Neural Network Design Patterns in Computer Vision
IN2107 Seminar Report, Fall 2023/24

Prasanga Dhungel
Student
prasanga.dhungel@tum.de

Ren Jeik Ong
Student
renjeik.ong@tum.de

Leonhard Siegfried Stengel
Student
leonhard.stengel@tum.de

Lukas Seitz
Student
lukas.seitz@tum.de

Minyu Liu
Student
minyu.liu@tum.de

Maanav Khungar
Student
m.khungar@tum.de

Zekun Jiao
Student
ge52vuj@mytum.de

Levente Tempfli
Student
lev.tempfli@tum.de

Roman Pflugfelder
Instructor
roman.pflugfelder@tum.de

Machine learning with neural networks has become the prevalent view of visual learning in computer vision. Each year, the scientific community proposes dozens of new models as solutions for various visual tasks. To better assess these models and compare them for a better understanding of their advantages and disadvantages, a deeper understanding of the building blocks that constitute a neural network is needed.
A similar problem is known in software engineering where progress in programming languages, the complexity of software and the need for reusability have triggered the appearance of *design patterns*. A software design pattern is a module or best practice to solve a particular programming problem in a very principal way. Examples are particular data structures or object-oriented best practices such as factories.
This seminar report attempts to distil in the same spirit design patterns of neural network models. We distinguish patterns arising at the level of network architecture and network layers. The architecture is a principled design of a neural network on a functional level, considering the possible inputs and outputs of a function and a functional classification. Generally, a neural network comprises several layers or sub-functions considering a specific sub-step or task in the overall network architecture.
The following neural design patterns result from eight students' seminar work. In the first step, the students proposed and discussed several neural models and identified neural design patterns. In the second step, each student chose one design pattern and analysed its intent, the history of the pattern, the limitations and its practical application. The results of these analyses are summarised in this report.

1. Skip Connection

In a simple neural network, the L^{th} layer transmits its activations solely to the $L+1^{th}$ layer during the forward pass and exclusively receives gradients from the $L+1^{th}$ layer during the backward pass. In contrast, a skip-connection allows the L^{th} layer to pass its activations to the $L+k^{th}$ layer, where k is a positive constant ensuring that $L+k$ does not surpass the network's depth. Consequently, during the backward pass, the L^{th} layer can directly obtain gradients from a layer situated deeper than the immediate succeeding layer, facilitating improved gradient flow throughout the network.

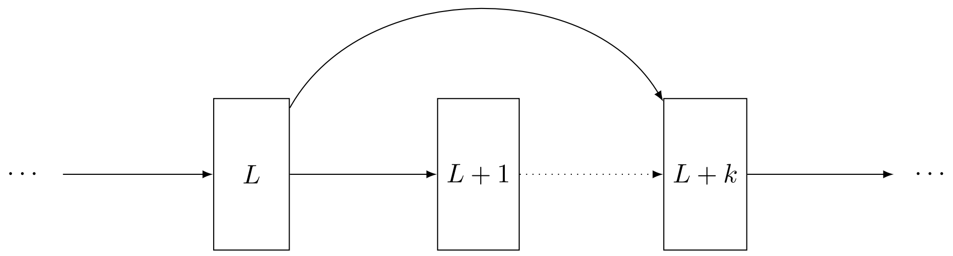


Figure 1. Block diagram of a Skip-connection. Each box represents a layer in the network. Each directed edge represents the propagation of activation. There is a skip connection from L^{th} layer to $L + k^{th}$ layer

1.1. Intend

Skip connection acts as a highway for gradients and solves the problem of vanishing gradients. Furthermore, skip-connection makes it easier to learn identity mapping over multiple non-linear layers [1]. He et al. experimentally demonstrated that the learned residual functions usually have small responses, indicating frequent occurrences of identity mappings.
In tasks such as object detection, and segmentation, there is a trade-off between localization accuracy and the use of context. Downsampling is necessary to capture the broader context, but it compromises localization accuracy. Consequently, the most prevalent architectures in these tasks adopt an encoder-decoder structure. The encoder reduces the original image resolution to leverage the context more effectively, while the decoder restores the downsampled features to the original resolution to enhance localization accuracy. In the encoder-decoder network, some information is inevitably lost due to the downsampling process in the encoder. Despite expectations that decoder parameter learning could recover this information, it does not typically occur in practice. Therefore, high-resolution information from the initial layers of the encoder is required, which is obtained via skip connections.
In addition, it has been shown that skip connections eliminate the singularities inherent in the loss landscapes of deep networks [2], making the loss landscape of networks with skip connections more convex than those without [3].

1.2. Block Diagram

1.3. Limitations

While skip connections, have proven to be highly effective in training deep neural networks and overcoming certain challenges, they are not a one-size-fits-all solution. Although skip connections help in mitigating the vanishing and exploding gradient problems [4], they don't completely eliminate the issues. Furthermore, in situations where the problem formulation and learning dynamics are ill-posed, skip connections will not offer any help.

1.4. History

Deep networks have been demonstrated to possess a higher representational capability than their shallower counterparts. In Convolutional Neural Networks, the early layers are responsible for learning basic features, while the more complex features are learned by the deeper layers [5]. This means that by adding more layers, the complexity and richness of the feature representation can be increased. Over time, this has led to an increase in the depth of ConvNets, which has subsequently improved their performance [6, 7, 8]. However, it has been observed that performance deteriorates beyond a certain depth. Adding more layers often results in poor gradients during backpropagation with Stochastic Gradient Descent, a phenomenon infamously known as the vanishing gradient problem [9]. The stacking of multiple non-linear transformations typically results in poor propagation of activations and gradients. While this issue can be somewhat alleviated with a proper choice of initialization, normalization [10], and activation function [11], it cannot be completely resolved.
Since the depth of representation is of crucial importance, several works have been proposed to train deep networks [12, 13, 14]. Inspired by LSTM [15], Srivastava et al. introduced a highway layer [16] that employs adaptive gating units to transmit

information through very deep layers without attenuation. This approach enabled them to train neural networks with over 1000 layers. Subsequently, He et al. proposed a deep residual learning framework, a non-parametric and data-independent variant of highway networks. Rather than learning the desired mapping $\mathcal{H}(x)$ directly, they explicitly learn the residual mapping $\mathcal{F}(x) = \mathcal{H}(x) - x$, hypothesizing that it is easier to optimize the residual mapping than the original one [1]. They realized $\mathcal{H}(x) = \mathcal{F}(x) + x$ with a skip connection. Skip connection enabled them to train very deep convolutional networks called ResNet, which was the state of the art on several computer vision challenges.

1.5. Application

Skip connections have found extensive applications across a multitude of tasks. Here are some notable examples:

- **Object Recognition**: ResNet [1] leverages skip connections to allow signals to be directly propagated from earlier layers to later layers. Furthermore, DenseNet [17] connects each layer to every other layer in a feed-forward fashion, thereby strengthening feature propagation.
- **Object Detection**: Feature Pyramid Network (FPN)[18] improves the process of object detection by integrating low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral connections.
- **Image Segmentation**: U-Net [19] employs an encoder-decoder structure for semantic segmentation, with skip-connections between mirrored layers in the encoder and decoder modules. Furthermore, Fully Convolutional Networks (FCN)[20] proposed the idea of using transposed convolutions for up-sampling and combined it with skip connections to fuse features from different layers.

1.6. Code Example

```
import torch
import torch.nn as nn

class SimpleResidualBlock(nn.Module):
    def __init__(self, channels):
        super(SimpleResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(channels, channels, kernel_size=3, stride=1, padding=1)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, stride=1, padding=1)

    def forward(self, x):
        out = self.relu(self.conv1(x))
        out = self.conv2(out)
        out += x
        out = self.relu(out)
        return out

simple_residual_block = SimpleResidualBlock(channels=64)
input_tensor = torch.randn((1, 64, 32, 32))
output_tensor = simple_residual_block(input_tensor)
```

2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have emerged as a fundamental design pattern in deep learning, revolutionizing various fields of computer vision and data analysis in general. CNNs are effective in capturing spatial hierarchies and patterns within locally correlated data, making them suitable to extract features from multidimensional input. The fundamental architecture consists of repeated application of convolutional filters, activation functions and in most cases some operation for dimensionality reduction or subsampling. The latter is usually performed by a pooling layer, but strided convolutions have also proven to be expedient [21]. Fully connected layers are often also counted as part of CNNs [22, 23], in this context we focus however on the pure design pattern of a CNN.

2.1 Intend

In many cases, data shows local correlation – images are a classical example for this. Neighboring pixels show often similar values, edges continue over several pixels and structures or textures result in locally repeating patterns. These can be extracted by the use of convolutional filters, which can detect arbitrary complex patterns if applied one after another and in multiple dimensions. Convolutions are however not limited to two-dimensional data like images. Locally correlated data can also be found in for example audio recordings, and CNNs are applied successfully to this type of input [24].
Common to all convolutions is the concept of weight sharing [25]. In contrast to fully connected layers, all input data is processed by the same convolutional filter or kernel, which is shifted over the input. This reduces the number of different weights per layer and hence storage consumption significantly. In consequence, the same type of feature is detected everywhere over the input – this property of CNNs is called translational equivariance [26]. The detection of features follows a hierarchical approach: in case of images, the first layer might only detect low-level features like a certain type of edge or corner everywhere on the image, for example. Following layers could then increase the complexity by combining them to basic patterns and finally to complex objects. Due to the merely sparse interaction of the kernel with the input data, this approach requires much fewer operations compared to fully connected layers.

2.2 Block Diagram

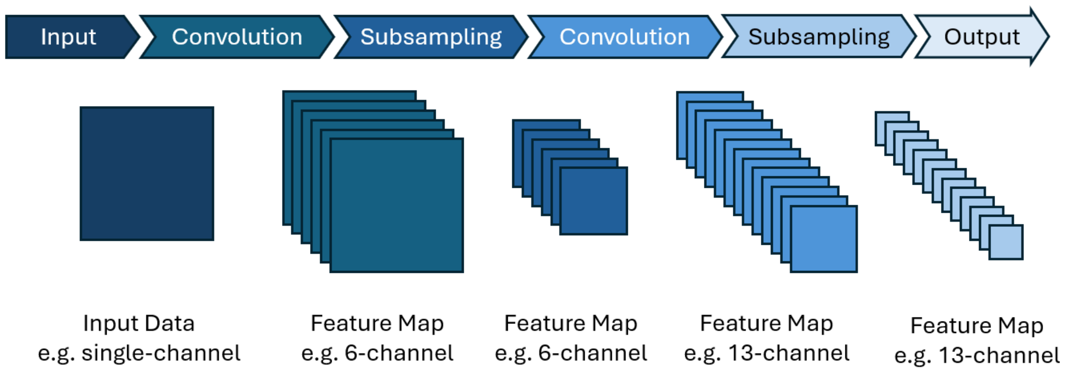


Figure 2. Block diagram of an examplary CNN consiting of two alternating layers of each convolution and subsampling (pooling). CNNs are often designed to increase the number of channels while spatial dimensions of the feature maps are reduced.

2.3 Limitations

A fundamental assumption when dealing with CNNs is that input data exhibits local correlations. In case of images, this is given by default, but sensor measurements in general (arranged in a table) are not required to show any local correlation. In this case, it might be favorable to use fully connected layers, which do not suffer from a limited field of view. Even in images, the receptive field of convolution layers might be a problem. The receptive field basically describes the size of the image region which influences one specific pixel in a following layer. It depends on various factors like pooling parameters, convolutional stride or the number of preceding layers. Depending on these factors, one neuron or pixel at the end of the CNN might not be aware of all parts of the input image. This becomes an issue if, for example, an object of interest appears in an unexpected region of the input image. If the network has not been trained to recognize the object at this position, it is likely to fail [26]. Most image recognition and classification networks compensate this limitation by a small number of following fully connected layers in order to connect also very distant parts of the input image. The property of computing the same output regardless of the position of an input feature is referred as translational invariance and is an active field of research [26, 27, 28]

2.4 History

The history of convolutional neural network reaches back for more than 40 years. The Neocognitron, pubished in 1980 [29], can be considered as one of the earliest precursors of CNNs, as it already exhibits several fundamental properties like the hierarchical structure, a local receptive field, weight sharing and pooling. Still it did not learn from backpropagation and lacks of the consistent structure found in modern CNNs. This is why LeNet-5 is mostly mentioned as the foundation of CNNs [25, 23]. In its final version published in 1998 [6], it was used for handwritten digit recognition to identify zip codes for the US postal service and later to read checks. With alternating convolution and pooling layers, the architecture of LeNet-5 resembles present CNNs, although a sigmoid activation function and average pooling was used, which is much less common in current networks. Following well-known network architectures include AlexNet, which was the first CNN-based network to win the ImageNet classification challenge in 2012 [7]. Compared to LeNet-5, it mainly increased the number of parameters and network layers and changed the activation function to ReLU and max-pooling instead of average-pooling. In the following, many refined architectures were published, like VGGNet [8] with a more homogeneous architecture ; GoogLeNet [30] with the newly introduced Inception module or ResNet [1], making use of skip connections.

2.5 Application

CNNs are widely used in many areas to extract meaningful features from data that shows local correlations, including:

- **Image classification:** Identifying and classifying of images has been one of the first applications of CNNs, starting with LeNet-5 [6], which was used for handwritten digit classification. Since the publication of AlexNet [7], CNNs are a major design pattern of image classification and have surpassed human-level performance in the ImageNet classification task [11].
- **Image Segmentation:** The process of dividing an image into different areas is often heavily based on the use of convolutions. Fully convolutional networks [20] and the U-Net [19] are highly cited representatives of this type of application.
- **Speech and natural language processing:** CNNs are well-suited to process one-dimensional signals [25], such that they have also been used for tasks in the field of audio processing. This includes automatic speech recognition, music information retrieval, environmental sound detection, localization, and tracking [31]. Significant progress has also been made in feature recognition of word vectors in natural language processing [32].

2.6 Code Examples

LeNet-5 [6] is often considered as the foundation of modern convolutional neural networks [25, 23]. The code snippet below is based on the convolutional block of LetNet-5, but uses max-pooling and ReLU as activation function, which is more common today. It consists of two layers, each including a convolution, activation function and max-pooling.

```
import torch.nn as nn

class ConvBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5, stride=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=5, stride=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        return out
```

3. Transformer

In the model of Recurrent Neural Network [33] (RNN) or LSTM, it leads to information loss or no control over forget gate in LSTM. Moreover, these models can't run in parallel because they read step by step in decoder, hence, making the model slow in training. As a result, Transformer model and self attention mechanism from Attention Is All You Need paper [34] are in favor to overcome parallelization and information loss problems.

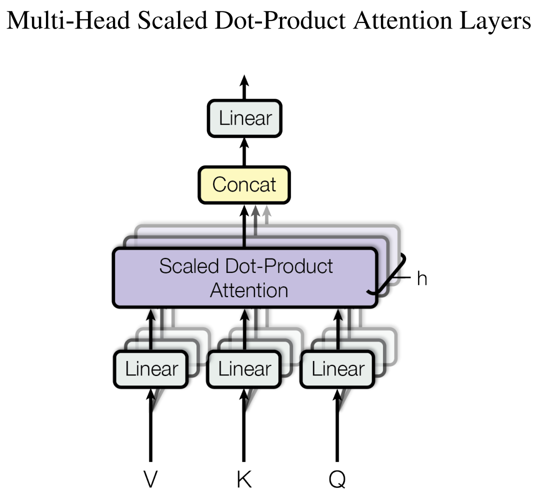
3.1. Intend

Self-attention is the black magic behind Transformer model. Self-attention pays attention to all input embeddings. In the context of LSTM, self-attention is attending to all hidden states that occurred previously. Thus, self-attention has long term context and the system figures out with its attention scores. Attention scores are used to determine the relevance among the embeddings within the sequence data in each training iteration.
In self-attention mechanism, there are 3 input tokens which are Query Q, Key K and Value V. The input tokens are computed independently using dot product matrix multiplication of $Q = XW^Q$, $K = XW^K$, $V = XW^V$ where $X, Q, K, V \in \mathbb{R}^{N\times d}$ and $W^Q, W^K, W^V \in \mathbb{R}^{d\times d}$ such that d is the features representation dimension and N is the sequence length. Thus, the formulation of self-attention is finalized as:

$$\text{SelfAttention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

where d_k is keys of features representation dimension. There are 2 types of Transformer architectures which are Encoder and Decoder. In Encoder, self-attention mechanism could attend the whole input sequence including past and future of the sequence in the view of their relevance among them. However, in Decoder, self-attention mechanism can only look into past input sequence because the training effect of the decoder is not strong if self-attention sees the future that is meant to produce. Therefore, we should mask the future input tokens when doing QK^T dot product matrix multiplication and produce stronger and reliable predictions in self-attention mechanism of Decoder.
All in all, there is no bottleneck compression and everything is done in parallel with dot product among Q, K, V, thus, solving the information loss and parallelization problem. To make Transformer model more interesting, Multi-Head Attention is introduced in the original paper [34] such that each head (which is self-attention mechanism) pays attention to different thing in h variants of heads. Nonetheless, the heads are stacked on top of each other in Multi-Attention layer and each head is trained separately. In the end, the multi-heads are concatenate together to form the final output. Finally, one could say Self-Attention mechanism solves autoregression problem such as text completion, text summarize and question answering tasks. Perhaps dot product is all you need!

3.2. Block Diagram


Multi-Head Scaled Dot-Product Attention Layers
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

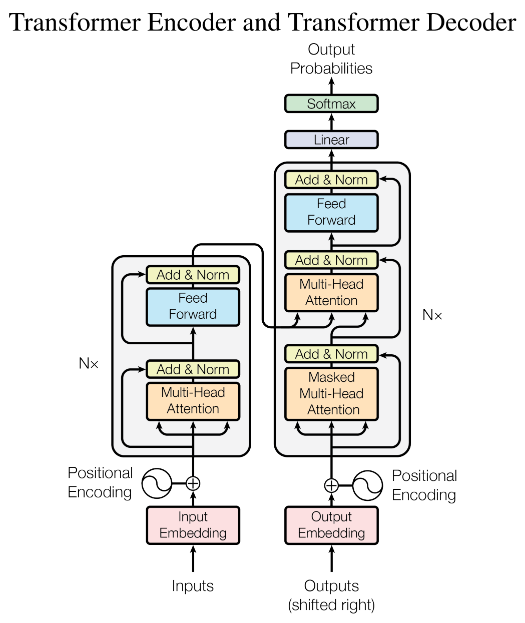


Figure 3. (left) Scaled Dot-Product Attention with h head Attention layers. (right) Transformer Encoder on the left, which passed the output of V and K, is having cross attention with Transformer Decoder on the right, which has Masked Multi-Head Attention beforehand. Diagrams are retrieved from [34]

3.3. Limitations

Even though self attention mechanism is current state of the art (SOTA) of early 2020s, constant supervision is required while training the models. Furthermore, it needs large dataset to gain promising accuracy that is comparable to Convolutional Neural Network [7] (CNN). In the context of the attention mechanism within Transformer models, Transformer models does not have inductive biases [7] of CNN models such that attention mechanism is fundamentally created for text sequential data whereas CNN is used for image processing. Therefore, Transformer model is not translation invariance, has no spatial encoding in 2D kernel structure and can't capture local pattern as self-attention extracts features globally. Last, attention mechanism might require longer training time due to slow convergence for some Transformer models [35]. Hence, many researchers do not train Transformer models from scratch, but use pretrained Transformer as transfer learning.

3.4. History

In the context of the evolution and development of Self-attention mechanism, Standard Hopfield Networks [36], which is introduced in the 1980s by John Hopfield, has update rule that is non-linear function and energy function that is aimed to minimize in training. Later, Standard Hopfield Network is evolved into Continuous Modern Hopfield Network [37] with new update rule as well as new energy function. As such, Softmax of Continuous Modern Hopfield Network is similar to Softmax of Self-attention mechanism with β as $\frac{1}{\sqrt{d_k}}$, ξ^T as Q and X as K^T.

$$\text{softmax}(\beta\xi^T X) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) \tag{2}$$

3.5. Application

Transformer architecture has been widely used as current state of art in early 2020s across Natural Language Processing and Computer Vision. Examples of transformer architecture in the field of Computer Vision are:

- **Image Classification**: Vision Transformer (ViT) [38] modifies the original Transformer paper [34] of sequence processing by splitting an image into fixed-size patches, thus, in principle, the image is being processed sequentially. Then, the patches are feed into Transformer Encoder for classification task. In addition, Swin transformer [39] is an enhanced version of ViT such that Swin transformer incorporate the inductive biases from CNN [7].
- **Object Detection**: Detection Transformer (DETR) [35] by Facebook AI predicts the bounding boxes of the classified objects in an image using Transformer Encoder, Transformer Decoder and CNN as the backbone of 2D input image representation.
- **Image Segmentation**: MaskFormer [40] is comparable to Panoptic FCN [41] model. MaskFormer could perform high accuracy across semantic and panoptic segmentation tasks. In MaskFormer paper, the main idea behind is using Transformer Decoder. First, the queries are initialised as potential masks. Next, Queries are crossed-attention with Keys and Values from backbone (e.g. CNN or Transformer Encoder) to generate kernels in Transformer Decoder. Moreover, Mask2Former [42] achieves higher SOTA accuracy across instance, semantic and panoptic segmentation tasks
- **Image Generation**: Stable Diffusion [43] utilizes modified version of cross-attention mechanism using Transformer Decoder in U-Net [19] model. At first, the input text prompt is passed into self attention mechanism to produce text embeddings in forward diffusion process. Later, in reverse diffusion process, the text embeddings and noisy image latent space are mapped together to predict noisy latent space in U-Net model. Eventually, denoising step is carried out to produce a new image in pixel-level representation.

4 Diffusion Denoising Probabilistic Model

4.1 Intend

DDPM aims to synthesize high-quality images while capturing the various modes of the underlying data distribution. In real world, the images exists on a complex subspace. The principle behind DDPMs is to incrementally add noise to data, thereby reducing its complexity and revealing its latent space, before reconstructing it from this noised latent state. [44] In forward diffusion, Gaussian noise is progessively added to the original data following a Markov Chain with controlled noise intensity at each step described by 3.

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_{x_{t-1}}, \beta_t I}) \tag{3}$$

Over time, the data´s original distribution transforms into a Gaussian noise distribution. Each step's intermediate state is recorded for use in the reverse process.
During the reverse diffusion, the model starts with data that follows a Gaussian distribution and attempts to reverse the noise addition. This process is described by the following equation:

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{4}$$

Ideally, in a generative model like DDPM, you'd want to make the recreated data as close as possible to the original data. This is known as maximizing the log-likelihood of the original data. However, this direct maximization is a highly complex task in the context of DDPMs, as it involves integrating over all possible paths of diffusion that the data could have been transformed $p_\theta(x_0) := \int p_\theta(x_0 : \tau) dx_{1:\tau}$, which would be a high-dimensional integration over the pixel space of the image over every time steps. To circumvent this problem, DDPM uses a variational approach. Rather than directly calculating the log-likelihood maximization, the authors employs a variational lower bound to approximate this integral. This approximation simplifies the objective, transforming a complex integration task into a more tractable optimization problem. This is simplified to a mean squared error loss function between the actual noise and the noise predicted by the model. [45]

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,x_0,\epsilon}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon, t\right)\right\|^2\right] \tag{5}$$

In each step, the noise of the step is predicted by the neural network ϵ_θ. The neural network architecture used here, typically a U-Net, is chosen for its feature to maintain the dimensionality of the data throughout the denoising steps. [44]

4.2 Block Diagram

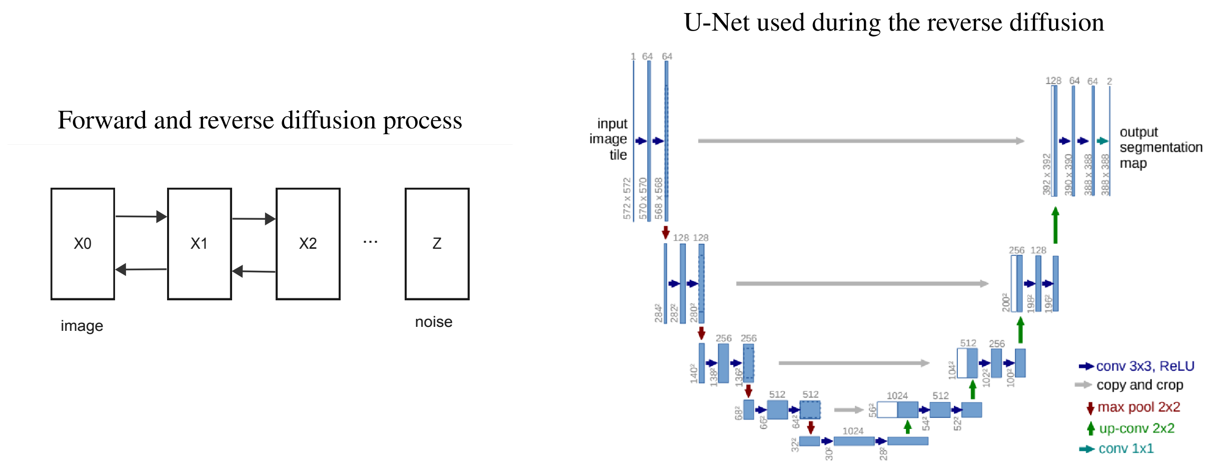


Figure 4. The block diagram illustrates the forward and reverse diffusion processes. It begins with the original image (X0) and shows the transition through various states of added noise (X1, X2, ...) until reaching a noisy state (Z). In the reverse phase, a U-Net [19] progressively denoises these states to reconstruct the original image.

4.3 Limitations

The first paper from Ho. et al [44] marked a significant advancement in producing high-quality images using diffusion models. However it encountered issues with a suboptimal log-likelihood when compared to other generative models, as well as high sampling times attributed to the long reverse diffusion step. In each step of this process, a forward pass through the neural network is required. It is important to note that a high log-likelihood in generative models is indicative of a good mode coverage and the capability of the model to learn and generate data with high complexity. Improvement were introduced in subsequent reasearch by Nichol and Dhariwal (2021) [46], which enhanced the log-likelihood of the model by optimizing the variational lower bound of the loss function to make the approximation more accurate and allowing the variance to be learnable. These modifications led a reduction of the number of denoising steps, thereby increasing the efficiency of the sampling process. Further developments by Dharival and Nichol(2021) [47] shows that diffusion models can outperform the state of the art GAN model, bigGAN-deep in terms of fidelity while maintaining a greater diversity through classifier guided diffusion. However, it´s important to mention that achieving higher fidelity with classifier guidance requires a labeled dataset. Furthermore, the sampling remains slower compared to GAN, as many denoising steps, each involving a forward passes of the neural network, are still necessary. As of now, the quality of the samples generated from a single reverse-diffusion step does not yet match the standard set by GAN models.

4.4 History

The concept of diffusion stems from non-equilibrium thermodynamics. There, diffusion describes a process that results in the movement of particles from a region of higher concentration to a region of lower concentration, eventually leading to an equilibrium state. During this process, the randomness, also called the entropy, is increased over time. An example of this phenomenon would be the dispersion of milk when poured into coffee, gradually spreading until homogenically mixed. The equivalence of this process in DDPM would be incrementically introducing noise to structured data until the data is completely corrupted. In Thermodynamics, reversing the diffusion process to return to a state of lower entropy lower entropy is generally not possible due to physical laws. RDDPM uses machine learning to perform what might be considered a computational reversal of diffusion for digital data. [48]
Building on this concept, Sohl-Dickstein et al. (2015) [45] formalizes the concept of forward and reverse diffusion steps using markov chains for generative models. Their contribution was crucial in advancing the understanding and application of generative models, especially for complex, real-world datasets. This principle is later adapted and expanded for image synthesis in the work by Ho et al. [44].

4.5 Application

Diffusion models have various application areas, such as image generation, image inpainting and image denoising.

- **Image denoising** [49]: In this paper, the authors trains a modified DDPM. The training here happens during the forward process. Here in the training Phase, noisy images are created by interpolating between clean images and real-world noisy images. A U-Net model is then trained to predict a clean image from these noisey variants. In the sampling phase, the trained U-Net denoises real-world noisy images back to their clean state.

- **Image inpainting**: In the application of DDPMs for image inpainting, noise is selectively applied to specific parts of an image during the forward process. he model then focuses on these noised areas in the reverse process, aiming to restore them. Some other approaches use a plain DDPM trained for image generation and do not train the model specifically for the inpainting task and achieved promising results. These approaches showcase the adaptability of DDPMs in handling selective image restoration challenges. [50]

- **Text to image**: Stable Diffusion, a famous text-to-image application, employs the Latent Diffusion Model (LDM) as outlined in [43]. This process begins with compressing the dataset into a lower-dimensional latent space by through an auto-encoder, whereafter the diffusion occurs. This makes the model more efficient. In the reverse diffusion phase, text inputs from a pre-trained text encoder conditions the noise at each step. This conditioning ensures that the generated images align with the semantic content of the text.

5. Encoder-Decoder (Autoencoder)

5.1. Intend

The central intention of encoder-decoder structures and autoencoders is to enable efficient feature extraction. Feature extraction is about identifying and extracting significant patterns, structures, or information from raw data. Helping machines understand and classify data efficiently, feature extraction is a fundamental aspect of machine learning and artificial intelligence. In computer vision, encoder-decoder structures can be used for a variety of applications.
An Encoder-Decoder is a common fundamental structure utilized in deep learning models. The architecture consists of two main parts: the encoder and the decoder. The encoder transforms the high-dimensional input data into a low-dimensional code. Typically, the encoder reduces the dimensionality, hence simplifying the complexity of the input data. The decoder, on the other hand, reconstructs the data from the low-dimensional code, effectively attempting to replicate the original input.
During training, the encoder-decoder model learns to compress the input into a low-dimensional code, and then recover the original input from the code. This is achieved by setting the target outputs to be the same as the inputs. The model's objective during training is to minimize the reconstruction loss between the input and the output produced by the decoder.
To avoid simply copying from input to output during training, i.e. overfitting, regularization is required. This can be done by adding a penalty term to the loss function. An undercomplete autoencoder is a type of encoder-decoder structure that leverages the concept of regularization. The undercomplete autoencoder is designed such that the code's dimensionality is less than the input data. This forces the autoencoder to construct useful features during the encoding process, thus learning a compressed, but informative representation of the input data.
The space in which the code lies is referred to as the latent space. Typically, the dimensionality of this space is significantly lower than that of the input space. The points in the latent space have the property that similar points decode to similar outputs. When training an autoencoder one is usually not interested in the output of the decoder but in the code taking on usefull properties [51].

5.2. Block Diagram

5.3. Limitations

One significant constraint is the requirement for the training set to be similar to the data to which they will be applied in real scenarios. If there's a substantial discrepancy, the model's predictions can be inaccurate, leading to suboptimal results.
Moreover, there's a persistent issue relating to the trade-off between data compression and feature extraction. While autoencoders compress information effectively, they sometimes fail to retain critical features from the original data during

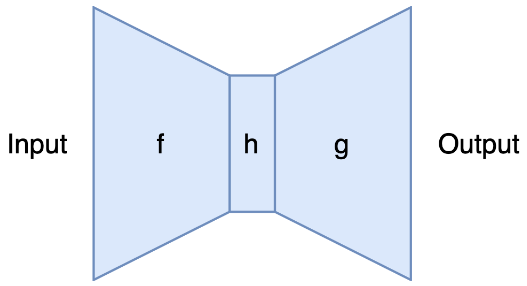


Figure 5. Block diagram of an encoder-decoder structure with encoder f, code h and decoder g.

the encoding process. This could potentially hamper the model's extrapolation efficiency, thus undermining its overall performance. Therefore, balancing between the amount of data compression and valuable feature extraction remains a key challenge.

5.4. History

The concept of autoencoders has been rooted in the history of neural networks for many years, dating as far back as the late 20th century. The fundamental idea behind an autoencoder is to extract, learn, and leverage useful, high-dimensional data properties related to the task at hand. Historically, the principal applications of autoencoders fell under the categories of dimensionality reduction and feature learning [51].
One of the early researchers to introduce something that comes very close to today's idea of autoencoders was Mark A. Kramer in 1991. He introduced an encoder-decoder structure as a method for Nonlinear Principal Component Analysis [52]. Even earlier in 1987 Dana H. Ballard introduced a method for unsupervised pretraining using an encoder-decoder structure [53].
Further research not only led to a change in terminology of autoencoders but also to more and more potential applications. Today, they also play a leading role in generative models with variational autoencoders [51].

5.5. Application

With their feature extraction capability, encoder-decoder structures and autoencoders can be used for various tasks in computer vision:

- **Semi supervised Classification**: To develop a classification model, without having enough labled data available, semi supervised learning with an autoencoder can be applied. For this, an autoencoder is first trained on an large unlabeled data set. The resulting encoder is then extended with additional layers that are trained for classification using labeled data [54].
- **Image Denoising**: Denoising Autoencoders are a type of neural network used to remove noise from images (or other kinds of signals). They are trained using noisy images as input and corresponding clean images as the target output. The network compresses the noisy input into a smaller representation during the encoding stage, then reconstructs it back to the original image size during decoding. The reconstruction loss is then calculated with the output and the original clean image, aiming to recreate the clean image. This technique is widely used in areas like medical imaging and photo enhancements where high-quality images are crucial [55].
- **Image Compression**: Autoencoders can be used in Image Compression. The Encoder is used to compress the original high-dimensional image into a lower-dimensional form, and the decoder reconstructs the original image from the compressed data. Although this is a lossy process and some image details might be lost, it allows significant reduction in storage space and faster transmission times, making autoencoders a valuable tool in efficient image management and transmission [56].
- **Anomaly Detection**: Autoencoders can be used for anomaly detection by training them on normal data and then using them to reconstruct new data. If the reconstructed output significantly differs from the input, this suggests the forwarded data is an anomaly. This approach is effective in identifying unusual patterns or outliers in datasets [54].

- **Image Segmentation**: The U-Net is an encoder-decoder type of Convolutional Neural Network (CNN) designed for semantic image segmentation. The encoder captures context via a standard CNN, which systematically reduces the spatial dimensions while increasing the depth. The decoder then uses transposed convolutions that recover spatial dimensions to reconstruct the output, taking advantage of skip connections from the encoder to retain high-resolution features (also see in chapter Skip-Connections). This design allows detailed image segmentation [19].

5.6. Code Example

The following code snippet shows a implementaion of a simple encoder-decoder structure. The encoder and decoder were implemented with fully connected layers. However, implementation with other architectures such as CNN is also possible.

```
import torch.nn as nn

class EncoderDecoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            torch.nn.Linear(28 * 28, 64),
            torch.nn.ReLU(),
            torch.nn.Linear(64, 32),
        )
        self.decoder = nn.Sequential(
            torch.nn.Linear(32, 64),
            nn.ReLU(),
            torch.nn.Linear(64, 28 * 28),
            nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return out
```

6 Pyramidal Networks

Object detection is an essential task in many fields, including autonomous cars, surveillance, medical imaging, etc. Traditional object detection methods typically use either one-stage detectors (e.g. YOLO[57]) or two-stage detectors (e.g. R-CNN family[58, 59, 60]). These methods often face challenges in detecting objects of different sizes within the same image, as they lack the architecture to effectively detect multi-scale objects. The aim of feature pyramid networks [18] is to help overcome this challenge. Unlike regular sequential convolutional neural networks, they build two pyramids of convolutional layers to capture high-level semantic information with higher resolution at each scale. This approach allows pyramidal neural networks to capture information at multiple scales, significantly improving the accuracy of object recognition tasks.

6.1 Intend

A feature pyramid network [18] constructs a pyramid of features with multiple levels with the idea of each level detecting objects at different scales. The core idea is combining low-resolution, semantically strong features with high-resolution, semantically weak features. This is achieved by utilizing a two-way architecture, one is bottom-up, and the other is top-down. See the Block Diagram
The bottom-up pathway consists of a regular feed-forward convolutional neural network, in practice usually a pre-trained ResNet [1] or similar). This computes a feature hierarchy consisting of feature maps at several scales. Each feature map is smaller than the last one by a factor of 2. The earlier, lower-level features should capture the features at high resolution, while the later higher-level features should capture high-level semantic information, but at a lower resolution.

The top-down pathway consists of the same number of levels as the bottom-up pathway, with the same feature map sizes, but the flow of information (or gradients) is in the opposite direction. It starts from the top level of the pyramid and ends with the bottom level, each level is the upsampled feature map from the previous level element-wise added to the lateral connection. The lateral connection is the feature map on the corresponding level of the bottom-up pathway transformed by a 1x1 convolution.
The strength of this architecture is that it combines the lower-level features coming through the lateral connections with the higher-level features coming through the upsamplings. This combination of higher resolution with better semantic information makes all the layers at various scales semantically strong, thus detecting objects at various scales becomes more effective.
To make the feature pyramid network a real object detector, it's combined with the Faster R-CNN [60] architecture. A ResNet [1] is used as a backbone, then a feature pyramid network is built on this ResNet with bottom-up and top-down paths. The resulting feature maps of the top-down path are used to generate the region proposals using a region proposal network (RPN). Then, the regions are fed into the classification and bounding box regressor networks to determine the objects on the image and their location. In the case of FPNs, these heads share parameters. Then, non-maximum suppression is used to eliminate duplicates. Because the feature maps fed into the classifier heads contain information at different scales, the region proposals are better able to handle objects of different sizes.

6.2 Block Diagram

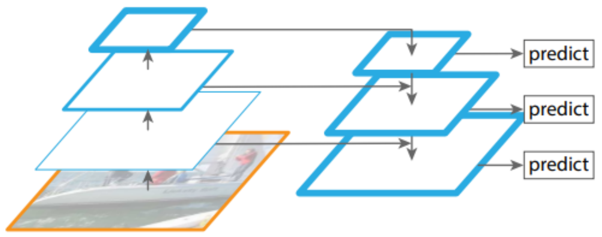


Figure 6. Block diagram of a feature pyramid network[18]. On the left with the bottom-up path and on the right with the top-down path

6.3 Limitations

Feature pyramid networks are not a one-solution-fits-all, they come with certain limitations. One of the limitations is the computational overhead caused by the added layers (upsampling and 1x1 convolutions) on the top-down path. This could be a problem during training since they require more memory and processing power to train the classifier of every level of the feature pyramid. The pyramid networks add not just computational complexity, but also implementation complexity, like it's harder to implement and debug. Furthermore, if one wants to combine FPNs with some other CNN model (eg. other than R-CNN), it would not be necessarily straightforward.

6.4 History

The idea of using image pyramids is not new, in computer graphics it's widely used to represent images at different scales and resolutions by for example applying a Gaussian Filter and for the further objects the lower resolution textures, for the nearer objects, the higher resolution images are used.
Already in 1984, a paper [61] theorized the possibility of using image pyramids in the context of computer vision. They wrote pyramid methods can be used to analyze textures and patterns at multiple image scales making it useful for tasks like object detection and recognition. They also mentioned other possible uses like image enhancement by reducing noise while sharpening details or image compression.
Later classical computer vision algorithms like HOG[62] or SIFT[63] also exploited the idea of image pyramids. In the case of HOG (Histogram of Oriented Gradients), they use sliding windows to scan through different image scales using an image

pyramid, calculate the HOG descriptor for each, and then run an SVM on it to classify the window. In the case of SIFT, they calculate the Laplacian of Gaussian on different octaves (scales), then based on these they find the local extremes, thus promising features. Image pyramids make the SIFT a scale-invariant feature detector.
One of the first few approaches to exploit the nature of image pyramids is the Single Shot Detector[64]. It is a one-stage object detector, very similar to conventional CNNS, it even uses a VGG16[8] as the backbone network. The difference lies in the detection head since it doesn't just get the information from the one last layer, but it's fed with the information from the last few layers, which have gradually reduced sizes.

6.5 Application

Feature pyramid networks are an effective architecture for object detection. Apart from the basic architecture, there are several different variations:

- Mask R-CNN [65]: for this model, the task is no longer just detecting and classifying, but also segmenting. It uses a very similar architecture to Faster R-CNN and FPNs, but the difference is that they add a new head for generating the binary mask for the objects.
- RetinaNet [66]: once-stage models are generally faster, but less accurate than two-stage detectors because of the oreground-background class imbalance. RetinaNet addresses this issue by introducing the focal loss function to improve performance. It is still a one-stage detector that uses an FPN to generate rich, multi-scale feature pyramids from a single-scale input.
- PANet [67]: improves the Mask R-CNN approach with a parallel bottom-up path augmentation to provide more effective feature representations. It also uses 'Adaptive Feature Pooling', which aggregates the features from all feature levels and these are used later in the subsequent detection stages. This makes the model even less sensitive to scale.
- NAS-FPN [68]: in this paper, they experimented with an architecture search related to FPNs.
- EfficientDet [69]: They propose a bidirectional feature pyramid network (BiFPN). Unlike standard FPNs which only process the features in a top-down manner, BiFPN proposes alternating layers of top-down and bottom-up feature fusion. This allows more efficient information flow across the scales.

7 Generative Adversarial Network

The Generative Adversarial Network (GAN) is a revolutionary generative model algorithm or framework introduced by Ian Goodfellow and his colleagues in 2014[70]. It employs a unique training algorithm in which the generator and discriminator undergo synchronized adversarial training, gradually making the generated data distribution more akin to the real distribution. Upon its introduction, GAN has sparked numerous transformative changes in the field of machine learning, achieving significant progress in various research domains that require the generation of data distributions. Particularly in computer vision, GAN has emerged as a widely employed design pattern, actively contributing to various tasks[71][72].

7.1 Intend

GAN stands out among other contemporaneous generative models primarily due to the uniqueness of its algorithm. Before the advent of GAN, generative models primarily relied on constructing an explicit density $p_{model}(x;\theta)$, employing the maximum likelihood principle to train the model by maximizing the log-likelihood. Constructing explicit densities in computation can be challenging, leading to the use of variational methods or Markov chains for approximation. The most notable models in this class are deep Boltzmann machine[73] and the Variational Autoencoders (VAEs)[74]. GAN, on the other hand, is an implicit algorithm that doesn't require the construction of explicit density functions. This characteristic allows GAN to bypass the computational challenges associated with variational approximations or Markov chain approximations that may arise in this process.
GAN employs an adversarial training approach to achieve a Nash equilibrium, obtaining a model that maximizes likelihood without computationally expensive approximation methods. In this training process, the adversarial counterparts are the generator G and discriminator D, which are the fundamental components of the GAN algorithm.

To learn the generator's distribution p_g over real data x, a prior input noise variable $p_z(z)$ is first defined for the generator. Thus, the mapping function of the generator G, parameterized by θ_g, can be expressed as $G(z, \theta_g)$. The expression for the discriminator is defined as $D(x, \theta_d)$ that outputs a single scalar, which measures the probability that the input comes from real data. GAN trains the discriminator D to maximize the probability of assigning correct labels to both training samples and generated samples provided by G. Simultaneously, G is trained to minimize the negative log likelihood, thereby maximizing the likelihood of the generated samples matching the distribution of real data. In other words, the training process of GAN can be formalized as a minimax optimization problem, and its loss function can be expressed by the following equation:

$$\underset{G}{min}\ \underset{D}{max}\ V(D,G) = E_{x \sim p_r(x)}[logD(x)]\ +\ E_{z \sim p_g(z)}[log(1 - D(G(z)))] \tag{6}$$

During the training process, the discriminator aims to make the output $D(G(z))$ close to 0, while the generator's objective is to make the output D close to 1. When both models undergo sufficient training, the game eventually reaches a Nash equilibrium. Ideally, when the minimum $p_r(x)/(p_r(x) + p_g(z))$ equals to 1/2, the model is optimal. It means that D cannot distinguish real samples and generated samples.
Due to the fact that GAN provides an algorithmic framework, the structure of the generator can be widely adapted to other network architectures. This imparts GAN with a powerful modeling capability, and practical evidence has shown that GAN often produces generated results that are closer to real data[72].

7.2 Block Diagram

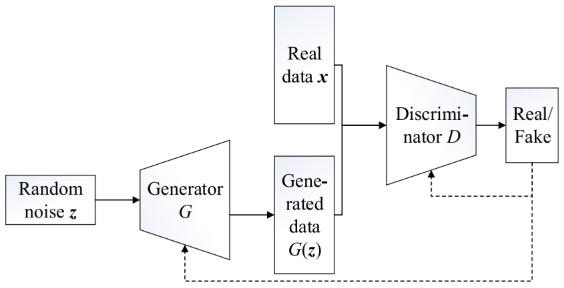


Figure 7. Block diagram of a basic GAN[71].

7.3 Limitations

One of the biggest challenges of GAN lies in the difficulty of its training. When the optimal discriminator is used, the loss function can be transcribed as $V(D, \theta^{(G)}) = 2JSD(p_r \parallel p_g) - 2log2$[71]. It follows that in the early stages of training, the overlap between the generated samples and the real data is low, resulting in a loss function of zero and thus the gradient vanishes. The adversarial training process also makes it difficult to ensure that both sides converge towards the saddle point, resulting in non-convergence problems such as pattern collapse, typically when an overly strong discriminator causes the generator to loop through a series of incomplete distributions in order to avoid penalties[72]. On the other hand, as a novel generative model, GAN currently lacks established metrics to comprehensively evaluate different models based on performance, accuracy, overfitting, and other aspects[75].

7.4 History

The historical development of GAN can be summarised into two important stages, one from the proposal of GAN to the proposal of DCGAN. Conditional GAN (CGAN) [76]appeared soon after the proposal of GAN as an optimisation scheme for convergence rate. DCGAN proposed in 2015 made various optimisations on the structure of GAN[77], the use of Batch Normalization algorithm, Strided convolutions and fractional-strided convolutions instead of pooling layers and other methods became the standard reference for the structure of subsequent GAN networks.

The second stage is to the proposal of Wasserstein generative adversarial network (WGAN) in 2017[78]. WGAN introduces the wasserstein distance to measure the difference in distribution between the generated data and the real samples, which solves the problem of gradient vanishing of JS divergence with low overlap. However, using Wasserteion distance needs to satisfy strong conditional lipschitz continuity, and cutting the weights for this purpose may lead to vanishing or exploding gradients. To solve this problem, WGAN with gradient penalty (WGAN-GP) is proposed by Gulrajani *et al*[79]. WGAN-GP constrains the gradient of the discriminator to prevent it from over-performing, thus reducing the requirement for generator-discriminator balance in training.

7.5 Application

GANs have found extensive applications in the field of computer vision, particularly excelling in the following areas related to generative tasks:

- Generation of high quality samples : GANs can learn the distribution of real data through supervised learning, semi-supervised learning or unsupervised learning[77][80]. It does not require the construction of function representations and performs model training through an end-to-end approach. GAN has been proven to produce higher quality images compared to other traditional generative models[72]. The problem of generating images with low resolution can also be solved by constructing a tandem network such as LAPGAN[81].
- Image restoration : GANs can be used to reconstruct images by generating similar missing images and finding the closest vectors, such as using semantic image generation to achieve pixel-level fidelity in image restoration, although this method is not accurate in the presence of large-scale image loss[82].GANs have also been used for super-resolution to restore image clarity, and have been found to perform better than ResNet structures[83].
- Image translation and style transfer : While traditional image translation tasks often require the application of different network frameworks for different task types, GAN can provide a unified framework for various tasks through adversarial training. For example Isola *et al.* [84] use GAN to cover a wide range of image transformation application examples.

8 Siamese Networks

Siamese Networks operate on the concept of Similarity Learning. It is a supervised machine learning technique in which the model learns a similarity function that measures how similar two objects are and returns a similarity score.

8.1 Intend

A Siamese neural network consists of twin networks which accept distinct inputs but are joined by an energy function at the top. The function calculates how similar or dissimilar the high level feature representations resulting on each side of the network. Weights of the sub-networks are tied, ensuring their symmetry. This guarantees that highly similar images cannot be mapped to vastly different locations in the feature space. There are two major loss functions used with Siamese networks, the Contrastive loss and the Triplet loss. The Contrastive loss aims to pull similar inputs closer in the embedding space while pushing dissimilar inputs apart. It is defined as:

$$(1-Y)\frac{1}{2}(D_W)^2 + (Y)\frac{1}{2}\{\max(0, m - D_W)\}^2 \tag{7}$$

where:

- $\mathcal{L}$ is the triplet loss function.
- Y is 0 for similar pairs, 1 for dissimilar ones,
- D_W is the Distance Measure, generally the Euclidean Distance
- m is the margin, a hyperparameter..

It works by minimizes the distance between pairs of training samples belonging to same class and maximizes the distance between pairs of training samples from different classes as long as this distance is smaller than a margin. This loss however is computationally challenging for most modern datasets as the run-time training complexity is $O(N^2)$[85]. Triplet loss, an extension of Contrastive loss, involves using triplets of anchor, positive, and negative examples to learn embeddings, where the distance between the anchor and positive examples is minimized, while the distance between the anchor and negative examples is maximized while maintaining a margin on separation. It is often defined as:

$$\mathcal{L}(A, P, N) = \max(0, d(A, P) - d(A, N) + m) \tag{8}$$

where:

- $\mathcal{L}$ is the triplet loss function.
- A is the anchor sample.
- P is the positive sample.
- N is the negative sample.
- $d(A, P)$ is the distance between anchor and positive samples.
- $d(A, N)$ is the distance between anchor and negative samples.
- m is the margin, a hyperparameter that determines the minimum difference between the distances for a loss to be incurred.

From a Design Pattern perspective, These networks evaluate the outputs of other network models like CNNs or Transformers. This illustrates the flexibility of Siamese networks to integrate with any architecture capable of compressing provided data into a latent space. This approach allows us to capture semantic similarity through the learning process.

8.2 Block Diagram

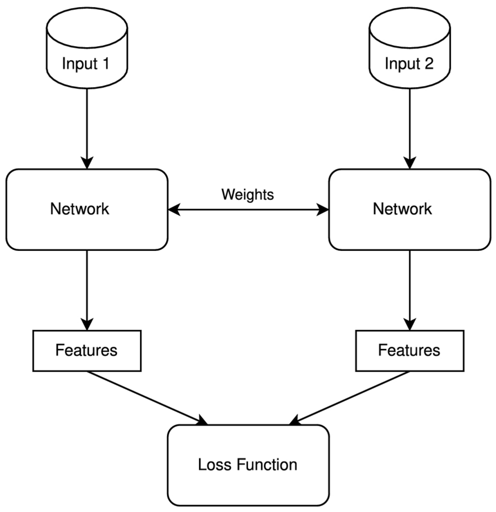


Figure 8. Block diagram of a basic Siamese Network[86].

8.3 Limitations

While Siamese Networks perform well in similarity learning and one shot learning, they do have some limitations. Understanding these limitations is crucial for determining when and how to use Siamese networks appropriately. Here are some common limitations:

- **Higher Training Time**: Siamese networks can sometimes have higher training times compared to other neural network architectures. These networks learn on triplets or quadratic pairs, leading to an augmented computational load and weight calculation process. This leads to higher training times
- **Output Predictions**: Unlike other neural network architectures, Siamese Networks output the distance from each class. This may pose difficulties in specific applications where outputs based on probabilities are more desirable.

8.4 History

Siamese networks were first introduced in the early 1990s by Bromley et al.[87] to solve signature verification as an image matching problem. The original motivation for Siamese networks was face verification and recognition, where the goal was to determine whether two face images belong to the same person or not. In 1997, The idea gained popularity in the context of face verification, where the task was to determine whether two face images belong to the same person. The concept of using a contrastive loss function, a key component of Siamese networks, was introduced by Samy Bengio, et al. in 2005[88].In 2015, Triplet loss, an extension of contrastive loss, became popular in Siamese networks.
In recent times, the architecture has found applications in various domains beyond face recognition, including signature verification, fingerprint matching, and more recently, in deep learning applications for natural language processing. Siamese networks were adapted to tasks such as text similarity and semantic matching. Siamese networks has also found applications in tasks like Named Entity Recognition (NER) and coreference resolution.

8.5 Application

The underlying idea of learning embeddings for similarity or distance measurement remains a valuable concept in various machine learning and deep learning applications and Siamese Networks are widely used in many areas including

- **Bio-metric Verification**: The architecture of Siamese networks makes it suitable for tasks that involve comparing and verifying biometric data such as Facial images or Fingerprints. Although Siamese networks originated in the domain of signature verification, they have also been applied successfully in the field of fingerprint verification[89]. Amouei et al.[90] is a perfect example of how Siamese Networks can be successfully applied to person re-identification.
- **Object Tracking**: Siamese Networks provide an effective approach for tracking objects in video sequences. During tracking, the network is used to locate the target object in subsequent frames by finding the region with the highest similarity score to the initial target region. The fully convolutional Siamese network (SiamFC)[91] is a pioneer example of how Siamese Networks can be used in object tracking effectively
- **One-Shot Learning**: This refers to the ability of a model to learn from only a single or a few examples of each class during training. Siamese networks are well-suited for one-shot learning tasks because they are designed to learn similarity or dissimilarity between pairs of inputs.

8.6 Code Example

```
import torch.nn as nn
class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 96, kernel_size=11,stride=4),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, stride=2),

            nn.Conv2d(96, 256, kernel_size=5, stride=1),
            nn.ReLU(inplace=True),
        )

        self.fc = nn.Sequential(
            nn.Linear(256, 512),
            nn.ReLU(inplace=True),

            nn.Linear(512, 128),
        )

    def network_forward(self, x):
        output = self.cnn(x)
        output = output.view(output.size()[0], -1)
        output = self.fc(output)
        return output

    def forward(self, input1, input2):
        output1 = self.network_forward(input1)
        output2 = self.network_forward(input2)

        return output1, output2
```

9. Conclusions

References

[1] K. He *et al.*, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] A. E. Orhan and X. Pitkow, “Skip connections eliminate singularities,” *arXiv preprint arXiv:1701.09175*, 2017.

[3] H. Li *et al.*, “Visualizing the loss landscape of neural nets,” *Advances in neural information processing systems*, vol. 31, 2018.

[4] G. Philipp *et al.*, “Gradients explode-deep networks are shallow-resnet explained,” 2018.

[5] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, Springer, 2014, pp. 818–833.

[6] Y. LeCun *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] A. Krizhevsky *et al.*, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.

[8] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[9] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[10] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pmlr, 2015, pp. 448–456.

[11] K. He *et al.*, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[12] C.-Y. Lee *et al.*, "Deeply-supervised nets," in *Artificial intelligence and statistics*, Pmlr, 2015, pp. 562–570.

[13] D. Sussillo and L. Abbott, "Random walk initialization for training very deep feedforward networks," *arXiv preprint arXiv:1412.6558*, 2014.

[14] A. Romero *et al.*, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] R. K. Srivastava *et al.*, "Training very deep networks," *Advances in neural information processing systems*, vol. 28, 2015.

[17] G. Huang *et al.*, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[18] T.-Y. Lin *et al.*, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[19] O. Ronneberger *et al.*, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, Springer, 2015, pp. 234–241.

[20] J. Long *et al.*, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[21] J. T. Springenberg *et al.*, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[22] S. Albawi *et al.*, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[23] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[24] S. Hershey *et al.*, "Cnn architectures for large-scale audio classification," in *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, IEEE, 2017, pp. 131–135.

[25] Z. Li *et al.*, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[26] V. Biscione and J. S. Bowers, "Convolutional neural networks are not invariant to translation, but they can learn to be," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 10 407–10 434, 2021.

[27] J. C. Myburgh *et al.*, "Tracking translation invariance in cnns," in *Southern African Conference for Artificial Intelligence Research*, Springer, 2020, pp. 282–295.

[28] C. Mouton *et al.*, "Stride and translation invariance in cnns," in *Southern African Conference for Artificial Intelligence Research*, Springer, 2020, pp. 267–281.

[29] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[30] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[31] H. Purwins *et al.*, "Deep learning for audio signal processing," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.

[32] S. Yang, "Natural language processing based on convolutional neural network and semi supervised algorithm in deep learning," in *2022 International Conference on Artificial Intelligence in Everything (AIE)*, IEEE, 2022, pp. 174–178.

[33] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *CoRR*, vol. abs/1912.05911, 2019. arXiv: 1912.05911. [Online]. Available: http://arxiv.org/abs/1912.05911.

[34] A. Vaswani *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[35] N. Carion *et al.*, “End-to-end object detection with transformers,” *CoRR*, vol. abs/2005.12872, 2020. arXiv: 2005.12872. [Online]. Available: https://arxiv.org/abs/2005.12872.

[36] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities.,” *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. DOI: 10.1073/pnas.79.8.2554. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.79.8.2554. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554.

[37] H. Ramsauer *et al.*, “Hopfield networks is all you need,” *CoRR*, vol. abs/2008.02217, 2020. arXiv: 2008.02217. [Online]. Available: https://arxiv.org/abs/2008.02217.

[38] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *CoRR*, vol. abs/2010.11929, 2020. arXiv: 2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929.

[39] Z. Liu *et al.*, “Swin transformer: Hierarchical vision transformer using shifted windows,” *CoRR*, vol. abs/2103.14030, 2021. arXiv: 2103.14030. [Online]. Available: https://arxiv.org/abs/2103.14030.

[40] B. Cheng *et al.*, “Per-pixel classification is not all you need for semantic segmentation,” *CoRR*, vol. abs/2107.06278, 2021. arXiv: 2107.06278. [Online]. Available: https://arxiv.org/abs/2107.06278.

[41] Y. Li *et al.*, “Fully convolutional networks for panoptic segmentation,” *CoRR*, vol. abs/2012.00720, 2020. arXiv: 2012.00720. [Online]. Available: https://arxiv.org/abs/2012.00720.

[42] B. Cheng *et al.*, “Masked-attention mask transformer for universal image segmentation,” *CoRR*, vol. abs/2112.01527, 2021. arXiv: 2112.01527. [Online]. Available: https://arxiv.org/abs/2112.01527.

[43] R. Rombach *et al.*, “High-resolution image synthesis with latent diffusion models,” *CoRR*, vol. abs/2112.10752, 2021. arXiv: 2112.10752. [Online]. Available: https://arxiv.org/abs/2112.10752.

[44] J. Ho *et al.*, “Denoising diffusion probabilistic models,” *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

[45] J. Sohl-Dickstein *et al.*, “Deep unsupervised learning using nonequilibrium thermodynamics,” *Proceedings of the 32 nd International Conference on Machine Learning*, 2015.

[46] A. Nichol and P. Dhariwal, “Improved denoising diffusion probabilistic models,” *Proceedings of the 38 th International Conference on Machine Learning*, 2021.

[47] P. Dhariwal and A. Nichol, “Diffusion models beat gans on image synthesis,” 2021.

[48] A. D. Vita, in *Non-equilibrium Thermodynamics*, vol. 1, Springer Cham, 2022, ISBN: 978-3-031-12221-7. DOI: 10.1007/978-3-031-12221-7.

[49] C. Yang *et al.*, “Real-world denoising via diffusion model,” 2023.

[50] A. Lugmayr *et al.*, “Repaint: Inpainting using denoising diffusion probabilistic models,” 2022.

[51] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[52] M. A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[53] D. H. Ballard, “Modular learning in neural networks,” in *Proceedings of the sixth National Conference on artificial intelligence-volume 1*, 1987, pp. 279–284.

[54] D. Bank *et al.*, “Autoencoders,” *CoRR*, vol. abs/2003.05991, 2020. arXiv: 2003.05991. [Online]. Available: https://arxiv.org/abs/2003.05991.

[55] P. Vincent *et al.*, “Extracting and composing robust features with denoising autoencoders,” 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:207168299.

[56] L. Theis *et al.*, *Lossy image compression with compressive autoencoders*, 2017. arXiv: 1703.00395 [stat.ML].

[57] J. Redmon *et al.*, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[58] R. Girshick *et al.*, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[59] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[60] S. Ren *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[61] E. H. Adelson *et al.*, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[62] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Ieee, vol. 1, 2005, pp. 886–893.

[63] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[64] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[65] K. He *et al.*, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[66] T.-Y. Lin *et al.*, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[67] S. Liu *et al.*, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8759–8768.

[68] G. Ghiasi *et al.*, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 7036–7045.

[69] M. Tan *et al.*, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[70] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[71] Y.-J. Cao *et al.*, "Recent advances of generative adversarial networks in computer vision," *IEEE Access*, vol. 7, pp. 14 985–15 006, 2018.

[72] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," *arXiv preprint arXiv:1701.00160*, 2016.

[73] R. Salakhutdinov and G. Hinton, "Deep boltzmann machines," in *Artificial intelligence and statistics*, PMLR, 2009, pp. 448–455.

[74] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[75] M. Lucic *et al.*, "Are gans created equal? a large-scale study," *Advances in neural information processing systems*, vol. 31, 2018.

[76] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[77] A. Radford *et al.*, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[78] M. Arjovsky *et al.*, "Wasserstein generative adversarial networks," in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[79] I. Gulrajani *et al.*, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[80] J. T. Springenberg, "Unsupervised and semi-supervised learning with categorical generative adversarial networks," *arXiv preprint arXiv:1511.06390*, 2015.

[81] E. L. Denton *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," *Advances in neural information processing systems*, vol. 28, 2015.

[82] R. A. Yeh *et al.*, "Semantic image inpainting with deep generative models," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5485–5493.

[83] C. Ledig *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[84] P. Isola *et al.*, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[85] T.-T. Do *et al.*, *A theoretically sound upper bound on the triplet loss for improving the efficiency of deep distance metric learning*, 2019. arXiv: 1904.08720 [cs.CV].

[86] N. Serrano and A. Bellogín, "Siamese neural networks in recommendation," *Neural Comput. Appl.*, vol. 35, no. 19, pp. 13 941–13 953, May 2023, ISSN: 0941-0643. DOI: 10.1007/s00521-023-08610-0. [Online]. Available: https://doi.org/10.1007/s00521-023-08610-0.

[87] J. Bromley *et al.*, "Signature verification using a "siamese" time delay neural network," in *Advances in Neural Information Processing Systems*, J. Cowan *et al.*, Eds., vol. 6, Morgan-Kaufmann, 1993. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf.

[88] R. Hadsell *et al.*, "Dimensionality reduction by learning an invariant mapping," Feb. 2006, pp. 1735–1742, ISBN: 0-7695-2597-0. DOI: 10.1109/CVPR.2006.100.

[89] Z. Li *et al.*, *Journal of Intelligent Systems*, vol. 31, no. 1, pp. 690–705, 2022. DOI: doi:10.1515/jisys-2022-0055. [Online]. Available: https://doi.org/10.1515/jisys-2022-0055.

[90] S. A. Sheshkal *et al.*, *An improved person re-identification method by light-weight convolutional neural network*, 2020. arXiv: 2008.09448 [cs.CV].

[91] L. Bertinetto *et al.*, *Fully-convolutional siamese networks for object tracking*, 2021. arXiv: 1606.09549 [cs.CV].