Flowers 102 (64 pixel) dataset using linear and KNN probe on the backbone model. Linear probe on the left, KNN on the right. The Y-axis represents the validation accuracy of the linear and KNN probes, while the X-axis indicates the training epoch of the backbone model.

7.2.1.3 Linear and KNN probe with STL-10

Linear probe: For both 32 and 64-pixel images, the model's downsample blocks performed the worst, while "Upsample 4" performed the best, with a validation accuracy of 0.71 for 32 pixels and 0.804 for 64 pixels. For the 64 $\times$ 64 images, the middle block performed significantly better, whereas for the 32 $\times$ 32 images, the middle block and "Upsample 3" achieved nearly identical performance. These results are all shown in figure 7.9.

KNN probe: For both image sizes, "Downsample 3" had the lowest performance, with validation accuracies of 0.62 and 0.65. Meanwhile, the "Downsample 4" and "Upsample 3" blocks performed almost identically. In both cases, the middle block showed significantly better performance, with "Upsample 3" achieving the best scores of 0.717 and 0.765 as shown on image 7.9.

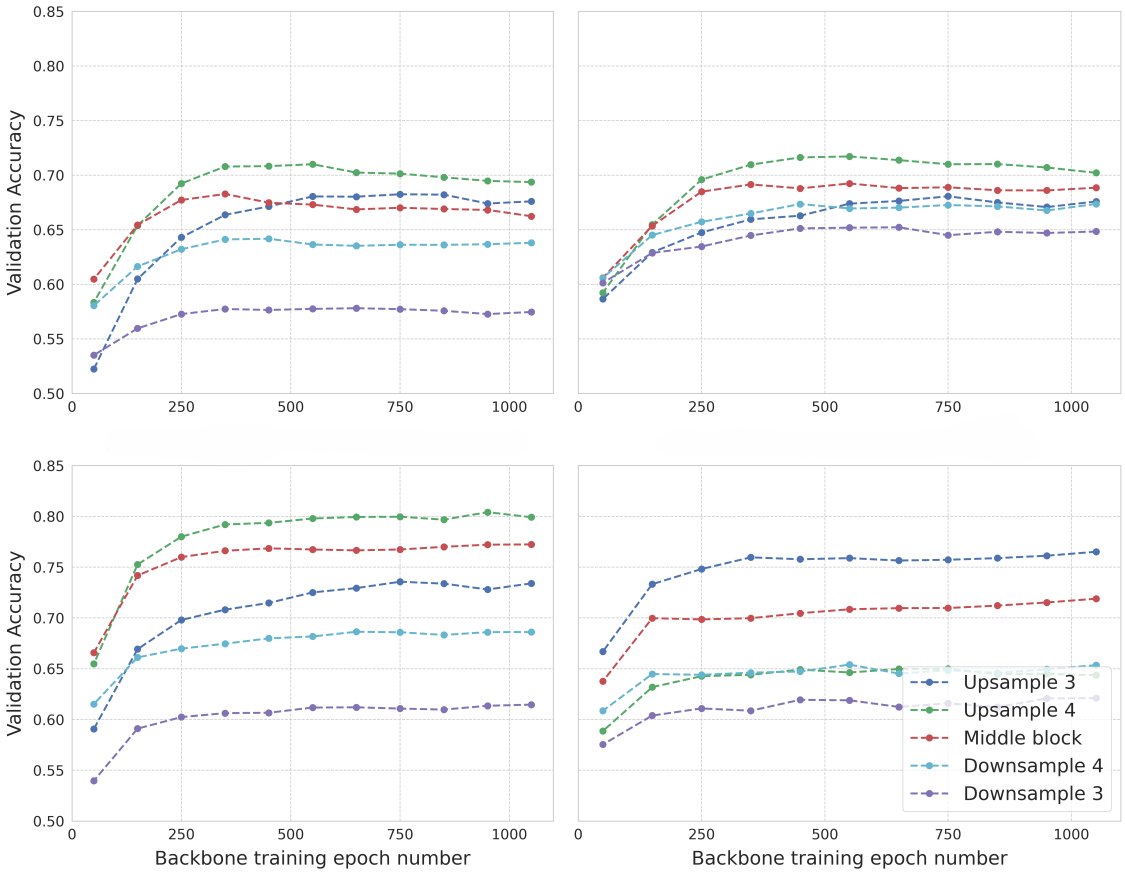

Figure 7.9: Performance of each block on STL-10 dataset using linear and KNN probe on the backbone model. Top left: 32 pixel linear probe. Top right: 32 pixel with KNN. Bottom left: 64 pixel with linear probe. Bottom right: 64 pixel with KNN. The Y-axis represents the validation accuracy of the linear and KNN probes, while the X-axis indicates the training epoch of the backbone model.

7.2.1.4 Differences between blocks for Linear and KNN probe

Regardless of the probing method, the upsample blocks consistently demonstrated the best performance during the models' most optimal training phases. Both metrics show degrading accuracy for upsampling blocks at later epochs, indicating the overfitting of these representations to the generation pretraining task. The linear probe metric also shows this effect for mid and downsample blocks. However, during the training of multiple models, it was observed that the mid and downsample blocks were significantly less prone to overfitting when evaluated using KNN. As shown in figure 7.7.

7.2.2 Image size

I tested the impact of increasing image size on the classification performance of the pre-trained feature extraction network using the STL-10 dataset. As shown in figure 7.10, increasing the image size significantly boosted the model's performance, improving it by approximately 10%. Increasing the image size required more computational resources, as the model needed to process larger inputs.

By increasing the image size, the model's blocks generated larger-dimensional latent spaces. However, this had no effect on which block achieved the best classification performance.

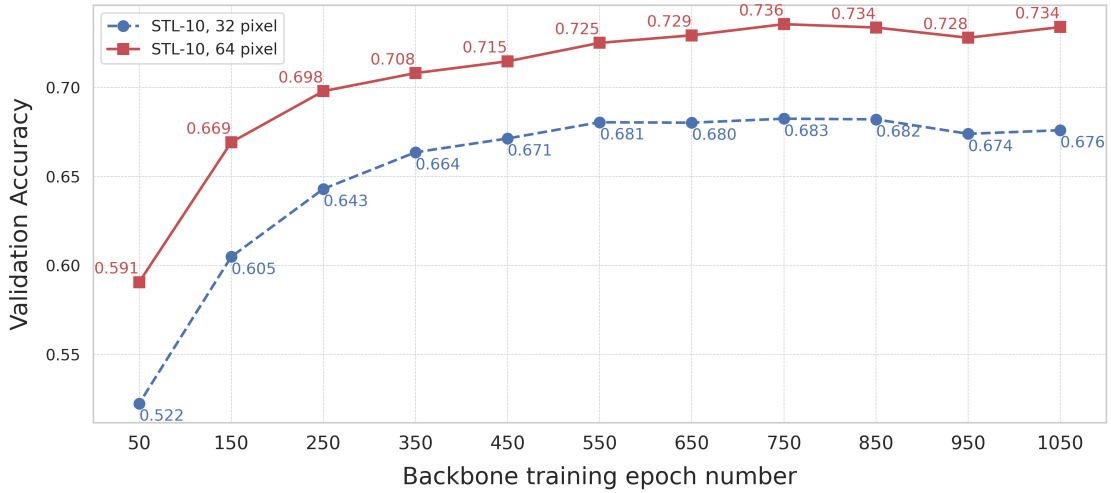

Figure 7.10: Performance of the best blocks on 32 and 64 pixel STL-10 dataset using linear probe on the backbone models. The Y-axis represents the validation accuracy of the linear probes, while the X-axis indicates the training epoch of the backbone model.

7.2.3 Impact of the attention layer

To evaluate how the attention layer influences the model's classification performance, I used the CIFAR-10 dataset. As previously mentioned, I trained two models on this dataset: one with attention and one without. The model's classification accuracy improved by 3% (as can be seen on figure 7.7) when attention was used, for 32-pixel images.

It is important to note that using an attention layer increases the GPU memory requirement for a given image quadratically. For 32-pixel images, this was not an issue with a graphics card featuring 40 GB of GPU memory. In contrast, for 64×64 images, simply doubling the image size (from 32×32 pixel to 64×64 pixels) results

in a fourfold increase in memory demand. This is further amplified by the square scaling of the attention layer, leading to a 16-fold increase in memory requirements.

A clear example of this is that training with a batch size of 24 for 32 × 32 images required only 2 GB of GPU memory, whereas training with 64 × 64 images consumed 32 GB of memory when the model included an attention layer.

Depending on the situation, the use of the attention layer should be carefully considered. While it can enhance the model's generative and classification capabilities, at a certain point, this comes at the cost of reduced batch size. A significant reduction in batch size can severely impair the model's performance.

7.2.4 Best performing epoch

An important consideration is at which stage of training the original backbone generated the most suitable latent space representation for classification. During the training on the CIFAR-10 dataset, the linear probe and KNN classification achieved the best results between the 250th and 350th epochs as shown in figure 7.11.

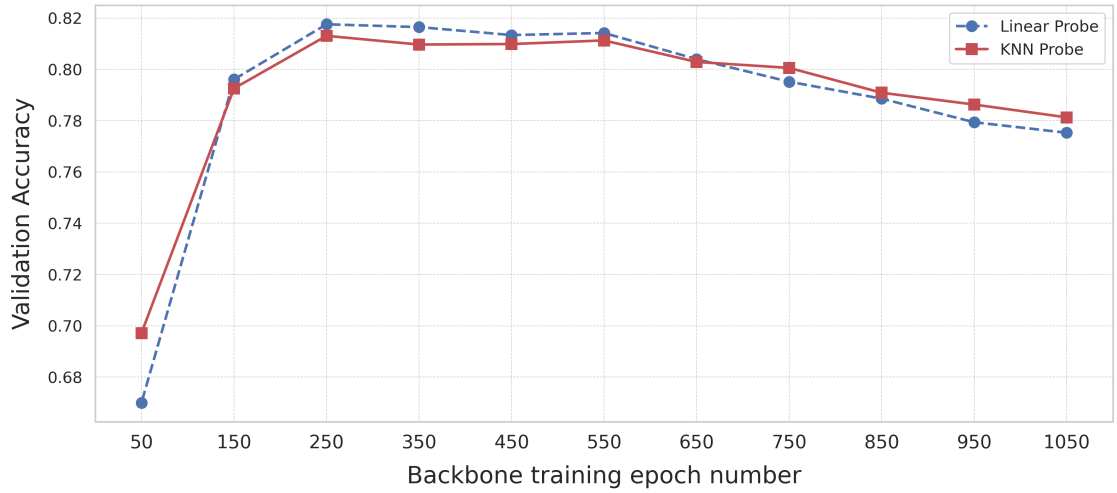

Figure 7.11: Performance of the best blocks on CIFAR-10 dataset using linear and KNN probe on the backbone models. The Y-axis represents the validation accuracy of the linear and KNN probes, while the X-axis indicates the training epoch of the backbone model.

The Flower102 dataset achieved the best classification performance for the linear probe between epochs 350 and 750, while the basic backbone performed best for KNN fitting between epochs 450 and 850 as can be seen in figure 7.12.

For the STL-10 dataset, the fluctuations in epoch quality were similar between linear probe and KNN classifications, with the primary difference stemming from the image

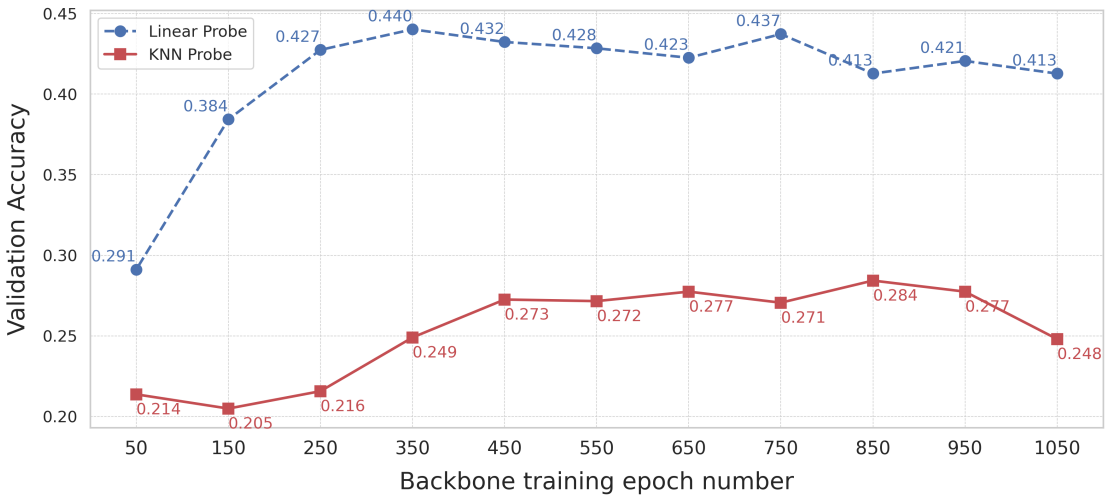

Figure 7.12: Performance of the best blocks on Flowers 102 dataset using linear and KNN probe on the backbone model. The Y-axis represents the validation accuracy of the linear and KNN probes, while the X-axis indicates the training epoch of the backbone model.

size. For 32-pixel images, the model performed best in both cases around epoch 550. In contrast, for 64-pixel images the radical increase in performance stopped at epoch 350 and thereafter, the improvement in performance was minimal, with only a 0.005 increase in validation accuracy by epoch 700 as can be seen on figure 7.13.

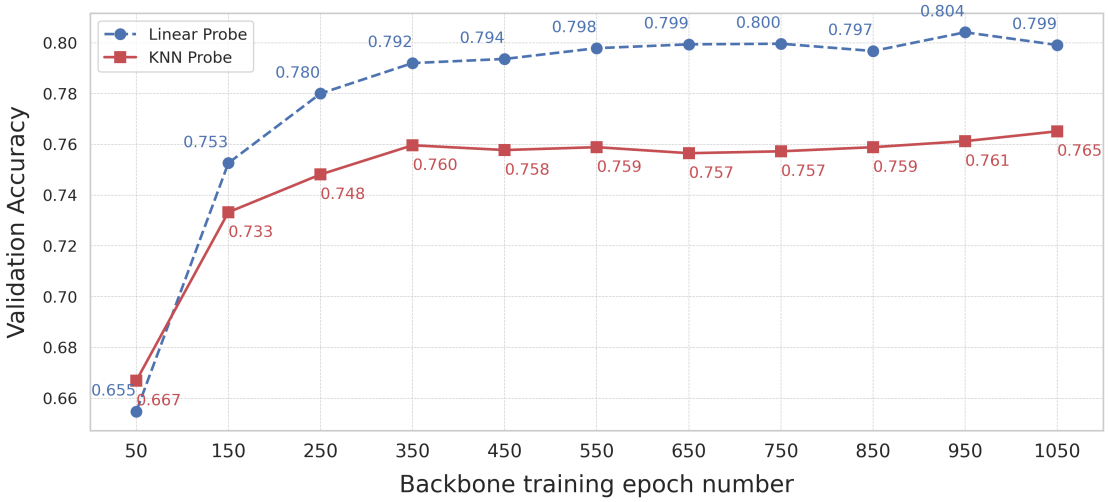

Figure 7.13: Performance of the best blocks on STL-10 dataset using linear and KNN probe on the backbone models. The Y-axis represents the validation accuracy of the linear and KNN probes, while the X-axis indicates the training epoch of the backbone model.

7.2.5 Noise during classification

During the original U-Net training, forward diffusion was applied to the images over 1000 steps. However, in the classification phase, adding even the smallest amount of noise as early as the 10th step out of 1000 significantly degraded the model's classification performance. This 10th step is an extremely minimal level compared to the noise levels used during U-Net training, yet it consistently reduced performance across all cases.

7.2.6 Error in normalization

At the beginning of my work, I made a mistake, which I quickly realized and corrected. However, during this time, I trained a Flowers102 model using incorrectly normalized images. The error occurred during the normalization of the training images, where the pixel values were transformed not into the range of -1 to 1, but approximately into the range of -1.1 to 1.1. This small deviation in normalization significantly impacted the model's classification capabilities, as seen when comparing it to a model trained on properly normalized images, illustrated in figure 7.14.

The impact is evident in the significant drop in maximum performance, and the model's classification capabilities showed a much steadier improvement and decline under overfitting across the different blocks.

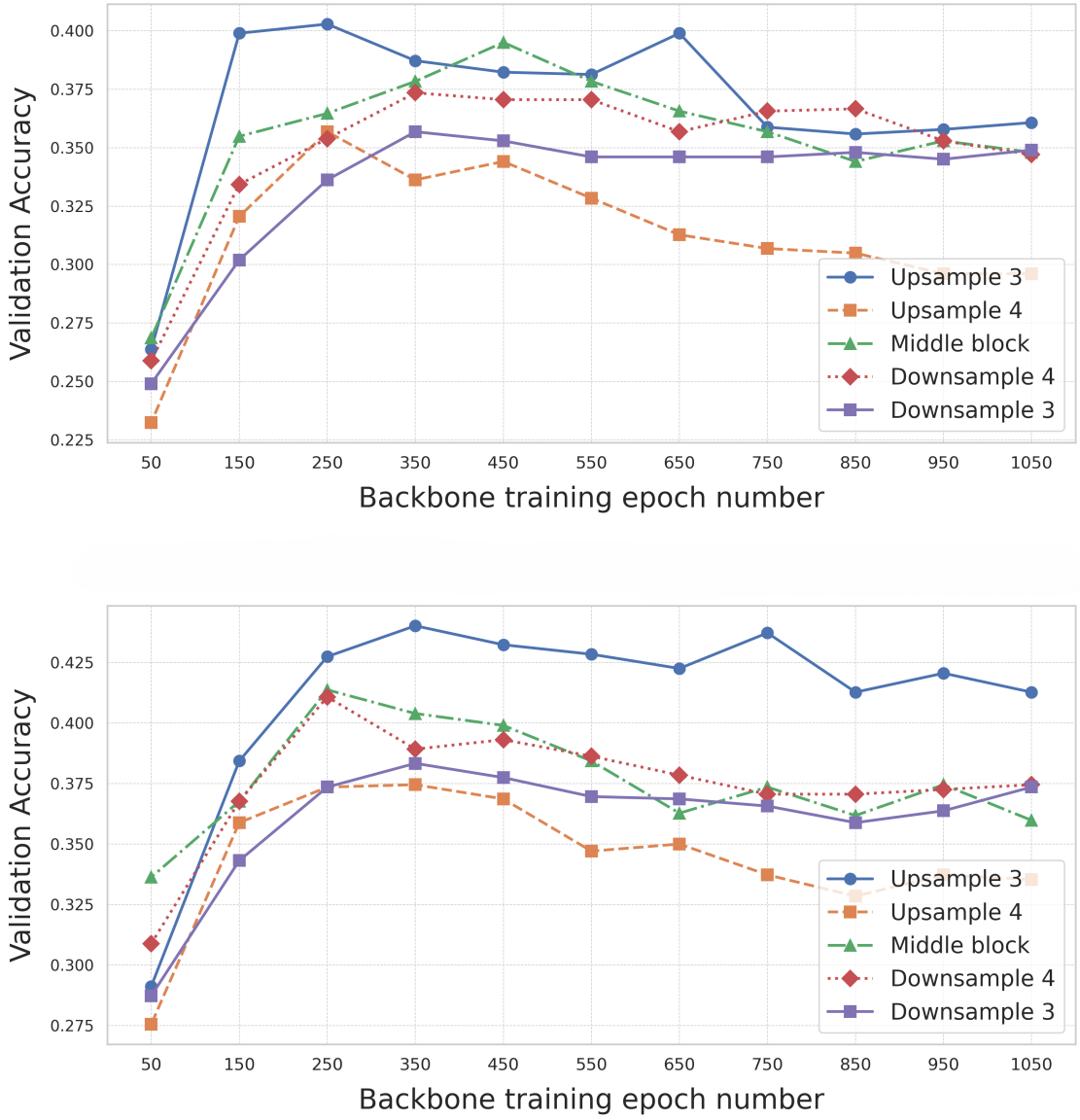

Figure 7.14: Performance of linear probe on the backbone models trained on Flowers10 dataset. At the top, with incorrect normalization, and at the bottom, with correct normalization. The Y-axis represents the validation accuracy of the linear probes, while the X-axis indicates the training epoch of the backbone model.

Chapter 8

Discussion

Testing the trained denoising diffusion probabilistic model showed that, in most cases, its classification ability was strongest during the earlier training epochs, while its generative ability was maximized only after significantly later training epochs, by which point the classification ability had already declined. These results showed that the dataset influenced which epoch yielded the best classification and generative performance for each trained model.

The classification and generative capabilities of the models indicated that the attention layer consistently made minimal contributions to performance. On average, the generative ability improved by 1.5 to 2 FID scores (5% improvement), while the classification ability saw an average increase of about 3 % in validation scores. Increasing the image size within a given dataset significantly impacted model performance. For STL-10, the best classification accuracy increased by 9.4 percent, while the FID score improved by 39.8 (15% improvement). Compared to the 32 $\times$ 32 pixel images, the 64 $\times$ 64 images, with four times as many pixels, clearly carried more information, leading to a notable increase in model performance.

One question that arose was whether adding a certain amount of noise to the classification could improve performance, given that the original U-net model (backbone) sees noisy images for most of the training. However, my tests showed that even the smallest amount of noise degraded the model's performance.

A very important question was which part of the U-net would be best suited to use its output for the classification task. For the STL-10 dataset, "Upsample 4" consistently performed best, regardless of image size or classification method. For CIFAR-10, "Upsample "4 outperformed for KNN, while "Upsample 3" was better for the linear probe. In the case of Flowers 102, "Downsample 3" worked best for KNN, while "Upsample 3" was better for the linear probe. These results suggest

that, except for the Flowers 102 dataset (where having 102 classes with only 10 training images per class may have posed challenges), upsample blocks dominated classification performance everywhere.

On the CIFAR-10 test set, linear probe fitting achieved an accuracy of 81.8%, while KNN achieved an accuracy of 81.3%. For STL-10 linear probe, a validation accuracy of 80.4% was achieved, while for KNN it was 76.5%. The backbone was not trained for classification and still gave far better results than a random guess. In addition, no optimization was performed on the classification layers and the model was not optimized for the dataset. Taking all this into account, these results are promising.

Training the flower 102 dataset proved to be much more difficult, as there were ten times as many classes and only 10 pictures per class. The model achieved an accuracy of 28.4% for KNN classification and 44.0% for linear probes. Learning many classes made the task difficult, which is the reason for the poor result of this classification. I believe better results could be obtained on this dataset by building and training a different diffusion model.

Chapter 9

Summary

The goal was to develop an unconditional diffusion model using self-supervised learning across three datasets for classification tasks. I aimed to test the resulting model's image generation and classification capabilities. The primary goal was to determine whether specific parts of the model are suitable for performing classification tasks and how certain factors influence their capabilities.

I was able to build and train diffusion models on different datasets and test these models' generative capabilities. In addition, I examined the latent spaces of the individual layers of these models using linear and KNN probing methods. I investigated how the performance is influenced by the use of attention layers, image sizes, various probing methods, and the addition of different amounts of noise to the classification images. I found out which parts of the trained models are most suitable for classification tasks in the overwhelming majority of cases.

To answer the most important question, I successfully created models using self-supervised learning methods that proved effective for classification tasks, even though they were neither trained for classification nor optimized for different datasets. Their performance significantly exceeded random choice, achieving up to 81% accuracy without any direct optimization for this objective.

Bibliography

[1] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances, 2021. URL https://arxiv.org/abs/2102.12452.

[2] Eyal Betzalel, Coby Penso, Aviv Navon, and Ethan Fetaya. A study on the evaluation of generative models, 2022. URL https://arxiv.org/abs/2206.10935.

[3] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10850–10869, September 2023. ISSN 1939-3539. DOI: 10.1109/tpami.2023.3261988. URL http://dx.doi.org/10.1109/TPAMI.2023.3261988.

[4] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018. ISSN 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.10.013. URL https://www.sciencedirect.com/science/article/pii/S0031320317304120.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

[6] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1), 2021. ISSN 2227-7080. DOI: 10.3390/technologies9010002. URL https://www.mdpi.com/2227-7080/9/1/2.

[7] Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Daniel Glasner, Ayan Chakrabarti, and Sanjiv Kumar. Rethinking fid: Towards a better evalua-

tion metric for image generation, 2024. URL https://arxiv.org/abs/2401.09603.

[8] Yu Liang, Maozhen Li, and Changjun Jiang. Generating self-attention activation maps for visual interpretations of convolutional neural networks. *Neurocomputing*, 490:206–216, 2022. ISSN 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.11.084. URL https://www.sciencedirect.com/science/article/pii/S0925231221017628.

[9] Antonio Mucherino, Petraq J. Papajorgji, and Panos M. Pardalos. k-Nearest Neighbor Classification. pages 83–106, June 2009. DOI: 10.1007/978-0-387-88615-2. URL https://ideas.repec.org/h/spr/spochp/978-0-387-88615-2_4.html.

[10] Alejandro Newell and Jia Deng. How useful is self-supervised pretraining for visual tasks?, 2020. URL https://arxiv.org/abs/2003.14323.

[11] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas, 2018. URL https://arxiv.org/abs/1804.03999.

[12] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

[13] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 759–766, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. DOI: 10.1145/1273496.1273592. URL https://doi.org/10.1145/1273496.1273592.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL https://arxiv.org/abs/1505.04597.

[16] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/sohl-dickstein15.html.

[17] Weilai Xiang, Hongyu Yang, Di Huang, and Yunhong Wang. Denoising diffusion autoencoders are unified self-supervised learners, 2023. URL https://arxiv.org/abs/2303.09769.

[18] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention, 2016. URL https://arxiv.org/abs/1502.03044.