POLITECNICO DI TORINO

Department of Computer Engineering, Cinema and Mechatronics

MSC in Data Science and Engineering

Master Thesis

DreamShot: Teaching Cinema Shots to Latent Diffusion Models

Exploring the diffusion model architecture and finetuning for cinematic shots generation

Supervisor:
prof. Tania Cerquitelli

Correlatore:
Dr. Bartolomeo Vacchetti

Candidate:
Tommaso Massaglia

Luglio 2023

Alla mia famiglia,
che mi ha reso quello che sono,
e ai miei amici,
che non mi hanno mai lasciato mentre lo diventavo.

The best solution to a problem is usually the easiest one!
GLaDOS

Acknowledgements

A special thanks to Professor Cerquitelli and Dr. Vacchetti for being incredibly supportive and encouraging in my endeavours. A huge thank you also goes to all the 55 people who partake in the survey giving even more value to this work, I'm sorry I couldn't thank you one by one. A mention as well to the community that formed around this wonderful subject for producing some wonderful material and tutorials which helped me understand the subject better.

Abstract

This thesis work presents a comprehensive overview of recent advancements in image synthesis models, exploring the recent developments of Diffusion Models [1] and their finetuning. The primary contribution consists in a novel approach that utilizes recently released techniques to tackle a relatively unexplored area in the literature: generating cinema-like shots to assist in the storyboarding process. Starting from the intuition that shot types can be learned as an artistic style, a fine-tuned version of Stable Diffusion [2] is leveraged to tailor the generation process specifically for this purpose. By utilizing a limited number of movie frames labelled with shot types and accompanied by brief descriptions, I use Dreambooth [3] along with Low Rank Adaptation [4] to teach a pre-trained model three specific shot types: close shot, medium shot, and long shot. Moreover, this approach is designed to run efficiently on low-power devices. The result is qualitatively more pleasing images that more closely align with the provided prompts and shot types. This improvement is then validated through a survey conducted on human subjects, in addition to an evaluation carried out using a setup similar to the one proposed in [3], demonstrating an increase in both CLIP-T and DINO scores, with the latter exhibiting a significant improvement compared to the baseline. A detailed and easily reproducible method for creating a dataset for finetuning purposes is presented along with the main matter, allowing, for example, to teach a specific filmmaker style. Finally, the impact of different generation parameters on the generative process is explored and comparisons between the traditional and this method of storyboarding are shown. In this thesis work, I show a method that produces improved output quality, increased adherence to shot types, and enhanced expressiveness in the generation of cinema-like shots, making it a valuable tool for the filmmaking industry and creative individuals.

Contents

Chapter 1

Introduction

In recent months, the transformer [5] based ChatGPT [6] became a worldwide phenomenon for its ability to answer pretty much any question that a user would throw its way, once again bringing Artificial Intelligence the forefront of news outlets and other mainstream media. Although born from the same *resurgence* that transformer-based models brought to the world of deep learning, what didn't raise as much noise was the release and *diffusion* of the most recent Diffusion Model [1] based text-to-image synthesis models, of which the three main actors, all released throughout the summer of 2022, are DALL-E 2 [7], Imagen [8], and the open source Stable Diffusion [2]. These three models are capable of using a textual input provided by the user to generate images that are semantically close to the text, and visually close to real images, often making it hard to distinguish between what's real and what's AI-generated.

Having the ability to generate realistic images and to guide the generation process has many applications within it, applications which still haven't been explored much yet; by using the available techniques it would be possible for example to generate automatically an ad campaign for a new product, to edit photos by only telling what edit we wanted, or even depict ourselves in many different styles and places. Of the many fields that widely use and creates reference pictures to improve the workflow, cinema is one of the more egregious. By having the ability to generate realistic pictures, generating reference pictures that show expressively an idea of the desired shot becomes suddenly a task open to anyone, without requiring the need of an extensive library of reference shots or the ability to draw the desired picture. These reference shots and sketches are often used in storyboarding, a widely spread

technique in filmmaking that helps in visualizing the story and in guiding the shootings better, resulting in more efficient work. When making reference shots, one of the most important aspects of it is the desired shot type, as it often is recognized as the element that most influence the viewer's focus and emotions [9].

In this thesis work, I propose an approach that makes use of an existing Diffusion Model (Stable Diffusion) and specific finetunings to generate film-like pictures with a determined shot type. Throughout the work we show the application of state-of-the-art techniques and discuss different aspects of the generation process, creating in the process a compendium that contains most of the information necessary to understanding fully the image synthesis field, from its inception to today.

1.1 A brief history of Image Synthesis models

The field of image synthesis through the use of a generative model effectively started with the advent of Variational Autoencoders (VAE) 3.2 and Generative Adversarial Networks (GAN) 3.1 in 2013/2014.

Variational Autoencoders are a class of probabilistic generative models that learn in an unsupervised manner. The model learns to effectively compress an input into a lower dimensional latent space (bottleneck) and then reconstruct the original input. The resulting model can then be used in two modes: by feeding a noisy image, it's possible as an example to reconstruct the original one, by giving it random noise instead, it's possible to generate novel images of the training domain.

The Generative Adversarial Network approach instead consisted in using two networks, a generator and a discriminator, to try and learn the underlying distribution (of pixels in this case) of the training domain. The generator's role was to, given a random noisy input, adapt the input to fit the target domain. The discriminator role was that of a binary classifier, a well-researched subject, whose goal was to determine whether the generated image was real or fake. Once training was complete, the generator could be used alone to synthesise images from the training domain.

For the next five years, most of the image generation landscape focused on GANs as the de-facto best method for image generations, with contributions such as 3.3, which explored for the first time text-conditioned image generation, StyleGAN 3.4, which revolutionized the GAN architecture and for the

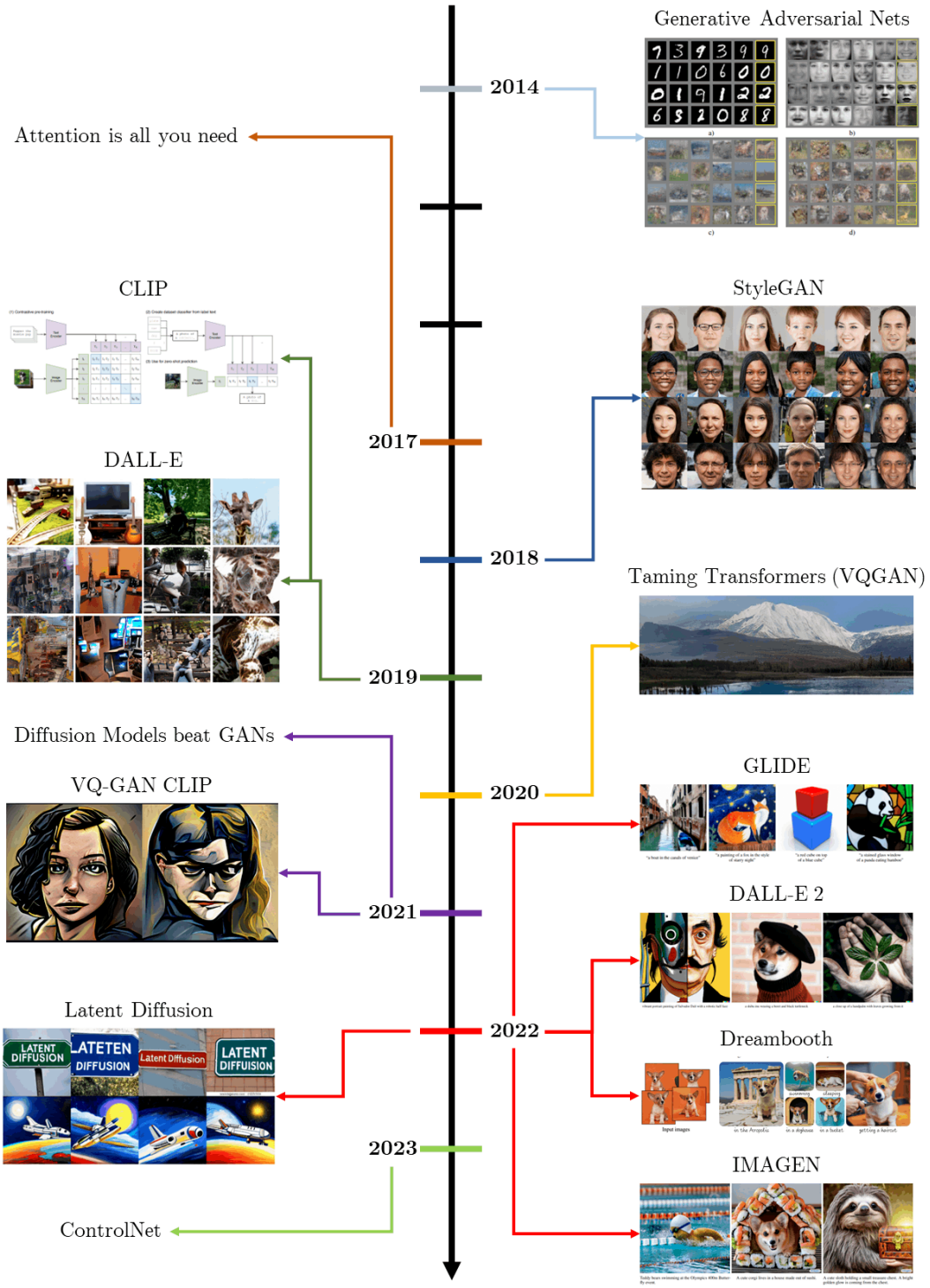


Figure 1.1: The timeline visualized.

first time achieved truly realistic image synthesis[1], and in the field of VAEs, VQ-VAE 3.11, which introduced an approach that allowed for even greater compression of the input into a latent space by using a discrete representation.

In 2017, the publication of the paper Attention is All You Need [5] revolutionized the field of Natural Language Processing forever. The Transformer 2.1.2 architecture enabled models to develop a deep understanding of the underlying structure of a language by determining which word each other word has to *attend*[2] to most. Not only NLP though, transformers saw very successful applications even in the field of computer vision for their ability to mimic the CNN architecture 2.2. By exploiting the transformer's ability to learn the relationships between elements, the GAN architecture, and the codebook representation of VQ-VAEs 3.5, the paper Taming Transformers for High-Resolution Image Synthesis 3.6 introduces transformers to the image generation field, allowing for high resolution and photorealistic generations, along with versions that accept different forms of conditioning, such as pose, depth maps, and image completion.

In June 2021, *Diffusion models beat GANs on image synthesis* [10] shows the potential that diffusion models 2.5, a class of probabilistic generative models that learn to reverse the so-called *diffusion process*, have in the image generation task, introducing a third actor in the image synthesis landscape that would later show major developments. Following in the footsteps of VQ-VAE, released in July 2021 together with the CLIP 2.3, an architecture which allows encoding images and text in a similar latent representation, DALL-E 3.13 was released. By composing text and image embeddings into the same data stream, DALL-E is trained to complete sentences with the missing image using an autoregressive generation typical of the transformers decoder architecture, all by using the predicted image embeddings as the base from which a VAE reconstructs the image.

Following the release of CLIP, an updated version of VQ-GAN that uses CLIP embeddings [11] as conditioning was also released to the public in late 2021, marking one of the last releases of a GAN-based image synthesis

[1] thispersondoesnotexist.com for example uses a StyleGAN based network to synthesize faces that, in fact, do not exist.

[2] To attend means to give more weight to in this scenario, the words with the highest attention are the ones that are considered to be more related to the one being considered.

model. 2022 was an extremely *lucrative* year for the image synthesis field. In March, GLIDE 3.8 was published, marking the release of the first text-to-image guided diffusion model. Starting from the architecture of [10], a text-encoding-based perturbance is added to move the generative process towards the desired output, using a *classifier free guidance* [12] to further move the generated samples away from random generations. Just a month after, developing entirely on GLIDE, DALL-E 2 3.9 was released. CLIP-based embeddings are used along the other image embeddings to guide the diffusion process, and a specialized diffusion upsampler is trained to allow for generations in higher resolutions. June saw the release of Imagen 3.11 which uses a large language model pre-trained on only text to produce prompt encodings that are more related to the prompt itself.

Arguably though, what really set an important milestone in the image synthesis landscape was the release in August of the **open source** large diffusion model Stable Diffusion 2.7. The model is a Latent Diffusion Model based on the architecture described in 2.6. The diffusion process is carried in a low dimensional latent space through the use of an encoder and a decoder, while conditioning is added through cross-attention layers using CLIP text embeddings. The open-source nature of the model lead to the formation of a thriving community around it, leading to the implementation of existing techniques and the development of new ones, as well as the "study" of prompt engineering and the effect that hyperparameters have on the generation.

Of such developments, two were the major ones, Textual Inversion 3.17 and Dreambooth 2.8. Both techniques allow adding subjects to an existing diffusion model using a few training images ($\approx 3-5$) without incurring in catastrophic forgetting or excessive class specialization and requiring much less time and resources than a full-fledged finetuning. The former trains textual encodings such that they influence generation towards the trained subject, while the latter, which achieved better results overall, uses unused tokens to bind a new class-specific subject.

Recently, in January 2023, ControlNet 3.12 was released as a way to add different forms of conditioning to Stable Diffusion without requiring an entire re-train. Said forms of conditioning can come from depth maps, pose schemes, scribbles, and normal maps as an example and allow for an extensive degree of control on the generation process.

1.2 Thesis Structure

This thesis work is subdivided into the following chapters:

Chapter 2 provides the necessary tools to understand the proposed approach at a technical level, showing all the different methods that converge into the final result.

Chapter 3 shows approaches and techniques that are relevant to the field of study and could be used as alternatives.

Chapter 4 gives a technical overview of the proposed approach highlighting our contributions.

Chapter 5 describes the training setup and shows outputs and relevant metrics to support our claims.

Chapter 6 resumes our findings and highlights the goal we reached.

Finally, chapter 7 explores possible developments on the same topic and how they could influence our work.

Chapter 2

Technical Background

2.1 Attention and Transformers

2.1.1 Attention

Attention is a type of learning with the goal of finding whether a certain part of the input is more important than another in order to compute the correct output. In the context of language processing, attention solves the common problem of degrading quality as the sentence becomes longer for RNNs.

A basic implementation of attention can be found in the RNN encoder-decoder language translator described in [13], where the mechanism is used to improve the importance that the decoder gives to certain parts of the input sentence.

Given a set of *annotations*[1] $(h_1, h_2, \ldots, h_n)$ generated by the encoder, their weighted sum is used to generate the context vector c_i for the output word y_i (2.1).

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.1}$$

The weight α_{ij} of each annotation h_j is computed by calculating the softmax:

[1] An annotation for a given word x_j is computed by concatenating the forward hidden state $\overrightarrow{h}_j$ and the backward hidden state $\overleftarrow{h}_j$, i.e. $h_j = [\overrightarrow{h}_j^\top; \overleftarrow{h}_j^\top]^\top$ of $\top$ words.

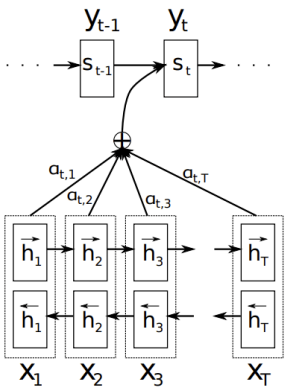


Figure 2.1: A visualization of an RNN with an attention mechanism; the model uses the source sentence $(x_1, x_2, \ldots, x_n)$.

to generate the word y_t, the attention mechanism is represented by the weights σ.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\Sigma_{k=1}^{T_x} \exp(e_{ik})} \tag{2.2}$$

where

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.3}$$

is defined as the alignment model, whose role is to output a numeric evaluation of how well the output at position i matches around position j (s_i is defined as the hidden state at position i), most commonly the dot product. α_{ij} in the case of sentence translation represents the probability that the target word y_i corresponds to a translation of a source word x_j. α_{ij} reflects how much weight, or importance, the annotation h_j with respect to every other state has, and in generating y. This is what it's called *attention*, the decoder decides which parts of the sentence to *attend* to.

2.1.2 Transformers

The paper *Attention is all you need* [5] works on the intuition that attention by itself is a powerful enough learning mechanism that requires no other network to learn, even on tasks unrelated to language. The following explanation takes largely inspiration from [14]. Suppose that we have a set of generic

input vectors x_i and a set of corresponding output vectors y_i of size and an embedding size d_{model}. Every input vector x_i is used in three different ways[2]:

- It is used unchanged as part of the weighted sum to compute each output vector once the weights have been established, usually called the *value* V
- It is compared to every other vector to establish the weights for its own output y_i to compute its *key* K
- It is compared to every other vector to establish the weights for the output of the $j-th$ vector y_j to compute its *query* Q

Three $k \times k$ weight matrices W_q, W_k, and W_v are added and the corresponding linear transformation is computed; these weights will store the computed relationships between each vector and the others:

$$
\begin{aligned}
q_i &= W^Q x_i = Q \\
k_i &= W^K x_i = K \\
v_i &= W^V x_i = V
\end{aligned}
$$

The attention for each input vector will then be:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.4}$$

The dot product QK^T is used to compute the alignment score for each vector with respect to the others; the scaling factor $\sqrt{d_k}$ (d_k is the size of the key) scales the dot product back to prevent the softmax function from growing too large. If for example, we want to learn the relationships between words in sentences we can set the input and the output of the attention to the same set of vectors; this type of attention is referred to as *self-attention*. Furthermore, it's possible to combine several attention mechanisms (that are able to capture different relationships to one another) with their own weights W_i^Q, W_i^k, and W_i^V (or *attention heads*) and then by concatenating their output:

[2]The names derive from the data structure of a key-value store; every key matches the query to a certain extent and the best is chosen to get the value.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W^O \tag{2.5}$$

$$\text{where} \quad \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.6}$$

To keep the speed of the model, rather than using h matrices of size $k \times k$ we reduce the size of W_i by a factor equal to the number of h matrices we want. The matrix W^O seen in (2.5) is used to "merge back" the split matrices into a single weight one for the subsequent layernorm and feed forward steps.

The *transformer* represented (fig 2.2) makes use of self-attention to process sequential input data all at once; the attention mechanism allows the transformer to access all previous states and weigh them accordingly to a learned measure of relevance, providing information for far away tokens.

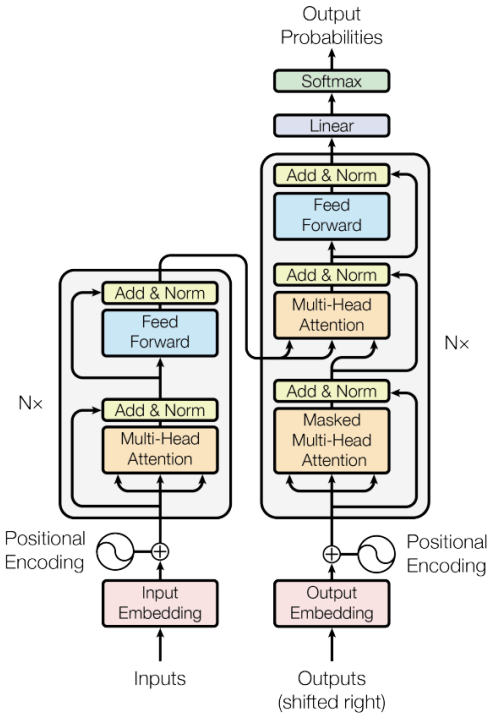


Figure 2.2: The model architecture of a transformer as shown in [5]. Each block can be stacked any N times.

First, the input is parsed into a token and converted into a vector; a positional embedding is then added. The encoder extracts and embeds information on which part of the input is relevant to each other through attention;

after a feed-forward layer the output is fed to the next encoder block and the operation is repeated. The decoder generates an output sequence by attending to the information provided by the encodings and the decodings up to and including that position by using a *masked attention* layer to preserve the autoregressive property[3] and a *cross attention* layer in which K and V come from the encoder output, and Q from the previous layer of the decoder block. More in general, a transformer block applies in sequence a self-attention layer, a layer normalization, a feedforward layer and a final layer normalization, all surrounded with residual connections.

Transformers are extremely adept at finding patterns between elements; the decoder and encoder blocks can be used alone or together depending on the use case.

2.2 Vision Transformers

The concept of using a self-attention mechanism to replace convolutional neural networks in the context of computer vision is introduced in [15] and further analyzed in [16]. Similar to a convolution, given a pixel $x_{ij} \in \mathbb{R}^{d_{in}}$ a local region of pixels in positions $ab \in \mathcal{N}_k(i,j)$ is extracted with spatial extent k centred around x_{ij} called *memory block*. Single-headed attention for computing the pixel output $y_{ij} \in \mathbb{R}^{d_{\text{out}}}$ is computed as follows (fig. 2.3):

A local region of pixels with size k centred around xij positioned at $ab \in \mathcal{N}k(i,j)$ is defined for each pixel $x_{ij} \in \mathbb{R}^{d_{in}}$, and is referred to as the *memory block*. To compute the pixel output $y_{ij} \in \mathbb{R}^{d_{\text{out}}}$ using single-headed attention, the following procedure is employed (see fig. 2.3):

$$y_{ij} = \sum_{a,b \in \mathcal{N}_k(i,j)} \text{softmax}_{ab}({q_{ij}}^{\top} k_{ab}) v_{ab} \tag{2.7}$$

Where the *queries* q, the *keys* k, and the *values* v (as seen in 2.1.2) are linear transformations of the pixel in position ij and the neighbourhood pixels. Each of q, k, and v has its own set of learned transforms $W_Q, W_K, W_v \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$.

While self-attention is used to aggregate nearby spatial information in a

[3] An output should not be predicted based on future outputs

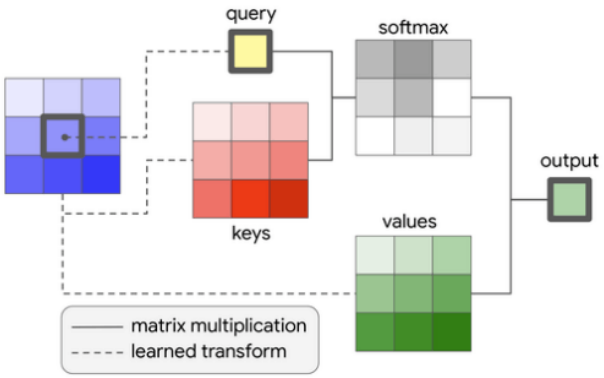


Figure 2.3: An example of a local attention layer over the spatial extent of $k = 3$. Picture from medium.

similar fashion to what a traditional convolution would do[4], the aggregation of faraway data is obtained through a convex combination of value vectors with mixing weights parameterized by content interactions (figure 2.3). By using multiple attention heads, thus increasing the depth, it's then possible to learn different representations of the same input. The distance between pixels and their relative position is modelled by adding a term $q_{ij}^{\top} r_{a-i,b-j}$ inside the softmax function in (2.7).

The culmination of this development is found in the paper *An image is worth 16x16 words* [17]. The paper introduces a fully attention-based image classification network named *Vision Transformer* (ViT), capable of attaining excellent results when compared to the state of the art with substantially faster training. The idea is to, rather than "mimicking" the approach of a convolutional neural network (as in [16] and [15]), the input image is subdivided into patches and then looked at as a whole. The standard transformer receives as input a 1D sequence of token embeddings.

Figure 2.5 provides an overview of the model. To effectively handle 2D images, the image $x \in \mathbb{R}^{H \times W \times C}$ is converted into a sequence of flattened 2D patches denoted as $x_p \in \mathbb{R}^{H \times (P^2 \cdot C}$ where:

- (H, W) is the resolution of the original image
- C is the number of channels

[4]Output of position ij in a regular convolution: $y_{ij} = \sum_{a,b \in \mathcal{N}_k(i,j)} W_{i-a,j-b} x_{ab}$

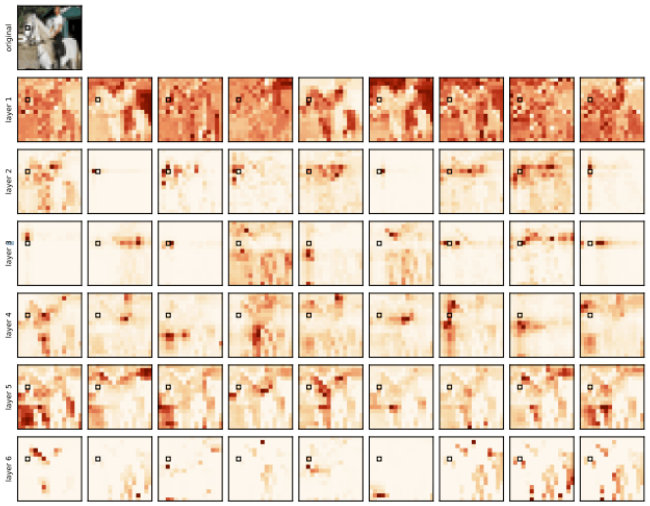


Figure 2.4: An attention probability heat map relative to a pixel positioned on the horse head, 6 layers and 9 heads. Picture from medium.

- (P, P) is the resolution of each patch
- $N = HW/P^2$ is the resulting number of patches, used for the input length of the transformer.

The embedding at position 0 is always a learnable embedding ($z_0^0 = x_{\text{class}}$) whose state at the output of the transformer encoder (z_L^0) serves as the image representation y. The position applied to each embedding is a simple 1D position as no significant performance gains were noticed by using more advanced ones. Given a patch embedding projection $\mathbf{E}$ the transformer encoder takes as input the embeddings vector (2.9), then it alternates layers of multi-headed self-attention (MSA) (2.10) and an MLP ((2.11)) blocks. Layernorm (LN) is applied before every block (differently from what we described in the original formulation 2.1.2, [18][19]).

$$z_0 = [x_{\text{class}}; x_p^1\mathbf{E}; x_p^2\mathbf{E}; \ldots; x_p^N\mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D} \tag{2.8}$$

$$z'_\ell = MSA(LN(z_{\ell-1})) + z_{\ell-1}, \qquad \ell = 1, \ldots, L \tag{2.9}$$

$$z_\ell = MLP(LN(z'_\ell)) + z'_\ell, \qquad \ell = 1, \ldots, L \tag{2.10}$$

$$y = LN(z_L^0) \tag{2.11}$$

$$\tag{2.12}$$

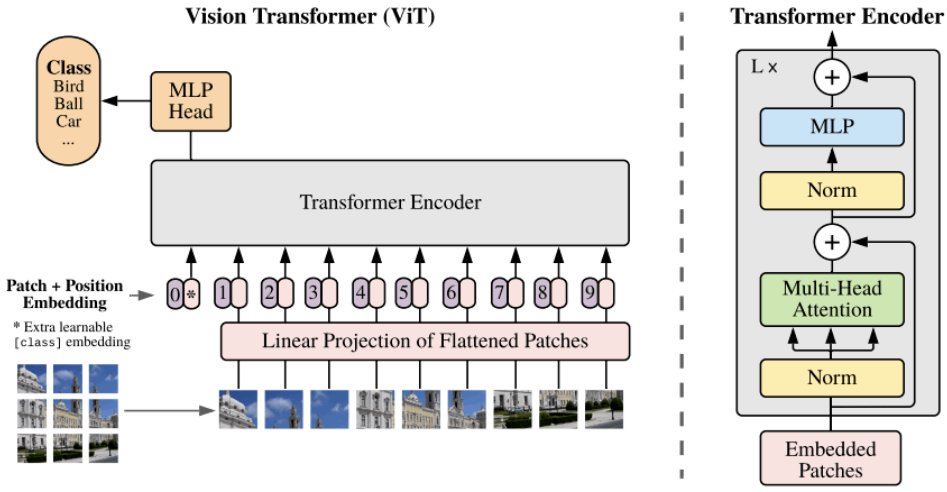


Figure 2.5: The image is split into patches, each patch is linearly embedded along with its position and fed to a standard transformer. An extra learnable token is added as input to perform classification.

The resulting model presents two main differences from a traditional CNN model: first, the model has much less image-specific *inductive bias*, secondly, the input of the transformer can be formed from other inputs such as feature maps rather than patches, allowing for a hybrid architecture. In figure 2.12 we can see the RGB embeddings filters obtained by the model; the filters are very similar to the ones produced in regular CNNs.

2.3 CLIP

CLIP [20], short for *Contrastive Language Image Pretraining*, is a technique developed to approach the zero-shot classification task by learning the contents of an image directly from a raw text description of it rather than from labels (such as the classes found in the ImageNet dataset).

Utilizing natural language for learning presents several notable advantages in contrast to alternative training methodologies. Its scalability surpasses that of conventional crowd-sourced labelling for image classification, as it eliminates the necessity for labels to conform strictly to the standard format, and the classification process differs from the typical 1-to-N best label voting system. Techniques applicable to natural language can easily leverage

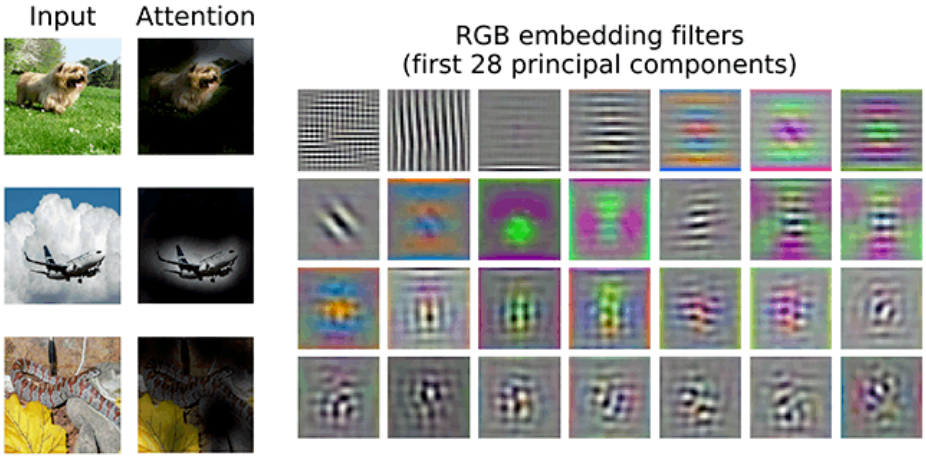


Figure 2.6: **left**: Representative examples of attention from the output token to the input space. **right**: Filters of the initial linear embedding of RGB values, from the original paper [16].

the abundant text available on the internet. The embedded representation obtained through CLIP shows a strong connection to language, facilitating zero-shot transfer capabilities.

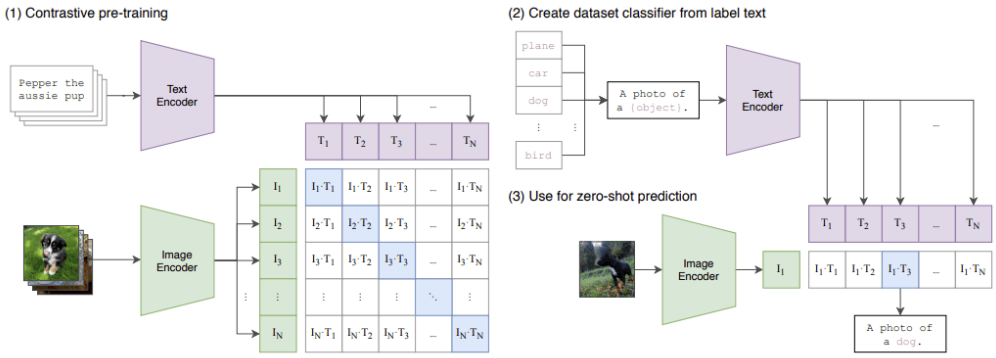


Figure 2.7: Summary of the CLIP approach as shown in the original paper.

Efficient training plays a pivotal role in effectively scaling the utilization of natural language supervision. In contrast to alternative methodologies that aim to predict the specific word representing the content of an image, its label, the approach introduced in [20] consists in trying to associate images to a whole description of them, without requiring an exact match. Rather

than a predictive objective, a *contrastive* objective is used[5]. CLIP is trained to predict the actual pairings across a batch of $N \times N$ (text, image) pairs. The output is a multi-modal space that closely links images to their associated description. The training objective consists in maximising the cosine similarity between the associated pairs of image and text embeddings while minimising the similarity to all the other pairings.

Two different architectures are proposed for the image encoder:

1. A ResNet-50 [21] as a base due to its proven performance and adaptation, augmented with improvements from recent years, and with its global average pooling layer replaced with a "transformer-style" multi-head QKV attention layer.

2. A Visual Transformer (ViT) that closely follows the implementation described in 2.2.

The text encoder is a Transformer (sec 2.1.2). The input text sequence is surrounded with `[SOS]` and `[EOS]` tokens, as it is usually done in LLMs; the activations for the `[EOS]` token at the highest level of the transformer are used as the feature representation for the text. These activations undergo layer normalization and linear projection to the multi-modal embedding space.

```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)
```

[5]Contrastive learning learns the general features of a dataset without labels by teaching the model which data points are similar or different. (definition from medium)

```
16
17 # scaled pairwise cosine similarities [n, n]
18 logits = np.dot(I_e, T_e.T) * np.exp(t)
19
20 # symmetric loss function
21 labels = np.arange(n)
22 loss_i = cross_entropy_loss(logits, labels, axis=0)
23 loss_t = cross_entropy_loss(logits, labels, axis=1)
24 loss = (loss_i + loss_t)/2
```

Listing 2.1: Numpy-like pseudocode for the core of an implementation of CLIP as seen in the original paper.

As natural language is used as the captioning and the prediction objective, **prompt engineering** becomes a meaningful task when training or inferencing using CLIP. Other than the issue of different labels being used for the same object among different datasets, the lack of context around a word raises the issue of polysemy[6]. To merge existing datasets to the captioned photos found on the internet, replacing the label to *"A photo of a [label]"* improved accuracy significantly. Specifying the category of the object, as in *"A photo of [label], a type of pet"* when training on a pet dataset, also improved on the final performance.

The main takeaways of the paper are the following: first, CLIP demonstrates high efficiency by training on noisy and diverse data, enabling zero-shot usage. It showcases flexibility and generality by directly learning various visual concepts from natural language, surpassing the capabilities of existing ImageNet models. However, CLIP faces challenges in tasks requiring abstract or systematic understanding, such as object counting, and in complex tasks like predicting the proximity of the nearest car in a photo. Lastly, CLIP's generalization abilities to images beyond its pre-training dataset are still limited.

2.4 Low Rank Adaptation (LoRa)

An important approach in natural language processing involves pretraining large-scale models (transformers sec. 2.1.2) on general domain data and then

[6]the coexistence of many possible meanings for a word or phrase.

fine-tuning them for specific tasks or domains. Other than the cost involved with training such large-scale models, which lies in the hundreds of GPU hours, this is to also reduce the carbon footprint that fine-tuning these models would involve. However, as models continue to grow in size, fully retraining all the parameters through fine-tuning becomes increasingly impractical. The approach proposed in *Low Rank Adaptation of Large Language Models*[4] named **Lo**w **R**ank **A**daptation was developed with the goal of reducing the number of parameters that need to be trained to finetune a transformer efficiently and with comparable performances.

The usual approach to developing downstream applications of existing large language models is to adapt, usually through fine-tuning, existing pre-trained models. As the size of the models increases and the number of parameters goes up the hundreds of billions, fine-tuning an existing model passes from an inconvenience to an extremely hard challenge, as the resources and time to perform this become more and more prohibitive. Other techniques introduced in the literature so far involved limiting the number of parameters to train or using an external network to adapt the input towards the desired goal, usually referred to as Hypernetworks. [22]. Despite reaching the desired goal, these techniques always led to a trade-off between efficiency and quality, almost never reaching the fine-tuning baselines.

The intuition is that "the change in weights during model adaptation also has a low intrinsic rank. LoRa allows training some dense layers in a neural network indirectly by optimizing rank decomposition matrices of the dense layers' change during adaptation instead while keeping the pre-trained weights frozen as shown in figure 2.8. Even a very low rank suffices for large models such as GPT-3 175B [23], making LoRa both storage and compute efficient" ([4] page 2).

The key advantages possessed by LoRa are:

- Rather than outputting a full mode, LoRa outputs a small, lightweight file that can be easily shared, stored, and switched to, empowering open-source development as well.
- LoRA makes training more efficient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters. Only the injected low-rank matrices are trained.
- By using LoRa the number of parameters is reduced up to threefold,

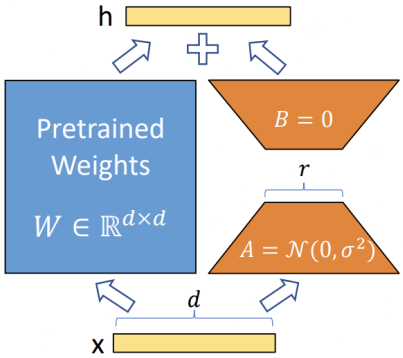


Figure 2.8: The reparametrization used in LoRa. Only A and B are trained. Taken from [4].

effectively lowering the hardware barrier to entry. In a scenario where machine learning is becoming more and more diffused, introducing techniques that lower training time and energy consumption become an important matter as well.

- As the weights are simply added to a pretrained model with a simple operation, the model operates as if it was fully finetuned, introducing no ulterior latency to the process.

The proposed solution consists of the following. Given a pre-trained LLM $P_\Phi(y|x)$ parametrized by Φ and a target language task represented by learning a mapping of context-target pairs $\mathcal{Z} = (x_i, y_i)_{i=1,\dots,N}$, with x_i and y_i being a sequence of tokens[7], a regular fine-tuning would consist in updating a set of initialized weights Φ_0 to $\Phi_0 + \Delta\Phi$ by maximising some sort of training objective:

$$\max_{\Phi} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log(P_\Phi(y_t|x, y_{<t})) \tag{2.13}$$

The drawback is that for each task we want to finetune a pretrained model for, a new different set of parameters $\Delta\Phi$ wish size $= \Phi_0$ is learned. The consequence of this is that the storage and deployment of a large set of

[7]For example if we are trying to convert the natural language to SQL then x_i would be NL and y_i the corresponding SQL.

finetunings can easily become a daunting task, requiring more and more storage space as the number of target tasks increases. Rather than learning and storing the entire set of parameters, the intuition is that the difference between the original and the output model is enough to efficiently store and deploy for a specific task. The proposed solution is then to further encode the change $\Delta\Phi$ by a much smaller set of parameters $|\Theta| \ll |\Phi_0|$ on which the learning is done:

$$\max_{\Theta} \sum_{(x,y)\in\mathcal{Z}} \sum_{t=1}^{|y|} \log(P_{\Phi_0+\Delta\Phi(\Theta)}(y_t|x, y_{<t})) \tag{2.14}$$

The low "intrinsic dimension" [24] that LLM possess allows them to efficiently learn a specific task despite a projection to a smaller subspace. The approach behind LoRa is based on the hypothesis that LLM also have a low "intrinsic rank" during adaptation. For a pretrained weight matrix $W_0 \in \mathbb{R}^{d\times k}$ [8], the update is constrained by representing it with a low rank decomposition $W_0 + \Delta W = W_0 + BA$, where $B \in \mathbf{R}^{d\times r}$, and $A \in \mathbf{R}^{r\times k}$, and the rank $r \ll min(d, k)$. During training, w_0 is frozen and does not receive gradient updates, while A and B contain trainable parameters. Given $h = W_0x$, the modified forward pass yields:

$$h = W_0x + \Delta W_x = W_0x + BAx \tag{2.15}$$

as seen in figure 2.8. In the case of transformers, this transformation is applied to the four weight matrices in the self-attention module (W_q, W_k, W_v, W_O), but in principle, LoRa could be applied to any subset of weight matrices in a neural network.

The output ΔW is a standalone "model" that can be applied on top of any existing set of weights.

2.5 Diffusion Models

The paper titled *Denoising Diffusion Probabilistic Models* (DDPM) [1] introduces a new class of specialized *probabilistic* image synthesis models named *diffusion model*. "A diffusion probabilistic model is a parametrized Markov

[8] The same naming conventions are the same as in 2.1.2.

chain trained using variational inference to produce samples that match the data after a finite time" ([1] pag. 2). The following explanation is roughly based on the one provided in [25].

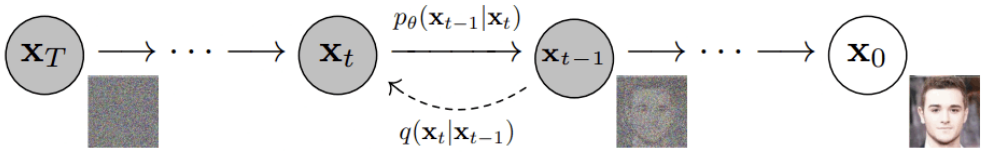


Figure 2.9: A visual representation of the diffusion process parametrized as a Markov chain. Noise is added to the sample (photo) x each time step from T to 0, and a posterior q is learned to reverse this process.

A diffusion process given a real sample $x_0 \sim q(x)$ consists in adding a certain amount of Gaussian noise to a sample over $\mathbf{T}$ steps until it is just random noise, and can be represented as a Markov Chain[9], usually referred to as *forward diffusion*. The step sizes are controlled by a variance schedule $\{\beta_t \in (0,1)\}_{t=1}^{\mathbf{T}}$.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t \mathbf{I}) \quad q(x_{1:\mathbf{T}}|x_0) = \prod_{t=1}^{\mathbf{T}} q(x_t|x_{t-1}) \tag{2.16}$$

As the step count t becomes larger and more and more noise is added, the initial sample gradually becomes more and more similar to random noise, eventually becoming equivalent to an isotropic Gaussian distribution at $\mathbf{T} \to \infty, x_{\mathbf{T}}$.

Given this process, by sampling from $q(x_{t-1}|x_t)$ it would be possible to, in theory, recreate the original sample from a Gaussian Noise input $x_{\mathbf{T}} \sim \mathcal{N}(0, \mathbf{I})$. Since it's impossible to easily estimate $q(x_{t-1}|x_t)$ without computing the entire solution space beforehand, we instead learn the *reverse process* $p_\theta(x_{0:T})$, which learns the Gaussian transitions from the considered step T up to the original sample $p(x_T) = \mathcal{N}(x_T; \mathbf{0}, \mathbf{I})$ parametrized as a Markov Chain.

[9]Markov Chain: a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. (Oxford Dictionary)

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \quad p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{2.17}$$

Diffusion models are a specific kind of latent variable model in which the approximate posterior $q(x_{1:T}|x_0)$ (the forward diffusion) is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1, \ldots, \beta_T$:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}), \quad q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}). \tag{2.18}$$

The variance $\Sigma_\theta(x_t, t) = \sigma_t^2\mathbf{I}$ is set to untrained time-dependent constant. The training objective consists in maximising the variational bound on negative log-likelihood (known as Evidence Lower Bound or ELBO):

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}_{\text{ıı}}[-\log \frac{p_\theta(x_{0:\mathbf{T}})}{q(x_{1:\mathbf{T}}|x_0)}] = \mathbb{E}_q[-\log p(x_\mathbf{T} - \sum_{t\geq 1} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}] =: L \tag{2.19}$$

The forward process variances β_t can be either learned through reparametrization (3.6) or held constant. By using different Gaussian conditionals in $p_\theta(x_{t-1}|x)$, different, one can express biases that are already known about the domain. The sampling x_t can be performed at any arbitrary timestep t. Using the notation $\alpha_t := 1 - \beta_t$ and $\hat{\alpha}_t := \prod_{s=1}^{t} \alpha_s$ (with $1 - \alpha_t$ = noise added at step t):

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\hat{\alpha}_t}x_o, (1-\hat{\alpha}_t)\mathbf{I}) \tag{2.20}$$

Rather than optimizing every parameter in (2.19), at each step a random term is chosen for optimisation, making training more efficient as a consequence. By rewriting L (2.19) through variance reduction and implementing KL divergence it's then possible to further improve the process.

$$\mathbb{E}_q[\underbrace{D_{KL}(q(x_T|x_0) \,||\, p(x_T))}_{L_T} + \sum_{t>1} \underbrace{D_{KL}(q(x_{t-1}|x_t, x_0) \,||\, p_\theta(x_{t-1}|x_t))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0}] \tag{2.21}$$

The full loss can be expressed as $L_{VLB} = L_T + L_{T-1} + \cdots + L_0$; every KL term compares two Gaussian distributions (the posterior and the corresponding prior) in closed form[10]. L_T is constant and can be ignored during training (q has no learnable parameters and x_T is Gaussian noise). L_0 is modeled using a separate discrete decoder derived from $\mathcal{N}(x_0; \mu_\theta(x_1,1), \Sigma_\theta(x_1,1))$. The reverse process mean function approximator μ_θ is trained to predict only $\tilde{\mu}_t$ or $\epsilon \sim \mathcal{N}(0, I)$ (the noise that was added rather than the Gaussian mean) by reparametrization (3.6).

By ignoring a weighting term and using a more simple training objective, the authors of the paper found that the diffusion model produced better outputs:

$$L_t^{simple} = \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}[||\epsilon_t - \epsilon_\theta(\sqrt{\hat{a}_t}x_0 + \sqrt{1-\hat{a}_t}\epsilon, t)||^2] \tag{2.22}$$

with the final simple objective being:

$$L_{simple} = L_t^{simple} + C \tag{2.23}$$

With C constant and not dependent on θ.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 2.10: The training and sampling algorithms for DDPM.

In practice, the reverse process is represented using a U-Net backbone with group normalization (figure 2.11). The U-Net makes use of a Transformer based architecture to share parameters across time through positional embeddings. Self-attention at the 16×16 feature map resolution is used.

Like other generative frameworks, diffusion models can be made to sample conditionally given a variable of interest such as a class label or a sentence description. One simple way to do this is to feed the conditioning variable

[10] In mathematics, a closed-form expression is a mathematical expression that uses a finite number of standard operations. (from Wikipedia)

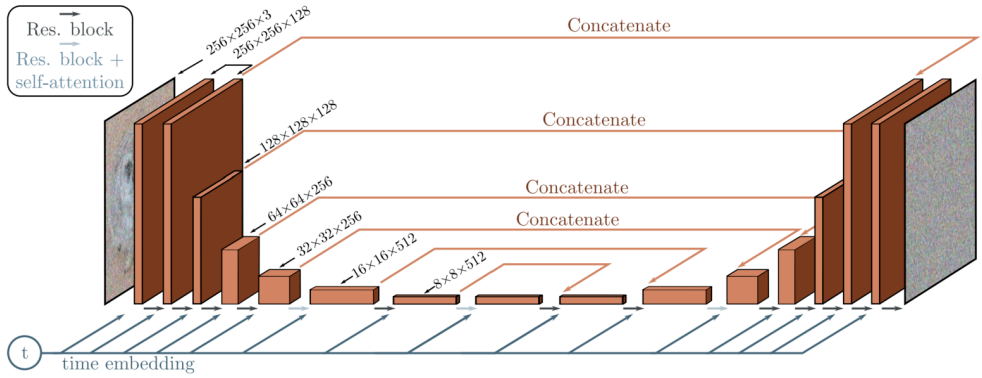


Figure 2.11: Visualization of the denoising U-Net of the diffuser. Picture from learn-opencv.

y as an additional input during $\epsilon_\theta(x_t, t, y)$, but it has been shown that the use of an external classifier specifically geared towards pushing the diffusion process towards the gradient of the desired label worked best 3.8. Another possibility that diffusion models offer is the so-called *inpainting*; the specially trained diffusion model is able to reconstruct missing parts of an image by knowing the surrounding pixels.

Diffusion models show promising results in the context of generalizing the output pictures domain and in understanding the different parts of the training images, but compared to GANs they are slower as the computation of a Markov chain with an unknown number of steps can be daunting when compared to the single forwards pass that GANs require.

2.6 Latent Diffusion

The paper *High-Resolution Image Synthesis with Latent Diffusion Models* [26] introduces a new class of diffusion models (DMs) called *latent diffusion models* that tries to solve the common issue of costly optimization and slow inference that previous iterations presented. Training the most powerful diffusion models can take hundred of GPU days and the required evaluations on the final product up to 5 GPU days. To enhance the accessibility of these models, and address the computational complexity associated with training and sampling, new methodologies need to be introduced. The proposed approach involves shifting the diffusion process from the pixel space to the

latent space. As a result, the training process is divided into two steps:

1. First, we train an autoencoder 3.2 that is capable of compressing an image to a more efficient space with lower dimensions. This learned latent space does not require excessive compression as the subsequent diffusion models are directly trained on it. This autoencoder requires a single training and can be reused for multiple DMs training to explore completely different tasks.

2. Then, image generation is carried in the latent space through a transformer-connected U-net backbone that can be conditioned through arbitrary token-based mechanisms.

The computational efficiency of the obtained DMs is enhanced by transitioning away from the high dimensional space, enabled by sampling in a lower dimensional space. Furthermore, the use of U-nets as the foundation for DMs leverages their inherent inductive bias, effectively mitigating the detrimental impact of aggressive quality-reducing compression levels. For the perceptual compression model, an autoencoder is used and trained using a combination of perceptual loss and a patch-based adversarial objective. Local realism, blurriness, and a more precise reconstruction are enabled by the use of this two-step process.

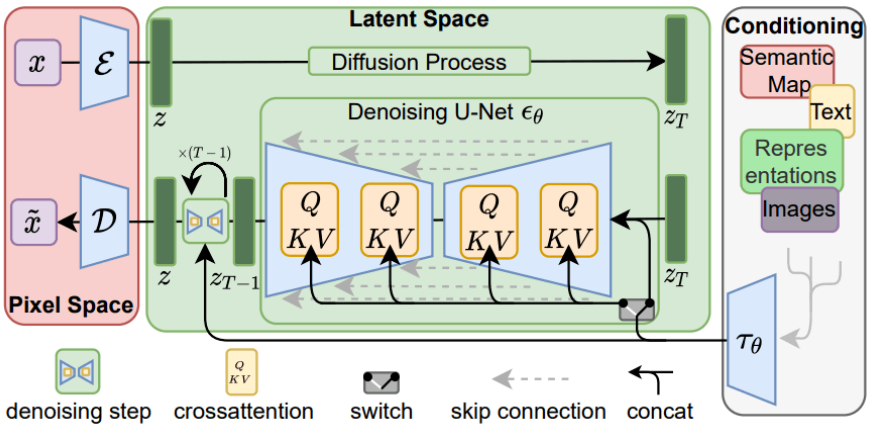


Figure 2.12: Structure of the model as shown in the original paper.

Given an image $x \in \mathbb{R}^{H \times W \times 3}$ in RGB space, the encoder $\mathcal{E}$ encodes x into a latent representation $z = \mathcal{E}(x)$, and the decoder $\mathcal{D}$ reconstructs the image from the latent, giving $\tilde{x} = \mathcal{D}(z) = \mathcal{D}(\mathcal{E}(x))$, where $z \in \mathbb{R}^{h \times w \times c}$.

The encoder downsamples the image by a factor $f = H/h = W/w$, and different downsampling factors $f = 2^m$, with $m \in \mathbb{N}$ are investigated. To avoid arbitrarily high-variance latent spaces, two different regularizations are used, *KL-reg* which imposes a slight KL-penalty towards a $\mathcal{N}$ on the learned latent, similar to a VAE 3.2, and *VQ-reg*, which uses a vector quantization layer within the decoder, similarly to a VQGAN [27].

The autoencoder works in the following way:

1. An image $x \in \mathbb{R}^{H \times W \times 3}$ in RGB space is the input.

2. An encoder $\mathcal{E}$ compresses the input image x into the latent space learned previously, creating the representation $z = \mathcal{E}(x)$. During compression, the image is downscaled by a factor $f = H/h = W/w$. Different downscaling factors f have been explored in the original paper.

3. At the same time, a decoder $\mathcal{D}$ is trained to reconstruct the original image so that $\tilde{x} = \mathcal{D}(z) = \mathcal{D}(\mathcal{E}(x))$

4. Two different forms of regularization, *KL-reg* which imposes a Kullback–Leibler loss towards $\mathcal{N}$ and a *VQ-reg* which uses a vector quantization layer, is used to contain the variance of learned latent space. The quantization applied is very mild when compared to other applications (such as DALL-E 3.7), rather than a 1D output a 2D one is produced, which keeps much more information on the inherent structure of z.

Thanks to the latent representation enabled by $\mathcal{E}$ and $\mathcal{D}$, likelihood-based modelling is a more suitable task as higher complexity details are abstracted away and the learning can focus on the important semantic bits of the data. Rather than using an autoregressive, attention-based approach, image-specific inductive biases can be taken advantage of. The diffusion process, rather than relying on a highly compressed autoregressive transformer, can be performed in a regular U-Net built primarily from 2D convolutions, with an objective that focuses on the task at hand:

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon \approx \mathcal{N}(0,1),t}[||\epsilon - \epsilon_\theta(z_t, t)||_2^2]. \tag{2.24}$$

During training, the process is made more efficient by fixing the forward diffusion process. Rather than compressing the image using the encoder and then performing diffusion on the compressed image, the image is compressed

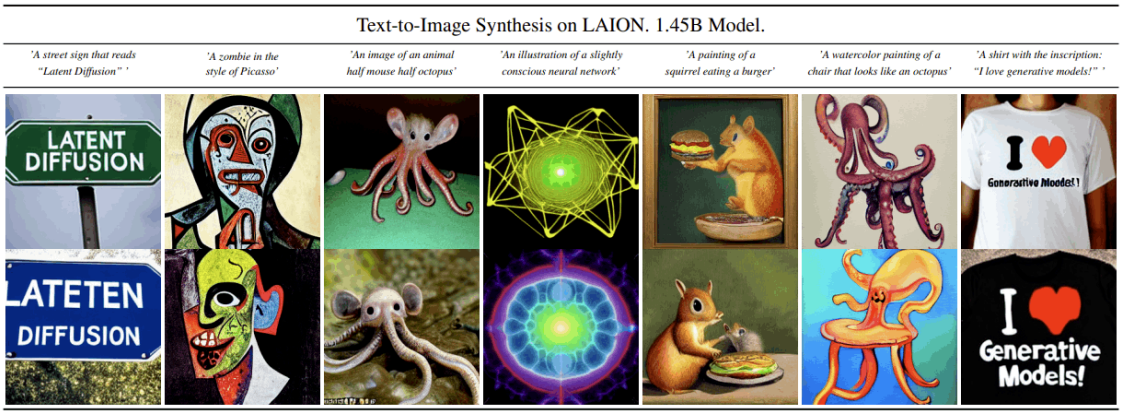


Figure 2.13: Examples of text-to-image conditioned generations shown in [26].

and made noisy in a single pass. Decoding the obtained latents to the pixel space similarly only requires a single pass.

Diffusion models are capable of modelling conditional distributions in the form $p(z|y)$. This can be implemented with a conditional denoising autoencoder $\epsilon_\theta(z_t, t, y)$, and allows inputs y such as text, semantic maps, or other image-to-image translation tasks. The underlying UNet is augmented with the cross-attention mechanism 2.1.1, which is effective for learning attention-based models of various input modalities. To preprocess y from various modalities, a domain-specific encoder τ_θ is introduced that projects y to an intermediate representation $\tau_\theta(y) \in \mathbb{R}^{M \times d_\tau}$, which is then mapped to the intermediate layers of the UNet via cross-attention layer implementing $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}}) \cdot V$, with

By using a conditional, denoising autoencoder $\epsilon_\theta(z_t, t, y)$, different types of distributions $p(z|y)$ can be used to condition the generative process. By using this mechanism, tasks such as image-to-image translation (similar to StyleGAN), Inpainting, and text-to-image generation become available. The mechanism by which the conditioning from y is applied to the diffusion process is through the use of a domain-specific encoder τ_θ whose role is to provide a representation of the conditioning y that is mappable to the cross-attention layers of the Unet $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}}) \cdot V$. Query, Key, and Values are computed from the encoded y as:

$$Q = W_Q^{(i)} \cdot \varphi_i(z_t), \quad K = W_K^{(i)} \cdot \tau_\theta(y), \quad V = W_V^{(i)} \cdot \tau_\theta(y).$$

During the training of the domain-specific encoder, the weights $W^{(i)}$ are updated. $\varphi_i(z_t) \in \mathbb{R}^{N \times d_\epsilon^i}$ is an intermediate representation of the denoising UNet. The conditional LDM is learned via:

$$L_{LDM} := \mathbb{E}_{\mathcal{E}(x),y,\epsilon \approx \mathcal{N}(0,1),t}[||\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y)||_2^2], \tag{2.25}$$

where both τ_θ and ϵ_θ are jointly optimized via the loss. To condition through a textual input specifically, the CLIP 2.3 was used. High-resolution images are obtained by using a super-resolution diffuser.

2.7 Stable Diffusion

Based on the Latent Diffusion model architecture 2.6, Stable Diffusion is the first large diffusion model to be released as open-source to the public [2]. One of the main contributing factors that allowed for this model training was the large availability of text-image pairs data that the recently released LAION-5B [28] dataset offered. LAION-5B is an image dataset consisting of 5.85 billion CLIP-filtered 2.3 image-text pairs with a largely noisy nature that was collected with the specific purpose of having an extremely large dataset of such data available to the general research landscape. Starting from Common Crawl[11], the data was filtered using an existing CLIP model by removing text-image pairs whose cosine similarity score fell under a certain threshold (0.28), this step alone removed over 44 billion images of the original 50. Furthermore, different subsets of LION-5B exist, depending on the perceived quality of the image (aesthetic), the resolution, or the language of the caption.

The public release of Stable Diffusion V1 was trained in four rounds, each of which used a different subset:

1. 237.000 steps at 256×256 on LAION-2B-en
2. 194.000 steps at 512×512 on LAION-high-resolution
3. 515.000 steps at 512×512 on LAION-improved-aesthetics
4. 390.000 steps at 256×256 on LAION-improved-aesthetics with 10% dropping of the text conditioning to favour classifier-free guidance.

[11] An open source repository of web crawl data https://commoncrawl.org/.

This version uses a custom-trained frozen CLIP ViT-L/14 model as a text encoder. The denoising U-Net is composed of three main parts: an encoder, a middle block, and a skip-connected decoder. The encoder and the decoder have 12 blocks, and the middle block is a single one, for a total of 25 blocks; 8 of those blocks are down or upsampling convolutional layers, 17 blocks are main blocks that each contain four resnet layers and two Vision Transformers with cross and self-attention mechanisms. The texts are encoded by CLIP and diffusion time steps are encoded through positional encoding.

A V2 version was later released [29] which replaces the encoder with a new one (OpenCLIP), improves the quality of the training subset, and enables the generation of images by default (with no upsampling) up to 768×768 pixels.

2.8 Dreambooth

Dreambooth[12] is an approach introduced in [3] to allow for a high degree of customisation of existing text-image diffusion models that requires just a few images to work, trying to "solve" the same issue that was dealt with in *Textual Inversion* 3.10 with a different method.

Figure 2.14: By only using a small set of training images, *Dreambooth* is can generate the subject in a variety of places/poses with a high visual fidelity. Picture from [3]

Large text-to-image models are able by design to generate a multitude of different subjects despite them being assigned to the same class but, as a by-product of the diffusion process, lack the ability to both reproduce the same subject in a different context and to mimic an existing subject into the

[12]From *dream photo-booth*.

generated image. The idea behind *Dreambooth* is to, given a few input images ($\approx 3-5$), bind the subject to a *unique identifier* such that when it is used in the prompt along with the class it belongs to (e.g. "A [V] dog"), the prior knowledge of the class is used along the new information to reconstruct the subject.

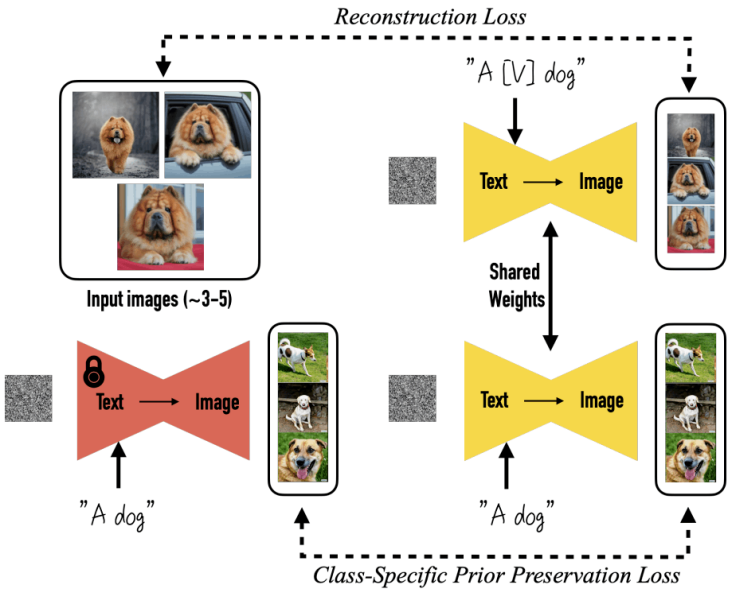


Figure 2.15: Representation of the finetuning process. In parallel, a reconstruction loss and a class-specific prior preservation loss work to teach a subject and prevent the model from *forgetting* the class prior [3].

The first step is to design *unique identifier*-subject pairs that can be implanted into the existing model dictionary. When finetuning a subject into an existing model, the caption assigned to each input picture is simply «"a [identifier] [class noun]", where [identifier] is a unique identifier linked to the subject, and [class noun] is a coarse class descriptor of the subject» (from [3] pag. 4). The class descriptor is used to leverage the knowledge present in the pre-trained model when learning the new subject, as it makes learning faster and more precise. The unique identifier that is bound to the novel subject is chosen so that it respects two characteristics:

- It has to be an unusual/rare token.
- It has to be tokenized "all together" (tokenizers usually decompose words into smaller tokens, and random letters could incur in the same).

The chosen approach was to, starting from the known dictionary of tokens, find unused three-letter tokens and invert them from the embedding to the textual space. An unusual token might be for example "SKS", making the descriptor for a new object of class "dog" that we want to add "A photo of a SKS dog". Finally, as the best result were found1 to come from fine-tuning all the layers of the model, two problems arose: *language drift*[13] and *reduced output diversity*[14]. To mitigate the two aforementioned issues, "*a new autogenous class-specific prior preservation loss is introduced to encourage diversity and counter language drift*" (from [3] pag. 1). During training, the model is supervised "*with its own generated samples*" in order to retain the prior knowledge of the class, and to use it along with the knowledge of the subject instance to generate new samples. The loss given an initial noise map $\epsilon \approx \mathcal{N}(0, \mathbf{I})$, a text encoder Γ, noise scheduling and sample parameters α_t, σ_t, w_t, and a prompt P becomes:

$$\mathbb{E}_{x,c,\epsilon,\epsilon',t}[w_t||\hat{x}_\theta(\alpha_t x+\sigma_t\epsilon, c)-x||_2^2+\lambda w_{t'}||\hat{x}_\theta(\alpha_{t'}x_{pr}+\sigma_{t'}\epsilon', c_{pr})-x_{pr}||_2^2], \quad (2.26)$$

where the second term is the prior-preservation term that supervises the model with its own generated images, and λ controls the intensity of said loss. The benefits of using this loss are better output diversity and less risk of overfitting.

As the requirements for applying this technique, although considerably smaller than a full-fledged finetune/retrain, can get expensive, an approach that became quickly popular among the userbase was to apply LoRa 2.4 to Dreambooth to both make the process faster and produce an output that's more easily shareable between users [30]. Rather than having the full model as output, a difference ΔW model is produced that weights several times less than the original one; furthermore, not all the parameters of the original model are trained as suggested in the original Dreambooth paper, but only the Q, K, V, O matrices of the cross-attention 2.1.2 layer.

[13] Typical of Large Language Models, as a consequence of fine-tuning the model can progressively lose the acquired language knowledge.

[14] When finetuning a text-to-image diffusion model on a subject there is the risk that the model reduces its variability when generating subjects of the finetuned class.

2.9 Shot Types

By definition in the Cambridge Dictionary, a movie shot is a *a photograph, or a short piece in a movie in which there is a single action or a short series of actions*. Movie shots can be classified by different means, the two most common forms of defining a shot are by field size and by camera placement. Our work is mostly focused on generating three specific kinds of field sizes:

- **Close shot**: in a close shot, or close-up the subject is close to the camera but far enough to show the shoulder line.
- **Medium shot**: a medium shot shows the subject from the waist up.
- **Long shot**: a long shot is usually an establishing shot that shows the environment around the subject along with the subject.

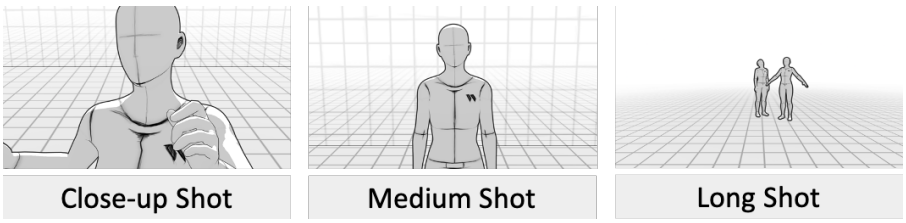


Figure 2.16: Representation of the three shot-types we deal with in this work.

Chapter 3

Related Works

3.1 Generative Adversarial Networks

Generative Adversarial networks are a kind of generative model released to the public in 2015. The Adversarial framework introduced in [31] describes the approach as follows: two networks are trained simultaneously, one is the *generator* network G whose goal is to generate samples that could have been drawn from the training distribution by learning in an unsupervised manner the latent variables of a distribution, the other is the *discriminator* network D whose goal is to predict a binary class label of real or fake (akin to a normal classification model) depending on whether the generated sample is considered to be from the target distribution or not.

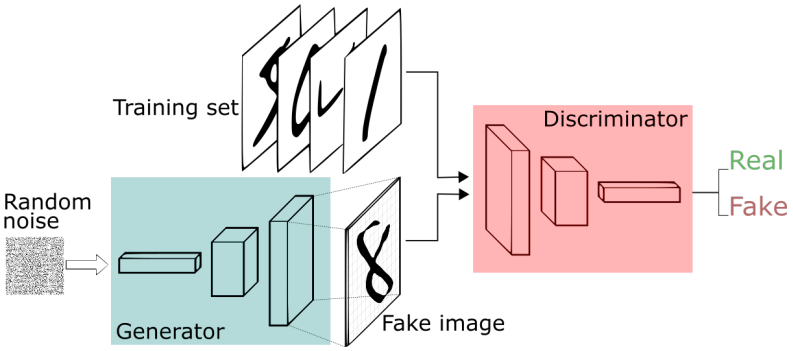


Figure 3.1: Visual representation of the Generative Adversarial Framework, from [32].

The goal of D is to minimize the probability of erroneously classifying a

sample generated by G, as such the framework becomes the same as a min-max two-player game with objective function (3.1); a unique solution to this game exists where G is able to recover the training data distribution and D assigns value $1/2$ everywhere, all trained with a single backpropagation pass each iteration.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))] \quad (3.1)$$

The goal of the generator is to learn a distribution p_g over data x. To do so, a prior $p_z(z)$ is initialized as the starting distribution with random values, and a differentiable mapping function G with parameters θ_g is defined such that $G(z;\theta_g)$, given input z, generates samples from the target distribution. A second function $D(x;\theta_d)$ is defined, with the role of outputting the probability that an input sample x comes from the true distribution rather than p_g. D is trained to maximize the probability of assigning the correct label; simultaneously G is trained to minimize $log(1-D(G(z)))$.

Once the training is considered to be complete, the network G can be isolated and used by itself in forward pass mode.

3.2 Variational Autoencoders

Generally speaking, an *autoencoder* [33] is a type of neural network that can learn to reconstruct input data in an unsupervised manner by compressing the information into a more efficient and compact representation. It achieves this by trying to reproduce the original input as closely as possible while reducing the amount of data needed to represent it. It consists of three main elements:

1. An *encoder* network: translates the original input from a high dimensional space into the latent low dimensional code; the input is larger than the output.

2. A *latent feature representation*: usually a tensor of real numbers.

3. A *decoder* network: recovers data from the code with increasingly large output layers.

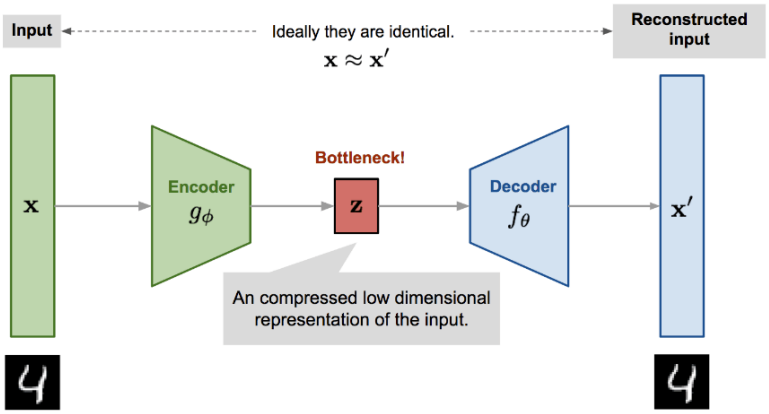


Figure 3.2: The autoencoder architecture, from [34].

The model contains an encoder function $g(\cdot)$ parametrized by ϕ and a decoder function $f(\cdot)$ parametrized by θ. The latent feature representation learned for input x in the *bottleneck* layer is $z = g_\phi(x)$ and the reconstructed input is $x' = f_\theta(g_\phi(x))$.

Training an autoencoder means finding a tuple of parameters (ϕ, θ) such that the original input is as similar as possible as the decoded output $x \approx f_\theta(g_\phi(x))$:

$$\underset{f_\theta, g_\phi}{arg\min} \quad [\Delta(x_i, f_\theta(g_\phi(x)))] \tag{3.2}$$

where Δ indicates a measure of how the input and the output of the autoencoder differ, also called *reconstruction loss*, which can be measured such as sigmoid or MSE.

There exist many different types of autoencoders, such as denoisers, sparse, and contractive, but the ones we are most interested in are *Variational Autoencoders* (VAEs) [35]. The basic concept of VAEs is to map the input data not to a single fixed vector but rather to a probability distribution that represents the possible variations in the input data. The following explanation is largely taken from [34].

Suppose that the distribution we want to map the input to is p_θ parametrized by θ; the relationship between the input x and the latent encoding vector z can be defined using three components: a prior $p_\theta(z)$, a likelihood $p_\theta(x|z)$, and a posterior $p_\theta(z|x)$. Assuming the value for parameter θ^* is known for the distribution we are trying to learn, then in order to generate a realistic data point x such that it would seem to come from the original distribution

we generate a sample $z^{(i)}$ using the prior $p_{\theta^*}(z)$, and then generate a value $x^{(i)}$ from a conditional distribution $P_{\theta*}(z|z = z^{(i)})$.

The goal of a VAE is to try and optimize the value θ^* such that the probability of generating a realistic sample is maximized:

$$\theta^* = arg\max_{\theta} \sum_{i=1}^{n} \log p_\theta(x^{(i)}) \quad \xrightarrow{encoding\ vector} \quad p_\theta(x^{(i)}) = \int p_\theta(x^{(i)}|z)p_\theta(z)dz \tag{3.3}$$

Using this method to define the optimal θ would require checking every possible value of z and thus be unfeasible; as such the idea is to use an approximation function $q_\phi(z|x) \approx p_\theta(z|x)$ parametrized by ϕ that is able to output what is likely a latent representation.

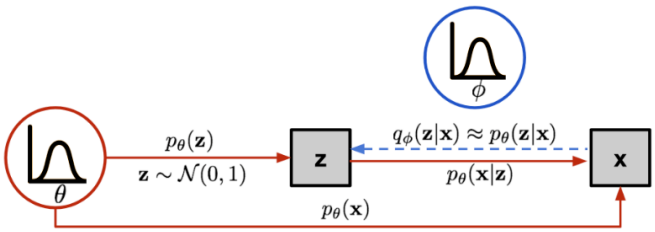


Figure 3.3: The graphical model of VAEs, from [34].

To estimate the posterior $q_\phi(z|x)$ the new objective function makes use of the KL Divergence [36] to quantify the distance between the estimated and real posterior; the KL divergence makes it so the learned distribution is not too far from the generating one. The loss function then becomes the following:

$$\max_{\phi,\theta} \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] \rightarrow L_{VAE}(\theta, \phi) = -\log p_\theta(x|z) + D_{KL}(q_\phi(z|x)||p_\theta(z)) \tag{3.4}$$

Minimizing $L_{VAE}(\theta, \phi)$ is equivalent to maximising the lower bound of the probability of generating real data samples; this loss function is known as *variational lower bound*.

$$-L_{VAE} = \log p_\theta(x) - D_{KL}(q_\phi(z|x)||p_\theta(z|x)) \leq \log p_\theta(x) \tag{3.5}$$

Finally, since the backward propagation would include a sampling step that is by nature stochastic and would therefore make back propagation unfeasible, the *reparametrization* trick is used to express the random variable z as a deterministic variable $z = \mathcal{T}_\phi(x, \epsilon)$ where ϵ is an auxiliary independent random variable and the transformation function $\mathcal{T}_\phi$ parametrized by ϕ converts ϵ to z. A common choice for $q_\phi(z|x)$ is a multivariate Gaussian with a diagonal covariance structure:

$$z \sim q_\phi(z|x^{(i)}) = \mathcal{N}(z; \mu^{(i)}, \sigma^{2(i)} I) \xrightarrow{rep.\,trick} z = \mu + \sigma \odot \epsilon \quad where\, \epsilon \sim \mathcal{N}(0, I) \tag{3.6}$$

This way the stochastic nature of z is conserved in the random variable ϵ that is fixed and therefore requires no backpropagation, while μ and σ become the trainable parameters.

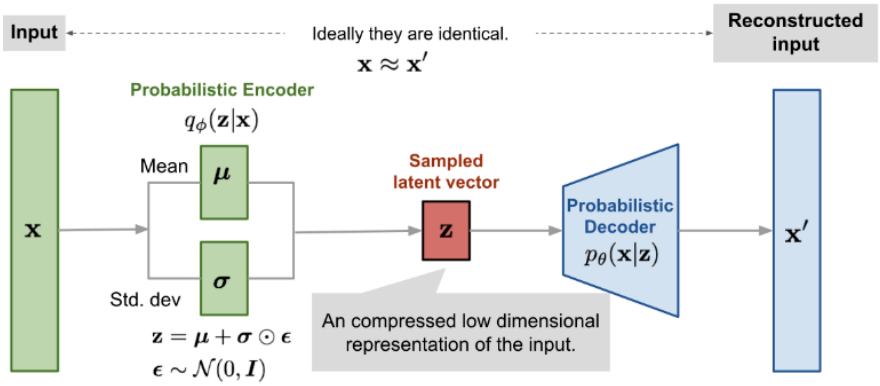


Figure 3.4: A variational autoencoder with a multivariate Gaussian assumption, from [34].

Other than data compression, VAEs are capable of generating data that is similar to the input data as they learn as a byproduct an intrinsic representation of the input/output domain from which they can sample.

3.3 Generative Adversarial Text to Image Synthesis

Having been used to varying success to synthesize images of specific categories, the first attempt at developing an architecture that uses text to guide

the generation process described in (3.1) [37]. The framework described divides the challenge into two sub-problems: "first, learn a text feature representation that captures important visual details; and second, use said features to synthesize a compelling image that a human might mistake for real" ([37] pp. 1-2). The plausible output of said generation though is multimodal by nature (there are many possible configurations of pixels that satisfy a given text input), as such the generative adversarial approach is a very natural fit for the challenge, by conditioning both discriminator and generator on side information (text) it's possible to naturally model the problem as the discriminator acts as a "smart" adaptive loss function.

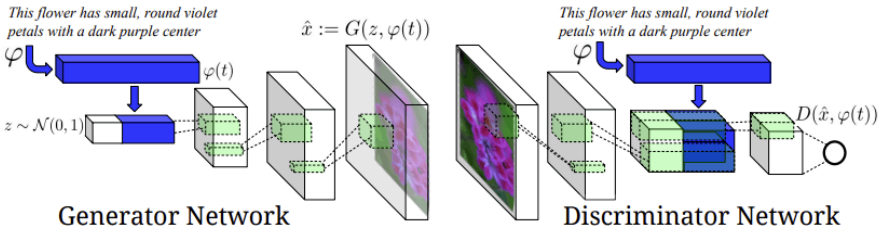


Figure 3.5: A visualization of the framework described in the paper, the text encoding $\varphi(t)$ both to generate the sample and to discern whether the sample is *real* or *fake*, from the original paper.

The generator G works by sampling from a noise prior $z \in \mathbb{R}^Z \sim \mathcal{N}(0,1)$, a text encoding obtained using the encoder φ is then used to compress a query t that is then added to the noise vector z. Using this information an image $\hat{x}$ is synthesized; this corresponds to a feedforward inference step for G. The discriminator uses the text-image pair to predict:

- If the image sample comes from a real distribution (similar to a regular GAN).
- If the pair is correct.

Using this prediction approach, the two outputs of the discriminator are used together to provide additional information to the generator, refining the process more at each step.

The main issue with using Generative Adversarial Networks to tackle the image synthesis task is the generalization that these networks allow, even when giving a textual prompt as input. Despite the images showing hints of understanding of the concept at hand, as the authors said, "upon close

inspection, it is clear that the generated scenes are not usually coherent" ([37] pag. 8).

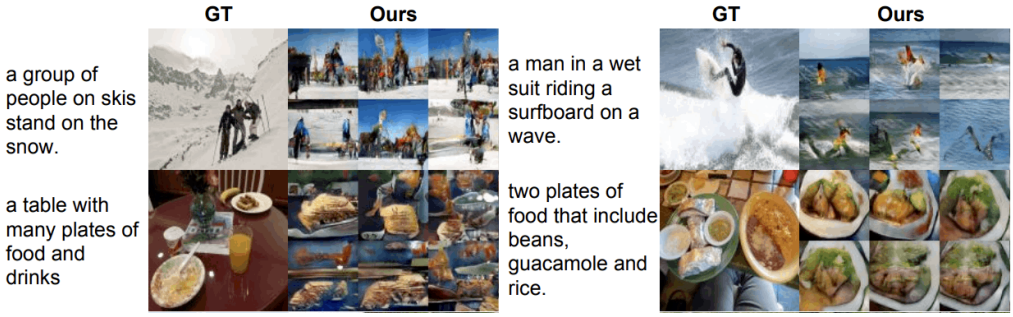


Figure 3.6: Examples provided by the original paper of results obtained when training with a wider dataset such as MS-COCO

Despite its *rudimentary* results, this can be considered the first instance of text-guided image generation and an attempt at generalizing the "knowledge" of the model.

3.4 Nvidia StyleGAN

In December 2018 Nvidia released their own approach to using GANs to synthesize images in *A Style-Based Generator Architecture for Generative Adversarial Networks* [38]. The novel approach proposed by Nvidia consists of a new generator architecture that allows greater control during the image synthesis process; the focus is put on understanding the various aspects of the image synthesis process, with a focus on the underlying image features and latent space understanding.

Compared to other approaches that make use of GANs, the one proposed in the paper starts from a constant input that is learned during training (rather than a random noise sample), and operates by adjusting the *style* of the image in each layer, with each convolution controlling either a high-level attribute (pose, identity, ...) or a stochastic variation (hair, facial imperfections, ...), which are separated during training. As a by-product of this, by moving in the learned latent representation of these attributes, it's possible to interpolate between styles (such as changing hair colour or style, changing skin colour, ...).

Figure 3.7 shows how the newly proposed style-based generator compares

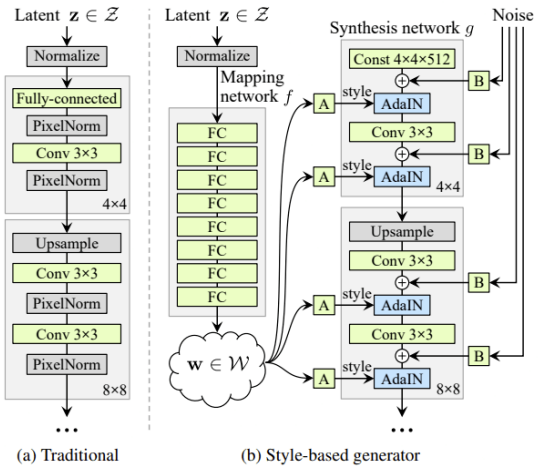


Figure 3.7: What a traditional generator process looks like on the left, and the newly proposed style-based generator on the right as shown in the original paper.

to a regular generator: in the traditional architecture, the input is fed once at the beginning only, comparatively, the approach proposed uses an intermediate latent space $\mathcal{W}$ representation of the input that is fed at different layers through adaptive instance normalization (AdaIN). The w that is used as input is specialized through learned affine transformations (A in the graph) at each step to a *style* that controls. The means by which the network is able to add stochastic variance is through learned per-channel scaling factors (B in the graph) that take Gaussian noise as input.

The main improvement when compared to previous generative networks is the ability through a larger number of training parameters and a framework that separates the generation in different steps to learn the intricacies and details of the subject, we can see from the output examples in figure 3.8 that the network was able to learn details such as glasses, facial hair, different kind of sheets and bedroom layouts, ..., and to reproduce them when asked to generate with the given class label.

The results obtained by StyleGan show the true potential of AI-generated images in generating pictures that are almost indistinguishable from real ones, but still lacking the generalization and control that later models will show.

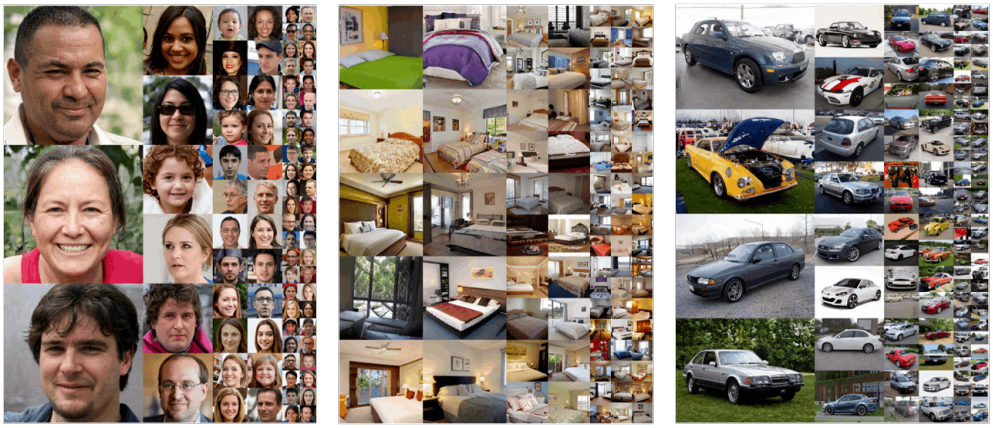


Figure 3.8: Uncurated sets of images generated when training with different datasets, in order faces, bedrooms and cars. From [38].

3.5 Vector Quantised Variational AutoEncoder (VQ-VAE)

Tasks that involve limited training data, adapting to different domains, or utilizing reinforcement learning techniques heavily depend on acquired representations from unprocessed information. The paper *Neural Discrete Representation Learning* [39] aims to develop a model that preserves the crucial characteristics of the information in its latent space while optimizing for maximum likelihood. Discrete representations are well-suited for various learning domains, such as language, as it is inherently comprised of distinct units, and images, which can often be easily described using language (through prompts). Moreover, discrete representations usually make predictive learning a simpler task, as the reduction of the learning space allows for more advanced techniques at a similar computation cost.

VQ-VAE uses a similar architecture to a conventional VAE 3.2. Instead of assuming Gaussian distributions with diagonal covariance for the *priors* and *posteriors*, VQ-VAEs use discrete latent variables. In this case, the prior and posterior distributions are categorical, and the obtained samples from these distributions serve as indices for embedding tables. The network works by building a *codebook* representation of the data: taking into example natural language, given a finite number of words each word would be assigned to a specific *code* in a sort of hashing table fashion. The resulting embeddings are

then utilized as inputs for the decoder network.

Conceptually, VQ-VAE are similar in approach to a regular VAE 3.2. Rather than having *priors* and *posteriors* be assumed as normally distributed with diagonal covariance (that allows for the reparametrisation trick (3.6)), VQ-VAEs use discrete latent variables where the prior and posterior distributions are categorical, and the samples drawn from these distribution indexes an embedding tables. These embeddings are then used as input into the decoder network.

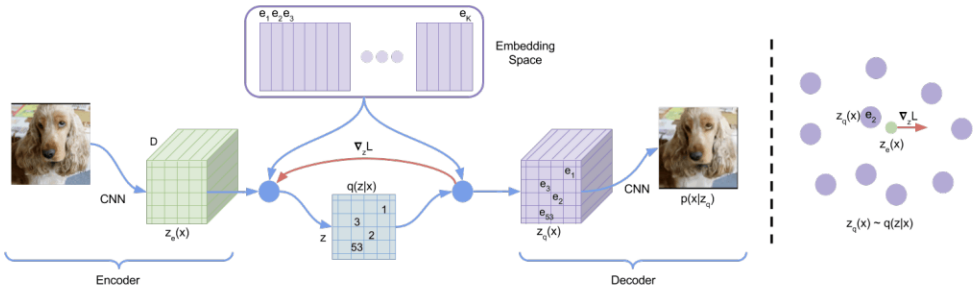


Figure 3.9: left: The VQ-VAE network. right: The output of the encoder $z(x)$ is mapped to the nearest point e_2. From [34].

We define a latent embedding space $e \in R^{K \times D}$, with K representing the size of the discrete latent space and D representing the dimensionality of each embedding vector e_i. The model takes an input x, and passes it through an encoder to obtain $z_e(x)$. The discrete latent variables z are determined via nearest neighbour lookup using the shared embedding space (3.7). The decoder utilizes the corresponding embedding vector e_k as input (3.8).

The posterior categorial distribution $q(z|x)$ probabilities are defined as one-hot as follows:

$$q(z = k|x) = \begin{cases} 1 & \text{for } k = \text{argmin}_j ||z_e(x) - e_j||_2, \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

The representation $z_e(x)$ first is represented in the bottleneck (3.7), then it is mapped to its nearest neighbour representation in the learned codebook (3.8)

$$z_q(x) = e_k, \quad \text{where} \quad k = \text{argmin}_j ||z_e(x) - e_j||_2 \tag{3.8}$$

The complete set of parameters for the model is the union of the parameters of the encoder, decoder, and the embedding space e. Since `argmin` prevents the flow of the gradient, backpropagation would be impossible. The gradient is approximated by copying gradients from the decoder input $z_q(x)$ to the encoder output $z_e(x)$ (figure [39] right). The overall loss function is composed of three terms: a *reconstruction loss* which optimizes the decoder and encoder, a *codebook loss* which uses a dictionary learning algorithm l2 error to move the embedding vectors e_i towards the encoder output, and a *commitment loss* that makes sure that the encoder commits to an embedding without fluctuating. The loss is then the following:

$$L = \underbrace{\log p(x|z_q(x))}_{\text{reconstruction loss}} + \underbrace{||sg[z_e(x)] - e||_2^2}_{\text{codebook loss}} + \underbrace{\beta||z_e(x) - sg[e]||_2^2}_{\text{commitment loss}} \tag{3.9}$$

Here, sg represents the *stopgradient* operator, which acts as an identity operator during forward computation and has partial derivatives equal to 0. This operator effectively restricts its operand to remain constant. The prior distribution $p(z)$ over the discrete latents is categorical and can be made autoregressive by incorporating dependencies to other zs within the feature map such as with a Transformer 2.1.2. By fitting an autoregressive distribution over z, $p(z)$, it's possible to generate x via ancestral sampling.

Figure 3.10: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512. From the original paper.

Figure 3.10 shows the particular effectiveness of VQ-VAE in the compression and reconstruction of images. Images contain a lot of redundant information and despite a reduction of $\frac{128\times128\times3\times8}{32\times32\times9} \approx 42.6$ in the embedded space, the reconstruction retains most of the information in the image. Another advantage of the VQ-VAE is that it successfully manages to avoid the "posterior

collapse"[1] issue that has been often observed in the VAE framework.

A second version of VQ-VAE named VQ-VAE2 was proposed in [40] which greatly improves the generation capabilities of the model by using a better prior model (through an attention-based CNN) and a hierarchical VQ-VAE that captures different level information at different resolutions of the image, with each layer being used along with the class label as input for the next layer.

3.6 VQ-GAN

Transformers are increasingly being adopted in a variety of tasks other than natural language processing. In the field of computer vision, compared to existing architectures such as CNNs, transformers introduce little to no inductive bias. This is generally seen as a benefit of using a transformer, but at the same time, compared to convolutions, it increases the computational cost of finding relationships quadratically, as the model takes into consideration each possible pairwise interaction. The paper *Taming Transformers for High-Resolution Image Synthesis* [27] proposes to augment the existing transformer architecture (that is capable of learning similarly to convolutions as in 2.2) with local connectivity through convolutions. Convolutions are more adept at finding low-level image structures, while transformers learn the global composition. "Allowing transformers to concentrate on their unique strength—modelling long-range relations—enables them to generate high-resolution images" ([27] pag. 2).

To leverage the powerful transformer architecture for image synthesis, the elements of an image have to be expressed as a sequence. Rather than composing the sequence based on individual pixels, a codebook of learned representations, similar to the methodology proposed in 3.5, is used. For any given image $x \in \mathbb{R}^{H \times W}$ there exists a representation formed from a spatial collection of codebook entries $z_q \in \mathbb{R}^{h \times w \times n_z}$ where n_z is the dimensionality of the codes. To efficiently learn said codes, the inductive biases of CNNs are directly incorporated. First, a convolutional model comprised of an encoder E and a decoder G is trained. The purpose of this training is for the model to acquire the ability to represent an image using a discrete codebook

[1] VAE models that have a powerful decoder often ignore the latents when reconstructing.

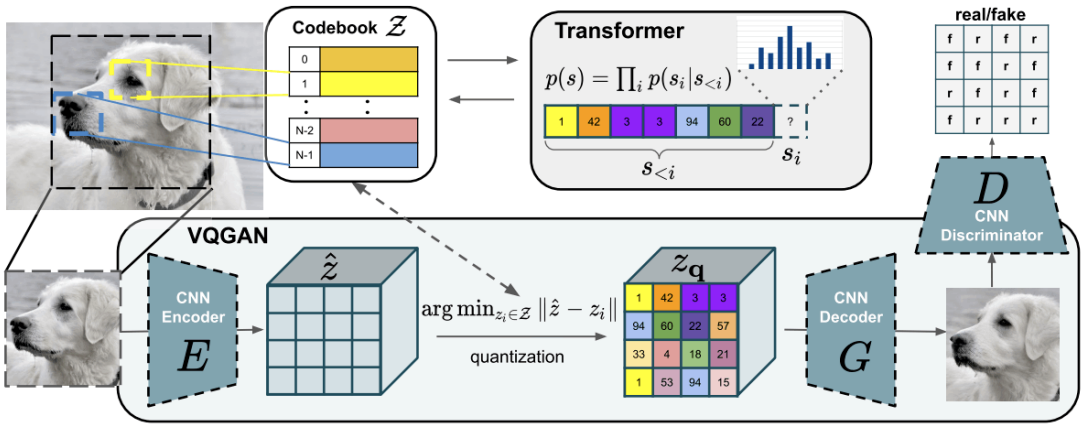


Figure 3.11: The proposed approach, as illustrated in the original paper. A convolutional VQGAN is used to learn a codebook of visual components. This codebook is then modelled using an autoregressive transformer architecture. To maintain semantic meaning while achieving significant compression, a patch-based discriminator is employed.

$\mathcal{Z} = z_k{}_{k=1}^{K} \subset \mathbb{R}^{n_z}$, as depicted in figure 3.11. The encoding of an image x $\hat{x} = G(z_q)$ is obtained by taking the closest codebook entry z_k after encoding:

$$z_q = q(\tilde{z}) := (\arg\min_{z_k \in \mathcal{Z}} ||\hat{z}_{ij} - z_k||) \in \mathbb{R}^{h \times w \times n_z}. \tag{3.10}$$

The reconstruction $\hat{x} \approx x$ is then given by:

$$\hat{x} = G(z_q) = G(q(E(x))). \tag{3.11}$$

As seen in the case of VQ-VAE, backpropagation through such a layer would be unfeasible, so a straight-through gradient estimation is used.

To learn a richer code book than a simple quantization, VQ-GAN is introduced. The L_2 loss employed for the recreational loss $\mathcal{L}_{\text{rec}}$ is replaced with a *perceptual loss*. Additionally, an adversarial training procedure is incorporated, utilizing a patch-based[2] discriminator D that distinguishes between real and reconstructed images.

$$\mathcal{L}_{\text{GAN}}(\{E, G, \mathcal{Z}\}, D) = [\log D(x) + \log(1 - D(\hat{x}))] \tag{3.12}$$

[2]The loss is computed in a patch-based fashion. Pixels that are more than a patch diameter of distance are considered independent.

The optimal compression model $\mathcal{Q}* = \{E*, G*, \mathcal{Z}*\}$ is then found using objective:

$$\mathcal{Q}* = \arg\min_{E,G,\mathcal{Z}} \max_{D} \mathbb{E}_{x\sim p(x)}[\mathcal{L}_{VQ}(E, G, \mathcal{Z} + \lambda \mathcal{L}_{\text{GAN}}(\{E, G, \mathcal{Z}\}, D)] \tag{3.13}$$

Where λ is an adaptive weight given by:

$$\lambda = \frac{\nabla_{G_L}[\mathcal{L}_{rec}]}{\nabla_{G_L}[\mathcal{L}_{GAN}] + \delta} \tag{3.14}$$

here, $\mathcal{L}_{REC}$ represents the perceptual reconstruction loss, while $\nabla G_L[\cdot]$ denotes the gradient of its input with respect to the last layer L of the decoder. To maintain numerical stability, a value of $\delta = 10^{-6}$ is employed. For comprehensive context aggregation, a single attention layer is utilized on the lowest resolution.

With the encoder and decoder available, images can be represented in terms of their codebook-indices encodings. By mapping indices of a sequence back to the corresponding codebook entry, it's possible to readily recover the image that originated that index. As such, once an ordering of the indices in the sequence s is established, image generation can be approached as an autoregressive prediction of the next index. Figure 3.12 shows the sliding attention window that is used to generate high-resolution images autoregressively. Despite the use of convolutions, the available context remains adequate for image modelling, as long as the dataset statistics exhibit spatial invariance.

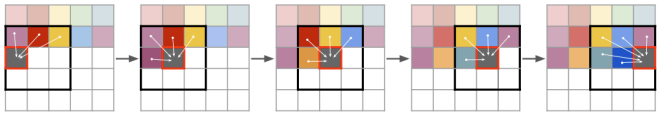


Figure 3. Sliding attention window.

Figure 3.12: Sliding attention window.

3.7 DALL-E

Traditionally, the development of text-to-image generation has primarily focused on refining modelling assumptions for fixed datasets. These assumptions often involve complex architectures, auxiliary losses, or additional information such as object part labels or segmentation masks, which are supplied

during training 3.3. DALL·E [41], a GPT-3[23] based model with 12 billion parameters, was specifically trained to generate images from text descriptions, utilizing a dataset comprising text-image pairs. The proposed approach involves training a transformer to autoregressively model both the text and image tokens as a unified stream of data. Directly using pixels would demand an excessive amount of memory for high-resolution images, with the modelling capacity primarily dedicated to capturing fine-grained details with high frequency, discarding the greater picture.

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

(d) the exact same cat on the top as a sketch on the bottom

Figure 3.13: Examples of DALL-E generation given a textual prompt provided in the original paper.

A two-stage training procedure similar to the one described in [40] 3.5 is proposed to address the issue: first, a discrete variational autoencoder (dVAE) is trained to compress 256×256 RGB images into a 32×32 grid of image tokens, each element of which can assume 8192 possible values. This compression reduces the transformer's context size by a factor of 192 while maintaining satisfactory image quality. Then, the 1024 image tokens are combined with up to 256 BPE-encoded[3] text tokens. An autoregressive transformer 2.1.2 is then trained to model the joint distribution of the text and image tokens.

The procedure can be seen as maximizing the Evidence Lower Bound (ELBO) (2.19) on the joint likelihood of the model distribution over images x, captions y, and tokens z representing the encoded RGB image. This distribution is modelled as $p_{\theta,\psi}(x, y, z) = p_\theta(x|y, z)p_\psi(y, z)$, resulting in the following lower bound:

[3]Byte Pair Encoding: a simple and robust form of data compression introduced in [42]

$$\ln p_{\theta,\psi}(x,y) \geq \mathop{\mathbb{E}}_{z\in q_\phi(z|x)}(\ln p_\theta(x|y,z) - \beta D_{KL}(q_\phi(y,z|x),p_\psi(y,z))). \quad (3.15)$$

The distribution q_ϕ represents the probability distribution of the image tokens, with size 32×32, which are generated by the dVAE encoder, given the input image x^2. p_θ represents the distribution over the generated images by the decoder of the dVAE model, given the image tokens. Lastly, the transformer model captures the joint distribution p_ψ over the text and image tokens. During training, the encoder produces a 1024-token-long encoding of the image, the decoder transformer receives as input the concatenation between the words encoding and the image encoding separated by a `<SOI>` (start of image) specialized token, and autoregressively tries to reconstruct the image encodings starting from the `<SOI>` token. As the input sentence is always 256 tokens long, the empty space is filled with a specialized token.

Once a joint prior p_ψ is learned, the decoder can be used autonomously by just tokenizing an input sentence and adding the `<SOI>` token.

Figure 3.14: Effects of increasing the number N of generated images on MS-COCO captions, from the original paper.

One final step uses CLIP 2.3 to further improve the output by performing a "best of N" approach. DALL-E is used to generate N images and then CLIP is fed both the image and the caption to compute how likely the generated images are associated with the given sentence. In figure 3.14 we can see the positive effect that this additional pass has on the output image.

3.8 GLIDE

The paper *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models* [43] explores the utilization of diffusion models 2.5 for text-conditional image synthesis. The aim is to leverage the capabilities that guided diffusion models offer in handling unrestricted prompts and apply them to the task of text-conditional image synthesis. Initially, a diffusion model with 3.5 billion parameters is trained, incorporating a text encoder for conditioning on natural language descriptions. Next, CLIP guidance and *classifier free guidance* [12] are compared to find which of the two yields better results. Along with the zero-shot model, an *inpainting* specific model is trained which allows to perform generation in a specific area delimited by the user.

Figure 3.15: Iteratively creating a complex scene using GLIDE by **inpainting** each component of the scene, as shown in [43].

Starting from the base diffusion model described in [1] (2.5), some improvements are applied to the base diffusion model. The variance schedule Σ_θ is added to the learnable parameters rather than fixed, as proposed in [44], effectively enabling high-quality image generation in fewer steps. To enhance the quality of generated samples, *guided diffusion* [10] is employed. This approach consists in utilizing a class conditional diffusion model with a mean $\mu_\theta(x_t|y)$ and variance $\Sigma_\theta(x_t|y)$ that is perturbed by the gradient of the log probability $\log p_\phi(y|x_t)$ predicted by a classifier for the target class y. The resulting perturbed mean $\hat{\mu}(x_t|y)$ is:

$$\hat{\mu}(x_t|y) = \mu_\theta(x_t|y) + s \cdot \Sigma_\theta(x_t|y)\nabla_{x_t} \log p_\phi(y|x_t) \tag{3.16}$$

The coefficient $s \geq 1$ is called the *guidance scale*; increasing the value of s improves sample quality by moving the generation away from random ones

but at the cost of diversity.

Two kinds of guided diffusion are proposed:

- **Classifier-free guidance:** by utilizing this technique the need for a separate classifier for guidance is eliminated. Instead of using a specific label y in a class-conditional diffusion model $\epsilon_\theta(x_t|y)$, a null label $\emptyset$ is introduced with a fixed probability during training. During sampling, the model's output is extrapolated in the direction of $\epsilon_\theta(x_t|y)$ and away from $\epsilon_\theta(x_t|\emptyset)$:

  $$\hat{\epsilon}_\theta(x_t|y) = \hat{\epsilon}_\theta(x_t|\emptyset) + s \cdot (\epsilon_\theta(x_t|y) - \epsilon_\theta(x_t|y))$$

  To enable classifier-free guidance using generic text prompts, the text captions are occasionally replaced with an empty sequence ($\emptyset$) during the training process. The main advantage of this approach is that it allows a single model to utilize its knowledge effectively, while also simplifying the guidance mechanism. This is often seen as a *hack*, and it is speculated that a better understanding of the transformer architecture would lead to CFG being unnecessary.

- **CLIP** 2.3 **guidance**: performed in a similar fashion to DALL-E best of N images (figure 3.14). CLIP is used to evaluate the similarity between an image and a provided caption. This is achieved by computing the pairwise cosine similarity between an image and caption encoding. The resulting score is high if the image and caption are similar and low otherwise. The mean of the reverse process is then perturbed by incorporating the gradient of the dot product between the image and caption encodings with respect to the image:

  $$\hat{\mu}(x_t|c) = \mu_\theta(x_t|c) + s \cdot \Sigma_\theta(x_t|c)\nabla_{x_t}(f(x_t) \cdot g(c))$$

  A specially trained noise-aware CLIP is used to obtain the correct gradient, even though regular models have been shown to work fine.

The model described is based on the *Autoregressive Diffusion Model* (ADM) architecture described in [10], with the inclusion of additional text conditioning information. For each noised image x_t and its corresponding text caption c, the model predicts $p(x_{t-1}|x_t, c)$. To incorporate text conditioning, the text is encoded into a sequence K and then passed through a transformer model. The output of the transformer replaces the embedding in the ADM model.

Furthermore, the output is separately projected to match the dimensionality of each attention layer in the ADM model. The key and value components of the transformer output are concatenated with the attention input at each layer of the ADM model, making use of cross-attention.

The final test showed that classifier-free guidance produced the most realistic samples when compared to CLIP guidance, as it is hypothesized that the model learned to produce adversarial examples to maximise the CLIP gradient.

3.9 DALL-E 2

Contrastive models like CLIP 2.3 have been shown to learn robust representations of images that capture both semantics and style. The approach proposed in *Hierarchical Text-Conditional Image Generation with CLIP Latents* [7] is to leverage these representations for image generation with a two-stage model.

Contrastive models like CLIP [20] have shown over multiple implementations (such as [11]) their ability to learn robust representations of images that are capable of capturing both semantics and style. Starting from this intuition, CLIP latents could be an effective way to replace the transformer-learned encodings in Diffusion Models. The paper *Hierarchical Text-Conditional Image Generation with CLIP Latents* [7] explores this possibility by proposing a two-stage model that makes use of CLIP representations to enhance the diffusion process.

Consider a dataset of pairs (x, y) of images x and corresponding captions y, let z_i and z_t be their embeddings respectively. There are two components involved in the process. A *prior* denoted as $P(z_i|y)$ generates CLIP image embeddings z_i conditioned on the given captions y. Then, a *decoder* $P(x|z_i, y)$ generates images x by utilizing CLIP image embeddings z_i and optionally using text captions y as conditioning factors. By using the two components together, we obtain a generative model $P(x|y)$. The text input is converted to the shared image-text latent space that CLIP learned and the decoder *inverts* these encodings back to the pixel space in a process that is named *unCLIP*:

$$P(x|y) = P(x, z_i|y) = P(x|z_i, y)P(z_i|y)$$

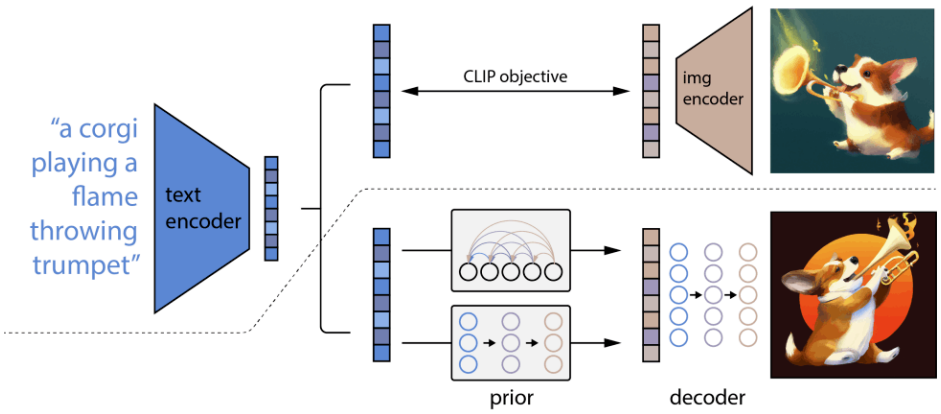


Figure 3.16: The two-stage process described in the paper. Above the line, we can see the CLIP [20] learning process as in the original implementation. Below, is the generation process. Given a textual input, the corresponding CLIP encodings are computed and used to condition the generative process. From [7]

The decoder is based on a diffusion model 2.5 that is conditioned on CLIP image embeddings. Specifically, the architecture is a variation of the one proposed in GLIDE 3.8. The CLIP embeddings are used for a dual purpose: first, as additional information in the existing timestep embeddings, and second, as a context that is concatenated to the text encoder. Classifier-free guidance is obtained by removing the conditioning provided from the text and from the CLIP embeddings some of the times during training. For the generation of high-resolution images, a diffusion upsampler is employed, progressively increasing the dimensions from 64×64 to 256×256 and then to 1024×1024. To enhance robustness, a slight corruption, in the form of Gaussian noise addition, is introduced before each step.

Two kinds of priors are tested. The Autoregressive (AR) prior involves converting the CLIP image embedding z_i into a sequence of discrete codes and predicting them autoregressively, conditioned on the caption y. The Diffusion prior directly models the continuous vector z_i using a Gaussian diffusion model, also conditioned on the caption y.

CLIP embeddings are computed as a deterministic function of the caption, making them a viable conditioning tool (variance in the output would lead to inconsistent generations given the same generation parameters and text) for the prior. Same as in previous experiences, Classifier Free Guidance is

used to improve prompt adherence and image quality.

Other than 0 shot generation, DALLE-2 allows for other operations such as variations, which apply a different amount of noises between reconstruction steps to change the generated output, interpolations, which traverse through interpolation between CLIP embeddings, and text diffs, which allow for image manipulation by computing a text diff vector $z_d = \text{norm}(z_t - z_{t_0})$ between the two captions and interpolating through that while changing from a caption to the other.

3.10 Textual Inversion

The development of large text-to-image models such as Imagen 3.11, DALL-E 2 3.9, and the latent diffusion 2.6 based Stable Diffusion 2.7, made available to the public the ability to generate images from a textual input, allowing for an already great degree of customizability. Text alone though is not enough to generate an existing subject, a possibility which would enable the same subject to be depicted in different places and contexts. Same as for Large Language Models, Diffusion Model's size makes them unfit for finetuning tasks due to time and cost, often leading to other issues such as catastrophic forgetting[4]. The paper *An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion* [46] introduces a technique to introduce new concepts in existing large diffusion models through inversion named *Textual Inversion*.

The general idea is to find a novel *pseudo-word*, denoted as S_*, which is still related to natural language embeddings while corresponding to a specific subject. This pseudo-word can then be utilized in textual queries for the generative model. The objective is to identify word embeddings for S_* that, when used in a textual prompt, will condition the generation towards the trained subject.

To accomplish this, given a text embedding space on which the inversion is carried, and a learned embedding v_* to which we want to bind our subject, during the training phase we update the value for v_* such that it injects the subject we want to teach into the existing vocabulary. Rather than updating

[4]The tendency of an artificial neural network to abruptly and drastically forget previously learned information upon learning new information, def. from [45].

Figure 3.17: By using pseudo-words found in the embedding space that depicts the subject, we can place it in new scenes. From [46].

the knowledge of the pretrained model, we leverage the existing knowledge to *hack* the new subject into it. The desired encoding is found through direct optimization of the standard LDM objective (2.25). To condition the generation process, randomly sampled context texts derived from CLIP are utilized. The complete objective, given the text conditioner denoted as $\tau_\theta \equiv c_\theta$, is as follows:

$$v_* = \arg\min_v \mathbb{E}_{z\sim\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}[||\epsilon - \epsilon_\theta(z_t, t, c_\theta(y))||_2^2]. \tag{3.17}$$

The same training scheme of the original Latent Diffusion Model [26] implementation is used (take an input text-image pair, add noise, try to reconstruct from text only, compute loss) for training.

The learned concepts can be used in a multitude of scenarios such as image variations, text-guided synthesis, concept composition (multiple learned concepts can be used in the same generation), and even style transfer.

3.11 Imagen

Latest to come out, *Imagen* [8] builds on the existing literature on diffusion models and extends it by proposing to use a large language model pretrained on only text (such as T5 [47] and GPT [23]) as a text encoder to condition the diffusion process.

The main building blocks of Image are the following (fig. 3.19): "a large

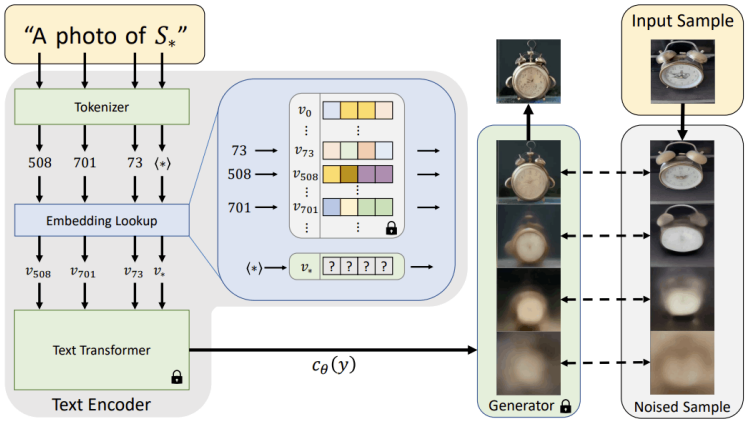


Figure 3.18: Outline of the text-embedding and inversion process. A string containing the placeholder is tokenised. The tokens are converted to embeddings v. Finally, the embedding vectors are transformed into a single conditioning code $c_\theta(y)$ which guides the generation. The embedding vector v_* associated with S_* is optimised. Taken from the original paper.

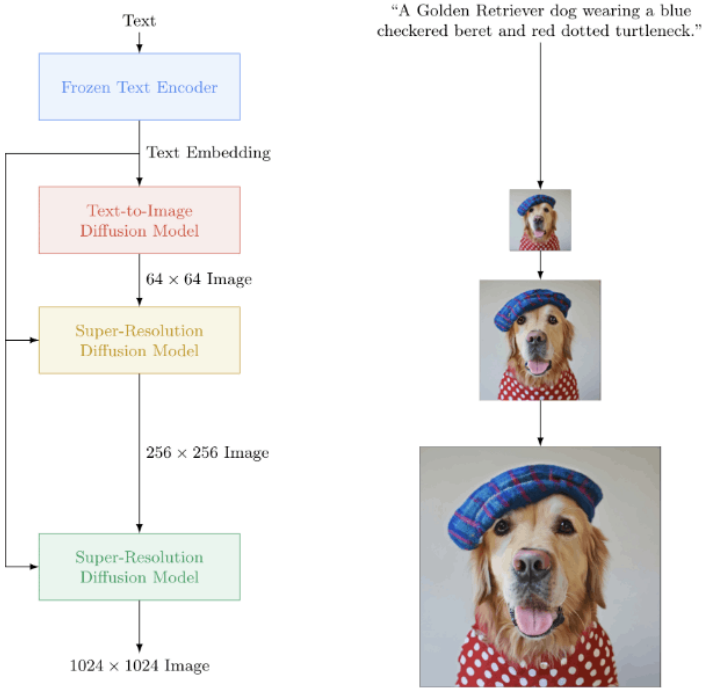


Figure 3.19: Visualization of Imagen. From the original paper.

frozen T5-XXL encoder to encode the input text into embeddings, a conditional diffusion model to map the text embedding into a 64×64 image, and finally a text-conditional super-resolution diffusion models to upsample the image" (from [8] pp. 1-2).

The main difference from models such as GLIDE 3.8 and DALL-E 2 3.9 is in the use of a large frozen model: the intuition behind the choice is that image-text trained encoders inherently possess the ability to capture visually and semantically significant representations that can be later leveraged for text-to-image generation. However, large language models (LLMs) can serve as a viable alternative for encoding text in the context of the same task. LLMs are exposed to a much more varied corpus of text, learn better and more meaningful language relationships, and are generally larger and with more parameters.

The diffusion model adopted in the paper is a standard conditional diffusion model that makes use of classifier-free guidance (CFG). On this topic, a solution to aberrations produced when using high guidance values is proposed along with the new architecture. Using a high guidance can make the x-prediction at timestep t $\hat{x}_0^t$ exceed the bound [−1,1]. Sampling iteratively from predictions that fall outside the bound causes the sampling process to diverge and produce unnatural images. Two solutions are analyzed to tackle this problem: Static Thresholding, in which the x-prediction is clipped to the fixed range of $[-1, 1]$, and Dynamic Thresholding, in which at each sampling step, the threshold value s is set to a specific percentile absolute value in $\hat{x}_0^t$. If s is greater than 1, $\hat{x}_0^t$ is then thresholded to the range of $[-s, s]$ and divided by s. This approach ensures that saturated pixels are pushed inwards, effectively preventing oversaturation.

The last step, consisting in a super-resolution augmentation to improve image quality, is performed using a cascaded diffusion model in which the noise added at the previous step is passed onto the next one as conditioning. The resulting model is thus aware of how much denoising is necessary, resulting in high-fidelity images and artefact removal.

3.12 ControlNet

With the development of large text-to-image models, generating a high-quality, coherent image requires only a short prompt written by a user. Often though, the generated image does not represent the idea that the user had

behind the prompt. As we are dealing with extremely large models though, trained on upwards of 5 billion data 2.7, both finding a comparable dataset to specialize the generation task and training the model itself becomes a daunting task to tackle. *ControlNet* [48] is proposed as an architecture that through specifically trained models manages to learn new conditioning patterns for existing diffusion models. Rather than finetuning the existing model directly, ControlNet clones an existing model into a "trainable copy" while retaining the original model as frozen weights. By updating the gradient on the cloned copy only, more aggressive task-specific learning can be performed without losing the generation capabilities of the original model.

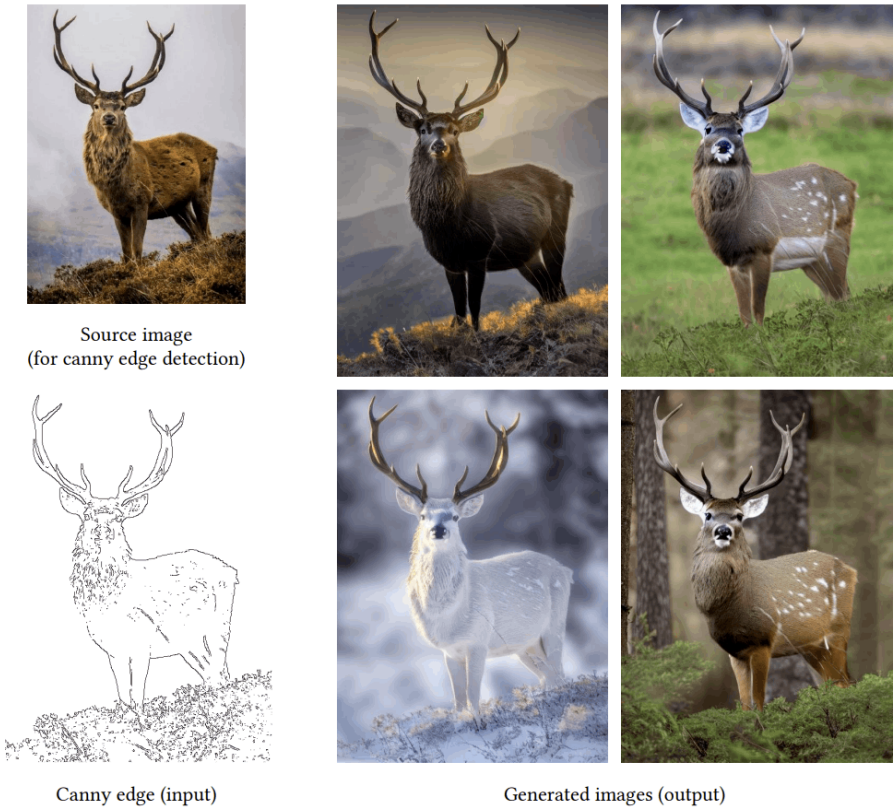


Figure 3.20: Stable Diffusion controlled using a canny edge map. First, from the input image, a canny edge map is generated. The images generated on the right make use of the map only, using it as additional input to guide the generation process.

The way in which ControlNet adds conditioning is by interacting with the

frozen model through the input layers of its network blocks[5], effectively influencing the whole neural network. Taking image $x \in \mathbb{R}^{h \times w \times c}$ with $\{h, w, c\}$ being height, width, and channels as an example, a neural block with parameters Θ that manipulates the image to obtain another output y is defined as $\mathcal{F}(\cdot, \Theta)$:

$$y = \mathcal{F}(x; \Theta) \tag{3.18}$$

For each network block with parameters Θ, ControlNet creates a copy Θ_c while keeping the original parameters frozen. The connection between the original and the cloned copy is then obtained through a novel layer called *zero convolution*, i.e., "a 1×1 *convolution layer with weights and biases initialized with zeros*" (from [48] pag. 4). Given the zero convolution operation $\mathcal{Z}()\cdot;\cdot$ and two network blocks $\{\Theta_{z1}; \Theta_{z1}\}$ that are used to build a basic network on top of which to apply ControlNet to:

$$y_c = \mathcal{F}(x; \Theta) + \mathcal{Z}(\mathcal{F}(x + \mathcal{Z}(c; \Theta_{z1}); \Theta_c); \Theta_{z2}), \tag{3.19}$$

where y_c becomes the output of the neural network block. The difference is shown in figure 3.21.

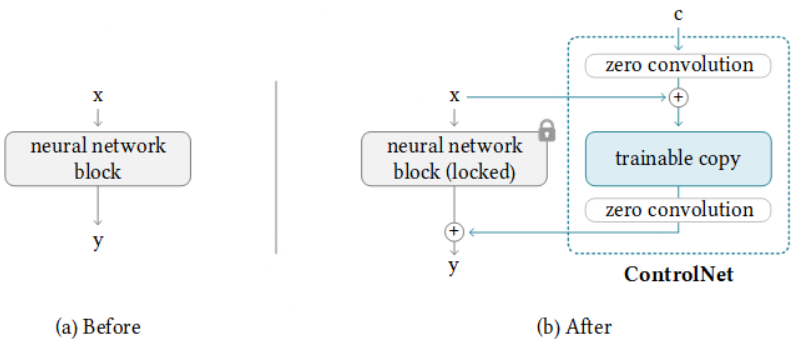


Figure 3.21: The approach to apply a ControlNet to an arbitrary neural network block. The x, y are deep features in a neural network. "+" refers to feature addition. "c" is an extra condition that we want to be added to the network. From [48].

[5]A series of neural layers that are put together frequently to form a unit, such as "resnet" block, "transformer" block, ...

When applied with no weights, the ControlNet acts as if it wasn't present, allowing the original network to work regularly. As the training progresses, the weights of the convolution are updated towards their optimised state.

Following is the application of a ControlNet to the denoising U-Net of Stable Diffusion (figure 2.12, described in 2.7). First, the input to the ControlNet has to be converted to a 64×64 feature space using an encoder network $\mathcal{E}(\cdot)$ that is trained along the full model to match the convolution size. The resulting image-space conditions given input c_i are $c_f = \mathcal{E}(c_i)$. The ControlNet is then connected to each block starting from the middle one in a way that requires no gradient computation in the original network.

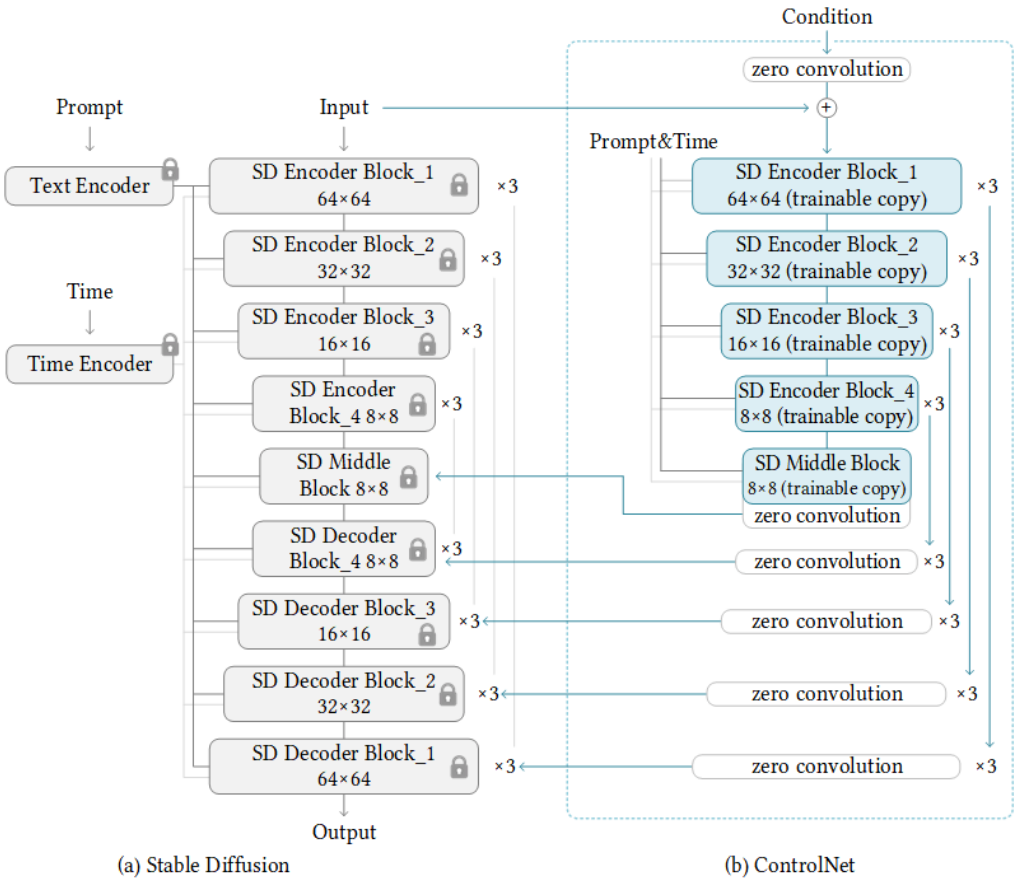


Figure 3.22: ControlNet in Stable Diffusion. The grey blocks are the structure of Stable Diffusion, the blue blocks are ControlNet. From [48].

Training is then performed. Given an image z_0, a noisy image z_t, timestep t, text prompts c_t, an original sample ϵ, and a task specific condition c_f, the

overall learning objective $\mathcal{L}$ is given by:

$$\mathcal{L} = \mathbb{E}_{z_0,t,c_t,c_f,\epsilon\sim\mathcal{N}(0,1)}[||\epsilon - \epsilon_\theta(z_t, t, c_t, c_f)||_2^2] \tag{3.20}$$

During training 50% of the prompts c_t are dropped to encourage ControlNet to learn from the provided conditioning (when no prompt is present the encoder tends to learn more semantics from the input control map as a replacement). Along with the paper, several controlnets that allow for output control were released, such as Canny Edge (which uses canny edge maps that can be easily obtained from canny edge detection algorithms), Human Pose (using human pose skeletons as described in [49]), Depth (which uses depth maps, widely used in the 3D rendering field), and many more.

3.13 Storyboarding

In recent years a growing number of studies have started to focus on the automation of video editing tasks. While these works, such as [50] and [51], achieve impressive performance in the generation of a video, either given as input a textual prompt [51], or a combination of textual prompt and image [50], they focus on the generation of motion and do not take into account the shot type used.

By having the ability to generate more scenographic shots, one of the many applications that become available is text-to-image storyboard creation. Existing storyboarding tools either extend digital painting applications (e.g. [52]), allow the user to place predetermined objects in a scene to compose the desired frame (e.g. [53]), provide a simple interface to create a reference of the desired scene (e.g. [54]).

For more deep learning-related approaches, StoryGAN [55] generates a sequence of images that describe a story written in a multi-sequence paragraph. To do this, the proposed framework uses a sequential Generative Adversarial Network [31] that consists of a Story Encoder, an RNN-based Context Encoder, an image generator conditioned on the story context, and an image/story discriminator that ensures consistency. Diffusion Models allow for high-quality generation on multiple domains without needing specific training, and a better understanding of the conditional text input than GANs. The conditioning based on previous frames could be a possible approach for increased temporal consistency even in LDMs.

Dynamic Storyboarding [56] approaches the storyboarding task directly by automatically composing scenes out of user inputs by simulating in a virtual environment the scene and discriminating the best proposal out of the available ones. This approach generates rich and complex dynamic (video) storyboards, but it lacks the customizability and intuitiveness that Diffusion Models offer through textual conditioning. Furthermore, by using ControlNet 3.12 trained networks it's possible to add conditioning through more inputs such as scribbles, which at the cost of a slightly higher effort can lead to much better generations.

Chapter 4

Contribution

4.1 Objective, Intuition and Architecture choice

Recently released models show an increasingly better ability to generate realistic samples, and the ability to guide the generation process towards a desired textual input, usually referred to as *prompt*, opens the doors to many tasks that were previously considered unfeasible. Generating captivating images and "AI art" are just one of the many possibilities that text-to-image diffusion models enable; the ability to insert novel and specific subjects or styles without needing a full-fledged (and therefore expensive) finetune that is enabled by Dreambooth and Textual Inversion 3.10 truly makes image generation a full artistic medium, allowing any end user with a reasonably ($\approx 8GB$ VRAM) powerful setup to customize generation with his own subject. We decided to apply this technique to the cinematographic field, with the goal of *teaching* to a pre-trained copy of Stable Diffusion (specifically *stable-diffusion-1-5* [2]) how to generate specific shot types: *close shot*, *medium shot*, and *long shot*, as the scale of a shot is one of the most determining factors in influencing the viewer perception of a subject [9]. Of the many applications that this ability offers, the final product is then tested on the storyboarding field: the ability to generate cinema-like shots that follow a given prompt and shot type suit well the storyboarding task, allowing the user to quickly generate a multitude of draft shots to choose from without having to rely on hand-drawing them.

The intuition that I followed to perform this training is that a specific

shot type can be considered akin to a style, which LDMs were shown to be capable of learning. The reasoning behind the intuition is the following: given that an LDM is able to learn the drawing style of an artist, imagine that this artist always painted their subjects as close shot portraits: when learning their style, the learned concepts would include this *idea* that the subject is always close to the camera.

Performing a full-fledged finetuning though would be unfeasible due to resources and time limitations, and tests to confirm that the approach is feasible would take longer than desirable. For that reason, I chose to test our intuition using either Dreambooth or Textual Inversion. To further close in into one of the two, other than the comparative results showing the better performance of Dreambooth in its own paper [3], I looked at the statistics of the top 100 models from Civitai[1] (tab. 4.1) using the publicly available API, and saw that over 90% of them are Dreambooth based models, solidifying our choice for a finetuning technique. Another further optimization that emerged after the release of Dreambooth was the implementation of Low Rank Adaptation 2.4 on top of it [30], which enables a faster and more efficient training as well as a significantly lower ($\geq \times 0.1$) model output size for minimal losses.

type	**number**	**downloads**
DreamBooth Checkpoint	70	5.575.099
Lora DreamBooth	26	1.670.288
Textual Inversion	4	348.187

Table 4.1: Total number and their respective downloads of the top 100 models hosted on Civitai. "DreamBooth checkpoint" include also checkpoints merged with a Lora Dreambooth.

Performing this finetuning is highly reliant on the training data used. As such, building a high-quality dataset is a necessity to generate high-quality samples and is one of the focuses of this work. My contributions are the following:

- A methodical outlining of the process necessary to finetune a style in an existing Latent Diffusion Model using state-of-the-art techniques, and a

[1] Civitai is one of the most popular checkpoint sharing-hosting sites that emerged after the release of Stable Diffusion

specific application towards shot types.

- A methodical approach in building a dataset for the finetuning task using external state-of-the-art tools, and its application towards building a 127.000 large cinema shots dataset from which to pool the training images.

- The application of the two aforementioned approaches to generate three shot-types specific checkpoints (close shot, medium shot, and long shot) and their application in the storyboarding task.

4.2 Method

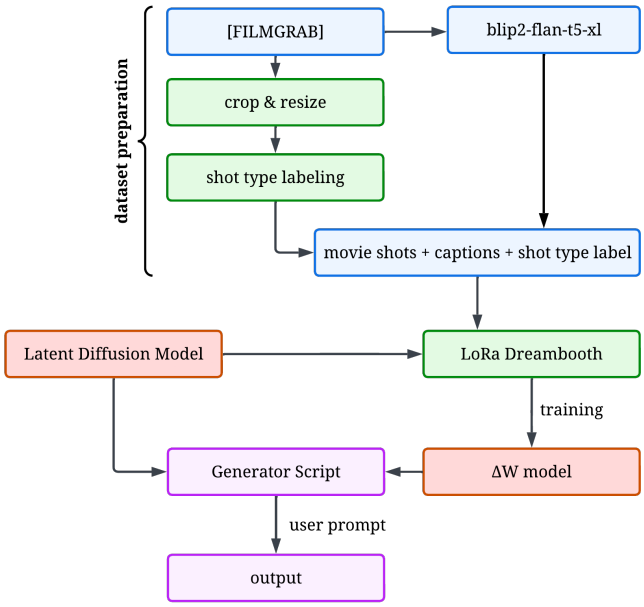


Figure 4.1: A visual representation of the full pipeline. The output weights, noted as ΔW, are used along their counterpart pre-trained model W' to generate samples.

4.2.1 Data Preparation

Similarly to other deep-learning models, the data that is chosen for the training is one of the most determining factors of a good result [57]. Looking at

the available datasets of movie shots, not many are readily available or of a high enough quality. CINESCALE for example has a very large amount of data, but the labels are often wrong (and thus require a double check by hand), the movies have lower resolution and quality than desirable, and the dataset is generated by using every available frame, rather than the *best looking* ones. For this reason, I decided to build a dataset ourselves starting from all the available photos from [FILMGRAB][2], which contains high quality, hand-picked movie frames. First, I built a scraper 4.2.1 to automatically download all the available movie galleries from the site. Looking at the html code of the page, each movie shots gallery has a consistent div of class `bwg_download_gallery` that has a tag `a` that contains a link to download the entire gallery as a zip file. By iterating over all the available movie galleries, all shots were downloaded.

```
1
2  from urllib.request import urlopen
3  from bs4 import BeautifulSoup
4  import requests
5  from tqdm import tqdm
6  import pandas
7
8  # This is a previously scraped list of the available
       movies in the format
9  # url, movie_name
10 filmgrab_movielist =
       pandas.read_csv("filmgrab_movielist.csv").dropna()
11 failedlist = []
12
13 # Iterates over all the movies using tqdm to keep track
       of the progress
14 for index, row in tqdm(filmgrab_movielist.iterrows(),
       total=filmgrab_movielist.shape[0], desc=f'Downloading␣
       Movie␣shots'):
15     url = row["url"]
16
17     # Two except statemets, one external for general
           errors, which keep track of the failed
```

[2]Open source for research purposes.

```
18      # movies. One internal in case the page is not
            loaded correcly. Only 12 movies failed.
19      try:
20          filename = url.split("/")[-2]
21          try:
22              page = urlopen(url)
23          except:
24              print(f"Error␣opening␣the␣URL␣{url}")
25
26          # The html parser BeautifulSoup is used.
27          soup = BeautifulSoup(page, 'html.parser')
28
29          # There is a consistent element in the pages
                that allows to download a .zip file
30          # of the movie shots gallery. We look for that
                and use requests to download the zip
31          content = soup.find('div',
                class_="bwg_download_gallery").a["href"]
32          open(f"filmgrab_zips/{filename}.zip",
                "wb").write(requests.get(content).content)
33      except:
34          failedlist.append(url)
35          continue
36
37  print(failedlist)
```

The collected shots totalled 127.000 from 2166 movies, with 12 failures in the scraping phase which I ignored as the collected frames were more than enough. Content-aware cropping [58] was applied to reduce the images to the maximum training resolution of 768×768 pixels without applying size reduction that would distort the images. The same process could be tested using the Seam Carving [59] algorithm, which wasn't considered during the first preprocessing. From the original images, all the pictures with less than 3 colour channels were pruned, as well as the ones coming from movies released before 2013 to guarantee a certain degree of image quality and resolution. Out of the remaining 41750 shots from 729 movies, only 600 (200 per shot type) were then to be selected. As the number of required pictures is relatively small, shot-type labelling was performed by hand, as there is no classifier with 100% accuracy, and the images would have to be checked anyway. Randomization was achieved by sampling single shots from all the available ones

and by assigning a label, thus adding it to the training set, if and only if the quality and crop were deemed to be appropriate. As the training set is small (compared to the size of the LAION-5B [28] dataset that was used for the original training), the finetuning is very sensitive to bad samples. A good movie variety was kept to not teach unwanted subjects.

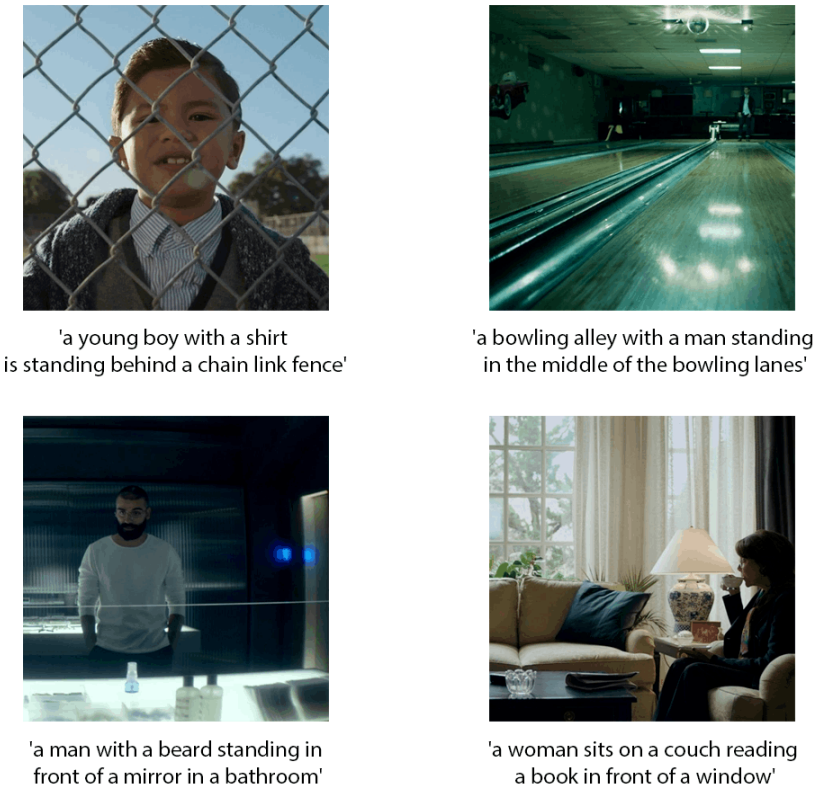


Figure 4.2: Some examples of the training data pairs used.

The final step was adding textual captions. To aid in the captioning process, the Vision-Language model *blip2-flan-t5-xl* [60] was used to generate a first *CLIP* 2.3 *style* caption which was subsequently checked by hand; as the learning process consists in comparing samples generated with a caption to the original image that caption was assigned to, by describing the contents of the training image thoroughly the model "picks up" the missing element, which in our case would be the shot type. The Final training data is composed of 3 subsets of 200 image-caption pairs. The same procedure can

be repeated for any starting dataset of any size, as the number of required pictures is no more than 200, in order to teach a different style or a subject.

4.2.2 Model Training

Following an analysis of the state of the art [3], a consideration of the available resources, and an analysis of user's preference 4.1, I decided to use Dreambooth 2.8 as our finetuning approach of choice. The idea behind DreamBooth is to, given a few input images ($\approx 3-5$), bind the subject to a *unique identifier* such that when it is used in the prompt, along with the class it belongs to (e.g. "A [V] dog"), the prior knowledge of the class is used along the new information to reconstruct the subject. A *"new autogenous class-specific prior preservation loss is introduced on top of the regular training objective to encourage diversity and counter language drift"* (from [3] pag. 1). During training, the model is supervised *with its own generated samples in order to retain the prior knowledge of the class and to use it along with the knowledge of the subject instance to generate new samples.*

By itself, DreamBooth already manages to significantly decrease the cost of adding a subject to an existing model. But, as a further optimization, I used Low Rank Adaptation 2.4 applied to the DreamBooth process (following the implementation found in [30]). LoRa allows efficient finetuning even in low-power devices while keeping a high-quality end result: instead of training the entire model, LoRa works by finetuning the residual: i.e. train ΔW rather than W'.

$$W' = W + \Delta W \tag{4.1}$$

Through matrix decomposition it's then possible to further decrease the number of parameters to finetune, hence reducing the size of the output model by an even larger degree.

$$\Delta W = AB^T \tag{4.2}$$

The attention layers parameters (Q, K, V, and O seen in figure 2.12) of the attention layers in the denoising U-Net of Stable Diffusion are enough to tune to obtain the desired output.

Given an existing diffusion model W, a LoRa of it is applied on top in the form of $W' = W + \alpha\Delta W$: when α is 0 the model is the same as the original one; when α is 1 the model is the same as the fully finetuned one. By applying this form of optimization to DreamBooth it's possible to achieve two major

goals: faster and less complex training, and a lightweight and more versatile output.

Once the training phase is finished, an output file is produced which contains the weights learned during training. The model is then used alongside the original one that was used as base during the finetuning process (in this case *stable-diffusion-1-5*) to synthesize images, but any other checkpoint based on the same model could be used.

Figure 4.3: *prompt*: a high-quality picture of a woman holding a cup of coffee in front of a brick building `<lora:lora_cs:1>`

The original Dreambooth paper binds a concept, such as a person or a style, to a unique unusual class identifier. One of the benefits of training using Low Rank Adaptation is that it's possible to specify how much our trained weights influence the generation process through the parameter α. As such, I decided to skip the unique identifier binding phase and just finetune the model towards our desired goal. The consequence of this is that when used, the generation is always in the trained style, which would be a detriment if the output file was expensively large as switching between and keeping several $\approx 6GB$ files can become daunting, but as the output is lightweight and easy to switch between one another, we can let the model generate only in one style, or in one shot type in this case, and just switch between finetunings when necessary, removing them altogether when not.

The way a ΔW set of weights is specified to be added to a base model W

in the generator script I used is directly in the prompt. By using the control sequence `<lora:loraname:alpha>` one can specify which weights to add and with what α.

The caption in figure (4.3) is the prompt that was used to generate the picture. The token `<lora:lora_cs:1>` instructs the generator script to use the LoRa *lora_cs* with $\alpha = 1$.

4.2.3 Generation

Once the model is successfully trained, the generative process can begin. Generation is performed by providing the model with a series of parameters along with a textual prompt describing the scene. Two kinds of prompt can be provided: a regular prompt and a 'negative prompt'. The generative process works by guiding the generation towards the text, or rather the text encodings, specified in the prompt field as done during training; at the same time, the generation is moved *away* from the encodings obtained from the negative prompt field similarly to how classifier free guidance works [12]. Usually, a negative prompt is not required, and was mostly not used in our experiments, but in more advanced generation processes it can be key in generating artistic samples. Prompt engineering[3] takes a big role in the generative process, with certain prompts such as *"high quality"* and *"masterpiece"* guiding the generated image towards more aesthetically pleasing results. This is a direct consequence of the training: as LAION-5B [28] is based on images scraped from the web, their description can often include qualitative words (imagine a wallpaper site where the description of the wallpaper is given by its tags, high-resolution would be an indicator of a high-quality wallpaper). By using these tokens the generation is influenced towards these training images, usually improving the end result. The most meaningful generation parameters are:

- **Sampler**: at each step of the diffusion process a certain amount of noise is predicted and subtracted from the image. The sampler takes care of both computing the predicted noise and scheduling the noise level at each sampling step so that an equally noisy image can be sampled at

[3]Prompt Engineering is to craft specific textual prompts that when used as input in large language models or text-to-image diffusion models improve the output by narrowing the output field or 'hacking' the generative process.

that step. There are many available with different benefits: those based on traditional ODE (Ordinary Differential Equation) solvers, such as Euler and Heun [61], DPM++ [62] enhanced with the suggestions from [63], DDIM [64], ... Furthermore, there exist a class of samplers named *ancestral* which add a low amount of noise after each iteration to further stimulate the generative process but might fail in converging consistently.

- **Steps**: changes how much noise is subtracted from the image at each step, the larger the number of steps the slower the generation process is, but finer details might be developed this way. Different samplers require different amounts of steps to perform at similar levels.
- **CFG Scale**: short for Classifier Free Guidance scale, classifier free guidance is a technique that moves the generated samples away from random unlabeled ones, essentially making the generated image adhere more to the provided prompt.
- **Seed**: determines the initial noise map, different seeds will result in different images.

Finally, the value α that determines how much the ΔW model weights are applied takes an important role in the generative process. As there is no deterministically perfect way to train a Lora DreamBooth model, sometimes lowering how much influence the finetune has can improve results.

Chapter 5

Results

This chapter contains an analysis of the results obtained. The testing setup closely follows the one proposed in the original Dreambooth paper [3], looking at the CLIP-T and DINO metrics, as well as using a human evaluation survey. Evaluating the performance of a latent diffusion model numerically can be awkward as the metrics that can be used to measure image similarity and adherence don't take into consideration qualitative aspects, which is why I'll show visual examples and human-based metrics.

5.1 Training Setup

The final training was performed using the process described in 4.2; the machine used for training was a regular home desktop with an $8GB$ RTX 3070, $16GB$ RAM and a 6-core Ryzen 5 processor, although available VRAM acted as the major bottleneck. Training took 7 hours for each model for a total of 21 hours with batch size 2. Due to $VRAM$ limitations in the training machine, the images were used at a lower resolution of 512×512, which is still an accepted training resolution for Stable Diffusion and should only cause the output to be of lower quality. The full training parameters are reported in table 5.1, the learning rate and text encoder learning rate are the ones proposed in the original LoRa implementation [4][30] (the text encoder learning rate is half of the u-net learning rate), the rest are the result of empirical experimentation and other users experiences. As far as I know, there are no optimal parameters for training, and the matter is still in active development.

resolution	512×512
unet learning rate	$1e-4$
text encoder learning rate	$5e-5$
network alpha/dim	172
optimizer	AdamW8Bit
learning rate scheduler	cosine with 10% warmup
epochs	15
steps per image	15
images	200
total steps per epoch	3000
total steps	45000

Table 5.1: The training parameters used. The script used for training can be found in [30].

5.2 Dataset

To compute the metrics, a testing dataset was generated following the same process described in 4.2.1. Out of the same subset of 41750 filtered and resized shots, 1800 shots were sampled with an even distribution between the three shot types. For each sample, a caption was generated using BLIP-2 [60] with no supervision. The image-caption pairs were then randomly sampled to generate two pictures with the same starting seed, one using the baseline model and one using ours, 1500 times for a total of 500 pairs of generated shots per shot type. For parameters, `DPM++ SDE Karras` was chosen as it is both one of the most used[1] and the one with which I managed to generate the most visually pleasing outcomes. The steps were set at 16, slightly above the suggested value for the sampler of 15, and the cfg_scale was lowered to 6 from the default 7 to improve image quality over prompt adherence.

[1]Looking once again at the API collected data from Civitai [65], which also hosts users generations, `DPM++ SDE Karras` is used in 23.4% of the generations, overcomed only by `Euler a` and `DPM++ 2M Karras`, both of which performed qualitatively worse.

sampler	DPM++ SDE Karras
steps	16
seed	random
cfg_scale	6
prompt	a high-quality [shot_type] picture of [caption]
size	512×512

Table 5.2: The pararameters used for generation during testing

5.3 Parameters Effect

In this section, I'll discuss the effect that the generation parameters have showing contextual examples. Unless differently specified, the setup is the same as described above, with one of the parameters being changed for each set of images in the x-axis and the three trainings in the y-axis. The prompt is the same for every picture: *"A high-quality picture of a man with a beard drinking coffee at the table* `<lora:loraname:1>`*"*. Each set of generations took no longer than 3 minutes.

5.3.1 Sampler

Figure 5.1: Comparison between different samplers. The generation is quite similar for every sampler

Changing the sampler boils down more to a matter of personal preference rather than one of noticeable difference, as it only changes how fast convergence is reached influencing the amount of noise being removed at each step. From a personal experience, `DPM++ SDE Karras` performed usually better when compared to the other ones (and it is also one of the most recent), and `UniPC` [66] (which was specially developed for fast convergence) performed best at low step counts (≈ 8), but the overall output at a step count that *guarantees* convergence is quite similar. A comparison is shown in figure 5.1

5.3.2 Steps

Figure 5.2: The output of a generation with an increasing number of steps. After 24 steps any difference between generations is hardly noticeable.

Increasing the number of generation steps usually improves the quality of the generated images, but only up to a limit. Even for the most demanding samplers such as `DPM++` the suggested amount of steps is around 15 to 20. After 24 steps, the differences between pictures is hardly noticeable, and are mostly due to the slight amount of noise that is added between steps. Figure 5.2 shows a comparison between different step sizes while keeping the other generation parameters equal.

5.3.3 Seed

Figure 5.3: The output of a generation with a varying seed. As expected, changing the seed can radically change the generation output. Navigating the "seed space" is often a technique adopted when generating pictures to pick the best one to work on.

Changing the seed means changing the initial noise map the reverse diffusion process is carried on. As a result of a changing seed, even if every other parameter remains the same, the image can wildly change between styles, poses and elements displayed. In a simple example like the one of figure 5.3 we can see 5 different backgrounds and slightly different subjects (although as previously said the similarity between subjects is a byproduct of the finetuning). One procedure that is usually adopted during the process of generating *AI art* consists in generating many pictures with different seeds at a low step count (≈ 10) and then selecting the best one to refine by slightly altering the prompt, the other generation parameters, and through inpainting[2].

[2]Inpainting means to select an area to generate into using the surroundings as conditioning, as described in 3.8.

5.3.4 Classifier Free Guidance Scale

Figure 5.4: Comparison between different CFG scale values. The higher the CFG the more aberrations we see, the lower the value, the blurrier and further from the prompt the generation.

By changing the value of the Classifier Free Guidance Scale (CFG) we are changing how much the picture is *moved away* from random generations. It was introduced in [12] and used for the first time in a diffusion model in GLIDE [43] where it is represented by the term s in (5.1).

$$\hat{\mu}(x_t|y) = \mu_\theta(x_t|y) + s \cdot \Sigma_\theta(x_t|y)\nabla_{x_t} \log p_\phi(y|x_t) \tag{5.1}$$

In the generated pictures 5.4, at low CFG values the image is blurrier and might miss some elements (the top left picture has no cup to drink coffee from), but at too high scale the image starts to show aberrations and excessive contrast. Usually, a value around 6/7 shows the best of both worlds, leaving the model enough room to generate correctly while guiding it towards the prompt. A higher CFG is often more desirable for more complex prompts where we require the model to follow the input more closely.

5.3.5 Alpha

Figure 5.5: A comparison between the three produced finetunings at different levels of α (y-axis). As α increases, the generated picture takes more and more aspects of the desired shot type. By changing the finetuning used, we can see a different picture, more related to the corresponding shot type, being generated.

5.4 Metrics

To get a quantitative result two metrics were adopted following the footsteps of the original DreamBooth [3] implementation testing approach. The first one is CLIP-T [20], the average pairwise cosine similarity (5.2) between the CLIP embeddings of the generated image and the prompt that generated it. The second metric, DINO [67], measures the average pairwise cosine similarity between the ViTS/16 DINO[3] embeddings of generated and real images, essentially measuring how similar the generated image is to its real counterpart. To compute these metrics, the Huggingface library `transformers` was used to fetch and compute the embeddings.

$$\text{Pairwise Cosine Similarity: } \cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{5.2}$$

The results shown in 5.3 show a slight (although significant for the considered metrics) increase for both CLIP-T and DINO scores over the baseline model. The lower increase seen in CLIP-T compared to DINO is justified as the model doesn't learn to represent more concepts/subjects (as CLIP looks at the tokens present in the picture rather than their composition I don't expect an improvement in this field) with the finetuning, but instead learns to represent them closer to the training image from a scale perspective. The section 5.4.1 expands on the shortcomings of the CLIP-T metric. From a qualitative analysis, we see that the finetuned model is more often able to generate pictures that are semantically close to the prompt used to generate them, sometimes even generating items present in the prompt that the baseline model ignored (e.g. one person when two were specified, a car not being present, etc. . .), overall showing a greater scene composition capability compared to the baseline. Of course, as there is *no free lunch* [68], although not tested on other tasks we expect the finetuned model to perform worse on other generative tasks as it loses some of its generalization capabilities; in the generated samples, for example, we can see that it often generates faces that are similar between each other.

As a secondary and ablation study, 600 more pairs of pictures were generated using the same setup as before but removing any information relative to

[3] A pretrained vision transformer used for classification tasks.

	CLIP-T	DINO
baseline	0.3221	0.4163
ours	**0.3269**	**0.4989**

Table 5.3: Results for the CLIP-T and DINO metrics on the 1500 pairs test.

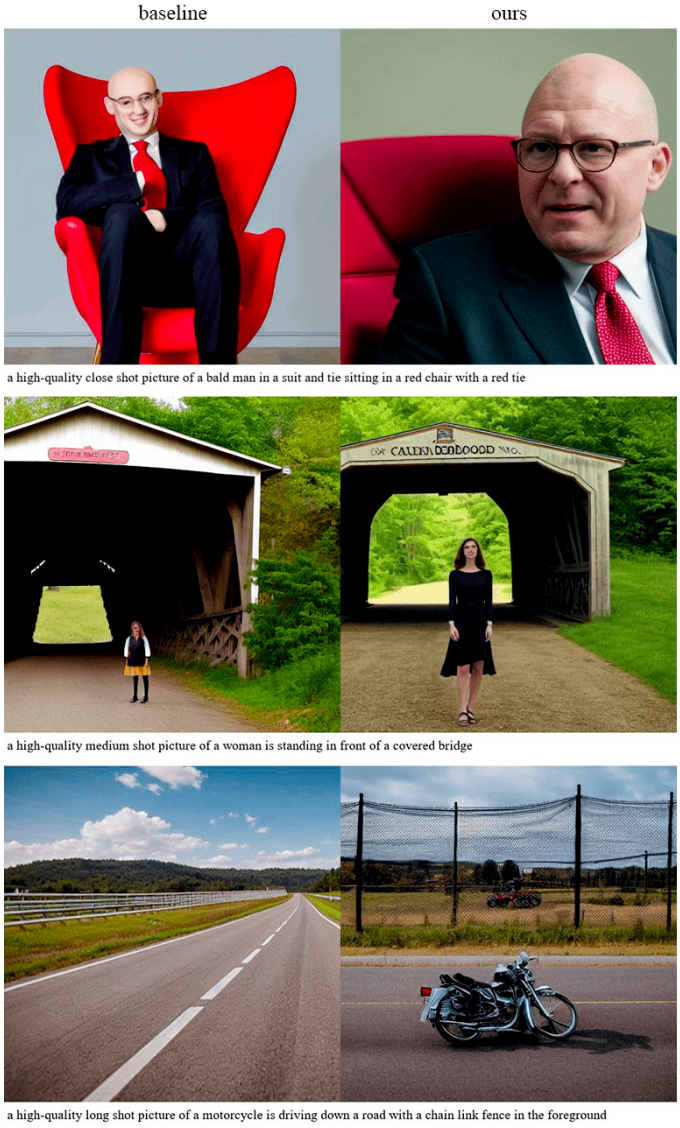
the shot type from the text conditioning. Looking at the resulting scores 5.4 We see a slight decrease in both CLIP-T and DINO scores, suggesting that the generation is not influenced much by specifying the desired shot type in the prompt.

	CLIP-T	DINO
baseline	0.3214	0.4014
ours	**0.3234**	**0.4803**

Table 5.4: Results for the CLIP-T and DINO metrics on the ablation test.

5.4.1 CLIP-T score shortcomings

Figure 5.6 shows three examples where there is a significant difference between the CLIP-T scores of the baseline and finetuned generated pictures (> 0.05). Taking as an example the third pair, the baseline picture got a CLIP-T score of 0.36, while the finetuned one got 0.27, which considering the nature of the score (that computes the similarity between the text and image embeddings) wouldn't be the expected result, as the baseline picture does not include the motorcycle specified in the prompt. The trend is reversed for the DINO metric which, for the pairs shown below, is always greater for the finetuned generation.



	CLIP-T	DINO
baseline	**0.346**	0.548
ours	0.294	**0.763**

	CLIP-T	DINO
baseline	**0.398**	0.335
ours	0.31	**0.48**

	CLIP-T	DINO
baseline	**0.36**	0.241
ours	0.276	**0.289**

Figure 5.6: Samples where CLIP-T shows a discrepancy between the expected result and actual result. The associated prompt is shown at the bottom of the picture and the relative scores in the tables on the right.

5.5 Qualitative Survey

Using the same setup, I conducted a survey on human subjects. Each subject was shown a total of 36 pairs of pictures A and B generated with the same setting and the same prompt, one from the baseline model and one from

the finetuned one. Whether a picture was labelled A or B was randomized. A very light form of supervision was applied to the generated samples to ensure the images were safe for everyone. Each pair of pictures was shown along with its associated shot type and generator prompt. For each pair of pictures, three questions were asked:

1. Which picture do you like most?
2. Which picture is closer to the associated shot type?
3. Which picture is closer to the associated prompt?

The possible answers for each question were A, B, or *neither/same* in case the two images were considered equal in a specific aspect. A total of 55 subjects with no required domain knowledge and a high degree of reliability answered the survey and the results are reported in 5.5. The data collected shows no anomalies such as subjects answering all As. We can see that even from human evaluations, my approach generates pictures that are more pleasing, closer to the associated shot type and to the prompt in close to or more than half of the cases.

Question	**baseline**	**ours**	**neither/same**
Which picture do you like most?	26.41	**57.53**	16.06
Which picture is closer to the associated shot type?	20.35	**56.82**	22.83
Which picture is closer to the associated prompt?	19.95	**49.6**	30.45

Table 5.5: Results collected from a survey conducted on 55 subjects.
The score are expressed as a percentage of the total number of answers.

Except for picture likeability, we can see that the baseline model obtained the lowest score of the three, suggesting that in most of the "worst cases", the generation is of equal quality to the non-finetuned one. Comparing the survey to the CLIP-T and DINO metrics, the results are in line with each other. The higher likeability and shot-type closeness are directly related to DINO and they are noticeably higher than prompt closeness and CLIP-T when compared to the baseline.

close shot			medium shot			long shot		
baseline	ours	neither/same	baseline	ours	neither/same	baseline	ours	neither/same
28.48	**61.67**	9.85	24.09	**58.79**	17.12	26.67	**52.12**	21.21
32.27	**46.67**	21.06	20.15	**51.36**	28.48	8.64	**72.42**	18.94
18.64	**51.97**	29.39	27.88	**42.73**	29.39	13.33	**54.09**	32.58

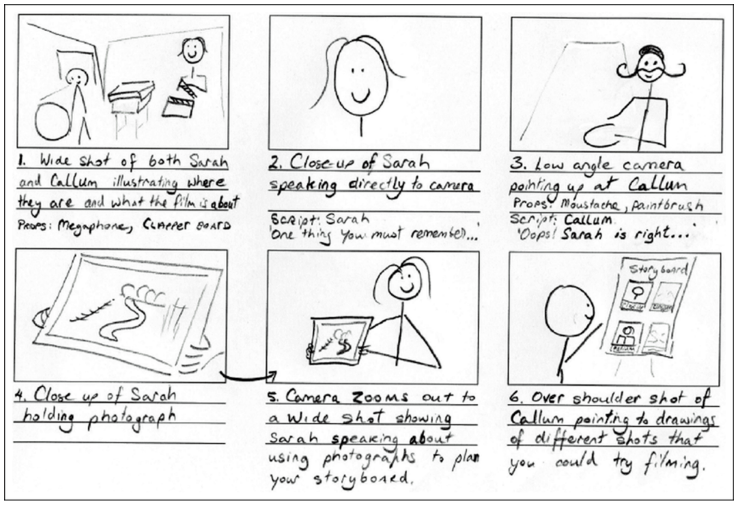
Figure 5.7: Results shown in table 5.5 subdivided by shot type.

Looking at the scores subdivided by shot type in 5.7 we can see the largest difference in scale adherence compared to the baseline for the long shot finetuning. Although shot type adherence doesn't register such a large difference for the close shot finetuning, we can see a huge difference in picture likeability and prompt adherence, supporting the claim of improved image quality (in terms of generating a cinema-like shot) when using our approach.

5.6 Storyboarding

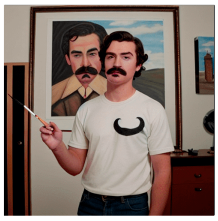
As a practical application of my method, we decided to test my approach for the storyboarding task. Figure 5.8 shows on the top a reference image (provided to entrants of the BBC's 'my place my space' competition) and on the bottom the *equivalent* picture generated using my finetunings and the prompt that was used to generate it. By generating cinematic shots, and empowering the generation with the ability to specify a shot type, the result is a storyboard with realistic and expressive images that easily convey the desired shot. Furthermore, the entire process of generating the shots took around 20 minutes, faster than it would take to generate sketches of comparable expressiveness. The shots shown were picked by generating 30/40 shots with a random seed and then by picking the best one. A simple caption encapsulating the shot was enough to generate similar images to the one in the reference.

In figure 5.9 I instead used a more detailed storyboard and enhanced it using ControlNet canny edge conditioning 3.12. A preprocessor detects the edges from the original image and uses them, along with the prompt, to condition the generation. The ability to influence the output to this degree allows multiple creation modalities, depending on the effort and the quality of the product that is needed.

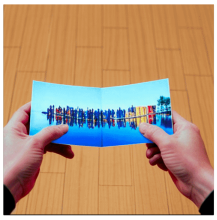


Long Shot, Sarah and Callum talking to each other in the middle of a room

Close Shot, Sarah speaking directly to the camera



Medium Shot, A front view of Callum with a mustache holding a paintbrush



Close Shot, two hands holding a photograph of something over a floor background

Long Shot, Sarah holding a small photograph in a room

Medium Shot, Callum seen from behind pointing at drawings

Figure 5.8

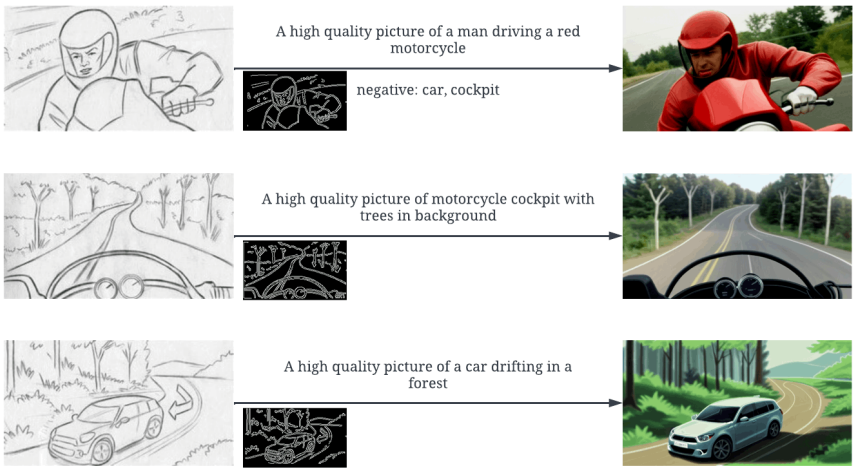

A high quality picture of a man driving a red motorcycle
negative: car, cockpit
A high quality picture of motorcycle cockpit with trees in background
A high quality picture of a car drifting in a forest

Figure 5.9

Chapter 6

Conclusions

First of all, this thesis work is presented as a compendium of the recent developments of image synthesis models, going in-depth into all the techniques that contributed to the development of the modern Diffusion Models. As my main contribution, I present an approach that uses novel techniques, released to the public during the last year, to approach a task which has not been approached much by the literature, generating cinema-like shots to aid in the storyboarding process. The proposed approach uses a finetuned version of Stable Diffusion to cater for the generation process towards this goal: by using a limited amount of shot-type labelled movie frames along with a brief description of the contents of it, I managed to teach specific shot types through the use of Dreambooth, making the learning process even more optimised by using Low Rank Adaptation, allowing the whole approach to run in a reasonable time even on a low power device compared to the industry standard. Our training results in qualitatively more pleasing output images, that more often adhere to the given prompt and shot type, as shown by a survey conducted on human subjects. For non-qualitative metrics, we test our approach using a similar setup to the one proposed in [3] and manage to obtain an increase in both the CLIP-T and DINO scores, with the latter showing a noticeable increase compared to the baseline. As a means of obtaining the required training images, I showed a detailed and easily reproducible way to create a dataset of images to use for training, which offers the possibility to easily swap the base dataset to teach a specific desired style. Finally, I showed the effects that the different generation parameters have in the generative process, and a comparison between a traditional sketch storyboard and one generated using my approach (which has much more expressiveness

and can be used by anyone), as well as a ControlNet enhanced storyboard (which manages to add expressiveness to an existing detailed storyboard in little time but requires a little extra knowledge).

Chapter 7

Future Developments

The field of image synthesis is still one that holds much potential for applications and developments, with new techniques being released and developed as we speak. While writing this work for example, a paper proposing a use for ControlNet to convert videos from one style to the other (such as realistic to anime) [69] released, showing a plausible solution for one of the most difficult tasks in image synthesis: subject and temporal consistency between frames. Regarding this work, there are several possible future developments. First, finetuning using a more powerful GPU is due: as the collected images are all in the highest possible resolution of 768×768, using a higher resolution would lead to an immediate improvement in image quality without further work necessary. The training parameters so far are mostly derived from empirical experience: the chosen parameters for training were decided through empirical experimentation and other user's experience, as there is no standard in the matter; a full grid search to optimize them would take a lot of time and resources, as such developments in the literature might lead to an improvement in the training results. Although it was deemed unnecessary as Dreambooth was shown to achieve state-of-the-art results, a comparison between our approach, full finetuning, and textual inversion might bring to light interesting results. Finally, an architecture to automatically build a storyboard out of a movie script might be developed using event-extraction: [70] proposes an approach to use an LLM to extract the relevant events out of an existing movie script which could be key in enabling this. Due to its recency, no code or model was made available yet, but by combining it with our approach, a more complete tool would be possible.

Bibliography

[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG] (cit. on pp. v, 1, 20, 21, 49).

[2] Runaway ML Stability AI. *Stable Diffusion release blog post*. https://stability.ai/blog/stable-diffusion-public-release. (accessed 23-May-2023). 2022 (cit. on pp. v, 1, 28, 62).

[3] Nataniel Ruiz et al. *DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation*. 2023. arXiv: 2208.12242 [cs.CV] (cit. on pp. v, 29–31, 63, 68, 72, 79, 86).

[4] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL] (cit. on pp. v, 18, 19, 72).

[5] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL] (cit. on pp. 1, 4, 8, 10).

[6] *ChatGPT*. https://openai.com/blog/chatgpt (cit. on p. 1).

[7] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV] (cit. on pp. 1, 51, 52).

[8] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022. arXiv: 2205.11487 [cs.CV] (cit. on pp. 1, 54, 56).

[9] Brendan Rooney and Katalin E. Balint. "Watching More Closely: Shot Scale Affects Film Viewers Theory of Mind Tendency But Not Ability". In: *Frontiers in Psychology* 8 (2018). ISSN: 1664-1078. DOI: 10.3389/fpsyg.2017.02349 (cit. on pp. 2, 62).

[10] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG] (cit. on pp. 4, 5, 49, 50).

[11] Katherine Crowson et al. *VQGAN-CLIP: Open Domain Image Generation and Editing with Natural Language Guidance*. 2022. arXiv: 2204.08583 [cs.CV] (cit. on pp. 4, 51).

[12] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. 2022. arXiv: 2207.12598 [cs.LG] (cit. on pp. 5, 49, 70, 77).

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL] (cit. on p. 7).

[14] Peter Bloem. "Transformers From Scratch". In: (2019). URL: https://peterbloem.nl/blog/transformers/ (cit. on p. 8).

[15] Prajit Ramachandran et al. *Stand-Alone Self-Attention in Vision Models*. 2019. arXiv: 1906.05909 [cs.CV] (cit. on pp. 11, 12).

[16] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. *On the Relationship between Self-Attention and Convolutional Layers*. 2020. arXiv: 1911.03584 [cs.LG] (cit. on pp. 11, 12, 15).

[17] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV] (cit. on p. 12).

[18] Qiang Wang et al. "Learning Deep Transformer Models for Machine Translation". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019, pp. 1810–1822. DOI: 10.18653/v1/P19-1176. URL: https://aclanthology.org/P19-1176 (cit. on p. 13).

[19] Alexei Baevski and Michael Auli. *Adaptive Input Representations for Neural Language Modeling*. 2019. arXiv: 1809.10853 [cs.CL] (cit. on p. 13).

[20] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV] (cit. on pp. 14, 15, 51, 52, 79).

[21] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (cit. on p. 16).

[22] David Ha, Andrew Dai, and Quoc V. Le. *HyperNetworks*. 2016. arXiv: 1609.09106 [cs.LG] (cit. on p. 18).

[23] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL] (cit. on pp. 18, 47, 54).

[24] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. *Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning*. 2020. arXiv: 2012.13255 [cs.LG] (cit. on p. 20).

[25] Lilian Weng. "What are diffusion models?" In: *lilianweng.github.io* (2021). URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/ (cit. on p. 21).

[26] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV] (cit. on pp. 24, 27, 54).

[27] Patrick Esser, Robin Rombach, and Bjorn Ommer. *Taming Transformers for High-Resolution Image Synthesis*. 2021. arXiv: 2012.09841 [cs.CV] (cit. on pp. 26, 44).

[28] Christoph Schuhmann et al. *LAION-5B: An open large-scale dataset for training next generation image-text models*. 2022. arXiv: 2210.08402 [cs.CV] (cit. on pp. 28, 67, 70).

[29] Runaway ML Stability AI. *Stable Diffusion V2 release blog post*. https://stability.ai/blog/stable-diffusion-v2-release. (accessed 23-May-2023). 2022 (cit. on p. 29).

[30] Simo Ryu aka cloneofsimo. *lora*. https://github.com/cloneofsimo/lora. 2023 (cit. on pp. 31, 63, 68, 72, 73).

[31] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML] (cit. on pp. 33, 60).

[32] Thalles Santos Silva. "A Short Introduction to Generative Adversarial Networks". In: *https://sthalles.github.io* (2017). URL: https://sthalles.github.io/intro-to-gans/ (cit. on p. 33).

[33] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG] (cit. on p. 34).

[34] Lilian Weng. "From Autoencoder to Beta-VAE". In: *lilianweng.github.io* (2018). URL: https://lilianweng.github.io/posts/2018-08-12-vae/ (cit. on pp. 35–37, 42).

[35] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML] (cit. on p. 35).

[36] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694 (cit. on p. 36).

[37] Scott Reed et al. *Generative Adversarial Text to Image Synthesis*. 2016. arXiv: 1605.05396 [cs.NE] (cit. on pp. 38, 39).

[38] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE] (cit. on pp. 39, 41).

[39] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. *Neural Discrete Representation Learning*. 2018. arXiv: 1711.00937 [cs.LG] (cit. on pp. 41, 43).

[40] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. *Generating Diverse High-Fidelity Images with VQ-VAE-2*. 2019. arXiv: 1906.00446 [cs.LG] (cit. on pp. 44, 47).

[41] Aditya Ramesh et al. *Zero-Shot Text-to-Image Generation*. 2021. arXiv: 2102.12092 [cs.CV] (cit. on p. 47).

[42] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL] (cit. on p. 47).

[43] Alex Nichol et al. *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. 2022. arXiv: 2112.10741 [cs.CV] (cit. on pp. 49, 77).

[44] Alex Nichol and Prafulla Dhariwal. *Improved Denoising Diffusion Probabilistic Models*. 2021. arXiv: 2102.09672 [cs.LG] (cit. on p. 49).

[45] Michael McCloskey and Neal J. Cohen. "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem". In: ed. by Gordon H. Bower. Vol. 24. Psychology of Learning and Motivation. Academic Press, 1989, pp. 109–165. DOI: https://doi.org/10.1016/S0079-7421(08)60536-8. URL: https://www.sciencedirect.com/science/article/pii/S0079742108605368 (cit. on p. 53).

[46] Rinon Gal et al. *An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion*. 2022. arXiv: 2208.01618 [cs.CV] (cit. on pp. 53, 54).

[47] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: 1910.10683 [cs.LG] (cit. on p. 54).

[48] Lvmin Zhang and Maneesh Agrawala. *Adding Conditional Control to Text-to-Image Diffusion Models*. 2023. arXiv: 2302.05543 [cs.CV] (cit. on pp. 57–59).

[49] Zhe Cao et al. *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2017. arXiv: 1611.08050 [cs.CV] (cit. on p. 60).

[50] Eyal Molad et al. "Dreamix: Video diffusion models are general video editors". In: *arXiv preprint arXiv:2302.01329* (2023) (cit. on p. 60).

[51] Uriel Singer et al. "Make-A-Video: Text-to-Video Generation without Text-Video Data". In: *ArXiv* abs/2209.14792 (2022) (cit. on p. 60).

[52] *Storyboarder*. https://wonderunit.com/storyboarder/ (cit. on p. 60).

[53] *Storyboardthat*. https://www.storyboardthat.com/ (cit. on p. 60).

[54] *Studiobinder*. https://www.studiobinder.com/storyboard-creator/ (cit. on p. 60).

[55] Yitong Li et al. *StoryGAN: A Sequential Conditional GAN for Story Visualization*. 2019. arXiv: 1812.02784 [cs.CV] (cit. on p. 60).

[56] Anyi Rao et al. "Dynamic Storyboard Generation in an Engine-based Virtual Environment for Video Production". In: *ArXiv* abs/2301.12688 (2023) (cit. on p. 61).

[57] Lukas Budach et al. *The Effects of Data Quality on Machine Learning Performance*. 2022. arXiv: 2207.14529 [cs.DB] (cit. on p. 64).

[58] jwagner. *GitHub - jwagner/smartcrop.js: Content aware image cropping — github.com*. https://github.com/jwagner/smartcrop.js. [Accessed 14-Jun-2023] (cit. on p. 66).

[59] Shai Avidan and Ariel Shamir. "Seam Carving for Content-Aware Image Resizing". In: *SIGGRAPH* 26 (2007). DOI: 10.1145/1276377.1276390 (cit. on p. 66).

[60] Junnan Li et al. *BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models*. 2023. arXiv: 2301.12597 [cs.CV] (cit. on pp. 67, 73).

[61] In: *Numerical Methods for Ordinary Differential Equations*. John Wiley and Sons, Ltd, 2016. Chap. 3, pp. 143–331. ISBN: 9781119121534. DOI: https://doi.org/10.1002/9781119121534.ch3. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119121534.ch3. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119121534.ch3 (cit. on p. 71).

[62] Cheng Lu et al. *DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models*. 2023. arXiv: 2211.01095 [cs.LG] (cit. on p. 71).

[63] Tero Karras et al. *Elucidating the Design Space of Diffusion-Based Generative Models*. 2022. arXiv: 2206.00364 [cs.CV] (cit. on p. 71).

[64] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: 2010.02502 [cs.LG] (cit. on p. 71).

[65] *Insights from analyzing 226k civitai.com prompts*. https://rentry.org/toptokens (cit. on p. 73).

[66] Wenliang Zhao et al. *UniPC: A Unified Predictor-Corrector Framework for Fast Sampling of Diffusion Models*. 2023. arXiv: 2302.04867 [cs.LG] (cit. on p. 75).

[67] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV] (cit. on p. 79).

[68] D.H. Wolpert and W.G. Macready. "No free lunch theorems for optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893 (cit. on p. 79).

[69] Ernie Chu, Shuo-Yen Lin, and Jun-Cheng Chen. *Video ControlNet: Towards Temporally Consistent Synthetic-to-Real Video Translation Using Conditional Image Diffusion Models*. 2023. arXiv: 2305.19193 [cs.CV] (cit. on p. 88).

[70] Qian Yi et al. "Movie Scene Event Extraction with Graph Attention Network Based on Argument Correlation Information". In: *Sensors* 23.4 (2023). ISSN: 1424-8220. DOI: 10.3390/s23042285. URL: https://www.mdpi.com/1424-8220/23/4/2285 (cit. on p. 88).

[71] Junnan Li et al. *BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation*. 2022. arXiv: 2201.12086 [cs.CV].