Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics

Evaluating Learned Representations of Denoising Diffusion Models for Predictive Tasks

Master's Thesis

Author
Márton Gergely Herkules

Advisor
Dr. Bálint Gyires-Tóth
András Kalapos
András Béres

May 23, 2024

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Herkules Márton Gergely*, szigorló hallgató kijelentem, hogy ezt a diplomaterv 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. május 23.

Herkules Márton Gergely
hallgató

Kivonat

A képgeneráló modellek napjainkban való elterjedtsége és fontossága több tényezőnek köszönhető. Az egyik legfontosabb oka, hogy a mesterséges intelligencia technológiák fejlődése lehetővé teszi, hogy a felhasználók könnyen hozzanak létre egyedi képeket, amelyek korábban csak hivatásos művészek képességei révén voltak elérhetők. Ezek a modellek számos lehetőséget kínálnak a felhasználók számára, hogy saját képeket alkothassanak és módosíthassanak pillanatok leforgása alatt.

Egyik igen érdekes felmerülő kérdés az, hogy ezek a modellek használhatóak-e képek generálásán kívül diszkriminatív feladatokra is, például képek klasszifikálására? Ha pedig alkalmasak erre, akkor érdemes-e ezeket a modelleket erre használni? Megfelelő a teljesítményük ahhoz, hogy versenyezni tudjanak a korszerű klasszifikációs modellekkel, amiknek ez az elsődleges és egyetlen feladatuk?

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

Ezen felül vizsgálom, hogy a model egyes rétegei által generált látens tér reprezentációi mennyire alkalmasak klasszifikációs feladatokra. Mindezt a már betanított súllyok segítségével és az arra való lineáris vagy KNN klasszifikáció illesztésével hajtom végre az eredeti adatbázisokon.

Abstract

Today's widespread use and importance of image-generating models can be attributed to several factors. One of the most significant reasons is that the advancement of artificial intelligence technologies allows users to easily create unique images, which were previously only achievable through the capabilities of professional artists. These models offer numerous opportunities for users to create and modify their images in seconds.

An interesting emerging question is whether these models can be used for tasks other than image generation, such as image classification. If they are suitable for this purpose, is it worth using these models? Is their performance good enough to compete with current classification models, which have this as their primary and only task?

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

Additionally, I test the model to see how suitable the latent space representation of the images generated by its individual layers is for classification tasks. I do all this with the help of already trained weights and by fitting the linear or KNN classification to the original databases.

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 representation and knowledge to solve predictive tasks may provide a new, effective paradigm for pre-training machine learning models on visual data.

My goal is to create and train an unconditional denoising diffusion probability model on two different datasets and test its image generation capability. I also need to analyze the model's internal representations of the images it receives as input, as this shows whether these latent space representations are suitable for classification tasks.

During my work I train an unconditional denoising diffusion probability model on the CIFAR 10 and Flower 102 datasets without using their predetermined labels. Thus, models need to learn the properties of each input image in a self-supervised learning way. In this approach, I can test the ability to discriminate between classes based on their representations in latent space alone. I fit a classifier to certain layers of the model using linear probe and KNN classification, which I perform in a supervised way to evaluate the ability of the model to extract the descriptive feature from the input image. Finally, 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 dataset 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. Finally, I will summarise the results and discuss future works.

Chapter 2

Literature research

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.

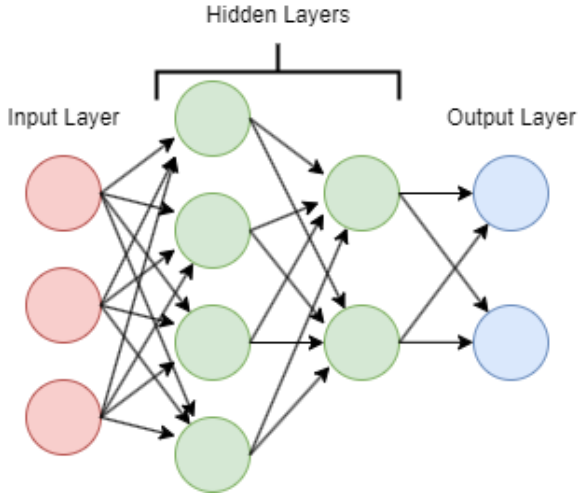

Figure 2.1: Basic neural network

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.

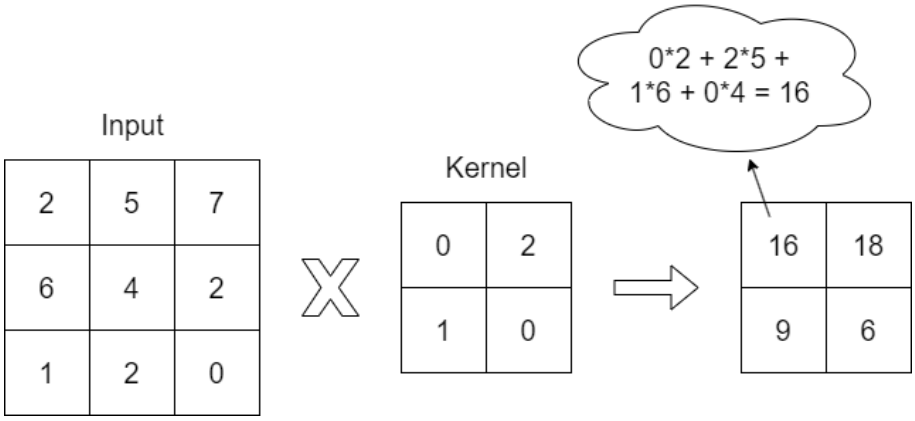

Figure 2.2: Convolution process

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. This network was designed to leverage less data effectively while maintaining speed and accuracy.

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.

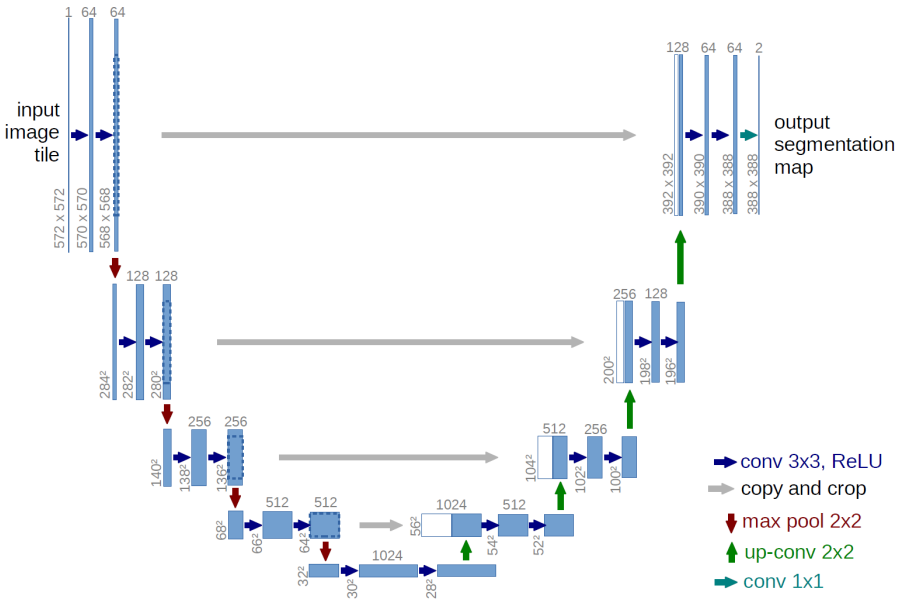

Figure 2.3: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The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

2.4.1 Soft Attention

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 fig 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.

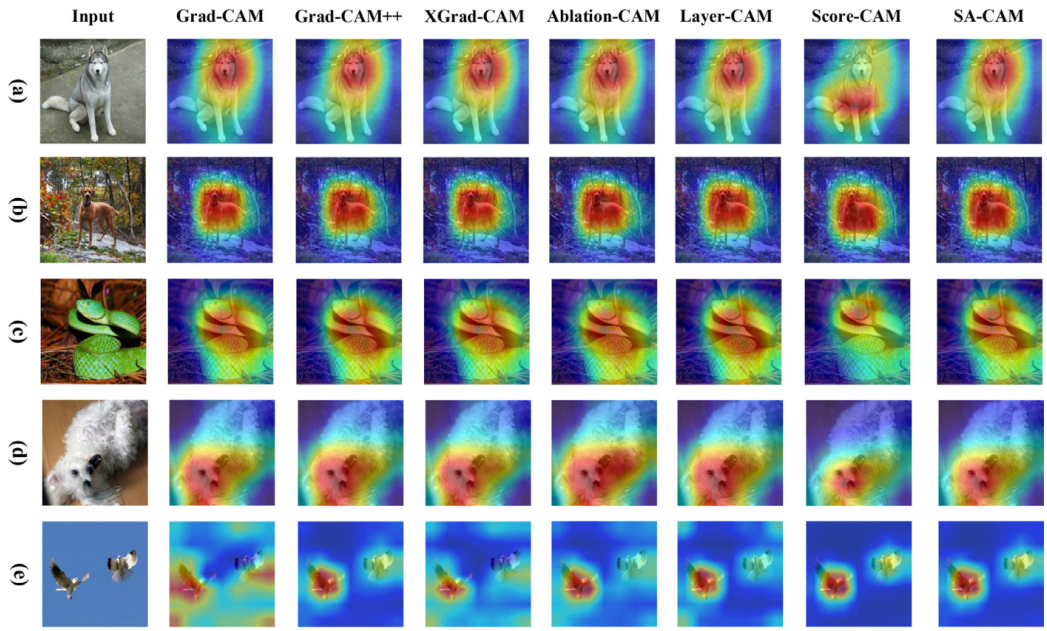

Figure 2.4: In the paper "Generating self-attention activation maps for visual interpretations of convolutional neural networks" [8], examples of how different models' attentions are on different images.

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, spatial information recreated is 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, varied-quality, 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.

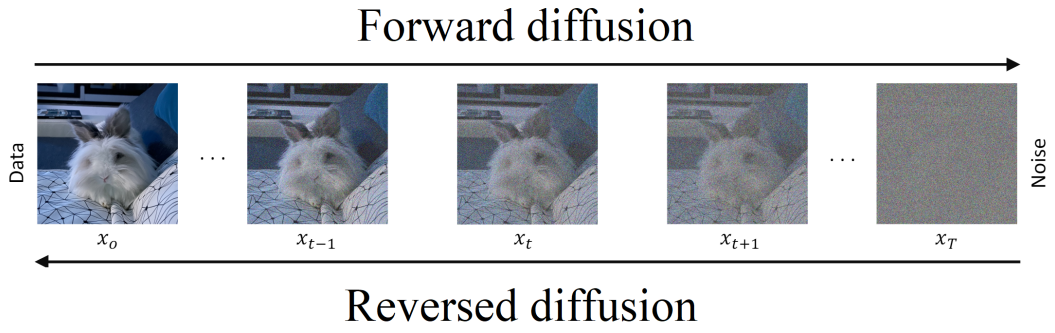

Figure 2.5: Diffusion process [3]

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.

2.6.3 Cumulative maximum mean discrepancy

The Cumulative Maximum Mean Discrepancy (CMMD) [7] is a metric designed for evaluating the quality of images generated by generative models, such as diffusion models, with several advantages over traditional metrics like the FID. CMMD is

particularly useful for evaluating progressive image generation models, where the quality of generated images improves over iterations.

One of the key advantages of CMMD over FID is its sample efficiency. Calculating FID requires estimating a large covariance matrix, which necessitates a large number of images, making FID less efficient in terms of sample size. In contrast, CMMD can provide consistent estimates even with small image sets, making it more practical for fast online evaluation during model development and for comparing a large number of models without the need for generating a massive dataset.

Another significant advantage of CMMD is its behavior under different sample sizes. Unlike FID, which may not reflect gradual improvements in iterative models accurately, CMMD monotonically improves (decreases) with the progression of iterations. This behavior is particularly beneficial for evaluating models like Stable Diffusion, where the quality of generated images improves over time. CMMD's ability to accurately reflect these improvements makes it a more reliable metric for assessing the quality of generated images.

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.

2.8 K-Nearest neighbors classification

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.

Despite its advantages, KNN faces challenges such as sensitivity to outliers and the curse of dimensionality, where the increase in the number of features makes computing distance and finding the nearest neighbors in high-dimensional space computationally more expensive.

2.8.1 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.

2.9 Probing classifiers

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.

Probing classifiers typically involves training a separate classification model on top of the pretrained model's representations. This additional classifier is trained to predict specific properties or features in the generated latent space. 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.

2.9.1 Linear Probe with SSL

A linear probe involves training a simple linear classifier on features extracted by a pretrained model. Once the SSL model is trained, it can 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 of that chosen layer. These output features serve as inputs for training the simple linear classifications layer on the labeled dataset.

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.

2.10 Article in focus

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:

2.10.1 Motivation

The challenge in machine learning of managing vast amounts of data with limited human annotations has prompted a shift from supervised to self-supervised pretraining, 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.

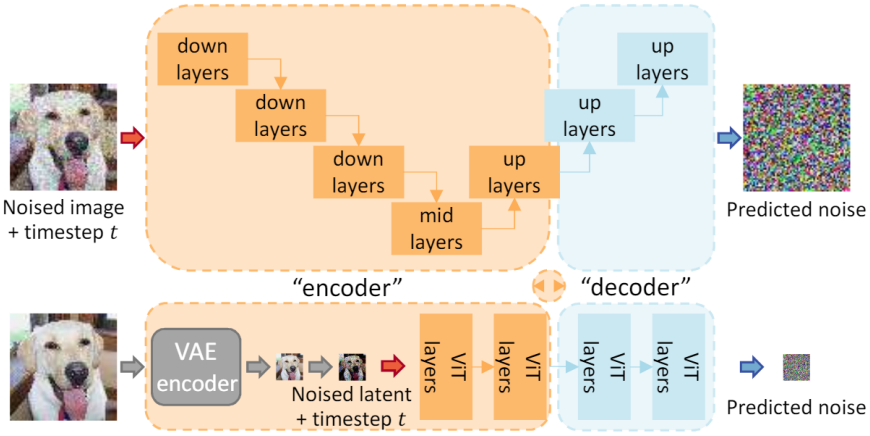

Figure 2.6: Denoising Diffusion Autoencoders (DDAE). Diffusion networks are essentially equivalent to level-conditional denoising autoencoders (DAE). The networks are named as DDAEs due to this similarity. The image is from the paper "Denoising Diffusion Autoencoders are Unified Self-supervised Learners"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

2.10.2 Evaluation

Evaluating discriminative learning capabilities in Denoising Diffusion Autoencoders (DDAEs) is challenging due to the complexity of extracting compact, classifiable in-

formation. 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.

2.10.3 Results

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 using self-supervised learning on two different datasets that can also be used for classification. I intend to test the resulting model's image generation capability using different methods.

Given that the main task is to determine whether specific parts of the model are suitable for performing classification tasks, I plan to use different parts of the model to test its ability to generate image representations.

Previous research and hypotheses suggest that the model's classification and generative abilities do not increase or decrease in parallel during training. Therefore, I plan to use checkpoints created during training to test the model at different stages until I find the most efficient scenario. In addition, I intend to test the parameters that influence the model's performance to achieve the best possible results.

My goal is also to test whether the model performs better with noisy or clean images since, during basic U-net training, the model encounters many noisy images.

Chapter 4

System design and methods

During my work, I have three main tasks. Training a U-net using unlabelled images. Generative evaluation of the neural network. To freeze the weights of the trained U-net model and use the output of each block from it for classification in two ways.

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 4.1. 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.

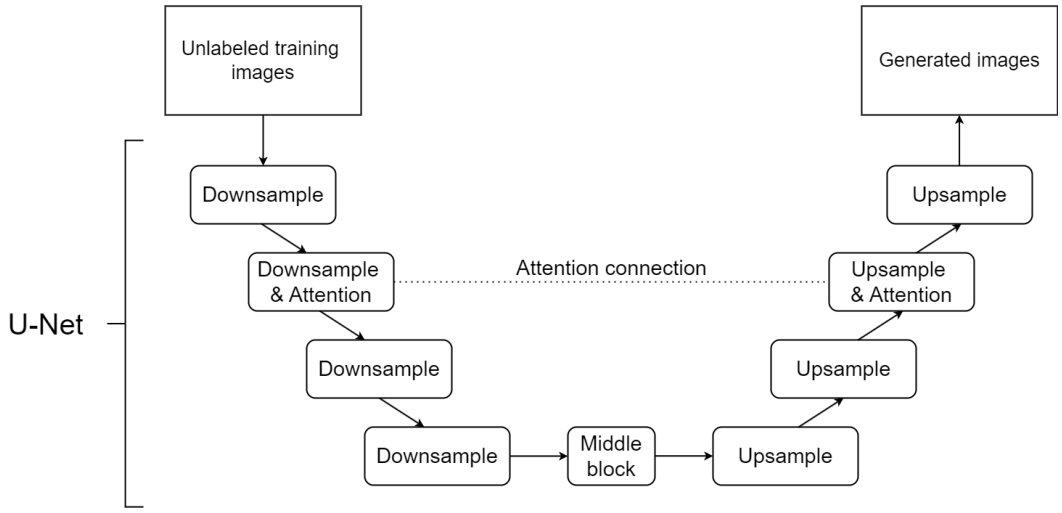

Figure 4.1: Training U-net on unlabeled data

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 4.2.

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.

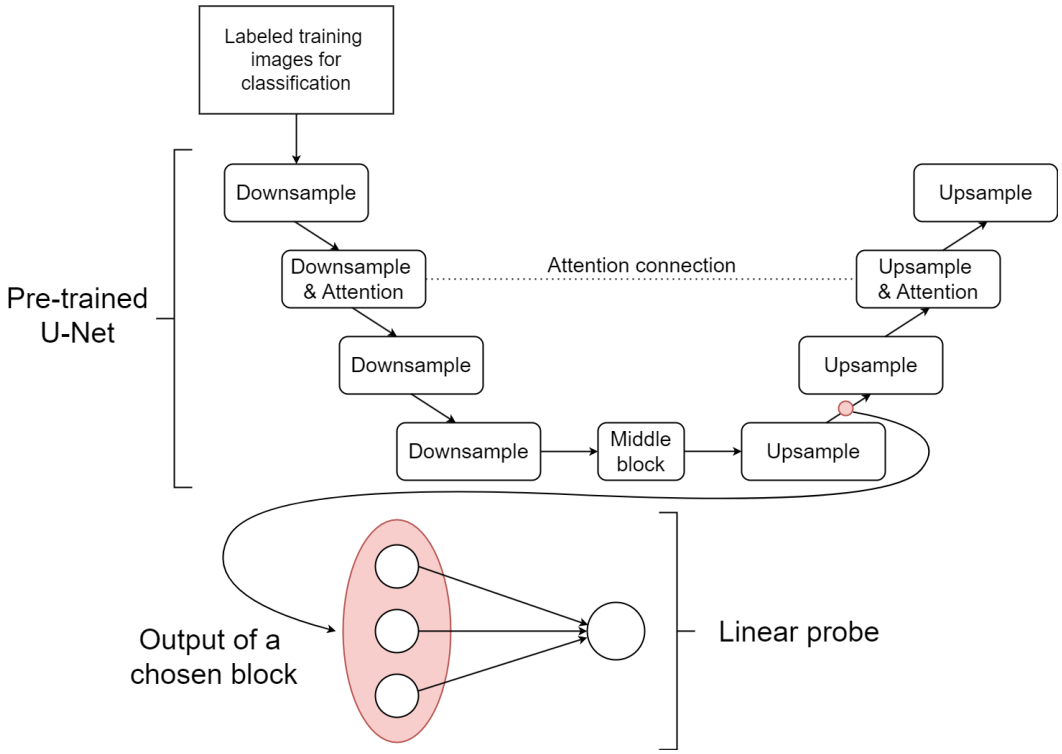

Figure 4.2: Linear probe

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.

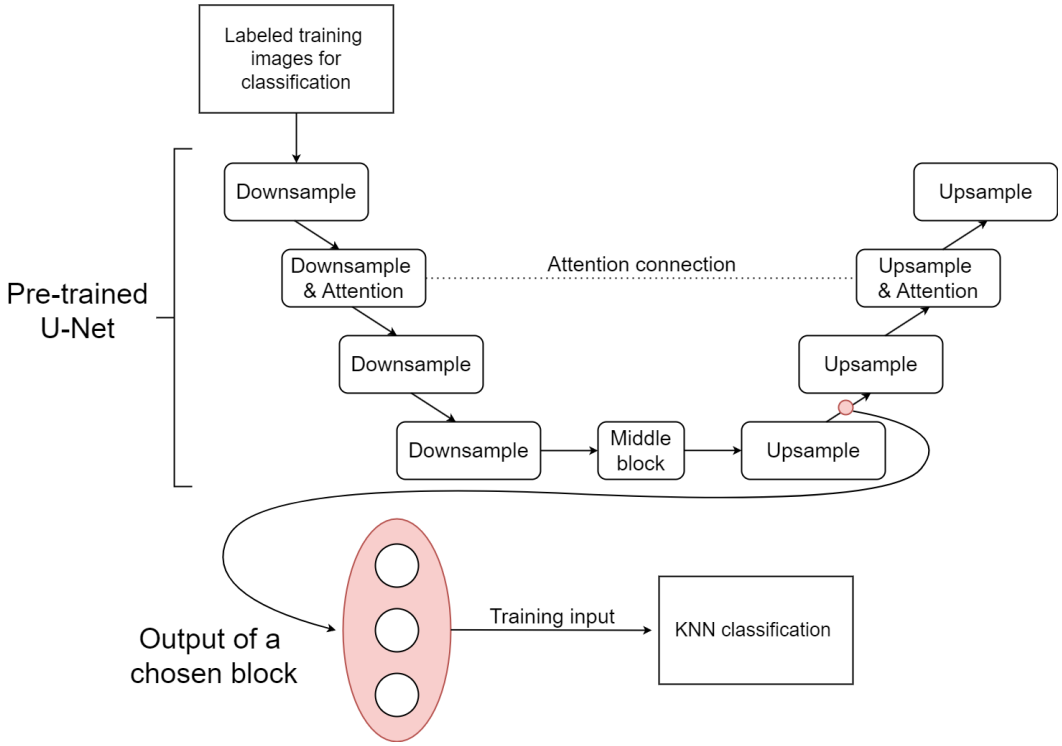

Figure 4.3: K-Nearest neighbors classification

Chapter 5

Datasets

I worked with two data sets, CIFAR 10 (figure 5.1) and Flowers 102 (figure 5.2). 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 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 32x32 color images divided into 10 different classes. 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.

5.2 Oxford 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. The images within the dataset exhibit significant variations in scale, pose, and lighting conditions.

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).

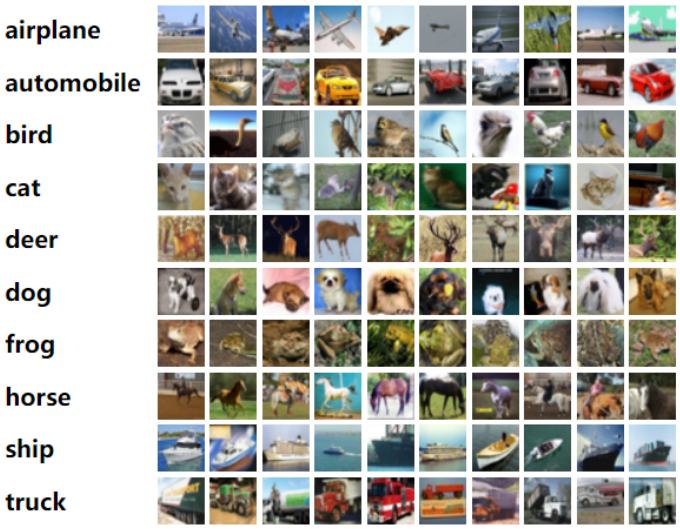

Figure 5.1: CIFAR 10

Figure 5.2: Flowers 102

Chapter 6

Implementation

6.1 Denoising Diffusion Probabilistic Model

My 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. 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.

6.2 DDPM model training

6.2.1 Data pre-processing and data loading

6.2.1.1 CIFAR 10

The dataset's training partition contained 50000 images, with 5000 32*32 pixel images per class. Unfortunately, given the resources, I was not able to use such a large dataset, as it would have taken too much time to train the model, and I was not the only user of the host machine. For this reason, I selected 1280 images from each class and 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.

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.

6.2.1.2 Flowers 102

This dataset does not cause problems with excessive runtimes, as it does not have nearly as many images as CIFAR 10. The only problem here is the different sizes of the images, so it was necessary to convert the images to one size in the pre-processing steps.

6.2.2 Training process

During the training I used AdamW optimizer with a learning rate of 0.0001. The process lasted for 2000 epochs, while a given epoch was done using 1000 steps forward diffusion for both datasets. During the process, an image was generated every 20-50 epochs to show what output the model could produce at the current state of the pipeline. However, every 100-150 epochs, I saved the weights of the model so later testing could be performed.

For images with 32*32 pixels I was able to use 320 batch sizes while training, but for images with 64*64 pixels only 20 batch sizes were possible. It's true that the images were 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 attension layers, they were 16 times the size. In practice, this meant that 20pcs of 32*32 images used 2.5 gigabytes of memory, while 64*64 images used 40. Approximately with both datasets the process took 12 hours.

6.3 Testing 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. Since it was specified in the definition that a 256-deep representation was expected, we got back a 1,256,4,4 tensor. The part (4,4) depends on the size of the processed image. For 64*64 images, the block returns (1,256,8,8) tensors. Given the hierarchical convolutional neural networks discussed at section 2.2, one possible consequence is that the given input is displayed in an

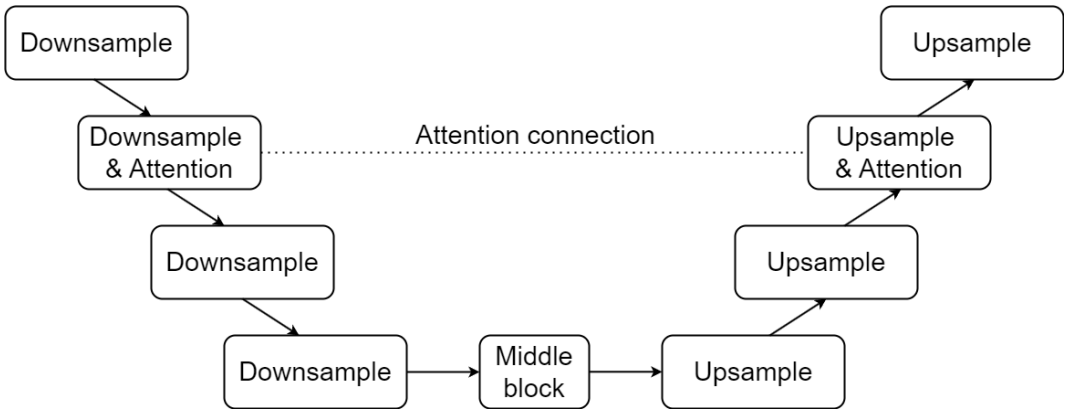

Figure 6.1: My simplified U-net model

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.

6.3.1 Testing 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. Depending on the dataset, the classification layer is trained for 10 or 40 epochs using the Adam optimizer and a learning rate of 0.001. In addition, the linear probe layer has not been further trained or optimized to see how representative the latent space representations of each block are.

6.3.2 Testing 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. 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

Evaluation

7.1 DDPM image generation

I was able to train the model to generate proper images, as shown in figures 7.1. I tested the images generated by my model in different training epochs on both datasets using the FID metric as explained in section 2.6.1.

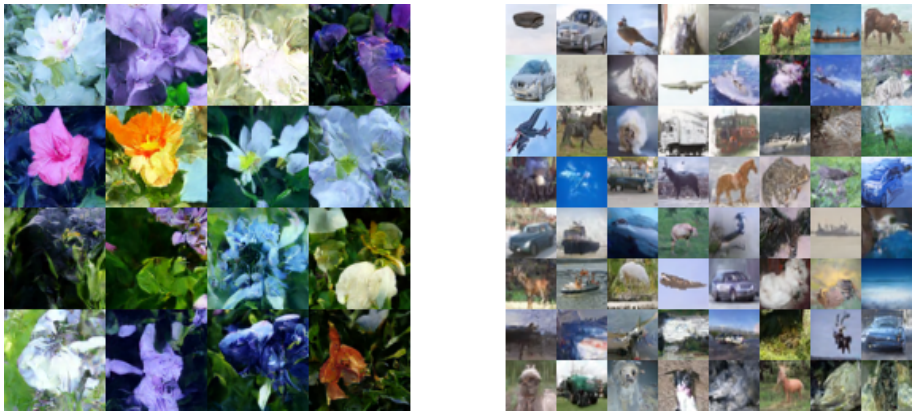

Figure 7.1: Diffusion model generated images based on Flowers 102 (left) and CIFAR 10 (right)

7.1.1 CIFAR 10

I compared the 512 images generated by the model with the 10000 test images provided by CIFAR10. I tested the model at different epochs, which showed that after the 1000th epoch, the model began to rapidly overfit the training data, which made its generalization ability worse, as shown in figure 7.2. Overfitting happens when the model starts to learn the data "too well". In the case of overfitting, the model sees the images so many times during training that it no longer memorizes

the features but learns the whole image perfectly so that when it generates the output later, it does not produce a similar image based on the learned features, but the training images themselves, which have already been memorized due to overtraining.

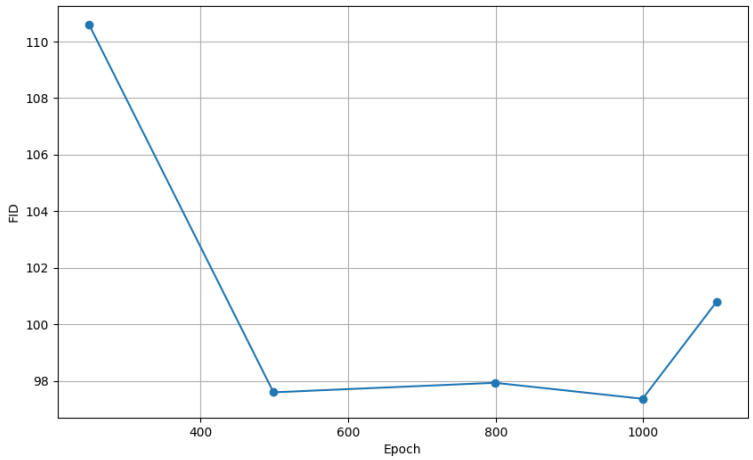

Figure 7.2: Diffusion model FID results on the CIFAR 10 test dataset at different training epochs.

7.1.2 Flowers 102

I compared the Flowers 102 test dataset with the 128 images generated by the model. As shown in figure 7.3, the model had the best generalization ability in epoch 1050, which then deteriorated rapidly due to overfitting in the following training steps in a similar way to the CIFAR 10 dataset.

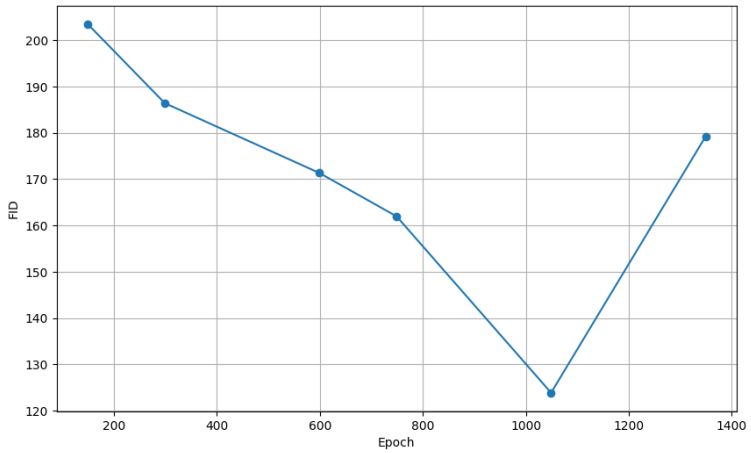

Figure 7.3: Diffusion model FID results on the Flowers 102 test dataset at different training epochs.

7.2 DDPM classification

I examined all 9 blocks of the model, but only 5 of them proved to be worthy of classification tasks based on the results, so I 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 7.4.

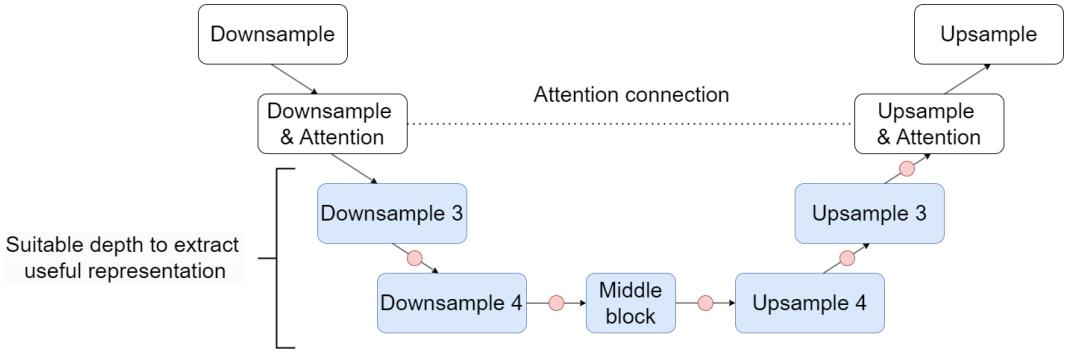

Figure 7.4: My simplified U-net model (the blocks tested are marked in blue and the values of the outputs marked in red are used for classification)

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, as 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 probe with CIFAR 10

The "Downsample 3" and "Downsample 4" blocks in the encoder part performed the worst with validation accuracies of 0.59 and 0.65. The middle block performed better, with a validation score of 0.67. In contrast, the "Upsample 4" and "Upsample 3" blocks in the decoder section achieved the best results with a validation accuracy of 0.699, as shown in figure 7.5.

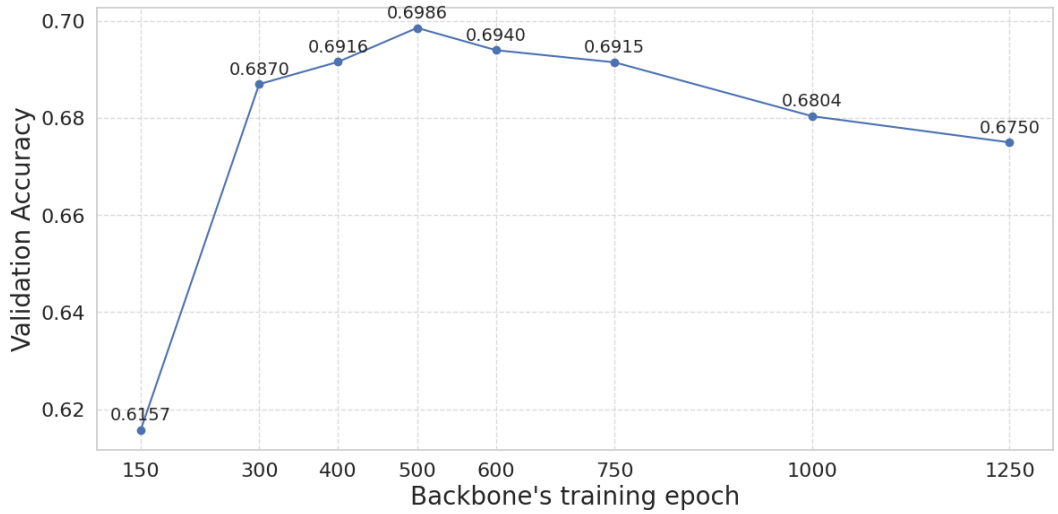

Figure 7.5: Performance of Upsample 3 on CIFAR 10 dataset using linear probe on the backbone model. The Y-axis is the validation accuracy of the linear probe. The X axis shows which epoch of its own training the backbone model was in.

7.2.1.2 Linear probe with Flowers 102

"Upsample 4" had the worst result; the U-net encoder layers (Downsample 3,4) performed better with 0.34 validation accuracy after training, while the middle and "Upsample 3" blocks performed best with about 0.373 scores as shown in figure 7.6.

7.2.1.3 KNN classification on CIFAR 10

The performance of the different blocks is almost identical for this task, but with a top-1 validation accuracy of 0.7378, the "Downsample 4" block had the best performance. 7.7.

7.2.1.4 KNN classification on Flowers 102

A KNN bound to the output of two blocks in the encoder layers (Downsample 3,4) gave a validation accuracy of 0.222 during training. The decoder layers gave slightly worse results, with a value of around 0.21. Meanwhile, the middle block was best suited to the task with a validation accuracy of 0.2335.

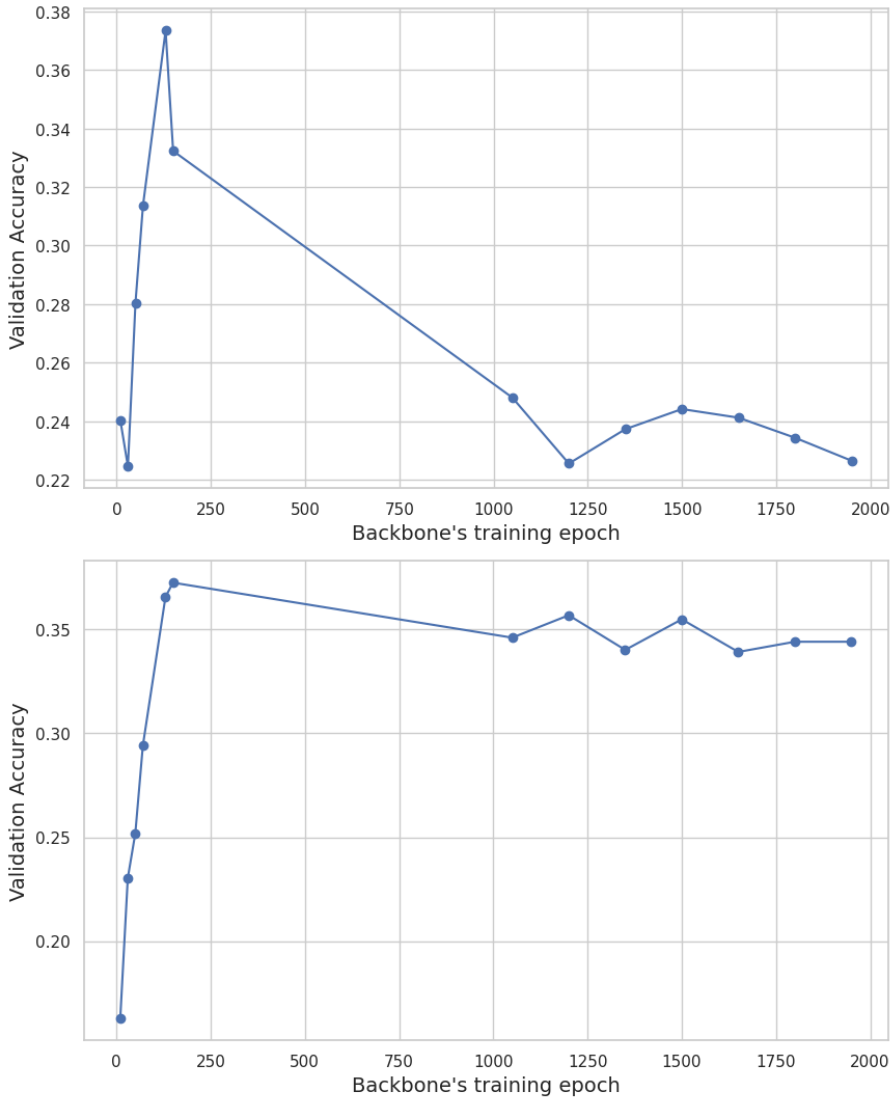

Figure 7.6: Performance of middle block (top) and upsample 3 (bottom) on Flowers 102 dataset using linear probe fitted to the frozen weights of the originally trained diffusion model at a given epoch. The Y axis is the validation accuracy of the linear probe. The X-axis is the number of epochs of the frozen backbone.

7.2.1.5 Overfitting

In testing, it can be observed that the "Upsample 3" block is much less exposed to the adverse effects of overfitting than the middle block. As seen in picture 7.6, the performance of the middle block deteriorates much more when the model is overtrained. It can be seen that "Upsample 3" does not lose nearly as much of its ability to generate a good latent space representation in the 1950th epoch of the U-net basic training as the middle block, which drops to two-thirds.

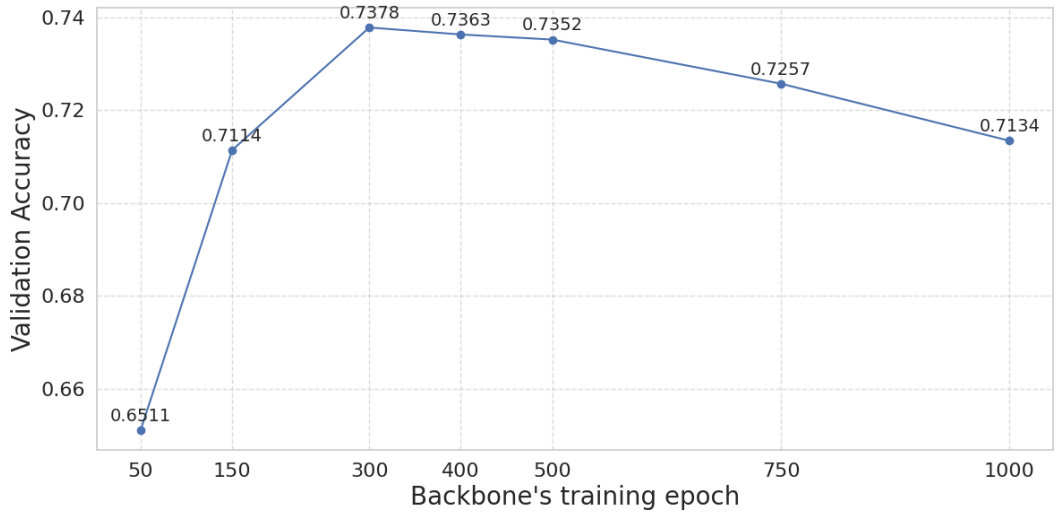

Figure 7.7: KNN using the output of the "Downsample 4" block on the different epoch states of the original U-net training as backbone with the CIFAR 10 dataset.

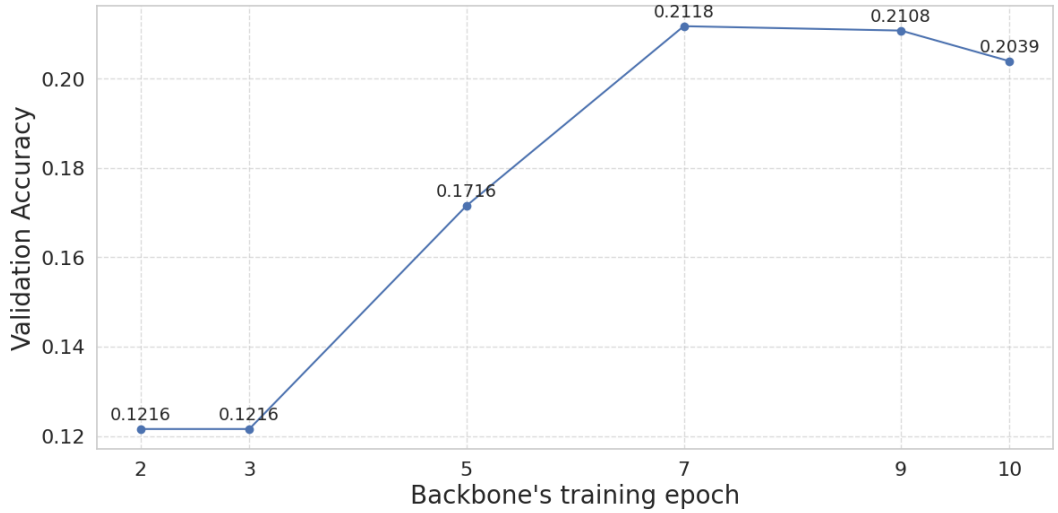

Figure 7.8: KNN using the output of the middle block on the different epoch states of the original U-net training as backbone with the Flower 102 dataset.

7.2.2 Best performing epoch

An important consideration is at which step of the training the original backbone was able to generate the most suitable latent space representation for classification. During the training of the CIFAR 10 dataset, the linear probe achieved the best results in the 500th training epochs, while the KKN classification achieved the best results between the 300th and the 500th training epochs.

In the Flowers 102 dataset, the linear probe was strongest between epochs 130 and 150, while the KNN started to deteriorate after the very early 6-8th epochs.

7.2.3 Noise during classification

During the original U-net training the forward diffusion was applied to the images in 1000 steps. In the classification phase adding even the smallest amount of noise as early as the 10th step out of 1000 significantly degraded the models' classification ability. It only reduced the performance in all cases.

Chapter 8

Interpretation of results

Testing the trained denoising diffusion probabilistic model showed that it was possible to train the model until it overfitted the training data. It could be seen that on both datasets, the generative capabilities of the model improved until around epoch 1000. In contrast, the latent space generating ability of the model for classification began to deteriorate in the much earlier. For linear probe, this number was 500 for the CIFAR dataset and 130 for Flowers 102. In the case of KNN, the Flower 102 dataset started to perform worse after 8 epochs, but the fact that the model only saw 10 images per class during learning may play a big role in this. This clearly shows that when training a diffusion model, the quality of the latent space generation is maximal at a different epoch than the image generation capability.

In testing, it can be observed that the "Upsample 3" block is much less exposed to the adverse effects of overfitting than the middle block, as shown in figure **??**. I hypothesize that a significant factor in this is that, for the reasons explained in the previous section 2.2, the output of the middle block is a [256, 8, 8] dimensional latent space representation. In contrast, the output of the "Upsample 3" block, located in the network's higher layers, is [256, 32, 32] dimensional. This contains significantly more information, which makes it more difficult to overfit.

A very important question was which part of the U-net would be best suited to use its output for the classification task. Unfortunately, I could not find a U-net part that would be universally best in all cases since different layers proved to be optimal for two different methods and two datasets. Even within the methods, the layers corresponding to the best values differed.

8.1 CIFAR 10 result

On the test set, linear probe fitting achieved an accuracy of 0.699, while KNN achieved an accuracy of 0.7378. The backbone was not pre-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.

8.2 Flower 102 result

Training the flower 102 dataset proved to be much more difficult, as there were ten times as many classes and only ten pictures per class. The model achieved an accuracy of 0.2335 for KNN classification and 0.373 for linear probes. Learning a lot of 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. Unfortunately, in this case, this result cannot compete with the currently used models created for classification tasks.

Chapter 9

Summary

9.1 Goals

My goal was to build and train an unconditional denoising diffusion probabilistic model using two different datasets. In addition, I had to analyse the different layers of the model to see which of the generated latent space representations is suitable for classification tasks.

9.2 Work completed

During the semester, I built and tested this model. I generated images with it and analysed its generative abilities in different training phases.

I fit linear probing and KNN classification to the different layers of the original U-net model, allowing me to test the model's latent space generating ability at different layers and epochs of the original training process. I then evaluated the classification models and compared their performance.

I examined how the latent space and image generating capabilities of the original model evolve relative to each other.

9.3 Opportunities for further development

- Compare models on similar datasets.
- Use a diffusion model with a different structure.
- Fine-tune the classification models to improve performance.

Bibliography

[1] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances, 2021.

[2] Eyal Betzalel, Coby Penso, Aviv Navon, and Ethan Fetaya. A study on the evaluation of generative models, 2022.

[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.

[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 evaluation metric for image generation, 2024.

[8] Yu Liang, Maozhen Li, and Changjun Jiang. Generating self-attention activation maps for visual interpretations of convolutional neural net-

works. *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, Panos M Pardalos, Antonio Mucherino, Petraq J Papajorgji, and Panos M Pardalos. K-nearest neighbor classification. *Data mining in agriculture*, pages 83–106, 2009.

[10] Alejandro Newell and Jia Deng. How useful is self-supervised pretraining for visual tasks?, 2020.

[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.

[12] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[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.

[16] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[17] Weilai Xiang, Hongyu Yang, Di Huang, and Yunhong Wang. Denoising diffusion autoencoders are unified self-supervised learners, 2023.

[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.