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

Thanh Huan Hoang
Student
huan.hoang@tum.de

Daniel Lehmberg
Student
daniel.lehmberg@tum.de

Jinxin Ai
Student
jinxin.ai@tum.de

Marko Alten
Student
marko.alten@tum.de

Kevin Dietrich
Student
kevin.dietrich@tum.de

Johannes Lehnerdt
Student
j.lehnerdt@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 six 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. Convolutional Neural Network

With the increasing popularity in recent years [1], convolutional neural networks (CNNs) have emerged and become a foundational design pattern for deep learning models. They are a type of feed-forward neural network whose architecture consists of alternating layers of convolutional filters and pooling operations, allowing them to efficiently extract image features and patterns while reducing dimensionality. This ability enables the models to learn a spatial hierarchy in images, ranging from simple to complex features, making them extremely powerful for computer vision tasks.

1.1. Intent

CNNs were designed to automatically extract features or patterns from structured data by leveraging local correlations and using kernels (convolutional filters) [1]. Images are an excellent example of local correlation in data, as nearby pixels tend to have similar colors and intensities. The hierarchical feature extraction process is accomplished by applying kernels that detect patterns within local regions, enabling CNNs, as they go deeper, to identify low-level features such as edges, mid-level features like shapes and textures, and high-level features such as objects and scenes. Each filter is trained to extract a distinct feature, enabling convolutional layers to incorporate multiple filters to detect various features.

Moreover, CNNs add more structure to the model through weight sharing in each filter, which reduces the number of parameters and computational cost compared to a multilayer perceptron (MLP) [2], where each neuron in one layer connects to every neuron in the subsequent layer. Weight sharing means the same filter (same set of weights) is applied across different input regions, allowing the network to detect the same feature regardless of its position. Due to this, CNNs exhibit translational equivariance. Using weight sharing and local connectivity, CNNs can effectively process high-dimensional data [1].

1.2. Block Diagram

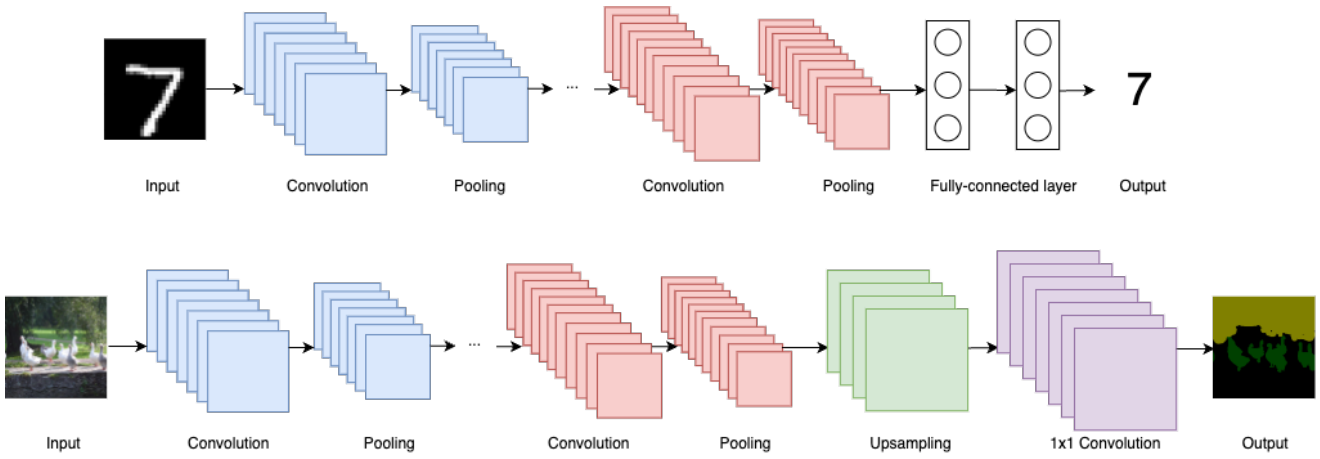

Figure 1: Architecture of CNNs for image classification and image segmentation

Figure 1 shows the architecture of CNNs in common tasks. A common pattern is alternating between convolution and pooling (subsampling) layers. The convolutional layer, which already includes the activation function, extracts features while the pooling layer selects them. As the network grows deeper, dimensions (width and height) decrease while the number of filters increases to capture more complex patterns. The final layers vary depending on the task. Usually, for image classification, fully-connected layers are used, while image segmentation utilizes upsampling followed by a 1x1 convolution layer.

1.3. Limitations

Despite numerous advantages, such as automatic feature extraction without manual feature engineering, efficient computation, and variable input size (for fully convolutional networks), CNNs still have some limitations. The first one is the

assumption about local correlation, which is true for grid-like structured data like images or time-series data like audio [3]. In the case of images, each pixel is related to its neighboring pixels, and in the case of audio, it captures temporal dependencies. However, this is not true for other data types like tabular data. Using other machine learning models, such as decision trees or MLPs, may be more appropriate for tabular data as they do not assume any data locality.

Convolutional layers have a restricted receptive field, which can hinder the network's ability to capture global context, especially when relevant features appear unexpectedly in distant regions. This can lead to reduced performance in tasks requiring comprehensive spatial understanding. Initially developed for natural language processing, attention mechanisms have been integrated into some CNN models [4] to address this limitation. These mechanisms allow the network to focus on specific parts of inputs, facilitating global context capture. Another solution is using dilated convolutions, which increase the receptive field without adding more parameters or computation [5]. By skipping input values with a particular step size, dilated convolutions enable the model to capture global context while maintaining computational efficiency.

1.4. History

The creation of CNNs was inspired by Hubel and Wiesel's experiments [6] on cats' visual cortex, which revealed that simple cells respond to oriented edges at specific locations, while complex cells are sensitive to edge orientations regardless of position. This prompted Kunihiko Fukushima in the 1980s to develop the Neocognitron [7] using alternating S-layers (equivalent to convolutional layers) and C-layers (equivalent to pooling layers). Since the Neocognitron lacked backpropagation and the structured design in modern CNNs, LeNet-5 [8] is often regarded as the foundational CNN model. Introduced by Yann LeCun in 1998, LeNet-5 was designed for handwritten digit recognition, using alternating convolutional and pooling layers, followed by fully connected layers. It uses the tanh activation function and average pooling. Modern CNNs prefer max pooling and ReLU-like functions for better feature capture and vanishing gradient mitigation. LeNet-5 was limited by the computational resources available at the time, resulting in shallower architectures compared to modern CNNs.

AlexNet [9], a significant milestone in 2012, drastically improved image classification on the ImageNet dataset by cutting the ImageNet top-5 error almost in half. This was achieved by using ReLU activations and GPU acceleration, allowing for training a deeper network and boosting performance. In 2014, VGGNet [10] further increased the depth of CNNs, demonstrating that deeper networks do not necessarily lead to better performance. This issue was addressed with ResNet's skip connections [11] in 2015. Also, in 2014, GoogLeNet introduced the inception layer [12], allowing for multiple kernel sizes within the same layer and incorporating dimension reduction techniques to enhance efficiency. EfficientNet [13], introduced in 2019, uses compound scaling to balance model complexity with computational efficiency.

Recent CNN models like HRNet [14], CoAtNet [15], and ConvNeXt [16] continue to innovate, focusing on high-resolution feature integration, integration of convolution and attention mechanisms, and flexible connectivity patterns, respectively.

1.5. Application

CNNs are applied not only in the field of computer vision but also in other domains where data exhibits local correlations:

- **Image classification:** Many CNN models were originally developed for this purpose. The Neocognitron [7] was used to recognize handwritten characters, while LeNet-5 [8] focused on handwritten digits. Following the introduction of the ImageNet dataset and the success of AlexNet [9], subsequent models aimed to achieve lower error rates in image classification tasks using this dataset.
- **Image segmentation:** This task involves assigning each pixel of an image to a predefined class, indicating the object class to which it belongs. Fully convolutional networks, like the U-Net architecture [17], are commonly used.
- **Object detection:** CNNs can separately detect objects using bounding boxes within an image. An example of this is using CNNs in smart cars to detect humans, traffic lights, or other vehicles in real time [18].
- **Natural language processing:** Although less common, CNNs demonstrate significant capability in this field. They can process and generate audio signals by leveraging temporal correlations. For instance, WaveNet [19] can synthesize speech, convert text to speech, compose music across various genres, and be adapted for speech recognition tasks.

2. Diffusion Models

Diffusion models are generative models that learn to gradually recover input data by reversing a process that iteratively adds noise to the training data [20]. Having made tremendous progress in recent years, diffusion models are now achieving truly impressive results [21] and have become one of the hottest topics in the field of computer vision [20].

2.1. Intent

Diffusion models aim to generate new, high-quality, and diverse data points. These data points can be images, videos, 3D shapes, or point clouds [20]. Apart from that, diffusion models can also be used for molecular design [22] or as a defense mechanism for adversarial attacks [23].

However, the most common application of diffusion models is in the image domain. In particular, they can be used not only for image generation, but also for other image-related tasks such as image super-resolution, image editing, image inpainting, or image segmentation [20].

2.2. Block Diagram

The fundamental principle underlying diffusion models is the diffusion process. As illustrated in Figure 2, the diffusion process can be divided into two distinct processes: the *forward process* and the *reverse process*.

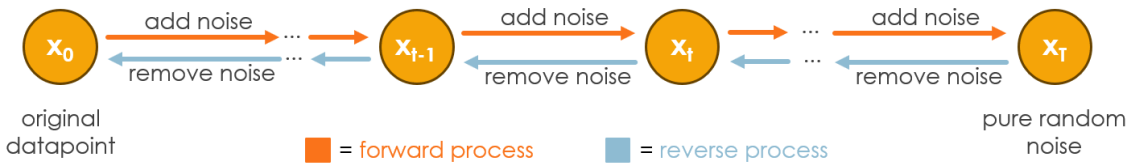

Figure 2: Distinction of the diffusion process into the forward process and the reverse process.

In the forward process, a small amount of Gaussian noise is gradually added, starting with the original data point x_0. This is done T times until the original data point is completely transformed into pure random noise x_T. The whole forward process is fixed, so there is no learning involved [24, 25]. This is not the case for the *reverse process*, which recovers the original data point by learning to gradually reverse the forward process. This means that, starting from pure random noise x_T, a small amount of Gaussian noise is removed for a total of T times. By doing so, the original data point x_0 is restored [20].

The reverse process can be approximated by a Gaussian probability distribution if the steps of the forward process are small enough. For figuring out the mean and the variance of this probability distribution a neural network is used, typically a modified version of the U-Net architecture [25]. Removing the noise is done in several steps, as this is easier for the model to learn than doing it in one step [22]. Figure 3 visualizes the results of the diffusion process, using the example of images as data points.

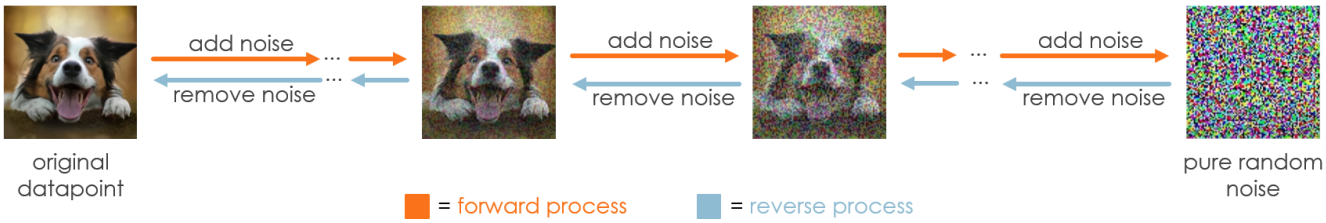

Figure 3: Visualization of the diffusion process for images as data points.

2.3. Limitations

Despite all the advantages that make diffusion models state-of-the-art in generative modeling, they also have some limitations. Most importantly, they have a low inference speed compared to other generative models. This is because they have to perform several steps at inference time [26, 20]. Diffusion models are also computationally expensive to train because training them involves repeatedly performing functions in a high-dimensional data space [27].

Apart from that, diffusion models are largely limited to generating samples that are similar to the training data [28]. This is because they attempt to learn the distribution of the training data during the training process and then sample from this distribution. Furthermore, diffusion models sometimes also generate data points that are inconsistent with the real world, as they can hallucinate non-existent objects. Therefore, they are not 100% accurate in terms of truthfulness [22]. Another limitation of diffusion models is that they can reveal training data. This is problematic if it contains sensitive or private information [27]. Finally, diffusion models tend to reproduce or exacerbate biases that are present in the training data [24, 27]. This could not only be discriminatory but could also reinforce stereotypes.

2.4. History

In 2015, Sohl-Dichstein et al. [29] introduced *diffusion probabilistic models*. The basic idea of these models is to gradually destroy the structure in a data distribution and then learn a reverse process that restores it. This technique has its origins in the field of statistical physics. However, its introduction into the field of machine learning sparked the development of a whole new branch of generative models. A huge leap in sample quality was achieved by Ho et al. [24] in 2020. This was the result of some groundbreaking changes such as the use of U-Nets and learning the reverse process by estimating the variance.

Nichol et al. (2021) [21] proposed several improvements to the diffusion process such as learning the variance of the reverse process or using a hybrid learning objective. This led to a better log-likelihood, similar sample quality, and much faster sampling. Furthermore, several improvements to the architecture of diffusion models were proposed by Dhariwal et al. [26] (2021). For example, increasing the number of attention layers and attention heads in the U-Net architecture and performing adaptive group normalization proved to be beneficial. As a result, the sample quality was improved significantly which led to diffusion models outperforming state-of-the-art generative models.

Over the past few years, several text-to-image models have been published that build on diffusion models and can generate really impressive and realistic images. The best known are *DALLE·2* [30], *Imagen* [31], *Midjourney* [32] and *Stable Diffusion* [27]. More recently, also good-performing text-to-video models have been published, such as *Imagen Video* [33] and *SORA* [34].

2.5. Application

As diffusion models have several advantages and limitations, it always depends on the specific use case if it is beneficial to apply them. In the following, an overview will be given of the use cases in which it makes sense to apply diffusion models and the use cases in which it does not.

Diffusion models are state-of-the-art in terms of generative modeling [26]. Consequently, they are a really good option if high-quality samples are needed [27]. Apart from that, diffusion models are a good choice if highly diverse data is needed. This is because they achieve a higher diversity than other generative models [26]. Further use cases for applying diffusion models are if the training process should be stable [20], if the used data has a complex distribution [27], or if completely new samples should be generated that have never been seen before by humans [25].

However, there are also use cases in which other types of generative models are preferable to diffusion models. In particular, if sampling needs to be very fast, such as for real-time applications, diffusion models are not suitable [26]. They are also usually not a good choice if a custom model should be trained unless lots of computing power and training data are available. Finally, if the generated data must be 100% accurate in terms of truthfulness, diffusion models are also not well suited [22].

3. Autoencoder

Autoencoders are an unsupervised learning technique used in artificial neural networks [35]. The network captures the most important features of the data by compressing the input into a lower-dimensional latent space and then reconstructing it back to its original dimension. This process involves two main components: the encoder and the decoder [36]. For practical implementation, refer to the Autoencoder code repository used in this seminar.

3.1. Intent

The primary intent of an autoencoder is to learn an efficient representation (encoding) of a set of data, typically for the purposes of dimensionality reduction or feature learning. Autoencoders aim to capture the most salient features of the input data by training a neural network to compress the input into a latent-space representation and then reconstruct the output from this representation [37].

3.2. Block Diagram

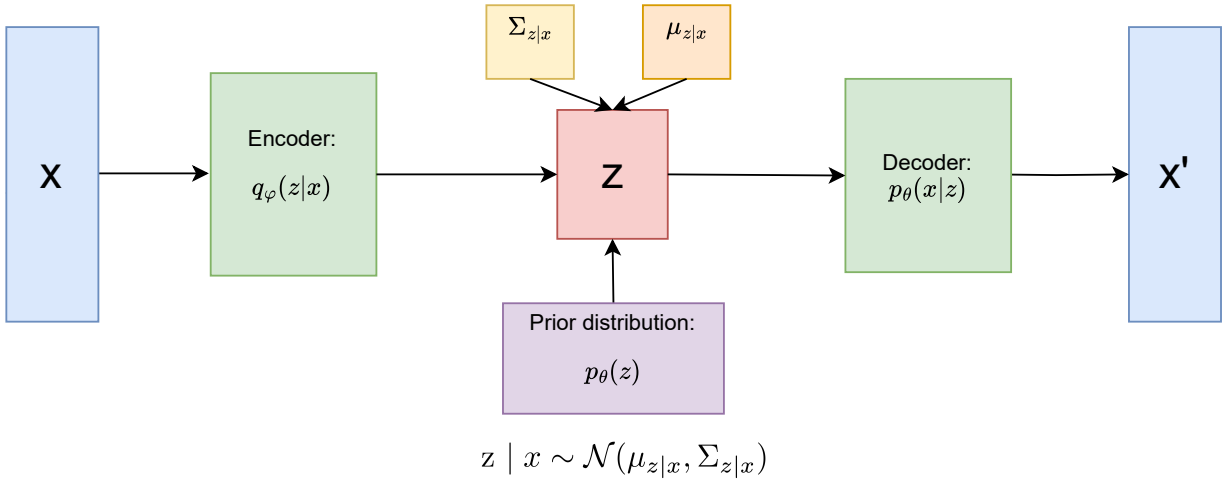

Figure 4: Variational Autoencoder

As illustrated in Figure 4 Kingma et al. [38] introduced a revolutionary and creative advancement in the field of autoencoders in 2014. This innovation significantly enhanced the traditional autoencoder architecture by introducing a unique manipulation of the hidden layer. In a standard autoencoder, the hidden layer learns a deterministic mapping of the input data. However, Kingma et al. [38] transformed this approach by treating the hidden layer as a probability distribution of a latent variable z. This is achieved by integrating the encoder and decoder into a probabilistic framework, allowing the model to learn not just a single point representation but a distribution that captures the uncertainty and variability in the data.
The key mathematical formulation can be summarized as follows:

$$
\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x,z) - \log q_\phi(z|x)\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)p(z) - \log q_\phi(z|x)\right] && \text{Chain rule of probability} \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log p(z) - \log q_\phi(z|x)\right] && \text{Split the expectation} \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - D_{KL}\left(q_\phi(z|x) \parallel p(z)\right) && \text{Definition of KL divergence}
\end{aligned}
$$

- Maximizing $\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$: This term represents the expected log-likelihood of the data given the latent variables, and maximizing it improves the reconstruction accuracy of the decoder.

- Minimizing $D_{KL}\left(q_\phi(z|x) \parallel p(z)\right)$: This term represents the Kullback-Leibler (KL) divergence between the approximate posterior distribution $q_\phi(z|x)$ and the prior distribution $p(z)$. Minimizing it ensures that the learned latent variable distribution is close to the prior distribution, promoting regularization and preventing overfitting.

3.3. Limitations

- **Autoencoders:** While adept at learning representations, face several limitations. They often suffer from lossy reconstruction, meaning the output may not perfectly replicate the input, leading to information loss. Additionally, they are prone to overfitting, particularly when the model is overly complex or the dataset is small. The interpretability of the learned latent space can be challenging, making it less useful for downstream tasks. Furthermore, autoencoders trained on specific data distributions may struggle to generalize well to unseen data or different distributions [39].
- **Variational Autoencoders:** Training VAEs can be computationally intensive and requires careful hyperparameter tuning to balance reconstruction loss and KL divergence. They often produce blurry reconstructions, suffer from mode collapse, and face approximation errors due to the simplified posterior distribution. The latent space can be challenging to interpret, and the model's performance is sensitive to the choice of prior distribution. Additionally, VAEs can struggle with scalability issues and typically do not match the generative quality of Generative Adversarial Networks (GANs) [40]. Optimizing the KL divergence term can be difficult, and the stochastic nature of VAEs introduces variability in the training process, making it less stable and requiring more iterations to converge [41].

3.4. History

Autoencoders were initially introduced in the 1980s by Hinton and the PDP group to solve the issue of "backpropagation without a teacher," by utilizing the input data as the teacher [42]. Another significant advancement is the denoising autoencoder introduced by Vincent et al. in 2008 [43]. Their core idea involves adding noise to the input signal to improve the model's generalization capabilities. This approach is inspired by human object recognition, where humans can accurately identify objects even in noisy or degraded images. In 2014, Kingma et al. introduced the variational autoencoder (VAE), marking a significant advancement in the field [38]. By employing a probabilistic approach to learning latent representations, VAEs enabled the creation of more robust generative models. This development is considered one of the most revolutionary improvements in the realm of autoencoders to date. Recent modifications to the autoencoder have primarily focused on altering the cost function. Tolstikhin et al. proposed using the Wasserstein distance instead of the KL divergence, due to the advantageous properties of the Wasserstein distance [44]. While there have been few fundamental changes to the autoencoder itself in recent years, significant advancements have been made in its application to fields such as computer vision, natural language processing, and medical modeling.

3.5. Application

Autoencoders have found numerous applications across various domains. For instance, Zhao et al. introduced an unsupervised autoencoder-based feature learning method specifically designed for cybersecuritym [45]. This approach leverages advanced neural network techniques to develop a hyperspectral image classification method, which excels in malware classification and network-based anomaly detection. Their method combines a Deep Stacked Autoencoder (DSAE) with a 3D Deep Residual Network (3DDRN), enhancing the capability to identify and classify intricate patterns in cybersecurity data [45].

Autoencoders have proven to be invaluable in healthcare applications, particularly in the realm of medical image processing. Noteworthy advancements have been spearheaded by Stefan Röhrl and his team, in collaboration with the Institute of Data Processing at the Technical University of Munich and Translatum [46]. Their pioneering work integrates Principal Component Analysis (PCA) methods with autoencoder techniques for the feature extraction and classification of white blood cells. This innovative approach aims to enhance the accuracy and efficiency of medical diagnoses. While this research is ongoing, its promising potential could significantly advance human healthcare in the future.

In conclusion, the Autoencoder is a versatile tool in the neural network design paradigm, providing a robust approach to representation learning, dimensionality reduction, and generative modeling. However, there is still a great deal of room for improvement, which offers significant potential for advancements that can benefit mankind.

4. Attention

Attention is an architectural pattern for neural networks that has gained significant popularity in recent years. This mechanism is now integral to a variety of network architectures. Attention enables the network to focus on important features of the input and/or output data. The primary components facilitating this behavior are queries, keys, and values.

4.1. Intent

In any dataset, certain parts are more crucial than others for tasks such as image classification, while some parts are less relevant. The purpose of the attention mechanism is to enable neural networks to dynamically focus on the most important parts of the data. This allows the networks to manage the inherent complexity and volume of the data more effectively. This capability is particularly advantageous in tasks like machine translation, text summarization, and image recognition. By employing attention, models enhance their ability to capture distant dependencies and complex patterns, thereby improving overall performance[47][48].

4.2. Block Diagram

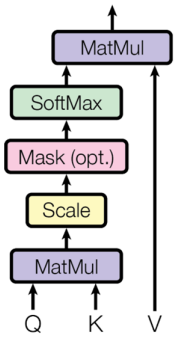

Figure 5: Single-Head Attention [47]

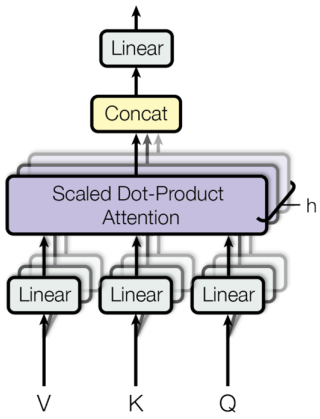

Figure 6: Multi-Head Attention [47]

Figure 5 shows the basic architecture of a scaled dot-product attention block named Single-Head Attention. Figure 6 shows the parallel arrangement of several Single-Head Attention blocks called Multi-Head Attention.

The mechanism depicted in figure 6 was introduced by Vaswani et al. [47] in 2017 and forms the foundation for transformer architectures. The basic concept revolves around three matrices: Queries, Keys, and Values, which are linear transformations of the input embedding X also called tokens. The weight matrices W^Q, W^K, and W^V, are modified during the training process of a model.

$$Q = XW^Q \quad (1) \qquad K = XW^K \quad (2) \qquad V = XW^V \quad (3)$$

Each of the three components addresses a different aspect of the input data:

1. Queries define the characteristics the model is searching for.
2. Keys contain the context of each token in the input data.
3. Values represent the actual information of the input data.

Following the formula for a single-head attention block:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (4)$$

The dot product of QK^T, also known as attention score, can become large in magnitude. To mitigate the risk of the softmax function operating in regions with small gradients, the dot product is scaled by the square root of d_k, the dimensions of the key matrix. By multiplying with V, the value of each token is adjusted, which is represented as the weighted sum of the input tokens. The output of the attention block is the context vector[47].

Such a trio of distinct queries, keys and values is called Single-Head Attention block. A single head focuses on certain aspects of the embedding. By parallelizing multiple attention heads through Multi-Head Attention, the network can focus on different aspects of the embedding. Multiple layers of Multi-Head Attention blocks and multi-layer perceptrons, allow each embedding to be influenced by its surroundings. The Multi-Head Attention blocks are the centerpiece of this design pattern [47].

4.3. Limitations

1. **Computational Complexity**: Attention mechanisms exhibit a complexity of $O(n^2)$ in terms of memory usage and computation time. This quadratic complexity arises from the need to calculate attention scores between all pairs of tokens in the input sequence. As a result the computational cost becomes prohibitive for long sequences [47].
2. **Data Requirements**: Training of neural networks leveraging attention mechanisms requires vast amounts of data. Large-scale datasets are essential to capture the important information in the data and allows the model to learn effective attention patterns [49].
3. **Scalability**: Scaling neural networks to handle larger datasets demands substantial computational power. Large models like GPT-3 with 175 billion parameters, require extensive computational infrastructure capable of parallel processing across many devices [50].

4.4. History

The development of attention mechanisms has significantly advanced computer vision and natural language processing. Early approches emerged in 2010 with Larochelle et al. [51], who aimed to reduce computational complexity in image processing. They introduced a network that focuses on important image areas, avoiding the need to process the entire image. In 2014, Bahdanau et al. [52] made a breakthrough with an attention mechanism for machine translation task. Their model used a bidirectional recurrent neural network with an attention layer, allowing it to focus on various hidden states during decoding, improving translation accuracy. A major milestone came in 2017 with Vaswani et al. [47]. They introduced the Transformer architecture based on self-attention. This mechanism allows the model to consider the entire input sequence simultaneously, enhancing efficiency and scalability in natural language processing tasks. In 2021, Dosovitsky et al. [53] introduced the Vision Transformer (ViT) for image classification. Unlike traditional convolutional neural networks (CNNs), ViT applies self-attention to image patches. This demonstrates the versatility of attention mechanisms beyond text processing.

4.5. Application

The attention mechanism is widely employed across various fields and domains. In natural language processing (NLP), attention is crucial for tasks like machine translation, text summarization, and question answering [54]. Attention enables models to focus on relevant text segments, thereby improving accuracy and fluency. For instance, in machine translation, attention allows the model to align source and target language segments effectively, leading to more coherent and contextually appropriate translations.

In computer vision, attention finds extensive application in tasks such as image classification [53], object detection [55], and action recognition [56]. Attention models for computer vision tasks focus on key regions and frames, enhancing performance by highlighting the most critical parts of an image or sequence. This selective focus helps in identifying objects with higher precision and understanding complex scenes more effectively.

Beyond NLP and computer vision, attention mechanisms are applied in various fields including speech processing, recommender systems, bioinformatics, and finance. These applications enhance performance by focusing on crucial features, such as audio signals in speech processing, user preferences in recommender systems, genetic sequences in bioinformatics, and market indicators in finance [57].

5. Siamese Network

Siamese networks are a network architecture proposed to solve the task of similarity comparison. While the initial proposal was to perform a simple distance comparison in the feature space of two inputs [58], the concept proved suitable for solving numerous other tasks. With significant advances in recent years, Siamese networks have become a commonly used tool for applications in computer vision and other fields where similarity is of interest.

5.1. Intent

Comparing two images based on their similarity can be quite challenging. Larger images require considerably more computational resources, and depending on how fine-grained the detection of differences between the images should be, further complexity arises. To solve this problem, Siamese networks do not compare the actual input images but the low-dimensional feature vector of the images. Since the main idea of the feature vector is to represent the details of the images, minor differences in the actual image will only lead to small changes in the feature vector. Accordingly, the distance between two feature vectors can be used as a metric for similarity comparison. Based on this metric, the inputs can not only be compared in pairs, but creating a ranking between the different inputs is also possible.

5.2. Block Diagram

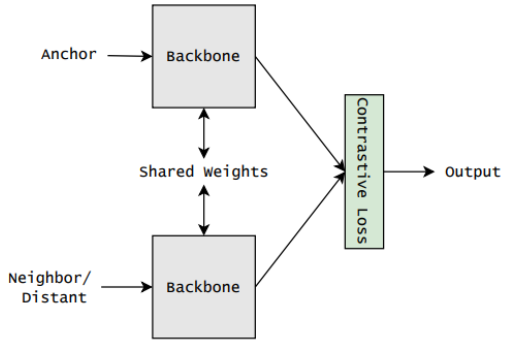

Figure 7: Siamese network with contrastive loss

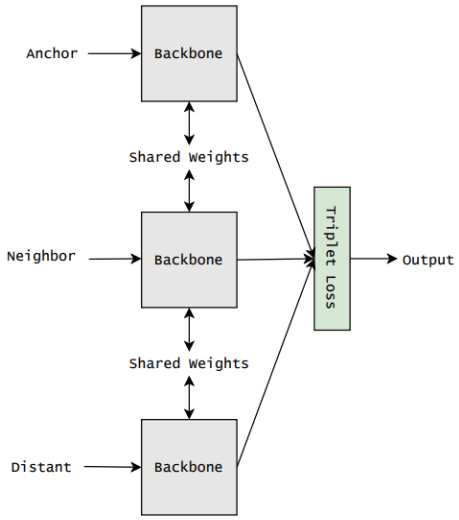

Figure 8: Siamese network with triplet loss

Figure 9: Comparison of Siamese networks with different loss functions [59]

Figure 7 represents the initially proposed structure of the Siamese network [58]. The same neural network is used for both inputs to allow a comparison of the feature vectors. The vectors are then forwarded to a contrastive loss [60]. This loss is defined as follows:

$$L_{con} = \frac{1}{2} * (Y_{truth} * D^2) + \frac{1}{2}(1 - Y_{truth}) * \{max(0, M - D)\}$$

Here, Y_{truth} is either 1 if the training images are similar or 0 otherwise. D is the distance between the feature vectors and can be calculated by using the Euclidean distance for example. Furthermore, M is a new hyper-parameter which defines the degree of discrimination [61] and thus specifies the distance beyond which the inputs should no longer be moved apart. As a result, the loss either increases or decreases the distance of the inputs based on their similarity. During training, the weights are shared between the two networks to update equally. Once trained, the forward pass of the network returns a distance-based similarity, which can classify images as similar or dissimilar in combination with a predefined threshold value.

While this basic structure already provides a powerful tool for similarity comparisons, further improvements and variations have been proposed. One commonly known variation is the triplet architecture [62]. As seen in Figure 8, this architecture utilizes the same network three times. Correspondingly, the inputs can be utilized in the following way. One image will be labeled as the anchor, while the other images are a neighboring, similar image and a distant, dissimilar image. This structure allows the network to cluster different classes more easily since images are grouped and separated simultaneously. The subsequent formula describes the corresponding triplet loss [63].

$$L_{tri} = max(D(f_a(x_a), f_n(x_n)) - D(f_a(x_a), f_d(x_d)), M)$$

D and M are defined as in the contrastive loss and $f_a(x_a)$, $f_n(x_n)$ and $f_n(x_n)$ are the feature vectors of the anchor, the neighboring and the distance images, respectively.

5.3. Limitations

Since Siamese networks consist of multiple neural networks, the required computational resources increase based on the number of networks. This is especially important for corresponding applications in which a similarity comparison should be performed on devices with limited computational power, such as embedded systems or IoT devices. As a result, this is an ongoing research field to allow such devices to utilize Siamese networks [64].
Another factor to consider is the longer data processing time required for Siamese networks. Since corresponding pairs must be created and labeled for training, more work must be invested compared to conventional data sets and networks. Forming effective training pairs can be crucial to ensure smooth training and positive results [65].
Furthermore, it is worth noting that Siamese networks do not generate a probability like other networks. Instead, the generated output will be a distance-based similarity. This difference must be considered when using Siamese networks in classification or similar tasks, as the class assignment must be adapted to this output format [66].

5.4. History

The original paper introducing Siamese networks was published in 1993 by J. Bromley et al. [58]. The paper was intended to provide a tool for verifying signatures and finding forged signatures among genuine ones. At the same time, P. Baldi and Y. Chauvin independently proposed a similar architecture for fingerprint recognition [67]. While the idea of Siamese networks offered new ways to solve similarity tasks, the previously mentioned limitations, especially the higher computational power required, proved challenging for further progress. Accordingly, significant progress in improving Siamese networks has only been made in recent years. Following the introduction of FaceNet [63] and the proposal to use Siamese networks in one-shot learning [68] in 2015, the architecture was reviewed for additional applications. Due to the limited amount of data in many application areas, the realization that Siamese networks can be used effectively even with a very small sample size has further strengthened interest. Further progress was made by Li et al. [69], who presented a combination of Siamese networks with regional proposal networks. This combination enabled Siamese networks for real-time tracking applications and resulted in further research for object tracking. He et al. [70] and Dong et al. [71] revisited established techniques like multiple branching and the triplet loss, respectively, for their approach on object tracking. Shen et al. [72], on the other hand, utilized the attention mechanism for this task. Other authors have tried to improve the Siamese architecture and worked on minimizing the limitations. Chen et al. [73], for example, showed that Siamese networks learn meaningful representations even without negative sample pairs, while Zhou et al. [74] proposed MASNet, a mutual-attention mechanism for improved performance on change detection datasets.

5.5. Application

Siamese networks' primary application area is similarity comparisons. Since this is a common problem, Siamese architecture can be utilized in many tasks. A direct comparison between two images, as the initial authors proposed for signatures [58] or face recognition like FaceNet [63], might come naturally as applications. However, utilizing the comparison in a clever way also allows for other use cases. As described previously, object tracking [69] can be implemented by comparing the difference between images in a time frame. Combining the Siamese architecture with other architectures can also open new applications. S. Zagoruyko and N. Komodakis [75] used a CNN-based Siamese model to approach the correspondence problem in computer vision. Furthermore, the underlying distance function can be used for unsupervised learning, as proven by X. Wang and A. Gupta [76]. Their research found that even unlabeled data can be used for visual representation by ranking the images with the triplet loss.

6. Region Proposal Networks

Region Proposal Networks (RPNs) are a type of neural network designed for generating region proposals within an image. A region proposal usually consists of a rectangular box that defines the bounds of an object, along with a confidence score that indicates the likelihood of the object being present in the region. RPNs pose a key role in modern object detection systems such as Faster R-CNN [77]. They provide a significant improvement over traditional image processing algorithms that were the norm before RPNs were introduced [78][79].

6.1. Intent

In 2015 Girshick identified region proposal generation as the main bottleneck in his proposed Fast R-CNN model [79]. These types of image processing algorithms usually came with drawbacks such as high computational cost and low accuracy due to the large number of region proposals. To address these issues, Ren *et al.* introduced a new neural network-based method in a subsequent paper called Faster R-CNN [77].

The main purpose of the RPN is to generate region proposals more efficiently than traditional methods. These improvements are a key element in enabling real-time object detection. The primary task of a RPN is to take an image as an input and output a set of region proposals along with their confidence scores, called objectness scores.

6.2. Block Diagram

Most RPNs share four key concepts that make them very powerful. The first concept is the use of an anchoring mechanism. This mechanism places rectangular boxes in the images that are refined by subsequent network layers. The second key aspect is the ability to perform end-to-end training. This means that the network can be trained with a joint objective that aims to improve objectness classification as well as refine the anchors to better fit the objects in the image. The third important concept, crucial for improving performance, is sharing convolutional features with subsequent network layers, which reduces redundant calculations. The fourth important distinction is that models using the RPN architecture are defined as two-stage detectors. This means that object classification is done separately from region proposal generation.

Ren *et al.* first proposed a RPN that takes the last feature map of a pre-trained backbone network as an input and outputs a set of region proposals along with their objectness scores (Figure 10). A sliding window over the feature map is used as an input along with a set of predefined anchor boxes that feed into two important layers. The first layer (*cls*) classifies the anchor boxes as either object or background. Another layer (*reg*) takes the same feature map window and outputs bounding box refinements for each anchor box, such as the offset needed to better fit the object in the image and adjustments to its width and height. Thus, for a feature map of size $W \times H$ with k anchor boxes, $W \times H \times K$ anchors are placed on the image. The number of outputted region proposals depends on the classification score threshold, commonly around 1000 region proposals.[1] When RPNs were first introduced in Faster R-CNN, a set of 9 distinct anchor boxes was used with 3 different scales and 3 different aspect ratios [77].

This anchoring mechanism provides two important properties to handle many different types of objects in an image. First, the anchors are translation invariant relative to their sliding window. This allows the network to guarantee the same region proposal and function output for the same object regardless of its position in the image. Second, anchors can handle objects at different scales and aspect ratios very efficiently without having to compute image pyramids or similar techniques.

To train the network, a set of pre-trained weights from a backbone network such as VGG16 [81] or ResNet [82] is used. As mentioned before, the network is trained with a joint objective. This is achieved by using an alternative training technique. In the original paper, the network is trained in four steps. Importantly, instead of using all proposed regions for training, a subset of randomly sampled regions is chosen to avoid bias towards negatives. The ratio of positive to negative samples can be up to 1:1. A sample is considered positive, if the intersection over union (IoU) is larger than a certain threshold, or larger than that of all other samples. An anchor is considered a negative sample, if the IoU is smaller than a certain threshold. All

[1] This principle can be shown using a simple Python-notebook that utilizes Torchvision's implementation of Faster R-CNN [80]. The following link leads to an example: https://github.com/lhnrdt/faster-rcnn-demo/blob/main/faster-rcnn-demo.ipynb

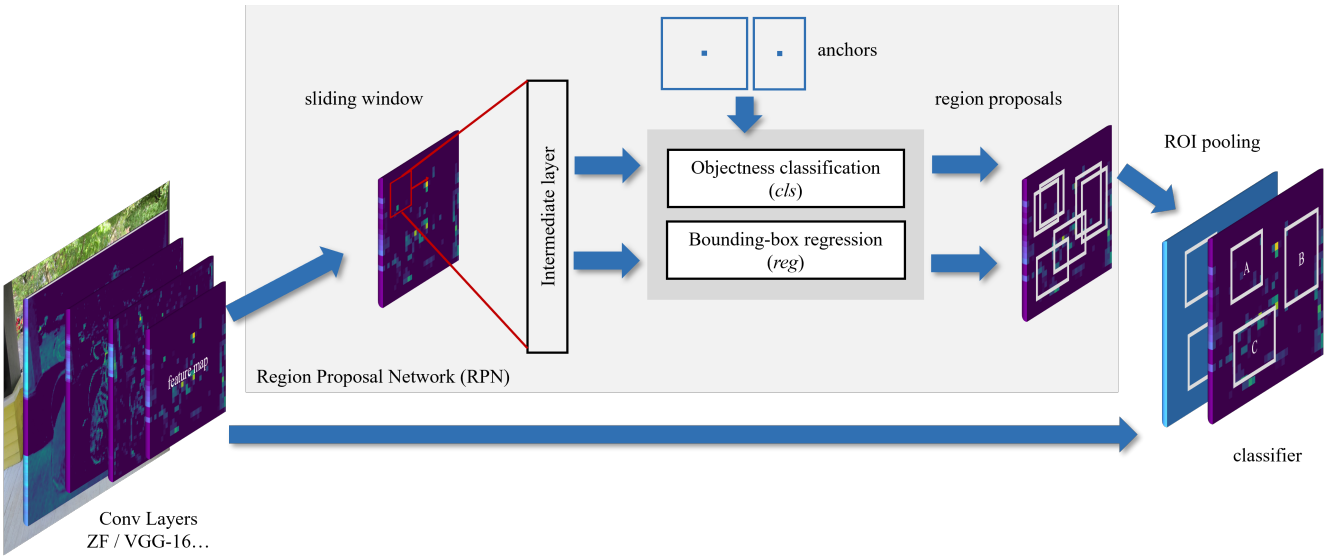

Figure 10: Block diagram of the RPN introduced in Faster R-CNN [77].

remaining anchors do not contribute to the training objective at all [77].

The joint objective is defined as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(p_i, p_i^*) \tag{5}$$

Where p_i is the predicted probability of anchor i being an object, p_i^* is the ground truth label, t_i is the predicted bounding box regression output, p_i^* is the ground truth bounding box regression output, L_{cls} is the log loss function, L_{reg} is the smooth L1 loss function, N_{cls} and N_{reg} are normalizing factors, and λ is a hyperparameter that balances the two losses. An important property to notice, is that the regression loss is only activated for positive samples ($p_i^* L_{reg}$). [77]

One important aspect of the RPN is that it shares convolutional features across many layers of the network. This applies not only to Faster R-CNN architectures but is also a key concept for most RPNs in general. This is done to reduce redundant calculations and improve performance. The shared features are used for both the RPN (to refine anchors and predict objectness scores) and the subsequent object classification network [77].

6.3. Limitations

Eventhough RPNs usually excel in terms of accuracy and precision, they come with some limitations. If we focus our attention at visual tasks alone, the main limitation is the computational cost. This usually results in rather slow inference times. This is due to the fact that the network has to generate a large number of region proposals and then classify them.

Another drawback, if we consider RPNs as a pattern outside of the computer vision domain, is that they are not very versatile. RPNs are designed for spatial regions and would require some substantial modification to be applied to sequential (text), temporal (video), or other types of data. For tasks like tracking an object in a video with a RPN in its pure form, regions in subsequent frames are purely generated based on the current frame and don't take into account the temporal context. Although with some modifications and a siamese network architecture RPNs can be applied to tracking tasks. [83]

As with most neural networks, the results are highly dependent on the quality and abundance of training data. This is especially true for RPNs as they usually heavily rely on a pre-trained backbone network that is shared throughout the network.

So the proposed regions are only as good as the features extracted by the backbone network.

The powerful anchor mechanism only works well, if the set of predefined anchors is well chosen. In the original proposal of RPNs these were chosen by trial and error. A poor choice of these anchors can lead to a poor performance of the network. [84] Typically, the anchors are placed at each spatial location on the feature map. This usually results in a large number of negative anchors that don't contain any objects. This can lead to a negative bias in the training data, which needs to be addressed by sampling a subset of the anchors for training and adds to the computational cost of the network.

6.4. History

With Girshick's introduction of Fast R-CNN [79] in 2015, the need for a proper neural network based method was born and introduced with the already discussed Faster R-CNN [77] in 2017. Quickly after, in 2016 Lin *et al.* identified that adding more anchors would not increase the quality of the proposals. Instead, they proposed an architecture that made use of the pyramid of features extracted by the backbone network. This allowed the network to make use of features at different scales and aspect ratios more efficiently. [84]

Around this time, although not directly related to classical RPNs, Redmon *et al.* introduced the YOLO (You Only Look Once) architecture. This network was designed to make object detection in real time possible. Instead of using a two-stage detector, YOLO used a single neural network to predict the bounding boxes and the class probabilities directly from the full image. This was a significant improvement in terms of speed compared to the two-stage detectors but still lacked behind in terms of precision. [85]

As some applications benefit from pixel wise segmentation, He *et al.* modified the RPN architecture by adding a mask generation branch to the network. This allowed to extract information like human pose estimations or pixel wise segmentation without adding large computational overhead. [86] Another contribution was Polygon R-NN by Castrejon *et al.* in 2017. This network tried to go beyond simple box predictions. Instead modified the architecture in such a way that it could sequentially output a set of vertices of a polygon that fits an object in the image. This was then used to make dataset annotations more efficient. [87] As previously mentioned in the limitations, until now, anchors were usually sampled uniformly across the image. Wang *et al.* addressed this issue by proposing a method that first generates a map of propabilities for a spatial location to contain the center of an object. These maps are then employed to sample more anchors in regions that are more likely to contain objects. This method was shown to improve the performance of the network. [88]

Around this time, single-stage detectors made significant improvements in terms of precision with contributions like CornerNet [89] and CenterNet [90] that focused on predicting the corners or centers of objects in the image.

With Cascade RPN [91] Vu *et al.* proposed a method that uses a cascade of RPNs to improve the quality of the region proposals. This was done by using the output of the previous RPN to refine the anchors for the next RPN. This method as of now is considered state-of-the-art in terms of region proposal based two-stage object detection. Later contributions included things like orienting the anchors to improve IoU performance. This was done by Cheng *et al.* in 2022 to be applied to aerial images. [92]

6.5. Application

Today RPNs can be applied to many different fields due to many contributions over the years. Eventhough two-stage detectors are not as fast as single-stage detectors, they are still widely employed for tasks that require precision and offer complex scenarios to detect. Tasks like autonomous driving are an interesting topic since they require precision but at the same time need to be fast. In a comparision by Carranza-García *et al.* that evaluates detection performance for autonomous driving tasks, Faster R-CNN networks were identified to acheive the best speed/accuracy tradeoff as well as being more reliable in the minority class detection. [93] Despite the difficulties of transferring RPNs to different domains, some interesting applications beyond 2D-image detection have been proposed. For example, Nabati and Qi proposed a method to detect objects in Radar and Lidar data. [94]

Another interesting application, where two-stage detectors are used, is in the field of medical imaging. For example, in a paper from 2020 [95] early signs of lung cancer could be detected with a two-stage approach. Liu *et al.* employed a RPN to successfully detect punctate white matter lesions in brain MRI images of preterm infants. [96]. In a paper from 2021 Ma and Luo managed to achieve about 90% accuracy in detecting bone fractures on X-ray images using a Faster R-CNN network. [97]

RPNs also play a key role in the field of intelligent surveillance systems. In a recent paper from 2024 Ding *et al.* proposed a method to detect specialty vehicles in airport surveillance systems. [98] Because of the irregular shapes of these vehicles, the need for high precision and pixel wise masking from Mask R-CNN [86] was solved. A paper from 2022 proposed a method to detect suspicious objects in video surveillance systems to prevent terrorist attacks. [99]

References

[1] F. Emmert-Streib *et al.*, "An introductory review of deep learning for prediction models with big data," *Frontiers in Artificial Intelligence*, vol. 3, p. 4, 2020.

[2] Y. LeCun *et al.*, "Generalization and network design strategies," *Connectionism in perspective*, vol. 19, no. 143-155, p. 18, 1989.

[3] D. Bhatt *et al.*, "Cnn variants for computer vision: History, architecture, application, challenges and future scope," *Electronics*, vol. 10, no. 20, p. 2470, 2021.

[4] L. Lu *et al.*, "Integrating local cnn and global cnn for script identification in natural scene images," *IEEE Access*, vol. 7, pp. 52 669–52 679, 2019.

[5] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[6] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[7] 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.

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

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

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

[11] K. He *et al.*, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[12] C. Szegedy *et al.*, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.4842.

[13] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 6105–6114.

[14] J. Wang *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 10, pp. 3349–3364, 2020.

[15] Z. Dai *et al.*, "Coatnet: Marrying convolution and attention for all data sizes," *Advances in neural information processing systems*, vol. 34, pp. 3965–3977, 2021.

[16] Z. Liu *et al.*, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[17] 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.

[18] W. L. Perera, *Objects recognition by cnn for the vision of smart car navigation*, Oct. 2019. DOI: 10.13140/RG.2.2.14857.60005.

[19] A. v. d. Oord *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[20] F.-A. Croitoru *et al.*, "Diffusion models in vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 10 850–10 869, Sep. 2023, ISSN: 1939-3539. DOI: 10.1109/tpami.2023.3261988. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2023.3261988.

[21] A. Nichol and P. Dhariwal, *Improved denoising diffusion probabilistic models*, 2021. arXiv: 2102.09672 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2102.09672.

[22] A. Amini and A. Amini, *Deep learning limitations and new frontiers*, Jan. 2023. [Online]. Available: http://introtodeeplearning.com/2023/slides/6S191_MIT_DeepLearning_L7.pdf.

[23] W. Nie *et al.*, *Diffusion models for adversarial purification*, 2022. arXiv: 2205.07460 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2205.07460.

[24] J. Ho *et al.*, *Denoising diffusion probabilistic models*, 2020. arXiv: 2006.11239 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2006.11239.

[25] A. Jain, *Diffusion*, Apr. 2024. [Online]. Available: https://docs.google.com/presentation/d/1bSqPL2hJEj_atOT1vLFEJipnOQbOiow70wjE3Pz2AO4/edit.

[26] P. Dhariwal and A. Nichol, *Diffusion models beat gans on image synthesis*, 2021. arXiv: 2105.05233 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2105.05233.

[27] R. Rombach *et al.*, *High-resolution image synthesis with latent diffusion models*, 2022. arXiv: 2112.10752 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2112.10752.

[28] A. Amini and A. Amini, *Deep generative modeling*, Jan. 2023. [Online]. Available: http://introtodeeplearning.com/2023/slides/6S191_MIT_DeepLearning_L4.pdf.

[29] J. Sohl-Dickstein *et al.*, *Deep unsupervised learning using nonequilibrium thermodynamics*, 2015. arXiv: 1503.03585 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1503.03585.

[30] A. Ramesh *et al.*, *Hierarchical text-conditional image generation with clip latents*, 2022. arXiv: 2204.06125 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2204.06125.

[31] C. Saharia *et al.*, *Photorealistic text-to-image diffusion models with deep language understanding*, 2022. arXiv: 2205.11487 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2205.11487.

[32] Midjourney, *Midjourney*. [Online]. Available: https://www.midjourney.com/home.

[33] J. Ho *et al.*, *Imagen video: High definition video generation with diffusion models*, 2022. arXiv: 2210.02303 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2210.02303.

[34] OpenAI, *Creating video from text*. [Online]. Available: https://openai.com/index/sora.

[35] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*, JMLR Workshop and Conference Proceedings, 2012, pp. 37–49.

[36] V. Badrinarayanan *et al.*, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[37] M. Tschannen *et al.*, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069*, 2018.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[40] Z. Pan *et al.*, "Recent progress on generative adversarial networks (gans): A survey," *IEEE access*, vol. 7, pp. 36 322–36 333, 2019.

[41] S. Bond-Taylor *et al.*, "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 11, pp. 7327–7347, 2021.

[42] D. E. Rumelhart *et al.*, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[43] P. Vincent *et al.*, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[44] I. Tolstikhin *et al.*, "Wasserstein auto-encoders," *arXiv preprint arXiv:1711.01558*, 2017.

[45] J. Zhao *et al.*, "A combination method of stacked autoencoder and 3d deep residual network for hyperspectral image classification," *International Journal of Applied Earth Observation and Geoinformation*, vol. 102, p. 102 459, 2021.

[46] S. Röhrl *et al.*, "Autoencoder features for differentiation of leukocytes based on digital holographic microscopy (dhm)," in *Computer Aided Systems Theory–EUROCAST 2019: 17th International Conference, Las Palmas de Gran Canaria, Spain, February 17–22, 2019, Revised Selected Papers, Part II 17*, Springer, 2020, pp. 281–288.

[47] A. Vaswani *et al.*, *Attention is all you need*, Jun. 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762 (visited on 06/15/2024).

[48] A. De *et al.*, *Attention, please! a survey of neural attention models in deep learning a preprint*, 2021. [Online]. Available: https://arxiv.org/pdf/2103.16775 (visited on 07/07/2024).

[49] J. Devlin *et al.*, *Bert: Pre-training of deep bidirectional transformers for language understanding*, May 2019. [Online]. Available: https://arxiv.org/pdf/1810.04805 (visited on 07/06/2024).

[50] T. Brown *et al.*, *Language models are few-shot learners*, Jul. 2020. [Online]. Available: https://arxiv.org/pdf/2005.14165 (visited on 07/06/2024).

[51] H. Larochelle and G. E. Hinton, *Learning to combine foveal glimpses with a third-order boltzmann machine*, Neural Information Processing Systems, 2010. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2010/hash/677e09724f0e2df9b6c000b75b5da10d-Abstract.html (visited on 06/13/2024).

[52] D. Bahdanau *et al.*, *Neural machine translation by jointly learning to align and translate*, arXiv.org, 2014. [Online]. Available: https://arxiv.org/abs/1409.0473 (visited on 06/22/2024).

[53] A. Dosovitskiy *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, Jun. 2021. [Online]. Available: https://arxiv.org/pdf/2010.11929 (visited on 06/22/2024).

[54] A. Galassi *et al.*, "Attention in natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 1–18, 2020. DOI: 10.1109/tnnls.2020.3019893.

[55] N. Carion *et al.*, *End-to-end object detection with transformers*, Jun. 2024. [Online]. Available: https://arxiv.org/pdf/2005.12872.

[56] R. Girdhar *et al.*, *Video action transformer network*, Dec. 2018. [Online]. Available: https://arxiv.org/pdf/1812.02707 (visited on 06/23/2024).

[57] G. Brauwers and F. Frasincar, "A general survey on attention mechanisms in deep learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021. DOI: 10.1109/tkde.2021.3126456.

[58] 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.

[59] B. Ghojogh *et al.*, "Fisher discriminant triplet and contrastive losses for training siamese networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9206833.

[60] R. Hadsell *et al.*, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[61] Y. Li *et al.*, "A survey on siamese network: Methodologies, applications, and opportunities," *IEEE Transactions on Artificial Intelligence*, vol. 3, no. 6, pp. 994–1014, 2022. DOI: 10.1109/TAI.2022.3207112.

[62] E. Hoffer and N. Ailon, *Deep metric learning using triplet network*, 2018. arXiv: 1412.6622 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6622.

[63] F. Schroff *et al.*, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015. arXiv: 1503.03832. [Online]. Available: http://arxiv.org/abs/1503.03832.

[64] B. Ríos *et al.*, "Deep learning for face recognition on mobile devices," *IET Biometrics*, vol. 9, Feb. 2020. DOI: 10.1049/iet-bmt.2019.0093.

[65] H. O. Song *et al.*, *Deep metric learning via lifted structured feature embedding*, 2015. arXiv: 1511.06452 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1511.06452.

[66] A. Fedele *et al.*, "Explaining siamese networks in few-shot learning," *Machine Learning*, pp. 1–38, Apr. 2024. DOI: 10.1007/s10994-024-06529-8.

[67] P. Baldi and Y. Chauvin, "Neural networks for fingerprint recognition," *Neural Computation*, vol. 5, no. 3, pp. 402–418, May 1993, ISSN: 0899-7667. DOI: 10.1162/neco.1993.5.3.402. eprint: https://direct.mit.edu/neco/article-pdf/5/3/402/812570/neco.1993.5.3.402.pdf. [Online]. Available: https://doi.org/10.1162/neco.1993.5.3.402.

[68] G. Koch *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, Lille, vol. 2, 2015.

[69] B. Li *et al.*, "High performance visual tracking with siamese region proposal network," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8971–8980. DOI: 10.1109/CVPR.2018.00935.

[70] A. He *et al.*, "A twofold siamese network for real-time object tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[71] X. Dong and J. Shen, "Triplet loss in siamese network for object tracking," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.

[72] J. Shen *et al.*, "Visual object tracking by hierarchical attention siamese network," *IEEE Transactions on Cybernetics*, vol. 50, no. 7, pp. 3068–3080, 2020. DOI: 10.1109/TCYB.2019.2936503.

[73] X. Chen and K. He, *Exploring simple siamese representation learning*, 2020. arXiv: 2011.10566 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2011.10566.

[74] H. Zhou *et al.*, *Masnet:improve performance of siamese networks with mutual-attention for remote sensing change detection tasks*, 2022. arXiv: 2206.02331 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2206.02331.

[75] S. Zagoruyko and N. Komodakis, *Learning to compare image patches via convolutional neural networks*, 2015. arXiv: 1504.03641 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1504.03641.

[76] X. Wang and A. Gupta, *Unsupervised learning of visual representations using videos*, 2015. arXiv: 1505.00687 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.00687.

[77] S. Ren *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017, ISSN: 2160-9292. DOI: 10.1109/tpami.2016.2577031.

[78] J. R. R. Uijlings *et al.*, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Apr. 2013, ISSN: 1573-1405. DOI: 10.1007/s11263-013-0620-5.

[79] R. Girshick, *Fast r-cnn*, 2015. DOI: 10.48550/ARXIV.1504.08083.

[80] Torch Contributors, *Faster r-cnn - torchvision main documentation*, pytorch.org. [Online]. Available: https://pytorch.org/vision/main/models/faster_rcnn.html (visited on 07/15/2024).

[81] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Sep. 2014. DOI: 10.48550/ARXIV.1409.1556. arXiv: 1409.1556 [cs.CV].

[82] K. He *et al.*, "Deep residual learning for image recognition," Dec. 2015. DOI: 10.48550/ARXIV.1512.03385. arXiv: 1512.03385 [cs.CV].

[83] B. Li *et al.*, "High performance visual tracking with siamese region proposal network," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018. DOI: 10.1109/cvpr.2018.00935.

[84] T.-Y. Lin *et al.*, *Feature pyramid networks for object detection*, 2016. DOI: 10.48550/ARXIV.1612.03144.

[85] J. Redmon *et al.*, *You only look once: Unified, real-time object detection*, 2015. DOI: 10.48550/ARXIV.1506.02640.

[86] K. He *et al.*, *Mask r-cnn*, 2017. DOI: 10.48550/ARXIV.1703.06870.

[87] L. Castrejon *et al.*, "Annotating object instances with a polygon-rnn," 2017. DOI: 10.48550/ARXIV.1704.05548.

[88] J. Wang *et al.*, *Region proposal by guided anchoring*, 2019. DOI: 10.48550/ARXIV.1901.03278.

[89] H. Law and J. Deng, *Cornernet: Detecting objects as paired keypoints*, 2018. DOI: 10.48550/ARXIV.1808.01244.

[90] K. Duan *et al.*, *Centernet: Keypoint triplets for object detection*, 2019. DOI: 10.48550/ARXIV.1904.08189.

[91] T. Vu *et al.*, *Cascade rpn: Delving into high-quality region proposal network with adaptive convolution*, 2019. DOI: 10.48550/ARXIV.1909.06720.

[92] G. Cheng *et al.*, "Anchor-free oriented proposal generator for object detection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–11, 2022, ISSN: 1558-0644. DOI: 10.1109/tgrs.2022.3183022.

[93] M. Carranza-García *et al.*, "On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data," *Remote Sensing*, vol. 13, no. 1, p. 89, Dec. 2020, ISSN: 2072-4292. DOI: 10.3390/rs13010089.

[94] R. Nabati and H. Qi, "Rrpn: Radar region proposal network for object detection in autonomous vehicles," in *2019 IEEE International Conference on Image Processing (ICIP)*, IEEE, Sep. 2019. DOI: 10.1109/icip.2019.8803392.

[95] H. Cao *et al.*, "A two-stage convolutional neural networks for lung nodule detection," *IEEE Journal of Biomedical and Health Informatics*, pp. 1–1, 2020, ISSN: 2168-2208. DOI: 10.1109/jbhi.2019.2963720.

[96] Y. Liu *et al.*, "Refined segmentation r-cnn: A two-stage convolutional neural network for punctate white matter lesion segmentation in preterm infants," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Springer International Publishing, 2019, pp. 193–201, ISBN: 9783030322489. DOI: 10.1007/978-3-030-32248-9_22.

[97] Y. Ma and Y. Luo, "Bone fracture detection through the two-stage system of crack-sensitive convolutional neural network," *Informatics in Medicine Unlocked*, vol. 22, p. 100 452, 2021, ISSN: 2352-9148. DOI: 10.1016/j.imu.2020.100452.

[98] M. Ding *et al.*, "Two-stage framework for specialty vehicles detection and classification: Toward intelligent visual surveillance of airport surface," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 60, no. 2, pp. 1912–1923, Apr. 2024, ISSN: 2371-9877. DOI: 10.1109/taes.2023.3342797.

[99] Z. Wen *et al.*, "Ai-based w-band suspicious object detection system for moving persons: Two-stage walkthrough configuration and recognition optimization," *Wireless Communications and Mobile Computing*, vol. 2022, K. Lakshmanna, Ed., pp. 1–16, Jun. 2022, ISSN: 1530-8669. DOI: 10.1155/2022/3690403.