UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Pavel Chizhov

# Self-Supervised Image Denoising Using Transformers

Master's Thesis (30 ECTS)

Supervisor: Mikhail Papkov, MSc

Tartu 2023

## Self-Supervised Image Denoising Using Transformers

**Abstract:**
Self-supervised image denoising is a computer vision task that implies image noise removal without access to clean data. This problem is critical to many domains, such as medical imaging, where clean images are often unobtainable. The absence of the true signal determines the main challenge, therefore self-supervised image denoising demands for specific model engineering. Modern solutions for this task mainly rely on convolutional neural networks, and there has been very limited research on the applications of rapidly developing transformer models to this task. To close this research gap, we adopt a ready-made transformer-based image restoration model for self-supervised image denoising and compare it to the convolutional counterparts. Apart from that, we propose a novel transformer autoencoder architecture, which not only shows more stable performance regardless of the noise type but also is the first model to prove the concept of zero-convolution end-to-end network for self-supervised image denoising. This work highlights the advantages and limitations of transformers in self-supervised image denoising and provides a conceptual foundation for further development in the field.

**Keywords:**
Image denoising, Neural networks, Self-supervised learning

**CERCS:** T111 - Imaging, image processing; P176 - Artificial intelligence

## Juhendamata piltide mürapuhastus transformeritega

**Lühikokkuvõte:**
Piltide mürapuhastus juhendamata viisil on masinnägemise ülesanne mille puhul ei ole võimalik mudeli treenimiseks kasutada müravabu pilte. Selline meetodika on olululine mitmetes valdkondades nagu näiteks meditsiiniline kujutamine, kus tihti ei ole võimalik müravabu pilte koguda. Juhendamata mürapuhastuse muudabki keeruliseks müravabade pildite puudumine ja seega vajab see mudelispetsiifilist lähenemist. Kaasaegsed mürapuhastus lahendused põhinevad peamiselt sidumnärvivõrkudel ja väga vähe on uuritud kuidas transformerid selle ülesandega hakkama saavad. Sellest lähtuvalt kohandatakse magistritöös olemasolevaid pildi taastamise transformereid juhendamata mürapuhastuse ülesande jaoks ja võrreldakse neid vastavate sidumnärvivõrkudega. Peale selle kirjeldatakse töös uudset autokodeerijaga transformeri arhitektuuri, mis hoolimata müratüübist saavutab stabiilsemaid tulemusi kui muud mudelid. Samuti on see esimene 'end-to-end' juhendamata mürapuhastuse närvivõrk, mis ei kasuta ühtegi sidumoperatsiooni. Käesolev magistritöö toob välja transformerite eelised ja puudused mürapuhastuse ülesande kontekstis ja loob kontseptuaalse aluse valdkonna edasiseks arenguks.

**Võtmesõnad:**
Piltide mürapuhastus, Närvivõrgud, Juhendamata õppimine

**CERCS:** T111 - Pilditehnika; P176 - Tehisintellekt

# Contents

# 1 Introduction

Image denoising is a computer vision task that seeks to reconstruct the underlying signal from the images corrupted by noise. Typical methods for this task include classical image restoration algorithms (*e.g.*, filtering) or supervised convolutional neural networks that are trained on pairs of noisy examples and the respective clean ground truth images. It is also possible to approach this task in a self-supervised manner when all the data present is noisy, and neither clear images nor alternative noise copies are available. Despite being challenging, self-supervised image denoising is highly valuable for a vast variety of domains because clear images are hardly obtainable in real-world tasks, including medical image restoration.

State-of-the-art solutions for self-supervised image denoising [49, 54, 58, 41, 39, 52, 66] typically use convolutional neural networks as backbones because of their invaluable benefits for image processing [27]. At the same time, since the first introduction of the Transformer architecture [35] for the natural language processing domain, transformers have been successfully applied to images [46] and have been developing rapidly ever since in application to a multitude of computer vision tasks, taking the lead from convolutional models. However, transformers have not yet found good use in self-supervised image denoising.

In this work, we aim to research the applicability of transformer-based models for the task. We build our solutions on the basis of Swin Transformer [56] — a powerful contemporary multi-purpose vision model. First, we integrate a Swin Transformer-based image restoration model (SwinIR) [55] directly into two frameworks for self-supervised image denoising — Noise2Self [39] and Noise2Self [52] and compare its performance to the conventional convolutional backbone used in the original works. Second, we develop the idea and design the first self-sufficient fully-transformer model for self-supervised denoising — SwinIA, which will work end-to-end as a simple image autoencoder without the need for external frameworks. We thoroughly test the models on a vast range of datasets with different kinds and levels of noise and reach new state-of-the-art results on several benchmarks.

In Section 2, we will give an overview of the image denoising task, describing the main kinds of noise and common classical denoising solutions. Then we will deliver the background behind the primary deep learning methods and describe how they have been applied to image denoising in supervised and self-supervised fashion. In Section 3, we will motivate the current work by reflecting on the background from the previous section. In Section 4, we will give details on the solutions we used for the current work, from the integration of SwinIR to the design of our own SwinIA model. We will also report auxiliary tools we used for development, training, and documentation. In Section 5, we will describe the experiments we conducted and their results, comparing the scores to the prior solutions and providing examples. In Section 6, we will discuss the results and speculate on the advantages and limitations of the investigated models.

# 2 Background

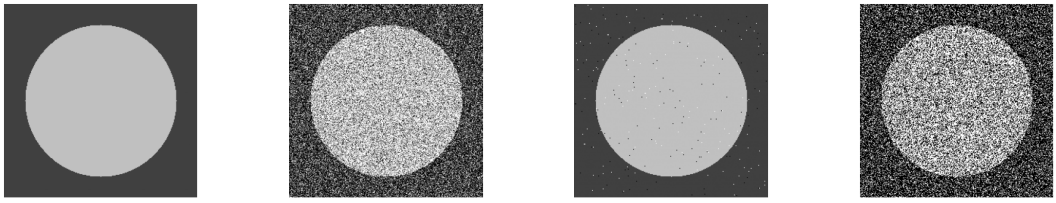
In this section, we present the background information on the current work. We start with the explanation of the denoising task, moving on to the overview of the methods that solve it: from classical unsupervised and supervised methods to deep learning solutions, including self-supervised algorithms.

## 2.1 Image denoising

Image noise is a term referred to random perturbations of signal in bitmap images. Such a defect is very common for bitmap images, it arises at any of the stages of image acquisition and processing due to apparate limitations and hindering environment conditions. Image denoising, which aims to remove noise from images, is one of the typical tasks in the computer vision domain. It is indispensable for image reconstruction and can find many usages in medical image analysis, where noise reduction might be a key factor for reliable diagnostics.

### 2.1.1 Types of noise

There are several main types of noise determined by the noise model, here we briefly describe the most common ones, give possible reasons for their occurrence, and provide examples (see Figure 1).



(a) Original image (b) Gaussian noise (c) Salt and pepper noise (d) Poisson noise

Figure 1. Illustrated types of noise on an example image.

*Gaussian noise* is a kind of additive noise that is characterized by adding random normally distributed values to pixels (eq. (1)).

$$X = S + N, N \sim \mathcal{N}(\mu, \sigma^2)\,. \tag{1}$$

Here $X$, $S$, and $N$ denote pixel value, true signal, and noise component, respectively. The noise is a random, normally distributed variable, therefore it is parametrized by mean

$\mu$ and standard deviation $\sigma$ of the Gaussian distribution. Typically, Gaussian noise is assumed to be zero-mean, thus it is centered around $0$ and the perturbations are equally likely to be positive and negative, with the noise level determined by $\sigma$.

Gaussian noise typically appears during the image acquisition procedure due to the limited capability of the sensor in troubled conditions, such as low illumination. Gaussian distribution is usually accurate in representing real-world random processes, therefore it may well describe natural physical conditions, for instance, high temperature affecting atom vibrations [25].

*Salt and pepper* noise is distinguished by a small partition of pixels of maximum (salt) and minimum (pepper) intensity randomly scattered over an image. Salt and pepper noise is parametrized by $p_w, p_b$ — the probabilities of imputing a pixel with white and black values. This kind of noise may be explained by faulty pixel-level elements of the sensor and errors in data transmission or storage when the original pixel value is simply lost and imputed [25]. Sometimes random black pixels, *i.e.*, pepper, are called Bernoulli noise with a parameter $p$, being the probability of occurrence of a black pixel.

*Poisson noise* is a type of image noise that is caused by the statistical properties of photon detection in imaging sensors. This often happens in low-light conditions due to the discretization of the electric signal. Determined by statistical fluctuations in the number of emitted photons, the noise follows Poisson distribution, commonly used to describe random processes involving counting [25].

$$X = \text{Poisson}(S \cdot \alpha) \tag{2}$$

The formula in eq. (2) represents the observed pixel intensity value in the presence of Poisson noise. $S$ is the true underlying pixel intensity value (signal), and $\alpha$ is a scaling factor that determines the overall level of noise in the image. The observed intensity $X$ is a random variable that follows a Poisson distribution with mean $\lambda = S \cdot \alpha$.

### 2.1.2 Classical denoising methods

Classical image denoising methods involve explicit filtering or smoothing of the image to reduce noise while preserving image features and details. The procedure is usually deterministic and often requires prior knowledge of the noise model and careful choice of parameters. We briefly describe the most popular classical methods of image denoising in the following paragraphs.

**Filtering methods**. In filtering methods, an image is filtered by a window of fixed size. Such a filter can be implemented with a simple aggregation function, such as averaging or median. Thus, each pixel is imputed by the value aggregated from its neighborhood. Window aggregation can be also performed in a weighted averaging way, for instance, by convolution with a Gaussian kernel, when every pixel is imputed by the normally-weighted average of the convolved window, or Bilateral filter [6], which

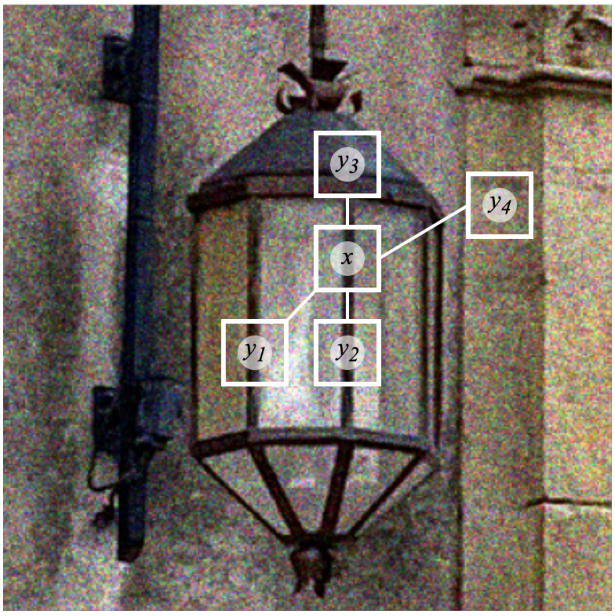



Figure 2. An example of non-local means patches comparison. Here, weights $w(x, y_1)$ and $w(x, y_2)$ will be greater than $w(x, y_3)$ and $w(x, y_4)$, because the patch centered in $x$ is more similar to the patches centered in $y_1$ and $y_2$.

assigns neighborhood weights accounting for both spatial distances and differences in color intensities. The latter is believed to preserve edges and patterns in the image. Morphological operations, *e.g.*, opening and closing, can also find their usage in denoising, as they may be considered a way to reduce single-pixel noise, such as salt and pepper.

**Filtering image properties**. Denoising can be done by filtering out certain parts of a specific image property range. For instance, total variation denoising algorithms assume that the noise is characterized by an excessive variation of the signal and thus the image is smoothed by limiting the variation in the image. Under the assumption that noise is represented by certain frequencies in the image, denoising can be done by filtering the image in the frequency domain obtained after Fourier transform [3] or excluding low-magnitude wavelet coefficients acquired by wavelet transform [22].

**Non-local means**. The Non-local means method [11] removes noise from an image by comparing similar patches of pixels instead of individual pixels. The method works by calculating a weighted average of the pixels in the image, where the weights are based on the similarity between the patches according to eq. (3).

$$\hat{u}(x) = \frac{1}{Z(x)} \sum_{y \in \Omega} u(y) w(x, y) \tag{3}$$

This formula represents the estimated denoised value of the pixel at location $x$, which is

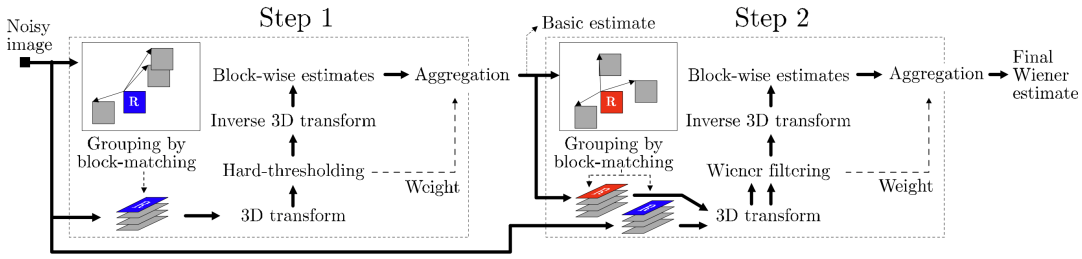



Figure 3. BM3D image denoising algorithm steps [13].

calculated by averaging the values of similar pixels within a search window $\Omega$ centered at $x$. The weights of each pixel in the averaging process are determined by the weight function $w(x, y)$, which measures the similarity between the patch centered at $x$ and the patch centered at $y$, as in Figure 2. $Z(x)$ is a normalization constant that ensures the sum of the weights is equal to one.

**Block-Matching and 3D filtering**. Block-Matching and 3D filtering (BM3D) [13] is a two-stage algorithm for image denoising that exploits the idea of redundancy in images. It is assumed that a noisy image can be decomposed into a combination of a sparse component (consisting of edges and structures) and a noise component. The complete algorithm is presented in Figure 3.

In the first stage, the noisy image is divided into overlapping blocks, and each block is compared to all the other blocks in the image to find similar blocks. The similarity is measured by calculating the Euclidean distance between the 2D block vectors formed by the pixels of the blocks. The similar blocks are then grouped together to form a 3D group, where the third dimension is the block index. A 3D transform (e.g., discrete cosine transform) is applied to each group, and the transform coefficients are thresholded to obtain a sparse representation of the group. The thresholded coefficients are then inverse-transformed to obtain the denoised block.

In the second stage, a Wiener filter is applied to the denoised blocks to further reduce the noise. The filter parameters are estimated from the noisy image and the denoised blocks.

The final denoised image is obtained by overlapping the denoised blocks and taking the weighted average of the overlapping pixels.

### 2.1.3 Evaluation

Denoising methods can be compared by evaluation on a denoising dataset, which consists of paired noisy and clean images$(x_i, y_i)$. During the evaluation, the images produced by the method $\hat{y}_i$ are compared to the clean images $y_i$. There are three main metrics that are commonly used in image denoising benchmarks.

**MSE**. Mean squared error, or MSE, measures the average squared difference between the output of the algorithm and the ground truth clean image:

$$MSE(\hat{y}_i, y_i) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \left[\hat{y}_i(p) - y_i(p)\right], \tag{4}$$

here $\mathcal{P}$ denotes the set of pixels in the image, over which the MSE is computed, and $\hat{y}_i(p)$ and $y_i(p)$ denote the values at pixel $p$ in images $\hat{y}_i$ and $y_i$, respectively.

**PSNR**. Peak signal-to-noise ratio, or PSNR, measures the ratio between the maximal possible pixel intensity value and the intensity of noise in the image. PSNR is calculated through MSE as follows:

$$PSNR(\hat{y}_i, y_i) = 10 \cdot \log_{10} \left( \frac{M^2}{MSE(\hat{y}_i, y_i)} \right), \tag{5}$$

where $M$ denotes the maximal possible pixel intensity value. For example, for 8-bit images, $M$ equals $255$, and for images normalized to $[0, 1]$ range, $M$ equals $1$.

**SSIM**. The structural similarity index measure, or SSIM [10], was proposed as a universal measure to compare two images. It consists of several components, which include luminance, contrast, and structure, and can be reduced to the following formula:

$$SSIM(\hat{y}_i, y_i) = \frac{(2\mu_{\hat{y}_i}\mu_{y_i} + c_1)(2\sigma_{\hat{y}_i, y_i} + c_2)}{(\mu_{\hat{y}_i}^2 + \mu_{y_i}^2 + c_1)(\sigma_{\hat{y}_i}^2 + \sigma_{y_i}^2 + c_1)}, \tag{6}$$

where $\mu_{\hat{y}_i}, \mu_{y_i}$ and $\sigma_{\hat{y}_i}, \sigma_{y_i}$ denote pixel mean value and variance in images $\hat{y}_i$ and $y_i$, respectively, $\sigma_{\hat{y}_i, y_i}$ denotes covariance between the two images, and $c_1$ and $c_2$ are constants that correct the denominator factors in case they are too small.

The minimal value for all three metrics is $0$, and SSIM is the only normalized metric out of the three and always ranges from $0$ to $1$. The score is better the lower the MSE, or the greater the PSNR or SSIM. In practice, the most commonly used metrics are PSNR, a task-specific version of universal MSE, and SSIM, the metric that might give additional information about the algorithm's ability to retain structural values.

## 2.2 Deep learning

Classical methods have limited efficiency because of their explicit nature and disregard for image context. On the contrary, deep learning methods open broader opportunities for sophisticated denoising solutions. Before going into details about task-specific architectures, in this section, we describe key deep learning concepts, from a formal definition of a deep learning model and its training and optimization to the evolution of artificial neural networks relevant to our work.

### 2.2.1 Deep learning conception

A neural network is a machine learning model that was inspired by a biological neural circuit. Mathematically, it is considered a function approximator that consists of multiple linear and nonlinear transformations. Such a model can be represented by a function of the form:

$$y = f(x|\theta), \tag{7}$$

where $x$ and $y$ are model inputs and outputs, and $\theta$ is a vector of trainable parameters, or weights.

The training process for a deep learning model involves adjusting the values of the parameters $\theta$ to minimize a loss function $\mathcal{L}(f|\theta)$ that measures the difference between the model's predictions and the true outputs. To adjust the parameters $\theta$, the gradient of the loss with respect to the parameters $\nabla_\theta L(f|\theta)$ is computed using backpropagation [4], which involves recursively applying the chain rule of calculus:

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial \theta_i}, \tag{8}$$

where $\frac{\partial \mathcal{L}}{\partial y}$ is the partial derivative of the loss with respect to the output $y$, and $\frac{\partial y}{\partial \theta_i}$ is the partial derivative of the output with respect to the parameter $\theta_i$.

The gradients are then used to update the parameters $\theta$ using an optimization algorithm, usually based on the gradient descent algorithm, which is performed according to the following equation:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \mathcal{L}(f|\theta_t) \tag{9}$$

where $\theta_t$ and $\theta_{t+1}$ are the parameter values at time steps $t$ and $t+1$, respectively, $\alpha$ is the learning rate. This process is repeated for a certain number of iterations or until the loss converges to a satisfactory level.

The gradient descent algorithm is computationally inefficient because it is computed on the whole dataset, which can be very big. In the context of neural network training optimization, this version of gradient descent is called batch gradient descent. A more computationally light version of it is called stochastic gradient descent (SGD) [1]. In SGD, the gradient is computed on one example from the data during each iteration. This approach is fast and has relatively acceptable convergence, yet it may be too radical. The version of the algorithm that is most commonly used is mini-batch gradient descent. It combines both ideas and computes the gradients on small partitions of data — mini-batches. Mini-batches are usually randomly sampled from the data, and the processing of one mini-batch is usually called a step or an iteration. Normally yet not necessarily,

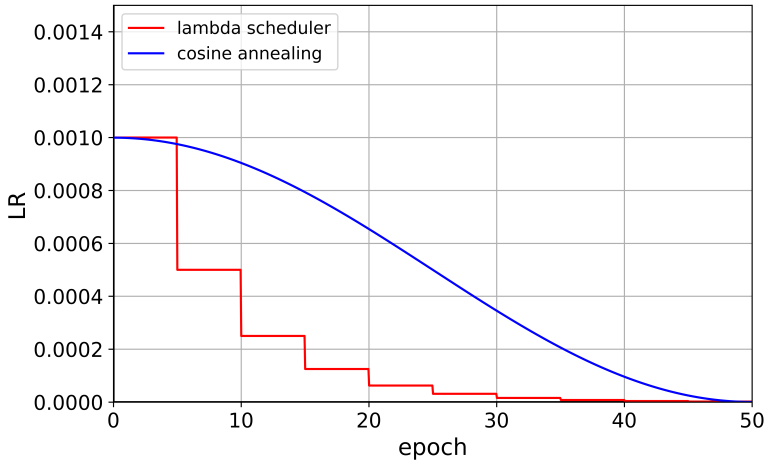



Figure 4. Learning rate (LR) scheduling examples. The training is set up for $50$ epochs, the initial learning rate is set to $10^{-3}$ for both schedulers, the final learning rate for cosine annealing is set to $10^{-6}$, and the lambda scheduler is set to decrease the learning rate by half every 5th iteration.

the whole data is shuffled and split into non-overlapping mini-batches. The processing of the dataset over a number of mini-batches is called an epoch.

There are several modifications of mini-batch gradient descent that have been developed for faster training convergence and avoiding classical caveats, such as local minimums and plateaus. For instance, Momentum [8] is a popular optimization algorithm that uses a moving average of the gradients to update the parameters, thus gaining velocity in parameter updates. Adagrad [17] adapts the learning rate for each parameter separately and adjusts the learning rate based on its past gradients. Adam [23] is an optimization algorithm that combines the ideas of momentum and Adagrad. In general practice, Adam is the most popular default choice that establishes fast and sound convergence in a lot of cases.

Optimizer is also typically combined with a learning rate scheduler. This means that the learning rate is adjusted at each epoch according to a specific schedule. For example, scheduling can be simply done by decreasing the learning rate by an equal coefficient for every certain number of iterations, this is often called lambda scheduling. Another type of scheduling is cosine annealing [32] when the learning rate is decreased by a cosine schedule from an initial value to a set final value. These types of scheduling are illustrated in Figure 4.

### 2.2.2 Multilayer perceptron

We start the overview of deep learning models with the most basic neural network — multilayer perceptron (MLP), which is also called a feed-forward or a fully-connected neural network. This architecture emerged from the concept of the perceptron [2] — one of the first prototypes of an artificial neuron. Essentially, the perceptron can be described as a thresholding function that receives a weighted sum of inputs:

$$y = f(x \cdot w + b), \tag{10}$$

here $x \cdot w$ represents a dot-product between the input vector $x$ and weight vector $w$, $b$ is the bias term. $f$ is an activation function that returns $0$ or $1$, in the simplest case it is a thresholding step-function.

This idea can be generalized to a linear layer with $m$ inputs and $n$ outputs, where each of the outputs is computed as follows:

$$\forall j = 1 \dots n : y_j = f\left(\sum_{i=1}^{m} x_i \cdot w_{ij} + b_j\right), \tag{11}$$

here $w_{ij}$ represents a weight of input $x_i$ in the computation of output $y_j$ and $b_j$ denotes the bias term for each output. Therefore, the computation of a linear layer can be optimized with vector operations:

$$y = f(xW + b), \tag{12}$$

here $W$ is a weight matrix of size $m \times n$, $x$ is a row vector of inputs of size $m$, and $y$ and $b$ are row vectors of size $n$ of outputs and biases, respectively.

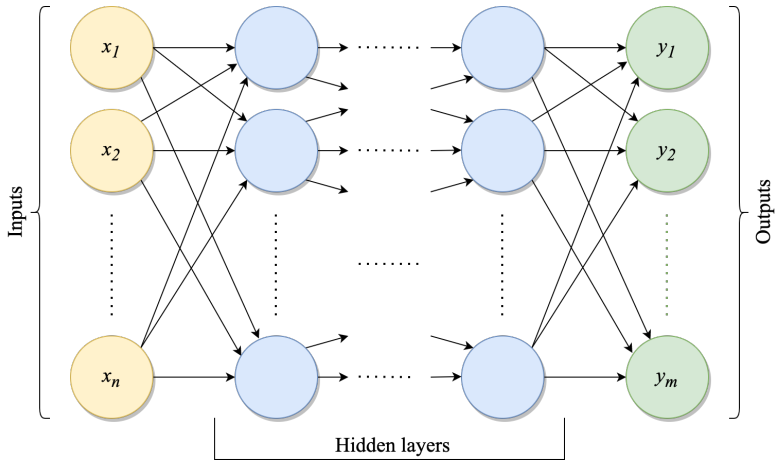



Figure 5. Multilayer perceptron.

Layers can be stacked one after another forming an MLP with several hidden layers (Figure 5). The outputs of the previous layer serve as inputs for the next layer, and each of the layers has separate parameters $W$ and $b$ and is computed as in eq. (12). Non-linear activation functions play a primary role in such stacking: without non-linearity two consequent layers would still do a linear transformation that can be represented with one transformation matrix, and, thus, one linear layer.

There are multiple types of non-linearity, in the list below and in Figure 6 we show the fundamental ones:

- Sigmoid function: $f(x) = \dfrac{1}{1 + e^{-x}}$;
- Hyperbolic tangent: $f(x) = \tanh(x)$;
- Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$;
- Gaussian Error Linear Unit (GELU) [31]: $f(x) = x\mathcal{P}(X \leq x), X \sim \mathcal{N}(0, 1)$.

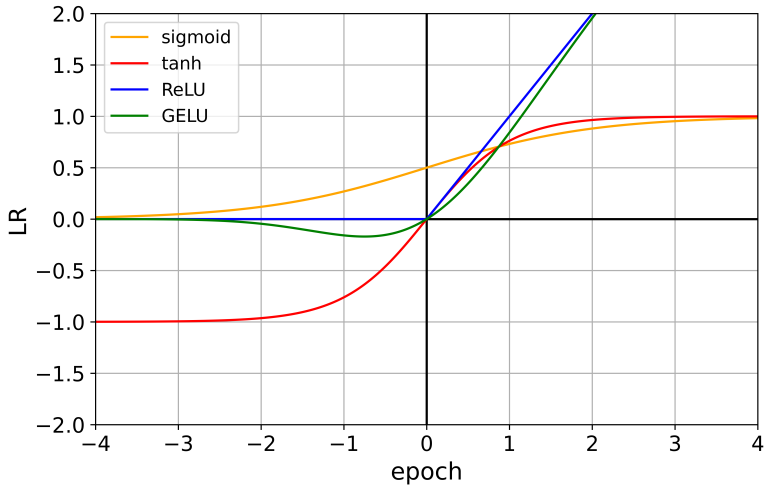



Figure 6. Activation functions.

Seemingly simple, ReLU is computationally cheap, does not suffer from vanishing gradients as sigmoid and hyperbolic tangent functions do, and generally leads to a faster convergence [21]. This made ReLU and its variants the default choice in practice. For example, GELU [31] is widely used in transformer models [40, 46], which will be discussed further.

### 2.2.3 Convolutional neural networks

When it comes to image analysis, the need for a specific solution arises. MLPs are able to process images, but they suffer from two main problems. First, a linear layer is unaware of the neighboring district around the pixel, it simply processes an image as a flattened sequence of pixel values. Besides, if we slightly shift or scale the image, this will go unnoticed by the human eye, but an MLP will consider this as a completely different image. Apart from the lack of context, MLPs suffer from image dimensionality. One linear layer has $\Omega(n^2)$ trainable parameters, where $n$ is the input size. This is critical for images because the number of pixels also grows quadratically with the increase in image resolution.

A convolutional neural network (CNN) [5] is a highly efficient architecture for image analysis. The core idea of CNNs is in the usage of convolutional filters, which are capable of image feature extraction, such as edge detection. A convolutional layer is composed of a set of convolutional filters with trainable parameters. Each convolutional layer applies a set of filters to the input image and then uses a non-linear activation function to generate a set of output image representations — feature maps. A convolutional layer is parametrized by kernel size, and optionally by padding, additional pixels that are added around the image, and stride, the length of a step that the filter takes going over the image. The latter two modifications are needed to tune the size of the output. In a CNN, a series of convolutional layers are used to detect increasingly complex patterns in an image.

In addition to convolutional layers, CNNs often include pooling layers, which are used to reduce the spatial size of the feature maps and make the network more computationally efficient. Pooling layers normally perform simple aggregation operations (such as maximization or averaging) over non-overlapping windows. CNNs can also include fully connected layers, which are used to make the final classification decision based on the extracted features.

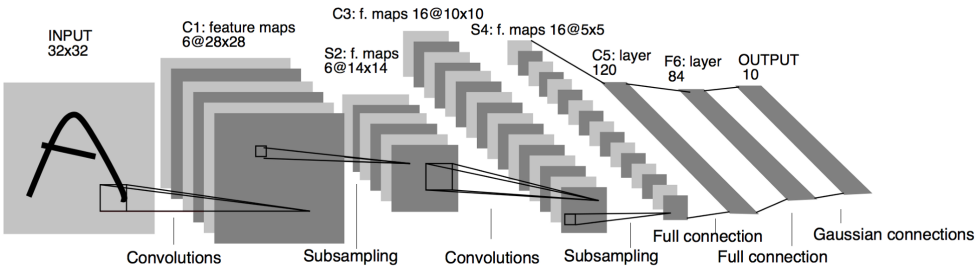



Figure 7. LeNet model architecture [5].

One of the first successful applications of CNNs was LeNet-5 architecture, introduced by Yann LeCun *et al.* in 1998 [5]. LeNet-5 was designed for handwritten digit recognition and consisted of several convolutional layers, pooling layers, and fully connected layers, the detailed architecture is illustrated in Figure 7.

| Model | #parameters | top-1 accuracy | top-5 accuracy |
|---|---|---|---|
| AlexNet [21] | 60M | 63.3% | 84.6% |
| VGG-16 [24] | 138M | 74.4% | 91.9% |
| GoogLeNet [28] | 6.8M | — | 93.3% |
| ResNet-50 [30] | 25M | 75.3% | 93.3% |
| ResNet-101 [30] | 40M | 78.3% | 94.0% |
| DenseNet-121 [33] | 5M | 75.0% | 92.3% |
| DenseNet-201 [33] | 20M | 77.4% | 93.7% |
| EfficientNet-B1 [43] | 7.8M | 78.8% | 94.4% |
| EfficientNet-B3 [43] | 12M | 81.1% | 95.5% |
| EfficientNet-B4 [43] | 19M | 82.6% | 96.3% |
| EfficientNet-B7 [43] | 66M | 84.4% | 97.1% |

Table 1. ImageNet [15] top-1 and top-5 accuracy for milestone CNN models.

CNNs have undergone a great evolution over the past two decades. This rapid evolution was driven by attempts to improve the size and efficiency of models, along with their score in the ImageNet challenge [15], a large-scale image classification competition. The key milestones of the development of CNNs are listed below, and their ImageNet scores and numbers of parameters are presented in Table 1.

**AlexNet** (2012) [21]. AlexNet was a deeper and more complex network compared to LeNet. AlexNet used ReLUs as activation functions and dropout regularization to prevent overfitting. The authors also introduced the Local Response Normalization, when the close ranges channels are normalized. Also, it was the first CNN model to utilize several GPUs.

**VGGNet** (2014) [24]. The VGGNet architecture consisted of up to 19 layers, which allowed it to learn more complex features than previous architectures. The key idea of the VGGNet was in the utilization of sequences of smaller $3 \times 3$ convolutional filters as an efficient substitute for large kernels.

**GoogLeNet** (2015) [28]. GoogLeNet used a combination of convolutional layers with different filter sizes, pooling layers, and inception modules parallelizing convolutions with different filter sizes that allowed the network to learn features at different scales.

**ResNet** (2016) [30]. ResNet introduced the concept of residual connections, also known as skip connections or shortcuts, that allowed the network to learn very deep representations by mitigating the vanishing gradient problem. The network became much deeper, with more than 50 layers (and later with more than 150 layers), this was possible because each layer of the network was given an opportunity to learn the

identity function if it was considered redundant during training. Also, ResNet utilized Batch Normalization [26] that normalizes inter-layer representations over mini-batches to prevent the distribution shifts and vanishing gradients.

**DenseNet** (2017) [33]. DenseNet introduced dense connections between convolutional layers: each layer aggregated the outputs of all its predecessors. Dense connections allowed the network to learn from all preceding layers in the network, which improved the flow of information and reduced the number of parameters.

**EfficientNet** (2019) [43]. EfficientNet was designed in a combination of model scaling, compound scaling, and neural architecture search to achieve state-of-the-art performance on image recognition tasks with a significantly smaller number of parameters.

### 2.2.4 U-Net

CNN models listed above showed competitive performance in image classification. However, such tasks as semantic segmentation and denoising required a conceptually different network that would be able to produce outputs of the same size as the input image, which is unfeasible for a plain CNN that uses downsampling and fuses local features to obtain global context.

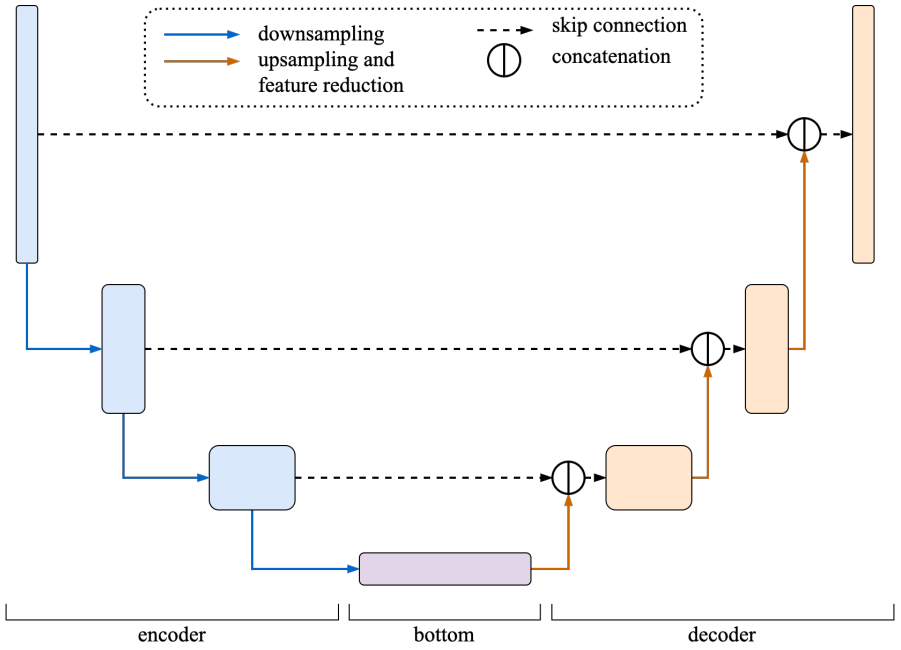



Figure 8. The U-Net architecture. Each block in the scheme represents a series of convolutional layers. The blocks' height and width indicate relative feature map resolution and the number of channels, respectively.

U-Net [27] is one of the most popular multi-purpose image analysis architectures,

which is widely used for semantic segmentation and other computer vision tasks. It consists of several convolutional layers, hierarchically stacked together into a downsampling encoder and an upsampling decoder with residual connections at each level, the architecture is illustrated in Figure 8.

The encoding part is a conventional CNN with several groups of convolutional layers and downsampling pooling layers between them. As in most CNNs, the number of channels grows with the decrease in feature map resolution. The encoder is followed by a bottom block with additional convolutional layers. In practice, any of the CNNs described in Section 2.2.3 can be used as an encoder for U-Net models.

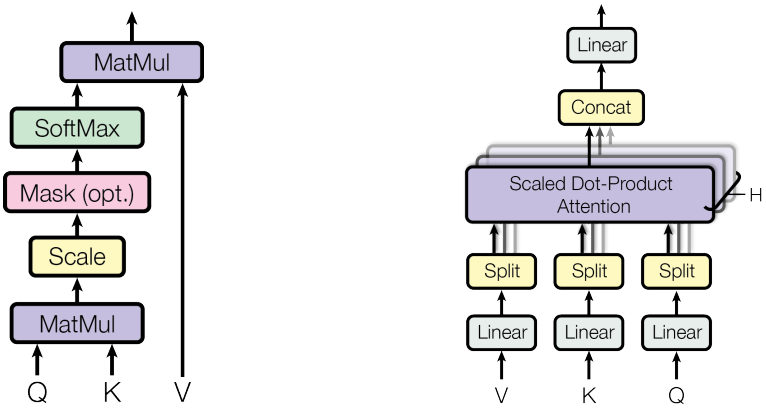
The decoding part is organized into a series of upsampling blocks that gradually increase the resolution of the feature maps, followed by a final convolutional layer that outputs the segmentation map. Each upsampling block in the decoder consists of an upsampling operation, a concatenation operation with the corresponding feature maps from the encoder, and a series of convolutional layers that process the concatenated feature maps. The upsampling operation is typically performed using transposed convolutional layers, which are convolutional layers with specific padding added around individual pixels and the whole image and a corresponding stride.

### 2.2.5 Transformers

First introduced in the seminal paper "Attention Is All You Need" by Vaswani *et al*. in 2017 [35], the Transformer architecture has become the state-of-the-art approach for many NLP tasks, including machine translation, text classification, and language modeling. Transformers are particularly well-suited for handling long-range dependencies between words in a sequence, and they do not require sequential processing, making them highly parallelizable and efficient. Their success in NLP has also led to their adoption in other areas of artificial intelligence, including computer vision and speech recognition. Before going into details about transformers in vision, which is the main focus of the current work, it is important to understand how transformers work and what makes them so efficient for processing sequential data.

The major component of the Transformer model is the multi-head self-attention (MHSA, or MSA) mechanism. It allows the model to attend to different parts of the input sequence during the encoding and decoding process.

Self-attention (Figure 9a) is performed on a sequence of tokens embedded into triplets — query $Q$, key $K$, and value $V$. A set of attention weights is computed to indicate how each word in a sequence influences all other words. These attention weights are computed by comparing each query to every key in the sequence via dot-product and generating a score that reflects the degree of similarity between each pair. The scores are additionally scaled down by a square root of dimensionality because of the exploding nature of dot-product values. The scores are then normalized using a softmax function, which produces a probability distribution over the entire sequence. Once the attention




(a) Scaled dot-product self-attention [35]. (b) Multi-head self-attention [35].

Figure 9. Multi-head self-attention mechanism visualization [35].

weights have been computed, they are used to compute a weighted sum of the input sequence, where each value is multiplied by the corresponding attention weight. All self-attention operations can be written in a single formula as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{13}$$

here $d_k$ is the dimensionality of $Q$, $K$ and $V$.

In order to diversify the mechanism, the embeddings are split channel-wise into parts (attention heads), over which the independent attention is computed, see Figure 9b. The heads are merged back after the attention computation, and a linear layer is applied to the result.

The complete original Transformer architecture (Figure 10) proposed by Vaswani *et al.* [35] consists of an encoder that extracts representations from a sequence and a decoder that generates the output sequence, word by word.

The encoder takes an input sequence of tokens and maps each token to a high-dimensional vector representation (input embedding). The encoder processes the input sequence through a sequence of transformer blocks, each of which is constituted by the MSA mechanism together with a feed-forward neural network and element-wise addition shortcuts around them, each followed by a layer normalization [29], which normalizes the output channel-wise. The output of the encoder is a set of context-aware representations of each token in the input sequence.

The decoder uses a similar attention mechanism as the encoder, but with an added mask that prevents the decoder from attending to future tokens in the output sequence. The decoder receives the encoded input sequence and an initial token as input, and generates the output sequence one token at a time. At each time step, the decoder additionally attends to the encoder's output to generate a context-aware representation of

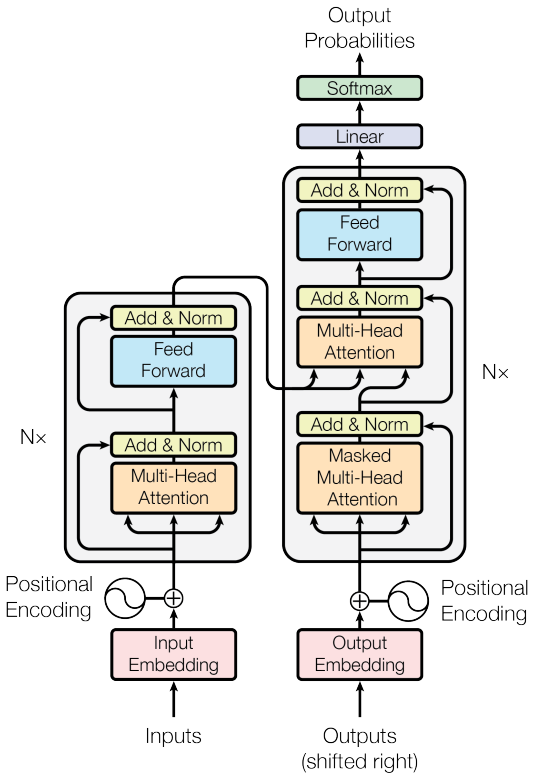



Figure 10. Transformer encoder-decoder architecture [35].

the input sequence. The decoder then uses this representation, along with the previously generated tokens, to predict the next token in the output sequence. The process is repeated until the decoder generates the entire output sequence.

Since MSA is unaware of the order of the tokens, a transformer does not have explicit representations of sequence order. Therefore, to enable the model to take into account the order of the words in the input sequence, positional encodings are added to the input and output embeddings. Each word in the input sequence is assigned a unique positional encoding vector added to its corresponding input embedding. The positional encoding vectors are designed to be sinusoidal functions of different frequencies and phases, allowing them to capture the relative position of words in the sequence. The choice of sinusoidal functions is motivated by the ability of these functions to model periodic patterns. However, sinusoidal functions are not the only option, and there are multiple ways to create positional encodings [59].

Transformer models are large and powerful, therefore they are suitable for unsupervised pre-training for whole data domains on vast corpora. One of the first transformer models that successfully incorporated unsupervised pre-training for texts was BERT (Bidirectional Encoder Representations from Transformers) [40].

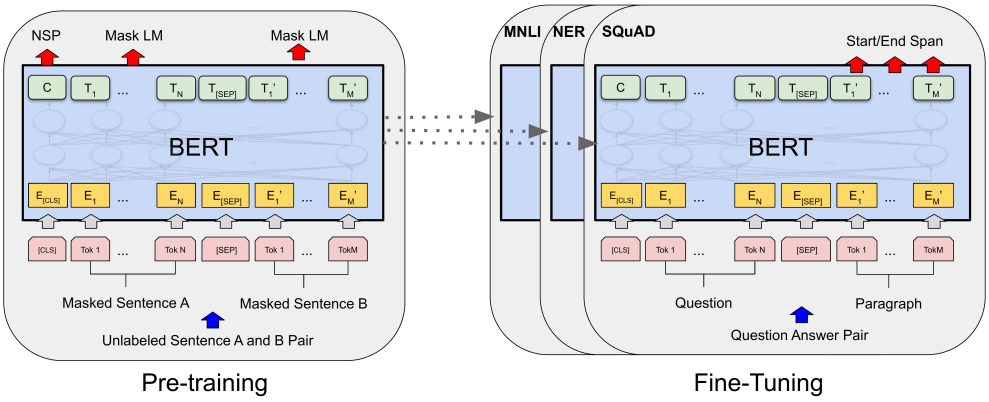
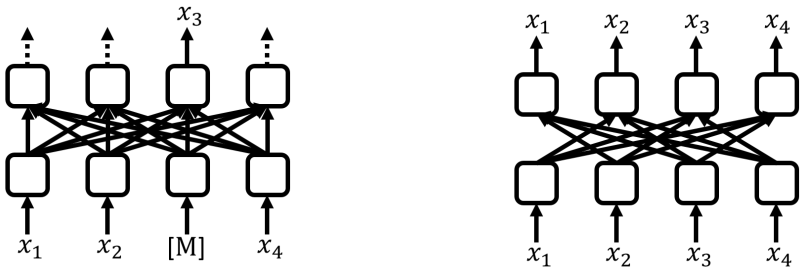



Figure 11. BERT pre-training and fine-tuning [40].




(a) BERT MLM objective [50].

(b) LAE objective [50].

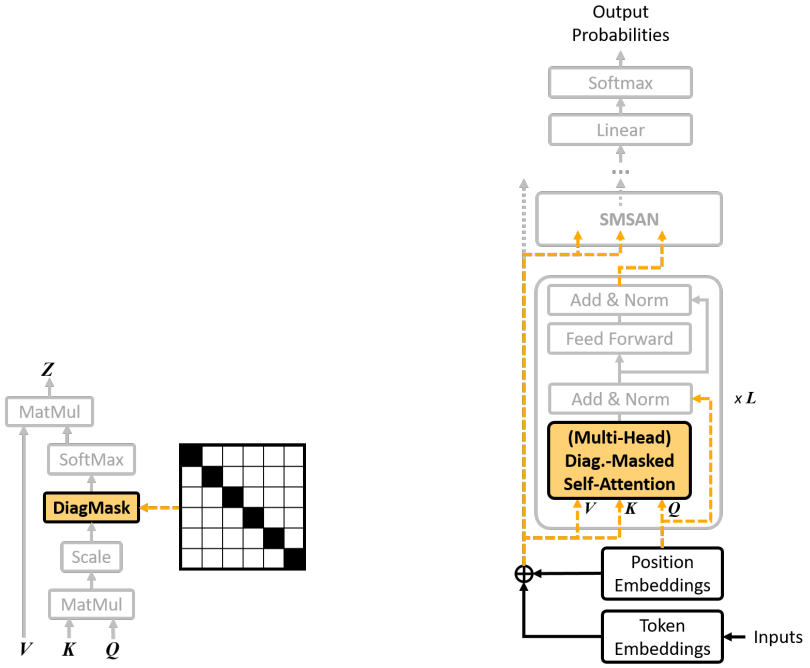
Figure 12. Building token representations in BERT and T-TA [50].

The main novelties of BERT were its pre-training objectives, such as masked language modeling (MLM) and next sentence prediction (NSP) tasks (see Figure 11), as well as the use of a transformer architecture with bidirectional attention. These techniques significantly improved the performance of the model on various NLP tasks, establishing a new state-of-the-art for several benchmarks.

In the MLM objective (Figure 12a), a certain percentage of words in the input sentence is randomly masked, and the model is trained to predict the masked words based on their context within the sentence. This encourages the model to learn representations that capture the relationship between different words and phrases in the sentence.

In the NSP objective, the model is trained to predict whether two input sentences are consecutive or not. This helps the model learn to understand the relationship between different sentences in a text and capture the context and meaning of the entire passage.

In addition, the authors introduced a novel method for incorporating positional information into the input embeddings, which is now known as learned positional embeddings. Unlike the fixed sinusoidal positional encodings used in the original Transformer paper, the positional embeddings in BERT are learned during pre-training




(a) Diagonally masked MSA [50]. (b) T-TA transformer block [50].

Figure 13. T-TA model architecture [50].

and can be fine-tuned during downstream tasks. This approach was found to be effective in capturing fine-grained positional information and further improved the performance.

BERT allowed for learning a deep understanding of natural language with later application to downstream tasks. However, if the main goal is to learn rich token representations, which might be required for such tasks as text ranking, BERT would be capable of doing this utilizing the MLM objective, but it would be too slow, as it is possible to mask out only a small partition of tokens. Transformer-based text autoencoder (T-TA) model proposed by Shin *et al.* [50] addressed this issue. The idea proposed in this work is to build token representations for each token at once using the language autoencoding (LAE) objective (compare the two objectives in Figure 12).

The LAE objective assumes that the output for each token is built upon all other tokens in the sequence. Thus, the only absent connections between inputs and outputs are for the corresponding tokens — this establishes the condition of self-unawareness [50]. Compared to BERT, this allows building the representations for all tokens at once.

Obviously, this concept demands considerable changes to the Transformer architecture (Figure 13). In order to hide the tokens' own values from their representations, an additive diagonal mask is applied to the attention matrix. This ensures that the attention weights of the tokens to their own values are zeros after the softmax function.

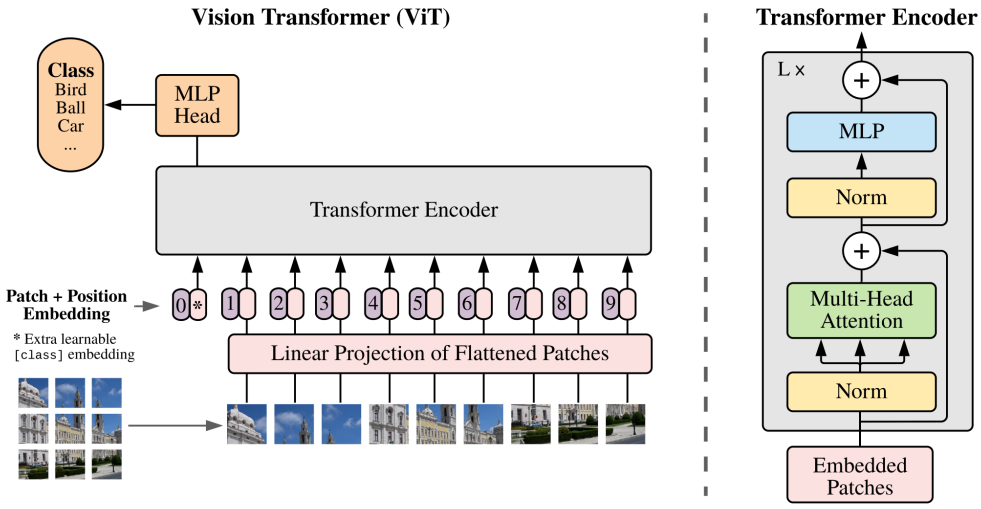



Figure 14. ViT architecture [46].

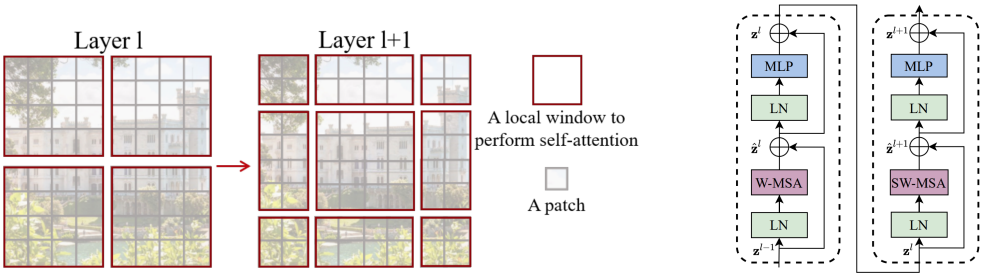
This change is enough for one transformer block. However, the power of transformers comes when they are stacked together, and the diagonal mask is simply avoidable by the permutation of neighboring tokens in two consecutive blocks. To tackle this issue, the authors proposed the mechanism of input isolation [50]. Only queries are propagated through transformer blocks, and keys and values are frozen at the input embedding stage and never change further. This ensures that context-aware representations never participate in one dot product, which mitigates the risk of learning permutation. In addition, the initial queries are built only from positional embeddings, so the queries are separated from the data.

According to the results presented in the original paper, T-TA achieved performance competitive to that of BERT with much faster training convergence.

### 2.2.6 Transformers in computer vision

Inspired by the success of transformers in NLP, researchers explored the possibility of adapting the architecture for image processing tasks. They realized that images could be represented as a sequence of patches, and each patch could be considered as a token, just like in NLP. By treating image patches as tokens, they could apply a transformer to process them and perform various computer vision tasks.

This led to the development of the Vision Transformer (ViT) [46], which replaces the convolutional layers of traditional computer vision models with self-attention layers, enabling the model to learn global relationships between different patches of an image. This approach has shown remarkable results in several image classification benchmarks and has paved the way for using transformers in other computer vision tasks.




(a) Window shifting operation [56].

(b) Two consecutive Swin Transformer blocks [56].

Figure 15. Window shifting in Swin Transformer [56].

The ViT architecture (Figure 14) consists of two main components: a patch embedding module and a transformer encoder.

The patch embedding module takes an image as input and divides it into a grid of non-overlapping patches. Each patch is then linearly projected to a low-dimensional vector using a fully connected layer and summed with trainable positional embeddings. The resulting vectors are treated as tokens and are fed into the transformer encoder.

The transformer encoder processes the sequence of vectors in a similar way to the transformers for natural language. The only difference is in normalization layers: in ViT they come before MSA and MLP layers, not after them. As usual, several transformer encoder blocks are stacked one after another. The output of the last transformer encoder layer is then passed through a linear classifier to predict the class of the input image.

ViT showed state-of-the-art performance for image classification, albeit it was not a good choice for many other computer vision tasks. The reason for this is the patch size ($16 \times 16$ by default), which limits the sequence length to acceptable numbers yet ensures that the main information is preserved. Such size is far from pixel level, which makes ViT hardly applicable to object detection, semantic segmentation, and other tasks that require local context. Even though there were examples of using ViT as an encoder for these tasks, *e.g.*, SETR [60] for semantic segmentation, it showed inferior performance compared to the models powered with a local context that came out later. One of the ways to move closer to the pixel level is to limit the attention mechanism to certain parts of the image, *i.e.*, to have local attention.

One of the most successful architectures that address the challenge of local attention is Swin Transformer, which was proposed as a universal encoder for image analysis [56]. One of the unique features of the Swin Transformer is its use of the window attention mechanism. In window attention, the input feature map is divided into non-overlapping windows of a fixed size, and self-attention is applied only within each window. Such a limitation allows for patch size reduction ($4 \times 4$ by default), but it also establishes a border between the windows, yet there are neighboring patches at the border that would benefit

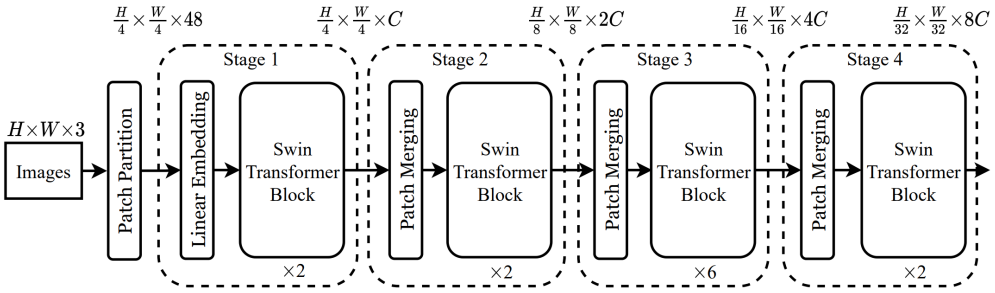
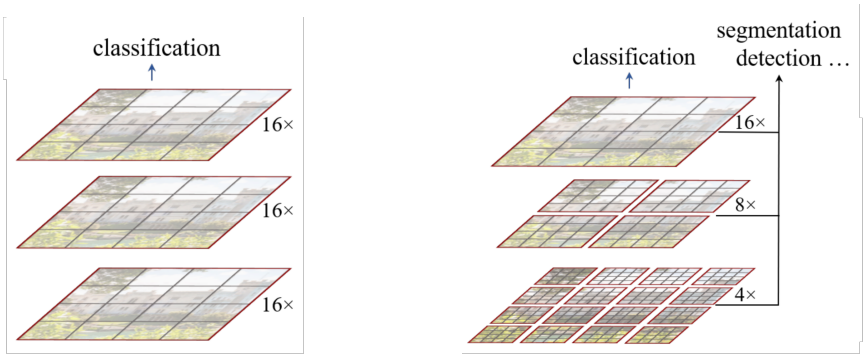



Figure 16. Swin Transformer architecture [56].



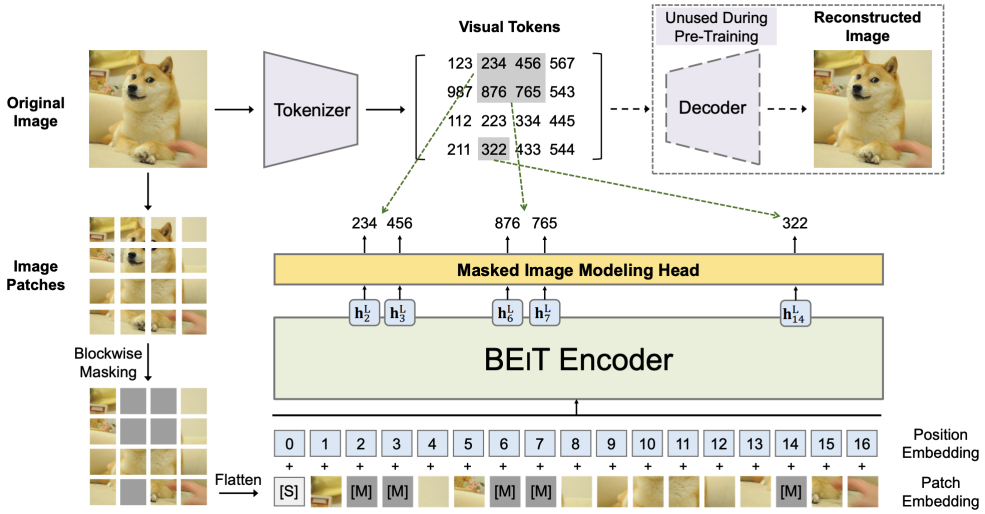

(a) ViT [46, 56]. (b) Swin Transformer [56].

Figure 17. Comparing patches and windows in Swin Transformer to ViT [56].

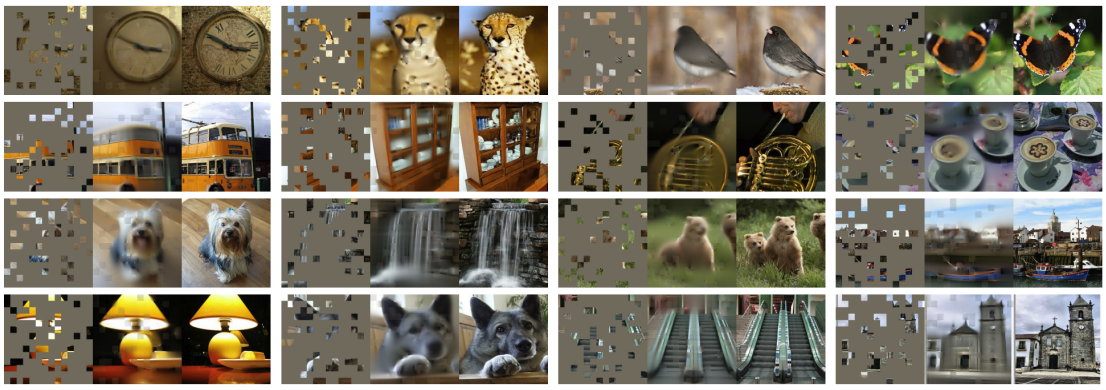
from mutual attention. To tackle this problem, Swin Transformer introduces window shifting operation (see Figure 15) — the windows are shifted in every second block by half of the window size, which allows the model to capture long-range dependencies efficiently. Such a modification is called (shifted) window multi-head self-attention, or (S)W-MSA.

Another important aspect of the Swin Transformer is its use of relative positional encoding, which allows it to capture more complex patterns in the input data. Unlike previous transformer models, which use fixed positional encodings to represent the position of each element in the input sequence, the Swin Transformer uses learned encodings that are relative to the position of other elements in the same window.

Nevertheless, the local windows might not capture all the relevant information for the task, especially when dealing with large images. To solve this problem, the Swin Transformer applies a hierarchical window attention mechanism, where the patches are flattened and further grouped into larger ones in a hierarchical manner. This allows the model to attend to both local and global information effectively. Thus, the whole architecture (Figure 16) is organized in stages, each of which consists of several Swin




(a) BEiT masked image modeling [53].



(b) Examples of masked autoencoding on ImageNet [15] validation set. In each group of three photos, there is a masked input, an output from the model, and a ground truth image [63].

Figure 18. Transformer-based masked image modeling approaches [53, 63].

Transformer blocks, and patch merging is applied between the stages to flatten and group neighboring patches. The outputs of the stages can now be used for a multitude of computer vision tasks, and the last output would still be suitable for image classification (see Figure 17b).

In efforts to adapt transformer models to vision, there have been successful attempts to organize unsupervised pre-training for images. For example, BEiT [53] adapted the masked language modeling objective of BERT to images. The idea of BEiT (Figure 18a) was to mask rectangular blocks of patches in an image and predict tokenized versions of the masked patches (visual patches). Such training does not require any ground truth data, which allows one to train a powerful large transformer encoder on a vast image

base. The pre-trained model can then be used for downstream tasks, the simplest one is image classification.

He *et al*. [63] developed the idea of BEiT and applied masked image modeling to pre-train an encoder for an image autoencoder model. They used masking more radically, masking out as much as 80% of tokens and the tokens for masking were chosen arbitrarily (Figure 18b). The results showed that even a small partition of tokens is enough for a transformer to understand an image and restore it in relatively good detail.

These works proved that transformers are capable of unsupervised pre-training on image data, however, this pre-training remains on the level of patches.

## 2.3 Denoising with deep learning

In this section, we describe the key ideas and methods that are applied for image denoising using deep learning models. We start with supervised models and move to the ideas of self-supervised image denoising, specifically focusing on blind-spot image denoising.

### 2.3.1 Supervised denoising

In a supervised setting, image denoising is mainly done with an autoencoder-like model that inputs and outputs images of the same size, *e.g.*, U-Net [27]. The simplest way to supervise such training is to use a dataset of pairs of noisy and clean images, this approach is often called Noise2Clean, or N2C (Figure 19). The loss function used during training is typically an MSE between the model's output and the ground truth clean image:

$$\mathcal{L}(f|\theta) = \mathbb{E}_{(x,y)}[MSE(f(x|\theta), y)] = \mathbb{E}_{(x,y)}\left[\frac{\|f(x|\theta) - y\|^2}{|\mathcal{P}|}\right], \tag{14}$$

here $\mathcal{P}$ denotes the set of pixels in an image, over which the MSE is computed, and $\mathbb{E}_{(x,y)}$ is the expected value over the whole data.

Theoretically, Noise2Clean training can be performed on any image dataset, in which the images are used as ground truth and their noisy versions are generated from the input. Such noise is called synthetic. Opposed to synthetic, natural noise datasets contain images with noise that occurs in real-world scenarios. One of the major difficulties with these data is that the noise there is can be comprised of different modalities and levels, which impedes the applicability of parametric models. Ground truth for such datasets is hardly obtainable and is typically present in the form of images taken with an enhanced acquisition procedure and/or better conditions. Another way of obtaining ground truth for natural noise datasets is by averaging several noisy images of the same scene.

Training on clean images is not the only option for supervised image denoising. In some cases, it is possible to obtain several noisy copies of the same scene. For instance, Noise2Noise, or N2N, is a model that is supervised by pairs of different noisy copies of the same image [38].

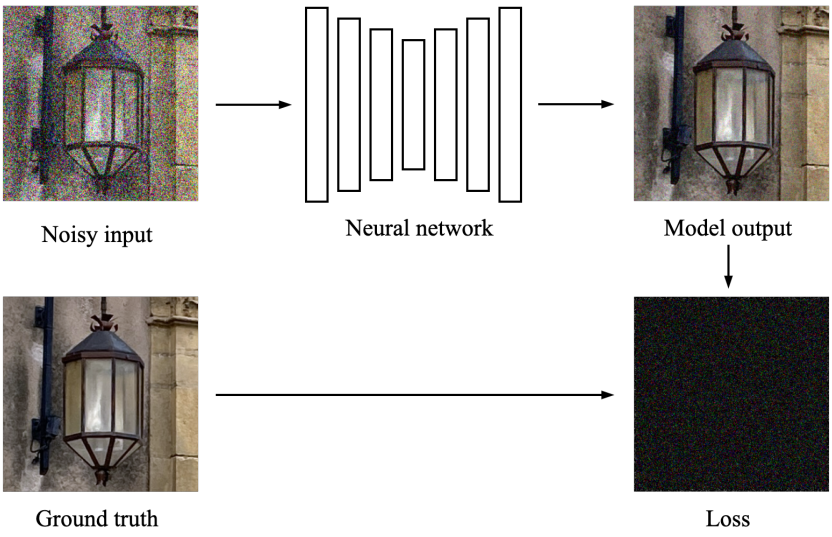



Figure 19. Noise2Clean supervised image denoising.

Noise2Noise uses the same training scheme as Noise2Clean, except for the ground truth — instead of clean images, the ground truth is represented by different corrupted images of the same scene. Thus, the dataset is comprised of pairs of noisy images $(x_i, x_i')$ under the assumption that the underlying clean signal for each pair is the same:

$$\mathbb{E}_{(x,x')}(x'|x) = y, \tag{15}$$

here $x$ and $x'$ represent input and target noisy copies, respectively, and $y$ is the unavailable true signal.

Although convolutional models are the most common choice for supervised image denoising, transformers were also successfully applied for the task: SwinIR architecture [55] recently established state-of-the-art performance on a variety of benchmarks.

SwinIR is entirely based on Swin Transformer with minor modifications. It utilizes patches of size $1 \times 1$, *i.e.*, each pixel is a patch, the authors claim that this is the only patch size suitable for image denoising. The windows, thus, have lower pixel resolution than in Swin Transformer, yet it does not affect the performance as in image denoising fine-grained details and textures matter more than the global context. Besides, the hierarchical structure of Swin Transformer is completely abandoned in SwinIR — patches are never grouped, and the output resolution is retained until the end of the network.

The complete architecture of SwinIR (Figure 20) consists of several groups of Swin Transformer blocks parametrized as described previously with a convolutional layer for additional smoothing and feature merging. There is a residual connection skipping each group and the whole set of groups. The network ends with additional convolutions that form the final output and reduce the number of channels.

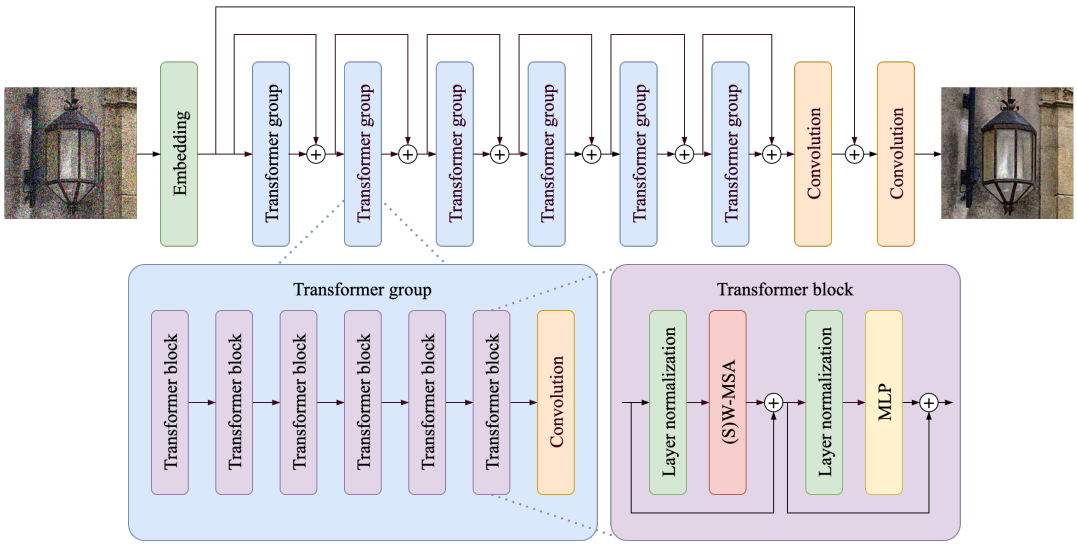



Figure 20. SwinIR model architecture [55, 69].

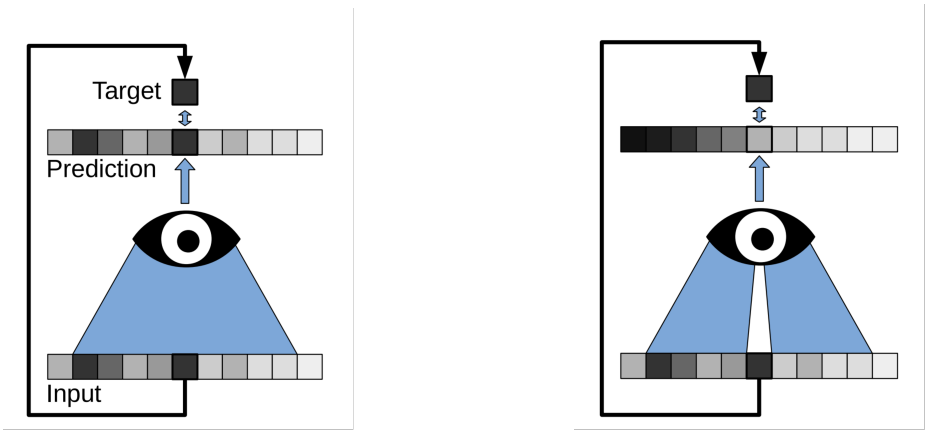
Other transformer-based architectures for supervised image restoration include Uformer [67], which is comprised of transformer blocks in a U-shaped structure and uses the mechanism of modulators to enhance the multi-scale processing, and Restormer [68], which incorporated depth-wise convolutions inside of the MSA and a gated form of MLP in the transformer.

### 2.3.2 Self-supervised image denoising

Self-supervised image denoising develops the idea of training without clean data even further. Self-supervised methods aim to reconstruct the uncorrupted signal from a noisy image only, without any additional data. This approach is invaluable for datasets with natural noise in various domains, such as medical imaging, where clear ground truth images are hardly ever available.

Self-supervised image denoising models usually require specific changes in model architecture, data processing, or training pipeline. For instance, Noisier2Noise [48] and Recorrupted2Recorrupted (R2R) [58] apply additional noise to the data during training to emulate the Noise2Noise method without additional noisy copies of the same data. These approaches show competitive performance, yet they are limited to known noise sources — the additional noise applied to the image should match the noise model that is present in the data.

There were other ways to adapt the Noise2Noise idea for noisy datasets without clean or noisy paired images. Neighbor2Neighbor [54] took a noisy image and subsampled it into two slightly different copies, treating one of them as an input and another — as a target. Self2Self [49] trained the model by applying Bernoulli dropout to noisy images




(a) Full local receptive field leads the network to learn the identity function [41].

(b) Restricted receptive field in a BSN [41].

Figure 21. Blind-spot network (BSN) design for image denoising [41].

and comparing the output to the original values of dropped pixels. At the inference stage, Self2Self aggregated the outputs of several corrupted copies for more robust results.

Another influential approach to self-supervised image denoising is to design a blind-spot network (BSN), which was contemporaneously introduced in Noise2Void [41] and Noise2Self [39]. Conceptually, a BSN is a deep learning model for image denoising that tries to predict the value of each pixel by its neighborhood, treating the pixels' own value as a blind spot. If the real pixel's value is not masked, the network will simply learn the identity function, as the loss will be computed on the input that the model can reproduce (compare the subfigures in Figure 21). Although BSP loses information about the pixel's own value, it is virtually possible to restore the pixel from its neighborhood, assuming that the pixel is similar to its neighbors and $\mathbb{E}\left[noise\right] = 0$. Therefore, such a setup works if the noise is zero-mean, independent, and identically distributed.

BSN would require a sophisticated model architecture, for example, in CNNs, it is impossible to just mask out the center in convolutional kernels: a model would simply learn pixel permutation as the filters are applied to neighboring pixels several times. To avoid complicated architecture design, the authors of Noise2Void and Noise2Self proposed using a simple U-Net as a backbone and masking a small random partition of pixels $\mathcal{J} \in \mathcal{P}$ in the image during each iteration. The loss was also modified, and an MSE would be computed only along the masked pixels:

$$\mathcal{L}(f|\theta) = \mathbb{E}_{\mathcal{J}}\mathbb{E}_{x}\left[\frac{\|f(x_{\mathcal{J}^c}|\theta)(\mathcal{J}) - x(\mathcal{J})\|^2}{|\mathcal{J}|}\right], \tag{16}$$

here $x$ is the noisy image, and $x_{\mathcal{J}^c}$ is the input image with a mask on $\mathcal{J}$, the expected value is computed over possible masks $\mathcal{J}$ and over data, $y$ is not present in the loss as

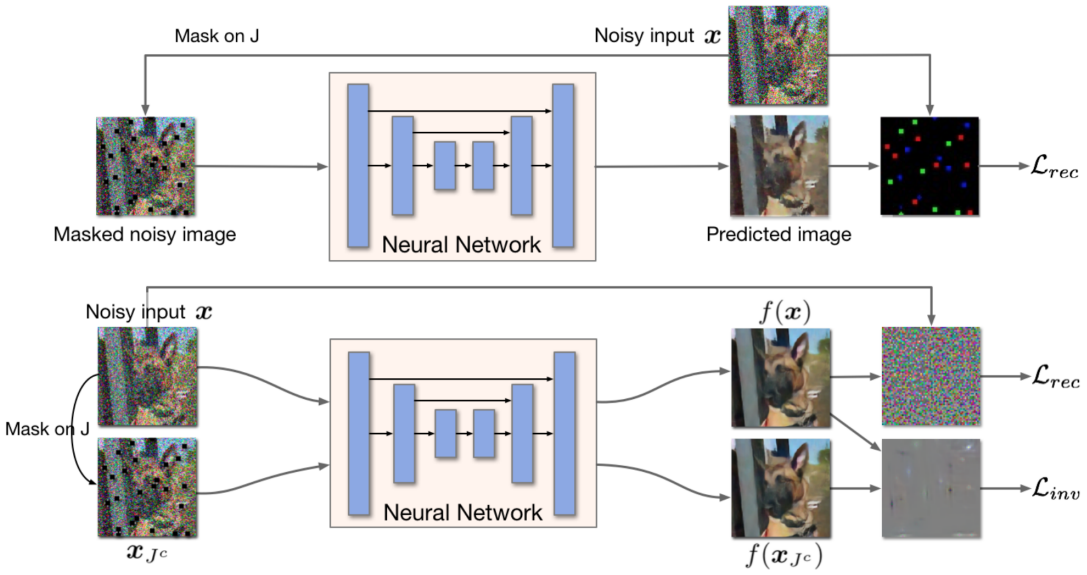



Figure 22. Noise2Void/Self (top) and Noise2Same (bottom) frameworks architecture and training settings [52].

there is no ground truth in self-supervised training. The difference between Noise2Void and Noise2Self lies in the masked pixel imputation: Noise2Void imputes the pixel with its random neighbor [41], whereas Noise2Self makes the pixel black and applies synthetic noise to it [39].

Noise2Same [52] further developed the idea of Noise2Self in order to enhance the performance of the BSN. The authors introduced an additional model pass without masking (Figure 22). The loss was also modified to a compound sum of losses:

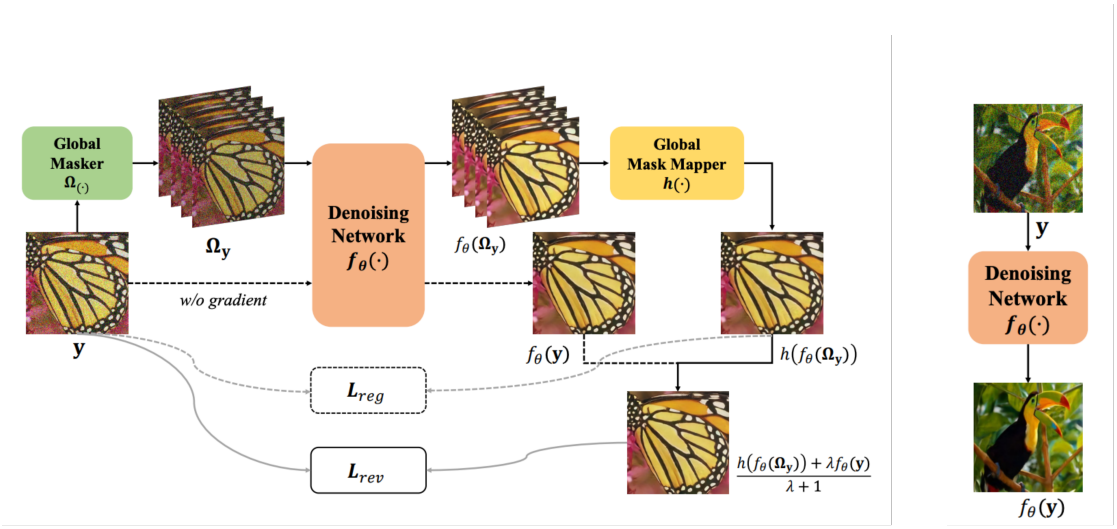
$$\mathcal{L}(f|\theta) = \lambda_{rec}\mathcal{L}_{rec}(f|\theta) + \lambda_{inv}\mathcal{L}_{inv}(f|\theta), \tag{17}$$

$$\mathcal{L}_{rec}(f|\theta) = \mathbb{E}_x\left[\frac{\|f(x|\theta) - x\|^2}{|\mathcal{P}|}\right], \tag{18}$$

$$\mathcal{L}_{inv}(f|\theta) = \mathbb{E}_{\mathcal{J}}\sqrt{\mathbb{E}_x\left[\frac{\|f(x|\theta)(\mathcal{J}) - f(x_{\mathcal{J}^c}|\theta))(\mathcal{J})\|^2}{|\mathcal{J}|}\right]}, \tag{19}$$

here $\mathcal{L}_{rec}$ is the reconstruction loss computed as an MSE between the input and the output of the unmasked run, $\mathcal{L}_{inv}$ is the invariance loss that is computed as a root of MSE between the unmasked and masked runs over the set of mask pixels, and $\lambda_{rec}$ and $\lambda_{inv}$ are hyperparameters that regulate the strength of each loss component.

Blind2Unblind [66] is another self-supervised image denoising scheme that utilized several model runs, it has recently established state-of-the-art performance. The main difference between this approach and Noise2Same is that instead of one random mask,




(a) Training phase [66].

(b) Inference phase [66].

Figure 23. Blind2Unblind image denoising [66].

Blind2Unblind creates a set of masks $\Omega_y$ with Global Masker $\Omega_{(\cdot)}$. Each mask hides a certain pixel in every $s \times s$ neighborhood. By default, $s = 2$, therefore, Blind2Unblind processes $4$ masked copies and assembles the output from them using Global Mask Mapper $h(\cdot)$. Additionally, the model processes the whole image without the gradient, and the losses are computed between the two outputs ($\mathcal{L}_{rev}$ — re-visible loss), and between the noisy input and the masked output ($\mathcal{L}_{reg}$ — regularization loss), see Figure 23.

There are several other worth-mentioning works that implemented the idea of BSN without masking. Laine *et al.* [42] implemented a BSN using four model branches for $0°, 90°, 180°, 270°$ image rotations. The convolutions in each branch have a receptive field limited to a certain direction and the final image is shifted by one pixel to hide the center pixel values. Honzátko *et al.* [47] adopted dilated convolutions to build a fully-convolutional BSN architecture. Wu *et al.* [51] also utilized dilated convolutions to create a network for unpaired noisy-clean learning — Dilated BSN, or DBSN. This method's main limitation was the restriction of the noise model, as one of the key stages of the network is noise parameter estimation.

So far, there has been limited research on the application of transformers to self-supervised image denoising. Denoising Transformer (DnT) [64], for example, uses the R2R training scheme and utilizes a transformer-based model with 2D MSA and additional convolutional feature extractors. This model, however, was proposed for single-image training and was not tested on larger denoising datasets. Another attempt to use transformers was in Context-aware Denoise Transformer (CADT) [71] combined the ideas of SwinIR [55] and Blind2Unblind [66]: they used Swin Transformer blocks,

yet at the level of patches, and Blind2Unblind masking scheme to train the denoising network. However, they claimed that the transformer alone is not enough for denoising and introduced additional convolutional context extraction. Thus, the features from Swin Transformer and convolutional feature maps are fused to produce the result.

# 3 Motivation

The solutions for self-supervised image denoising discussed above mainly concentrate on framework architecture, while using a conventional U-Net as a denoising network backbone [49, 48, 54, 41, 39, 52, 58, 42, 66]. U-Net shows good results, but it loses in comparison with state-of-the-art transformer models in supervised training. Therefore, there is room for experiments with transformer models as backbones in the self-supervised setting. We chose to integrate SwinIR [55] model into two frameworks — Noise2Self and Noise2Same, as these frameworks do not require backbone modifications and do not propagate the same input through the backbone more than two times, which is crucial for a heavy transformer model.

Another solution that is missing among the contemporary architectures is a true blind-spot network, a model that would establish the concept of reconstructing each pixel from its neighborhood at once without explicit image masking [41, 39, 52, 66] or multiple runs through the backbone [52, 66, 42]. Besides, there has been no self-sufficient transformer architecture for self-supervised image denoising — existing solutions [64, 71] still rely on convolutions. Combining the two demands, we came up with the SwinIA — Swin Transformer-based Image Autoencoder, the first fully-transformer end-to-end model for self-supervised image denoising that requires neither data manipulations nor auxiliary forward passes and can be trained with a single-component MSE loss [70].

# 4 Methods

In this section, we give details on the methods applied in the current work. First, we describe the integration of the SwinIR model into the Noise2Self and Noise2Same frameworks. Second, we propose a novel image autoencoder architecture for self-supervised image denoising based on Swin Transformer (SwinIA) [70]. Finally, we document the technologies used in our work.

## 4.1 Self-supervised image denoising with SwinIR

We integrated SwinIR into the framework that supports both Noise2Self and Noise2Same modes (Figure 24). We took the original SwinIR with six groups with six Swin Transformer blocks and a $3 \times 3$ convolution in each group [55]. We removed inner input normalization as the normalization is already done in the framework and discarded the shortcut connection from input to output as it was not motivated in the original paper and affected the performance in self-supervised training [69].

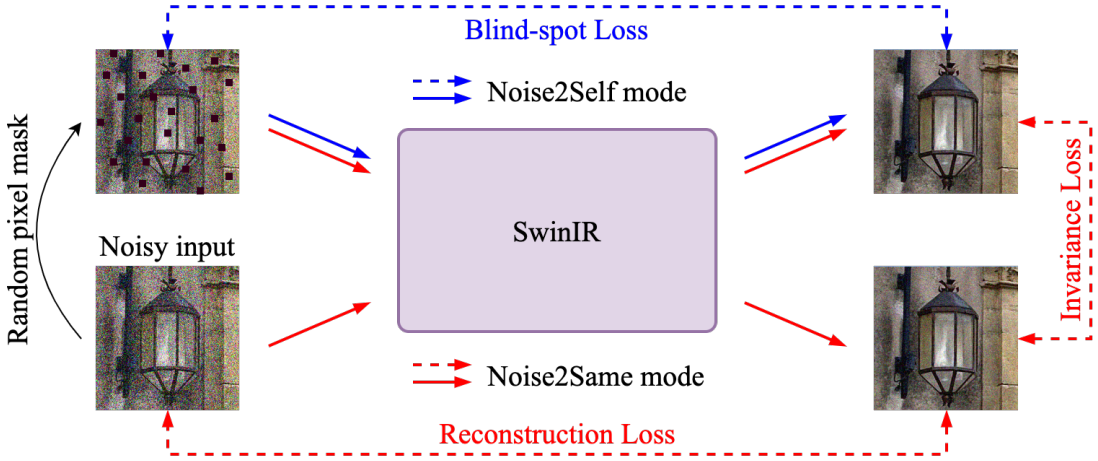



Figure 24. Proposed framework architecture, it combines Noise2Self [39] (blue flow) and Noise2Same [52] (red flow) and uses SwinIR as a backbone [69].

Following SwinIR [55], we use embedding dimensionality of $96$ with $6$ attention heads. To avoid unnecessary padding, we use windows of size $8 \times 8$ instead of $7 \times 7$, as we train on images of size $64 \times 64$.

To obtain a random pixel mask for both model modes, we mask random $0.5\%$ of pixels and impute them with zero-mean Gaussian noise with $\sigma = 0.2$. For Noise2Same mode, we use $\lambda_{inv} = 2$ to equalize loss components.

## 4.2 SwinIA

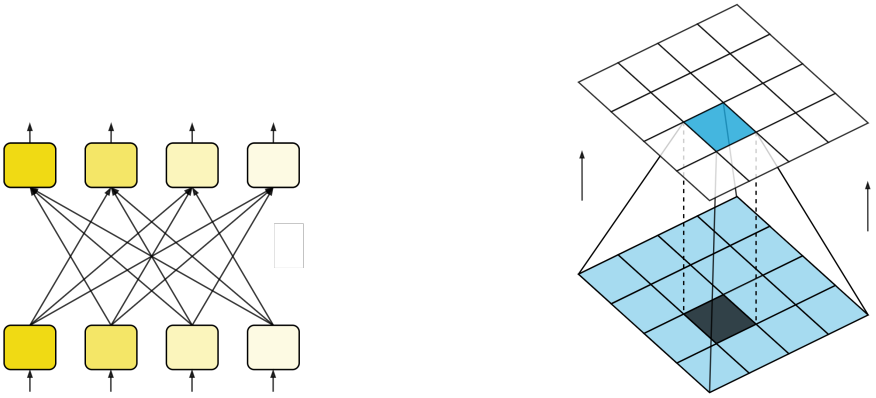
In this section, we describe the proposed SwinIA architecture. We start with the model design, formulating the main idea and the architecture requirements. Then we describe the architecture bottom-up.

### 4.2.1 Design

The idea of SwinIA is to create a true BSN which would be trained by a simple MSE loss between input and output:

$$\mathcal{L}(f|\theta) = \mathbb{E}_x \|f(x|\theta) - x\|^2. \tag{20}$$

The model should not require any additional forward passes or any data manipulations. Any classical denoising backbone would simply learn an identity function in such a setting, so it is the architecture that should establish the blind-spot property.



(a) Language autoencoding with self-unaware tokens [50, 70].

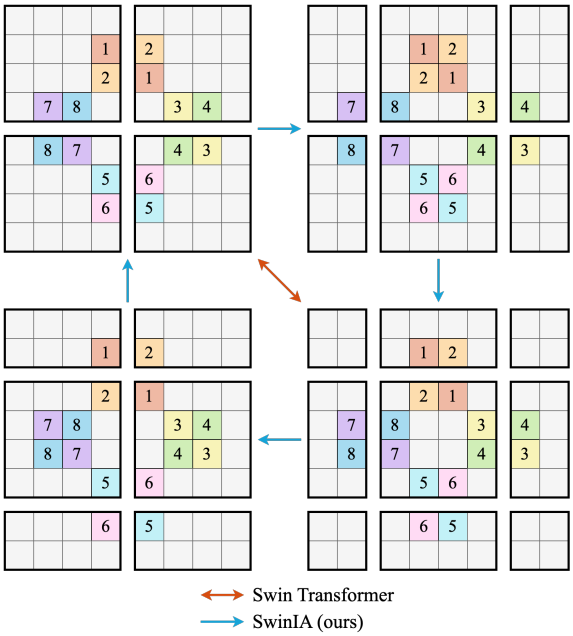
(b) Image autoencoding with self-unaware patches [70].

Figure 25. Self-unaware autoencoding in text and images [70].

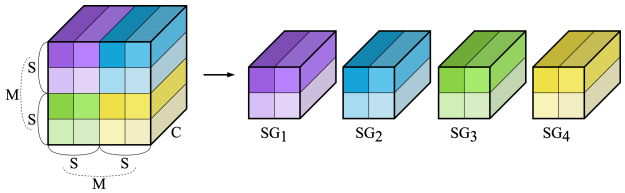
We adapt the idea of language autoencoding objective with self-unaware tokens from T-TA [50]. The transformer architecture that we propose is trained with self-unaware patches when each patch does not attend to its own value (compare subfigures in Figure 25), which establishes the blind-spot property on the level of patches. We also follow the idea of SwinIR [55] to use window self-attention with $1 \times 1$ patches to ensure the blind-spot property on the pixel level.

To implement the idea of a true fully-transformer BSN described above, we formulate the following list of architecture requirements.

**Self-unawareness**. This is the main requirement to establish the blind-spot property: in every block of the network, none of the pixels should be able to access either their own values or the values of the pixels that accessed its value.




(a) Swin Transformer and SwinIA window shifting approaches. Pairs of adjacent pixels that never participate in the same self-attention in Swin are enumerated $1-8$ [70].




(b) Shuffle group partition example for shuffle $S = 2$. A window of size $M \times M = 2 \times 2$ shuffle groups is partitioned into $4$ shuffle groups [70].

Figure 26. SwinIA design decisions [70].

**Pixel level**. Image denoising requires pixel-level interactions, so the transformer blocks should operate on pixel-size patches.

**Continuous field of view**. The window shifting scheme of Swin Transformer leaves several pairs of adjacent pixels unaware of each other under the restriction of input isolation (Figure 26a). Taking this into account, we demand that neither of the neighboring pixel interactions can be neglected.

**Long-range interactions**. Input isolation restriction cancels the receptive field growth essential to Swin Transformer. Therefore, there must be a mechanism to establish a broader context for each pixel.

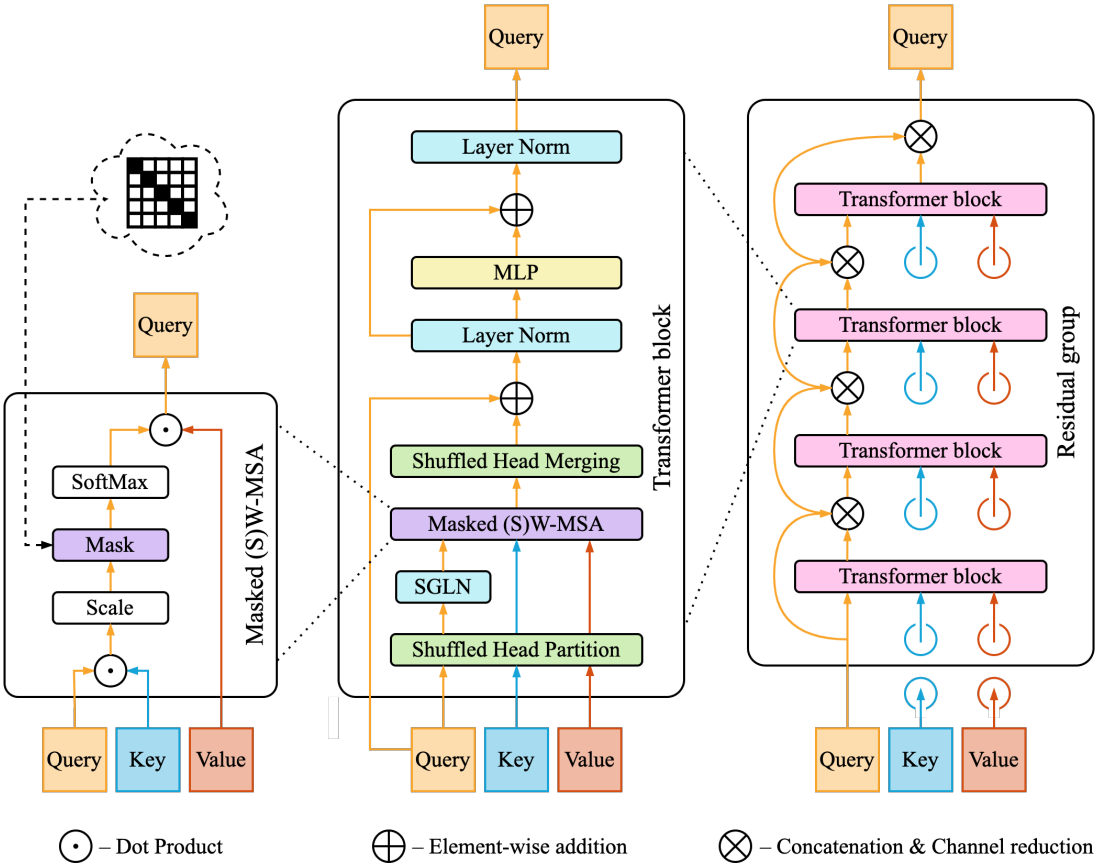



Figure 27. SwinIA MSA, transformer block, and residual group architecture [70].

**Encoder-decoder structure**. The output of the network should be an image, therefore different encoder levels should be properly decoded to ensure the restoration of the local context.

### 4.2.2 Architecture

Here we give a detailed description of SwinIA architecture in a bottom-up approach [70]. We will explain key architectural solutions referring to the requirements formulated above.

**Transformer block**. SwinIA transformer block (Figure 27) follows the idea of T-TA as a diagonal mask is applied to self-attention and only queries are propagated through the transformer [50]. As in Swin Transformer, SwinIA uses window MSA [56]. Combined, these ideas satisfy the requirements of *self-unawareness* and *pixel level*.

Another distinguishing feature of SwinIA is that it operates at the level of shuffle groups — square groups of pixels of size $S \times S$. Each shuffle group is treated as a single

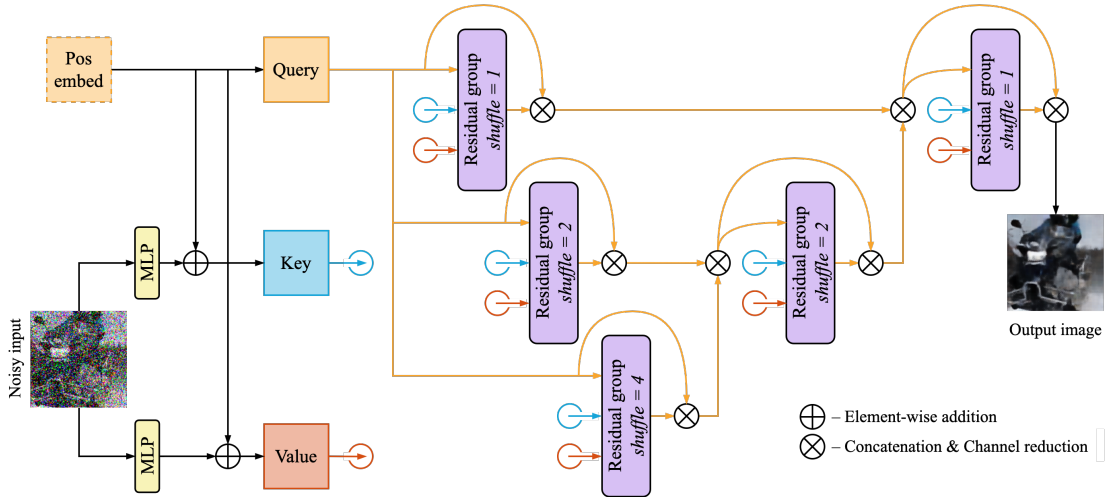



Figure 28. SwinIA model architecture [70].

token for self-attention. By adjusting $S$, we can make SwinIA transformer blocks operate at the pixel level ($S = 1$) or in the broader context ($S > 1$). This allows us to enlarge the attention windows to gain *long-range interactions*. The window size $M$ now corresponds to a square window of size $M \times M$ shuffle groups.

Following the omnipresent practice in vision transformers, we normalize the tokens before self-attention (shuffle group layer normalization — SGLN) and MLP blocks. There is an additional layer normalization after the MLP, which improved the performance of the model (see ablation study in Section 5.5).

In our experiments, we use the embeddings of size 144, windows of size $8 \times 8$ for each shuffle group size, and 16 attention heads in every transformer block.

**Residual group**. SwinIA residual group (Figure 27) consists of four transformer blocks. This number is determined by the number of shifts done in the cyclic shifting scheme that we propose (see Figure 26a). The window is first shifted by half of the window size to the right, then to the bottom, then to the left, and finally back up. Such a shifting scheme satisfies the requirement of *continuous field of view*, which was unobtainable with the shifting scheme of Swin Transformer.

To achieve *self-unawareness* through input isolation, the keys and values that come as inputs to the residual group are never changed and only query is propagated through the transformer blocks. Additionally, there is a shortcut around each transformer block in the group. The shortcut is performed as two operations, concatenation and channel reduction back to the original number of channels, as concatenation doubles the channel dimension.

**Full model**. The complete SwinIA model (Figure 28) has an *encoder-decoder structure* that forms a U-shape. This allows to extract features on different levels and use them to form the final pixel-level output. Each encoder and decoder block is a residual

group described above. Before the encoder, an input image is separately embedded into key and value, while query is formed from the trainable positional embeddings matrix. These embeddings are also added to the keys and values.

The encoder consists of three blocks with shuffles $S = 1, 2$ and $4$. The inputs are not passed down through the encoder and are the same for each encoder block. Otherwise, with increasing $S$, shuffle groups aware of their neighborhood will assemble into bigger tokens, inside of which exists a certain mutual correspondence between the parts, which contradicts the requirement of *self-unawareness*.

The decoder consists of two blocks with shuffles $S = 2$ and $1$ that correspond to the similar encoder blocks, the block with $S = 4$ is treated as a bottom block. Unlike the encoder, the decoder sequentially propagates tensors through blocks, because $S$ decreases by this flow and *self-unawareness* retains. The input of each decoder block is comprised of the output from the previous block and from the corresponding encoder block, these outputs are combined via concatenation and channel reduction.

Additionally, there is a skip connection around each residual group with concatenation and channel reduction.

## 4.3 Technologies used

In this section, we describe the technologies and instruments we used to implement the models, organize training and logging, and create visualizations and documentation.

### 4.3.1 Implementation and training

All models were implemented in Python 3.8.3 with PyTorch 1.12.1 [34] with `einops` [65] library for most operations of tensor reshaping and dimensions permutation. The training was accelerated with NVIDIA V100 32GB GPUs and NVIDIA A100 80GB GPUs (driver version: 470.57.02, CUDA version: 11.4).

For training logging and visualization, we used Weights & Biases (W&B) software [45]. Through W&B we also visualized training and validation examples and compared the models.

### 4.3.2 Visualization and documentation

We visualized the denoising examples using `matplotlib` [14] library. More complex result visualizations and model architecture diagrams were drawn using `diagrams.net` [57].

The text of this thesis was written using Overleaf editor [20]. We used the assistance of ChatGPT [61] when writing the background section (Section 2). This language model was used to enrich the text variety and to make the overview paragraphs more readable. To correct grammatical errors and misused vocabulary, we used Grammarly [16].

# 5 Experiments and results

In this section, we describe the experiments that we performed. We start with the general training and testing settings (Section 5.1). Then we continue to details and results of the experiments that we completed. We conducted a series of experiments on datasets that consisted of clean images, to which we applied synthetic noise for training while keeping clean images unavailable for the model. These results are presented in Section 5.2. We also used prepared mixed noise datasets from Noise2Self [39] and Noise2Same [52] papers, these results are presented in Section 5.3. In addition, we tested the model on natural noise data (Section 5.4). In Section 5.5, we describe an ablation study on different training settings and model parameters. Finally, in Section 5.6, we report failed experiments that are worth mentioning.

When presenting the results, we will highlight wherever possible separate groups for supervised methods (N2C [27] and N2N [38]) and methods that achieve the blind-spot property via backbone architecture. The latter is done for a fair comparison of our SwinIA model with other true-blind spot methods.

## 5.1 Training settings

We follow Noise2Same [52] and Blind2Unblind [66] in most of the training settings. We train models for $50000$ iterations (there are several exceptions, they are mentioned in the experiments section) with Adam optimizer [23]. During each iteration, a model processes $32$ random images of size $64 \times 64$ cropped from the training dataset. Each crop is scaled to the $[0, 1]$ range and independently standardized to $\mu = 0$ and $\sigma = 1$, except for the experiments with Poisson noise. Then, several random augmentations are applied, including rotation and vertical and horizontal flipping.

In Noise2Self and Noise2Same models training (both with U-Net and SwinIR inside), we use lambda learning rate scheduling with an initial learning rate of $4 \cdot 10^{-4}$ and learning rate decay by half every 500th iteration. In SwinIA training, we use cosine annealing learning rate scheduling with an initial learning rate of $10^{-3}$ and final learning rate of $10^{-6}$, as it is recommended for transformers in image restoration [62]. We also apply padding to images so that they become divisible by $8$ (the window size) in SwinIR, and by $32$ (the window size multiplied by the maximum shuffle group size) in SwinIA.

## 5.2 Synthetic noise

Following Blind2Unblind [66], we conduct a series of experiments on a range of clean image datasets by applying noise to them as a special augmentation for training. The clean data is still unavailable for the training. In the following subsections, we describe the experiments on grayscale and sRGB data.

### 5.2.1 Grayscale synthetic noise

We use BSD400 [36] dataset for grayscale denoising training. It consists of $400$ grayscale natural images of different resolutions. For testing, we use BSD68 [9] and Set12 datasets, consisting of $68$ and $12$ images, respectively. As in Blind2Unblind [66], we do several testing iterations for each dataset, providing the average value as a result. We repeat BSD68 $4$ times and Set12 $20$ times, obtaining $272 + 240 = 512$ images in total. We apply zero-mean Gaussian noise with $\sigma = 15, 25, 50$, the experimental results are presented in Table 2, and example visualizations are in Figure 29, Figure 32, and Figure 33.

| Noise Type | Method | Network | BSD68 | Set12 |
|---|---|---|---|---|
| Gaussian $\sigma = 15$ | N2C [27] | U-Net [27] | 31.58/0.889 | 32.60/0.899 |
| | R2R [58] | U-Net [27] | **31.54/0.885** | **32.54/0.897** |
| | Noise2Self† [39] | U-Net [27] | 30.63/0.843 | 29.88/0.840 |
| | Noise2Same† [52] | U-Net [27] | 30.85/0.850 | 30.02/0.849 |
| | Noise2Same† [52] | SwinIR [55] | 30.80/0.848 | 30.11/0.850 |
| | Blind2Unblind [66] | U-Net [27] | 31.44/0.884 | 32.46/**0.897** |
| | Ours† | SwinIA [70] | 31.07/0.856 | 30.37/0.857 |
| Gaussian $\sigma = 25$ | N2C [27] | U-Net [27] | 29.02/0.822 | 30.07/0.852 |
| | R2R [58] | U-Net [27] | 28.99/0.818 | 30.06/0.851 |
| | Noise2Self† [39] | U-Net [27] | 28.88/0.789 | 28.37 0.799 |
| | Noise2Same† [52] | U-Net [27] | 29.13/0.800 | 28.54/0.814 |
| | Noise2Same† [52] | SwinIR [55] | 29.06/0.798 | 28.57/0.814 |
| | Blind2Unblind [66] | U-Net [27] | 28.99/**0.820** | **30.09/0.854** |
| | Ours† | SwinIA [70] | **29.17**/0.801 | 28.72/0.817 |
| Gaussian $\sigma = 50$ | N2C [27] | U-Net [27] | 26.08/0.715 | 26.88/0.777 |
| | R2R [58] | U-Net [27] | 26.02/0.705 | 26.86/0.771 |
| | Noise2Self† [39] | U-Net [27] | 26.19/0.664 | 25.56/0.692 |
| | Noise2Same† [52] | U-Net [27] | **26.75**/0.714 | 26.13/0.744 |
| | Noise2Same† [52] | SwinIR [55] | 26.60/0.708 | 25.93/0.734 |
| | Blind2Unblind [66] | U-Net [27] | 26.09/**0.715** | **26.91/0.776** |
| | Ours† | SwinIA | 26.61/0.706 | 26.03/0.736 |

Table 2. Denoising results on grayscale images with synthetic Gaussian noise. The results are presented as average PSNR (in dB)/SSIM scores. In each series of experiments, the best score is highlighted in **bold**, and the second-best score is underlined. † denotes our implementations of the models that we evaluated ourselves [70].

As part of Noise2Same, SwinIR shows relatively comparable performance to that of U-Net, overcoming Blind2Unblind on BSD68 with strong noise. SwinIA mostly outperforms Noise2Self/Noise2Same models, beating the state-of-the-art Blind2Unblind model on BSD68 with higher noise levels.

| Noise Type | Method | Network | KODAK | BSD300 | SET14 |
|---|---|---|---|---|---|
| Gaussian $\sigma = 25$ | N2C [27] | U-Net [27] | 32.43/0.884 | 31.05/0.879 | 31.40/0.869 |
| | N2N [38] | U-Net [27] | 32.41/0.884 | 31.04/0.878 | 31.37/0.868 |
| | CBM3D [12] | — | 31.87/0.868 | 30.48/0.861 | 30.88/0.854 |
| | Self2Self [49] | U-Net [27] | 31.28/0.864 | 29.86/0.849 | 30.08/0.839 |
| | N2V [41] | U-Net [27] | 30.32/0.821 | 29.34/0.824 | 28.84/0.802 |
| | Noisier2Noise [48] | U-Net [27] | 30.70/0.845 | 29.32/0.833 | 29.64/0.832 |
| | R2R [58] | U-Net [27] | 32.25/0.880 | 30.91/0.872 | 31.32/0.865 |
| | NBR2NBR [54] | U-Net [27] | 32.08/0.879 | 30.79/0.873 | 31.09/0.864 |
| | Blind2Undlind [66] | U-Net [27] | 32.27/0.880 | 30.87/0.872 | 31.27/0.864 |
| | Noise2Same† [52] | U-Net [27] | 30.77/0.841 | 29.50/0.834 | 29.53/0.827 |
| | Noise2Same† [52] | SwinIR [55] | 30.91/0.839 | 29.63/0.828 | 29.81/0.822 |
| | Laine19-mu [42] | U-Net [27] | 30.62/0.840 | 28.62/0.803 | 29.93/0.830 |
| | Laine19-pme [42] | U-Net [27] | 32.40/**0.883** | 30.99/**0.877** | **31.36/0.866** |
| | DBSN [51] | DBSN [51] | 31.64/0.856 | 29.80/0.839 | 30.63/0.846 |
| | Honzatko20 [47] | U-Net [27] | **32.45**/ — | **31.02**/ — | 31.25/ — |
| | Ours† | SwinIA [70] | 30.12/0.819 | 28.40/0.789 | 29.54/0.814 |
| Gaussian $\sigma \in [5, 50]$ | N2C [27] | U-Net [27] | 32.51/0.875 | 31.07/0.866 | 31.41/0.863 |
| | N2N [38] | U-Net [27] | 32.50/0.875 | 31.07/0.866 | 31.39/0.863 |
| | CBM3D [12] | — | 32.02/0.860 | 30.56/0.847 | 30.94/0.849 |
| | Self2Self [49] | U-Net [27] | 31.37/0.860 | 29.87/0.841 | 29.97/0.849 |
| | N2V [41] | U-Net [27] | 30.44/0.806 | 29.31/0.801 | 29.01/0.792 |
| | R2R [58] | U-Net [27] | 31.50/0.850 | 30.56/0.855 | 30.84/0.850 |
| | NBR2NBR [54] | U-Net [27] | 32.10/0.870 | 30.73/**0.861** | 31.05/**0.858** |
| | Blind2Undlind [66] | U-Net [27] | 32.34/**0.872** | 30.86/**0.861** | 31.14/0.857 |
| | Noise2Same† [52] | U-Net [27] | 30.78/0.835 | 29.49/0.823 | 29.34/0.817 |
| | Noise2Same† [52] | SwinIR [55] | 31.28/0.840 | 29.85/0.826 | 29.66/0.813 |
| | Laine19-mu [42] | U-Net [27] | 30.52/0.833 | 28.43/0.794 | 29.71/0.822 |
| | Laine19-pme [42] | U-Net [27] | 32.40/0.870 | 30.95/**0.861** | 31.21/0.855 |
| | DBSN [51] | DBSN [51] | 30.38/0.826 | 28.34/0.788 | 29.49/0.814 |
| | Honzatko20 [47] | U-Net [27] | **32.46**/ — | **31.18**/ — | **31.25**/ — |
| | Ours† | SwinIA [70] | 30.30/0.820 | 28.40/0.785 | 29.49/0.809 |

Table 3. Denoising results on sRGB images with synthetic Gaussian noise. The results are presented as average PSNR (in dB)/SSIM scores. In each series of experiments, the best score is highlighted in **bold**, and the second-best score is underlined. † denotes our implementations of the models that we evaluated ourselves [70].

### 5.2.2 sRGB synthetic noise

For color denoising training, we use $44328$ sRGB images from ImageNet validation set [15] with resolutions varying from $256 \times 256$ to $512 \times 512$. For testing, we use KODAK [7], BSD300 [9], and Set14 [18] datasets. The datasets consist of $24$, $100$, and $14$ images, and we repeat them for evaluation $10, 3$, and $20$ times, respectively. Thus,

| Noise Type | Method | Network | KODAK | BSD300 | SET14 |
|---|---|---|---|---|---|
| Poisson $\lambda = 30$ | N2C [27] | U-Net [27] | 31.78/0.876 | 30.36/0.868 | 30.57/0.858 |
| | N2N [38] | U-Net [27] | 31.77/0.876 | 30.35/0.868 | 30.56/0.857 |
| | Anscombe [19] | — | 30.53/0.856 | 29.18/0.842 | 29.44/0.837 |
| | Self2Self [49] | U-Net [27] | 30.31/0.857 | 28.93/0.840 | 28.84/0.839 |
| | N2V [41] | U-Net [27] | 28.90/0.788 | 28.46/0.798 | 27.73/0.774 |
| | R2R [58] | U-Net [27] | 30.50/0.801 | 29.47/0.811 | 29.53/0.801 |
| | NBR2NBR [54] | U-Net [27] | 31.44/0.870 | 30.10/<u>0.863</u> | 30.29/<u>0.853</u> |
| | Blind2Undlind [66] | U-Net [27] | <u>31.64</u>/<u>0.871</u> | **30.25**/0.862 | <u>30.46</u>/0.852 |
| | Noise2Same† [52] | U-Net [27] | 27.73/0.747 | 26.69/0.714 | 26.78/0.735 |
| | Noise2Same† [52] | SwinIR [55] | 27.70/0.735 | 26.72/0.713 | 26.71/0.728 |
| | Laine19-mu [42] | U-Net [27] | 30.19/0.833 | 28.25/0.794 | 29.35/0.820 |
| | Laine19-pme [42] | U-Net [27] | **31.67/0.874** | **30.25/0.866** | **30.47/0.855** |
| | DBSN [51] | DBSN [51] | 30.07/0.827 | 28.19/0.790 | 29.16/0.814 |
| | Honzatko20 [47] | U-Net [27] | **31.67**/ — | **30.25**/ — | 30.14/ — |
| | Ours† | SwinIA [70] | 29.51/0.805 | 27.92/0.775 | 28.74/0.799 |
| Poisson $\lambda \in [5, 50]$ | N2C [27] | U-Net [27] | 31.19/0.861 | 29.79/0.848 | 30.02/0.842 |
| | N2N [38] | U-Net [27] | 31.18/0.861 | 29.78/0.848 | 30.02/0.842 |
| | Anscombe [19] | — | 29.40/0.836 | 28.22/0.815 | 28.51/0.817 |
| | Self2Self [49] | U-Net [27] | 29.06/0.834 | 28.15/0.817 | 28.83/0.841 |
| | N2V [41] | U-Net [27] | 28.78/0.758 | 27.92/0.766 | 27.43/0.745 |
| | R2R [58] | U-Net [27] | 29.14/0.732 | 28.68/0.771 | 28.77/0.765 |
| | NBR2NBR [54] | U-Net [27] | 30.86/<u>0.855</u> | 29.54/<u>0.843</u> | <u>29.79</u>/<u>0.838</u> |
| | Blind2Undlind [66] | U-Net [27] | **31.07/0.857** | **29.92/0.852** | **30.10/0.844** |
| | Noise2Same† [52] | U-Net [27] | 27.44/0.738 | 26.36/0.700 | 26.37/0.721 |
| | Noise2Same† [52] | SwinIR [55] | 27.09/0.713 | 26.29/0.695 | 26.27/0.713 |
| | Laine19-mu [42] | U-Net [27] | 29.76/0.820 | 27.89/0.778 | 28.94/0.808 |
| | Laine19-pme [42] | U-Net [27] | <u>30.88</u>/0.850 | <u>29.57</u>/0.841 | 28.65/0.785 |
| | DBSN [51] | DBSN [51] | 29.60/0.811 | 27.81/0.771 | 28.72/0.800 |
| | Ours† | SwinIA [70] | 29.06/0.788 | 27.74/0.764 | 28.27/0.780 |

Table 4. Denoising results on sRGB images with synthetic Poisson noise. The results are presented as average PSNR (in dB)/SSIM scores. In each series of experiments, the best score is highlighted in **bold**, and the second-best score is <u>underlined</u>. † denotes our implementations of the models that we evaluated ourselves [70].

the test set amounts to $240 + 300 + 280 = 820$ images. For SwinIA training, we follow Blind2Unblind [66] and increase the number of training steps to $80000$.

Following Blind2Unblind [66], we experiment with these datasets by applying zero-mean Gaussian noise with $\sigma = 25$ and $\sigma \in [5, 50]$ (the results are in Table 3) and Poisson noise with $\lambda = 30$ and $\lambda \in [5, 50]$ (the results are in Table 4).

In Noise2Same experiments, SwinIR outperforms U-Net in Gaussian denoising, especially with a varying $\sigma$ parameter. In Poisson noise removal, all Noise2Same experi-

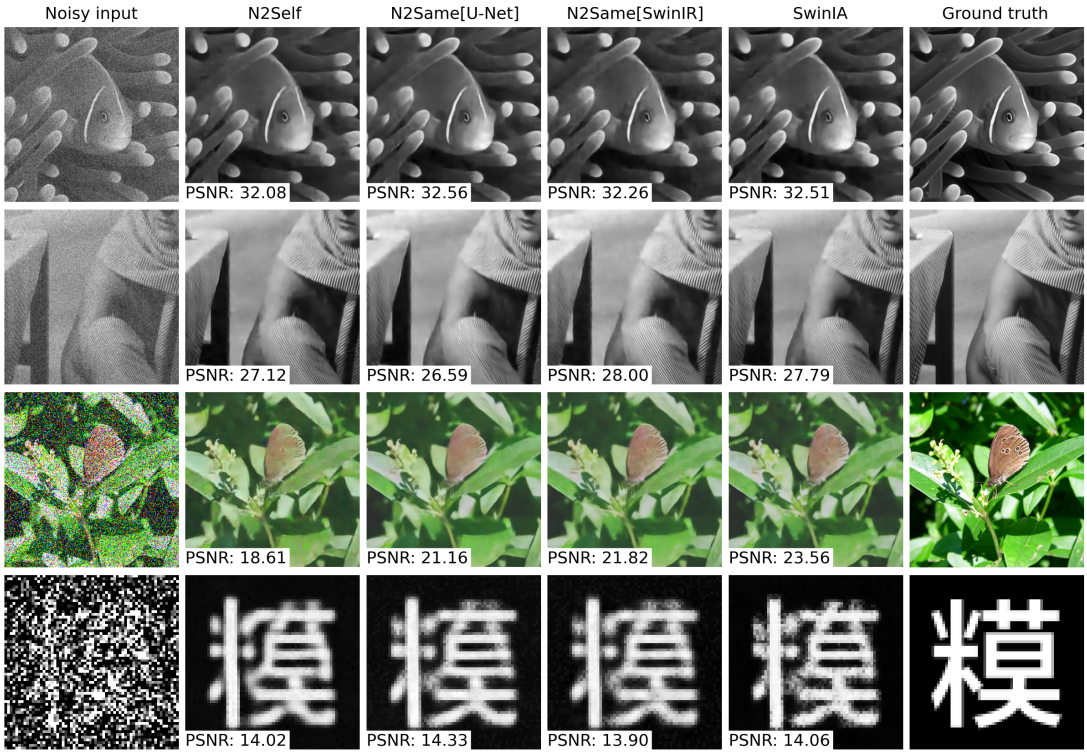



Figure 29. Denoising examples from BSD68 [9] and Set12 grayscale datasets with Gaussian noise ($\mu = 0, \sigma = 25$), ImageNet [15, 52], and HànZì [52] prepared datasets with mixture noise. There are six columns with noisy input, denoising results from Noise2Self (N2Self) [39], Noise2Same (N2Same) with U-Net [52] and SwinIR [69] as backbones, SwinIA, and ground truth images. Each denoising result has a caption with PSNR (in dB) for this image [70].

ments showed lower performance than concurrent methods, while SwinIA maintained the performance on par with other true blind-spot networks. Comparing Noise2Same and SwinIA, the latter was slightly worse with Gaussian noise, yet it showed stable scores regardless of the noise model.

## 5.3 Prepared mixed synthetic noise

Prepared noisy data was taken from Noise2Same [52] repository to compare the results with those presented in the paper. These datasets were prepared by Noise2Self [39] and Noise2Same [52] authors, thus, the noisy copies are generated and saved before training.

HànZì dataset of hand-written Chinese characters contains $13029$ characters and consists of $78174$ $[0, 1]$-ranged noisy images of size $64 \times 64$. Each image was corrupted with a mixture of zero-mean Gaussian noise with $\sigma = 0.7$ and Bernoulli noise with

| Method | Network | ImageNet $[0, 255]$ [52] | HànZì $[0, 1]$ [39] |
|---|---|---|---|
| | | Poisson $\lambda = 30$<br>Gaussian $\sigma \in [5, 50]$<br>Bernoulli $p = 0.2$ | Gaussian $\sigma = 0.7$<br>Bernoulli $p = 0.5$ |
| N2C [27] | U-Net [27] | 23.39/ — | 15.66/ — |
| N2N [38] | U-Net [27] | 23.27/ — | 14.30/ — |
| NLM [11] | — | 18.04/ — | 8.41/ — |
| BM3D [13] | — | 18.74/ — | 10.90/ — |
| Noise2Void [41] | U-Net [27] | 21.36/ — | 13.72/ — |
| Noise2Self† [39] | U-Net [27] | 21.33/0.574 | 14.16/0.512 |
| Noise2Self†‡ [39] | SwinIR [55] | 19.41/0.470 | 12.18/0.388 |
| Noise2Same† [52] | U-Net [27] | 22.85/0.625 | **14.85**/0.542 |
| Noise2Same† [52] | SwinIR [55] | 22.95/0.626 | 14.24/0.510 |
| Laine19 [42] | U-Net [27] | 20.89/ — | 10.70/ — |
| Ours† | SwinIA [70] | **23.36/0.638** | 14.35/**0.556** |

Table 5. Denoising results on images with mixed synthetic noise. The results are presented as average PSNR (in dB)/SSIM scores. In each series of experiments, the best score is highlighted in **bold**, and the second-best score is underlined. † denotes our implementations of the models that we evaluated ourselves. ‡ denotes clipping gradient norm to $1.0$ [70].

$p = 0.5$. The data is randomly split into training and test sets with a ratio $9 : 1$ [39].

Another mixed noise dataset was made of $50000$ images from the ImageNet validation set. $60000$ crops of size $128 \times 128$ were cropped from the first $20000$ images. Then the images were exposed to the mixture of Poisson noise with $\lambda = 30$, zero-mean Gaussian noise with $\sigma = 60$, and Bernoulli noise with $p = 0.2$. Another two subsets of ImageNet validation of size $1000$ images each are used for validation and testing [39].

Noise2Self [39] also used BSD datasets with training on BSD400 and testing on BSD68 with Gaussian noise ($\mu = 0, \sigma = 25$). However, we have already conducted such experiments in Section 5.2.1, where the training scheme was more reasonable as the noisy images were not prepared beforehand. Therefore, we did not use these data in this section.

The results for these experiments are presented in Table 5 and visualized in Figure 29, Figure 34, and Figure 35. Noise2Self with SwinIR as a backbone failed in comparison to the original Noise2Self. The reason for this might be in gradient norm clipping applied in this experiment. Without clipping, the gradients exploded at the very beginning of the training, likely due to the stochastic nature of Noise2Self training, as loss is computed on an exiguous partition of pixels. Noise2Same with SwinIR outperforms the original Noise2Same on ImageNet data while failing on HànZì. The reason for the latter might be the strong noise level on HànZì data and the lack of high-frequency structures.

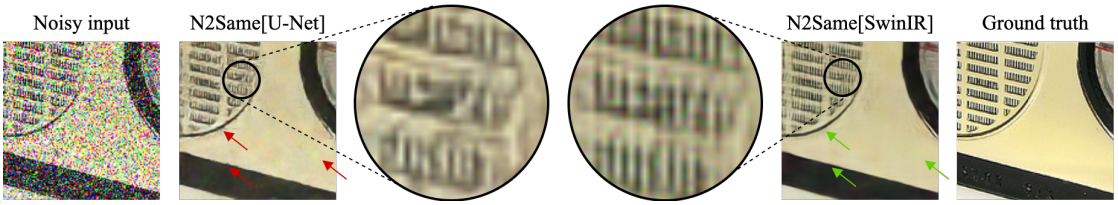



Figure 30. Detailed structure restoration comparison between Noise2Same with U-Net [52] and SwinIR [69] as backbones. The structure is zoomed in for comparison. Additional parts of the image for comparison that include color spots and fractured lines are highlighted with arrows.

SwinIA shows state-of-the-art performance on both datasets, yielding the best score only to Noise2Same on HànZì data in PSNR, while still beating it in SSIM. Besides, SwinIA outperformed Noise2Noise on both datasets and reached a score close to that of a strongly supervised Noise2Clean on ImageNet. Compared to another true BSN by Laine *et al*. [42], SwinIA showed $+2.47$dB PSNR on ImageNet and $+3.65$dB PSNR on HànZì.

We noticed the difference in the PSNR-SSIM score relation between convolutional and transformer backbones. In some cases, transformers show lower PSNR than convolutional U-Net backbone, but better SSIM at the same time. We assume that transformers are more capable of fine texture restoration, whereas CNNs are prone to smooth detailed parts of an image. We show an example of such a comparison in Figure 30. The same behavior is demonstrated in grayscale images: in Figure 29, the image from Set12 in the second row has a striped pattern, which is mostly smoothed by U-Net, yet it is more distinguishable in the experiments with SwinIR and SwinIA.

## 5.4 Natural noise

Apart from synthetic noise, we used natural noise datasets in experiments. We conducted experiments on Fluorescent Microscopy Denoising (FMD) datasets [44] that contain a natural mixture of Gaussian and Poisson noise. We took Confocal Fish, Confocal Mice, and Two-Photon Mice subsets from the dataset. Each subset consists of $20$ views, each view contains $50$ images of the same scene. The ground truth for each view is obtained by averaging the $50$ images. We leave the 19th view for testing and use all other views for training.

Since the datasets are small and consist literally of only $20$ different views, we adjust the training settings, lowering the number of steps to $20000$ and setting up a weight decay rate of $10^{-8}$.

The results of the experiments with natural noise are presented in Table 6 and visualized in Figure 31. SwinIR does not give any improvement to the Noise2Same

| Method | Network | Confocal Fish | Confocal Mice | Two-Photon Mice |
|---|---|---|---|---|
| N2C [27] | U-Net [27] | 32.79/0.905 | 38.40/0.966 | 34.02/0.925 |
| N2N [38] | U-Net [27] | 32.75/0.903 | 38.37/0.965 | 33.80/0.923 |
| BM3D [13] | — | 32.16/0.886 | 37.93/<u>0.963</u> | <u>33.83</u>/**0.924** |
| N2V [41] | U-Net [27] | 32.08/0.886 | 37.49/0.960 | 33.38/0.916 |
| NBR2NBR [54] | U-Net [27] | 32.11/0.890 | 37.07/0.960 | 33.40/<u>0.921</u> |
| Blind2Undlind [66] | U-Net [27] | **32.74/0.897** | **38.44/0.964** | **34.03**/0.916 |
| Noise2Self† [39] | U-Net [27] | 31.96/0.877 | 36.45/0.960 | 31.61/0.910 |
| Noise2Same† [52] | U-Net [27] | 32.36/0.893 | 37.64/0.960 | 33.55/0.917 |
| Noise2Same† [52] | SwinIR [55] | 32.07/0.881 | 37.44/0.959 | 33.39/0.914 |
| CADT[71] | CADT[71] | <u>32.52</u>/<u>0.895</u> | <u>38.21</u>/0.962 | 33.64/0.914 |
| Laine19-mu (G) [42] | U-Net [27] | 31.62/0.849 | 37.54/0.959 | 32.91/0.903 |
| Laine19-pme (G) [42] | U-Net [27] | 23.30/0.527 | 31.64/0.881 | 25.87/0.418 |
| Laine19-mu (P) [42] | U-Net [27] | 31.59/0.854 | 37.30/0.956 | 33.09/0.907 |
| Laine19-pme (P) [42] | U-Net [27] | 25.16/0.597 | 37.82/0.959 | 31.80/0.820 |
| Ours† | SwinIA [70] | 31.79/0.871 | 37.65/0.960 | 33.25/0.915 |

Table 6. Denoising results on fluorescent microscopy images with natural noise. The results are presented as average PSNR (in dB)/SSIM scores. In each series of experiments, the best score is highlighted in **bold**, and the second-best score is <u>underlined</u>. For Laine *et al*. [42], G stands for Gaussian noise, and P stands for Poisson noise. † denotes our implementations of the models that we evaluated ourselves [70].

model in this data, showing slightly lower scores for all three subsets. However, SwinIA shows the best performance among the true blind-spot solutions, outperforming all models by Laine *et al*. [42], except for a minor loss in PSNR on Confocal Mice data.

We also set up the experiments for Smartphone Images Denoising Dataset (SIDD) [37], which contained high-resolution sRGB images, where each pair of images was shot in the same scene on a smartphone and on a professional camera. However, all our models managed to learn the identity function on these data. The reason behind this is that the noise in these images is spatially correlated [51], while most of the methods we use assume that the noise is pixel-level and spatially uncorrelated. Obviously, Noise2Self and Noise2Same methods easily restore masked pixel value from its neighboring pixels in such a setting. SwinIA also operates on the pixel level, yet there is room for experiments with shuffle group size, which, in theory, could help to overcome the spatial correlation.

## 5.5 Ablation study

In this section, we describe the ablation study that we performed during SwinIR integration and SwinIA design. We give reasons for the ablations and summarize the results.

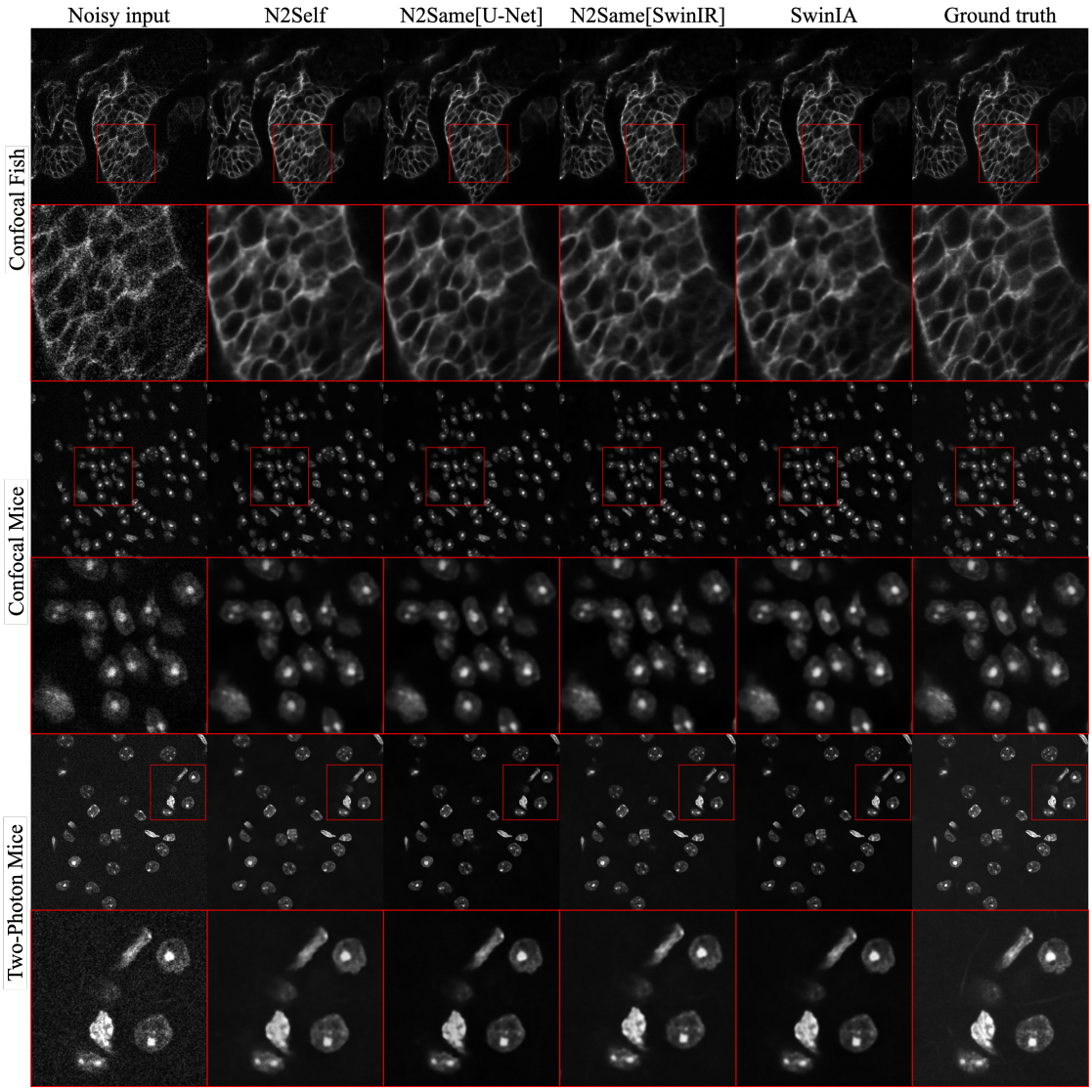



Figure 31. Denoising results on fluorescent microscopy denoising (FMD) dataset [44]. Each pair of rows contains an example from a part of the dataset and a zoomed patch for comparison, and there are six columns with noisy input, denoising results from Noise2Self (N2Self) [39], Noise2Same (N2Same) with U-Net [52] and SwinIR [69] as backbones, SwinIA, and ground truth images [70].

### 5.5.1 SwinIR

SwinIR [55] uses patches of size 1, while patch size equals 4 by default in Swin Transformer [56]. The authors did not perform an ablation study of window size with regard to this change. They adopted the window size equal to 7 from Swin Transformer. Thus,

window resolution in pixels was implicitly reduced from $28 \times 28$ to $7 \times 7$.

We investigated the opportunity to increase the window size in order to obtain a field of view similar to that of Swin Transformer. In Table 7, we show the scores of SwinIR in Noise2Self and Noise2Same modes with window sizes of $8$ and $16$. The sizes were chosen to avoid unnecessary padding for training crops of size $64 \times 64$.

| Method | Window size | HanZi PSNR | HanZi SSIM | ImageNet PSNR | ImageNet SSIM |
|---|---|---|---|---|---|
| Noise2Self | 8 | 12.18$^\dagger$ | 0.388$^\dagger$ | 19.41$^\dagger$ | 0.470$^\dagger$ |
| | 16 | **13.57** | **0.475** | **20.21**$^\dagger$ | **0.493**$^\dagger$ |
| Noise2Same | 8 | 14.24 | 0.510 | 22.95 | 0.626 |
| | 16 | **14.35** | **0.513** | **23.05** | **0.635** |

Table 7. SwinIR window size ablation study. PSNR (in dB) and SSIM scores are compared across window sizes of SwinIR in each method separately, and the best scores are highlighted in **bold**. $^\dagger$ denotes the experiments where gradient norm was clipped to $1.0$ [69].

Even though a window size of $16$ showed superior scores in all of the experiments, multiplying the window size by the factor of $2$ increases the computational complexity by $4$ times. In the situation of limited resources and knowing that SwinIR by itself demands by orders of magnitude more resources than U-Net, we consider the improvement coming from this parameter change insignificant.

### 5.5.2 SwinIA

First of all, we conducted an ablation study on the key architectural concepts that we combined in our SwinIA model. We took our best scoring model on ImageNet data and tested its variants without each design feature. The design features that we tested in this ablation were:

- Cyclic shifting scheme of $4$ consecutive window shifts instead of diagonal shifts from Swin Transformer;
- Cosine annealing learning rate scheduling instead of lambda;
- Additional shuffle group layer normalization before self-attention;
- Shuffle groups instead of utilizing all transformer blocks on pixel level;
- U-shaped structure instead of a plain sequential pixel-level scheme as in SwinIR.

Table 8 shows that all the architecture concepts substantially contributed to the best score. The last two experiments, with only pixel-level transformers without shuffle groups, did not converge, so the score was unobtainable, with or without a U-shaped structure.

| | | Diagonal shift [56] | Cyclic shift | Lambda LR [52] | Cosine LR [32] | Shuffle Group LN | Shuffle groups | U-Shape | PSNR/SSIM | Δ |
|---|---|---|---|---|---|---|---|---|---|---|
| | Our best | | ✓ | | ✓ | ✓ | ✓ | ✓ | 23.36/0.638 | |
| Ablation | No Cyclic shift | ✓ | | | ✓ | ✓ | ✓ | ✓ | 23.10/0.623 | -0.26/-0.015 |
| | No Cosine LR | | ✓ | ✓ | | ✓ | ✓ | ✓ | 23.16/0.627 | -0.20/-0.011 |
| | No SGLN | | ✓ | | ✓ | | ✓ | ✓ | 23.30/0.631 | -0.06/-0.007 |
| | No Shuffle groups | | ✓ | | ✓ | ✓ | | ✓ | — | |
| | No U-shape | | ✓ | | ✓ | ✓ | | | — | |

Table 8. SwinIA design decisions ablation study on ImageNet dataset with mixed noise [52]. The first line represents our baseline with chosen design decisions and the rest of the lines are ablations for each decision [70].

We also separately verified other architectural decisions that did not play a major part in our reasoning process. We tested a model without shortcuts around each transformer block, as they were not used in SwinIR [55], by which our model was inspired. We also trained a model without layer normalization at the end of transformer blocks, as it is not common for transformers in vision [46].

| | HànZì [39] | ImageNet [52] |
|---|---|---|
| Our best | 14.35/0.556 | 23.36/**0.638** |
| No shortcuts | **14.36/0.559** | 23.24/0.624 |
| No last norm | 13.91/0.543 | **23.40/0.638** |

Table 9. Additional ablation study for SwinIA transformer group architecture details on HànZì [39] and ImageNet [52] mixed noise datasets [70].

Table 9 shows that each of the changes improved the score on one of the datasets. However, at the same time, it substantially degraded the performance on the other dataset. Therefore, we decided to proceed with the most stable solution and kept both changes in the model.

## 5.6 Failed SwinIA experiments

While designing the SwinIA model, we tested a large variety of configurations for different architectural concepts. Eventually, not all of the ideas we had proved their utility. Here, we mention the ideas that seemed promising but did not find their place in the final architecture [70]. This list could be valuable for possible future work on the model.

**Noise2Same training scheme**. We implemented a SwinIA variant that allows for training in two forward passes, as it is done in Noise2Same [52]. During the masked pass, we utilized SwinIA without any changes. During the unmasked pass, we canceled the diagonal attention mask, allowing the tokens to access their own values, and loosened the input isolation, propagating keys and values through the transformers starting from the second block. In this setting, we computed the original Noise2Same compound loss between the two outputs. This training had convergency issues, we speculate that this is due to the considerable modality shift between the two forward passes through the same model. However, we do not eliminate the possibility of returning to this idea in future work.

**Dilated attention**. As an alternative to pixel shuffle, we considered dilated window attention as a mechanism to increase the receptive field under the restriction of input isolation. However, the interactions between non-adjacent pixels violated the importance of the continuous field of view and produced sharp single-pixel artifacts.

# 6 Discussion

Our experimental results indicate that transformers are capable of image denoising in a self-supervised manner. We experimented with a state-of-the-art transformer backbone for image restoration integrated into classical frameworks and with our own model that does image denoising without outer frameworks. Thus, we can evaluate the performance of transformers both as a backbone and as a standalone blind-spot network.

SwinIR integrated into Noise2Self did not show good results as the model did not converge on high noise levels and required an additional constraint on gradient value. This indicates that the stochasticity of Noise2Self, determined by its single-component loss on a small partition of pixels, is harmful to a complex transformer model.

As a part of Noise2Same, SwinIR showed performance comparable to that of U-Net, overcoming its results in some cases, for instance, on sRGB data with Gaussian or mixed noise. Besides, we have shown that SwinIR is more capable of restoration of fine structures and patterns (Figure 30). However, such an improvement comes at a considerable computational cost, the details of training and inference time are presented in Table 10 and the complexity of the models is compared in Table 11 in Appendix II. We have also shown that SwinIR denoising can be further improved by increasing window size, albeit it would demand quadratically more resources due to window MSA time complexity.

SwinIA came as a good proof of concept and showed decent results on all benchmarks, which was not the case with Noise2Same that degraded on Poisson noise data (Table 4) regardless of the backbone. This proves that SwinIA is designed in such a way that it does not require knowledge of the noise model, unlike many solutions for self-supervised image denoising, including true blind-spot approaches. This property is essential, for example, no matter how much better SwinIR is compared to U-Net in supervised denoising, its capabilities become limited once it is put into a denoising framework. SwinIA is the first convolution-free architecture that is capable of self-supervised denoising done by model properties solely and optimized by tuning a simple MSE.

The only assumption that we are yet to eliminate for SwinIA is the spatial uncorrelatedness of noise. As it was shown in failed experiments with SIDD in Section 5.4, SwinIA is able to learn an identity function on spatially correlated noise, since it operates on the pixel level. Therefore, we leave this idea for future work and it is yet to determine whether increasing the initial shuffle group size will overcome this requirement.

All in all, SwinIA is subject to further development, which can not only help the model to generalize to different kinds of noise better but also to improve the scores, establishing a new state of the art in self-supervised image denoising. In addition, we hypothesize that SwinIA might be developed into a powerful general-purpose computer vision encoder capable of efficient self-supervised pre-training.

# 7 Conclusion

Self-supervised image denoising is invaluable for the industry. It allows for image quality enhancement when clean images are unavailable, which is crucial for such domains as medical imaging, where image analysis plays a key role in a variety of processes. In this work, we managed to fill the gap in modern research about transformers in self-supervised image denoising by thoroughly testing a ready-made solution within an existing framework and designing a novel transformer-based blind-spot architecture. Our results show that one can definitely benefit from using transformers for the task, yet the computational costs have to be taken into account. Apart from the insights into the performance of transformers in self-supervised denoising, we provided a novel proof-of-concept network, which is valuable for the further development of self-supervised transformers in computer vision.

# 8 Acknowledgements

The work was funded by Biomedical Computer Vision research group of the University of Tartu jointly with PerkinElmer, Inc. The author acknowledges help and support from the research group. The author also expresses high gratitude to his supervisor, Mikhail Papkov, for his thorough guidance, positive attitude and support, and the time he devoted to the project. Special thanks to Joonas Ariva for helping with the Estonian translation and to Anna Nikonorova for helping to find the most beautiful English words.

# References


[1] H. Robbins and S. Monro. "A stochastic approximation method". In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407.

[2] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[3] E. O. Brigham and R. E. Morrow. "The fast Fourier transform". In: *IEEE Spectrum* 4.12 (1967), pp. 63–70. DOI: 10.1109/MSPEC.1967.5217220.

[4] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. URL: http://dx.doi.org/10.1038/323533a0.

[5] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[6] C. Tomasi and R. Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846. DOI: 10.1109/ICCV.1998.710815.

[7] Rich Franzen. "Kodak lossless true color image suite". In: *source: http://r0k.us/graphics/kodak* 4.2 (1999).

[8] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. URL: http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[9] D. Martin et al. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics". In: *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*. Vol. 2. 2001, 416–423 vol.2. DOI: 10.1109/ICCV.2001.937655. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=937655.

[10] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[11] A. Buades, B. Coll, and J.-M. Morel. "A non-local algorithm for image denoising". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. 2005, 60–65 vol. 2. DOI: 10.1109/CVPR.2005.38.

[12] Kostadin Dabov et al. "Color Image Denoising via Sparse 3D Collaborative Filtering with Grouping Constraint in Luminance-Chrominance Space". In: vol. 1. Sept. 2007, pp. I–313. ISBN: 978-1-4244-1437-6. DOI: 10.1109/ICIP.2007.4378954.

[13] Kostadin Dabov et al. “Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering”. In: *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society* 16 (Sept. 2007), pp. 2080–95. DOI: 10.1109/TIP.2007.901238.

[14] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[15] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[16] Alex Shevchenko, Max Lytvyn, and Dmytro Lider. *Grammar and spelling check with Grammarly*. Software available from grammarly.com. 2009. URL: https://www.grammarly.com/.

[17] John C. Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” In: *COLT*. Ed. by Adam Tauman Kalai and Mehryar Mohri. Omnipress, 2010, pp. 257–269. ISBN: 978-0-9822529-2-5. URL: https://dl.acm.org/doi/abs/10.5555/1953048.2021068.

[18] Roman Zeyde, Michael Elad, and Matan Protter. “On single image scale-up using sparse-representations”. In: *International conference on curves and surfaces*. 2010, pp. 711–730.

[19] Markku Makitalo and Alessandro Foi. “Optimal Inversion of the Anscombe Transformation in Low-Count Poisson Image Denoising”. In: *IEEE Transactions on Image Processing* 20.1 (2011), pp. 99–109. DOI: 10.1109/TIP.2010.2056693.

[20] John Lees-Miller et al. *Overleaf, Online LaTeXeditor*. Software available from overleaf.com. 2012. URL: https://www.overleaf.com/.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[22] Guomin Luo and Daming Zhang. “Wavelet Denoising”. In: Apr. 2012. ISBN: 978-953-51-0494-0. DOI: 10.5772/37424.

[23] Diederik Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[24] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[25] Ajay Boyat and Brijendra Joshi. “A Review Paper: Noise Models in Digital Image Processing”. In: *Signal & Image Processing : An International Journal* 6 (May 2015). DOI: 10.5121/sipij.2015.6206.

[26] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 448–456.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. cite arxiv:1505.04597Comment: conditionally accepted at MICCAI 2015. 2015. URL: http://arxiv.org/abs/1505.04597.

[28] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[29] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. cite arxiv:1607.06450. 2016. URL: http://arxiv.org/abs/1607.06450.

[30] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR ’16. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459.

[31] Dan Hendrycks and Kevin Gimpel. “Gaussian Error Linear Units (GELUs)”. In: 2016. arXiv: http://arxiv.org/abs/1606.08415v3 [cs.LG].

[32] Ilya Loshchilov and Frank Hutter. “Sgdr: Stochastic gradient descent with warm restarts”. In: *arXiv preprint arXiv:1608.03983* (2016).

[33] Gao Huang et al. “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

[34] Adam Paszke et al. “Automatic differentiation in pytorch”. In: (2017).

[35] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[36] Kai Zhang et al. “Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising”. In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3142–3155.

[37] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. “A High-Quality Denoising Dataset for Smartphone Cameras”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[38] Jaakko Lehtinen et al. *Noise2Noise: Learning Image Restoration without Clean Data*. cite arxiv:1803.04189Comment: Added link to official implementation and updated MRI results to match it. 2018. URL: http://arxiv.org/abs/1803.04189.

[39] Joshua Batson and Loic Royer. “Noise2Self: Blind Denoising by Self-Supervision”. In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, Sept. 2019, pp. 524–533. URL: https://proceedings.mlr.press/v97/batson19a.html.

[40] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[41] Alexander Krull, Tim-Oliver Buchholz, and Florian Jug. “Noise2void-learning denoising from single noisy images”. English. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2129–2137.

[42] Samuli Laine et al. “High-quality self-supervised deep image denoising”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[43] Mingxing Tan and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. cite arxiv:1905.11946Comment: Published in ICML 2019. 2019. URL: http://arxiv.org/abs/1905.11946.

[44] Yide Zhang et al. “A Poisson-Gaussian Denoising Dataset with Real Fluorescence Microscopy Images”. In: *CVPR*. 2019.

[45] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[46] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. cite arxiv:2010.11929Comment: Fine-tuning code and pre-trained models are available at https://github.com/google-research/vision_transformer. ICLR camera-ready version with 2 small modifications: 1) Added a discussion of CLS vs GAP classifier in the appendix, 2) Fixed an error in exaFLOPs computation in Figure 5 and Table 6 (relative performance of models is basically not affected). 2020. URL: http://arxiv.org/abs/2010.11929.

[47] David Honzátko et al. "Efficient blind-spot neural network architecture for image denoising". In: *2020 7th Swiss Conference on Data Science (SDS)*. IEEE. 2020, pp. 59–60.

[48] Nick Moran et al. "Noisier2noise: Learning to denoise from unpaired noisy data". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 12064–12072.

[49] Yuhui Quan et al. "Self2self with dropout: Learning self-supervised denoising from single image". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1890–1898.

[50] Joongbo Shin et al. "Fast and Accurate Deep Bidirectional Language Representations for Unsupervised Learning". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, July 2020, pp. 823–835. DOI: 10.18653/v1/2020.acl-main.76. URL: https://aclanthology.org/2020.acl-main.76.

[51] Xiaohe Wu et al. "Unpaired learning of deep image denoising". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV*. Springer. 2020, pp. 352–368.

[52] Yaochen Xie, Zhengyang Wang, and Shuiwang Ji. "Noise2Same: Optimizing A Self-Supervised Bound for Image Denoising". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020, pp. 20320–20330.

[53] Hangbo Bao et al. "Beit: Bert pre-training of image transformers". In: *arXiv preprint arXiv:2106.08254* (2021).

[54] Tao Huang et al. "Neighbor2neighbor: Self-supervised denoising from single noisy images". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 14781–14790.

[55] Jingyun Liang et al. "SwinIR: Image Restoration Using Swin Transformer". In: *arXiv preprint arXiv:2108.10257* (2021).

[56] Ze Liu et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021.

[57] JGraph Ltd. *diagrams.net, draw.io — an open-source visual editor*. Software available from diagrams.net. 2021. URL: https://www.diagrams.net/.

[58] Tongyao Pang et al. "Recorrupted-to-recorrupted: unsupervised deep learning for image denoising". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 2043–2052.

[59] Kan Wu et al. "Rethinking and improving relative position encoding for vision transformer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10033–10041.

[60] Sixiao Zheng et al. "Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers". In: *CVPR*. 2021.

[61] John Schulman et al. *Introducing ChatGPT*. 2022. URL: https://openai.com/blog/chatgpt.

[62] Liangyu Chen et al. "Simple Baselines For Image Restoration". In: *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VII*. Tel Aviv, Israel: Springer-Verlag, 2022, pp. 17–33. ISBN: 978-3-031-20070-0. DOI: 10.1007/978-3-031-20071-7_2. URL: https://doi.org/10.1007/978-3-031-20071-7_2.

[63] Kaiming He et al. "Masked autoencoders are scalable vision learners". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009.

[64] Xiaolong Liu et al. "DnT: Learning Unsupervised Denoising Transformer from Single Noisy Image". In: *Proceedings of the 4th International Conference on Image Processing and Machine Vision*. IPMV '22. Hong Kong, China: Association for Computing Machinery, 2022, pp. 50–56. ISBN: 9781450395823. DOI: 10.1145/3529446.3529455. URL: https://doi.org/10.1145/3529446.3529455.

[65] Alex Rogozhnikov. "Einops: Clear and Reliable Tensor Manipulations with Einstein-like Notation". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=oapKSVM2bcj.

[66] Zejin Wang et al. "Blind2unblind: Self-supervised image denoising with visible blind spots". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 2027–2036.

[67] Zhendong Wang et al. "Uformer: A general u-shaped transformer for image restoration". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 17683–17693.

[68] Syed Waqas Zamir et al. “Restormer: Efficient transformer for high-resolution image restoration”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5728–5739.

[69] Pavel Chizhov and Mikhail Papkov. *Self-Supervised Image Denoising with Swin Transformer*. 2023. URL: https://openreview.net/forum?id=EARgl3EH-nq.

[70] Mikhail Papkov and Pavel Chizhov. *SwinIA: Self-Supervised Blind-Spot Image Denoising with Zero Convolutions*. 2023. arXiv: 2305.05651 [cs.CV].

[71] Dan Zhang and Fangfang Zhou. “Self-Supervised Image Denoising for Real-World Images with Context-aware Transformer”. In: *IEEE Access* (2023).

# Appendix

# I. Visualization

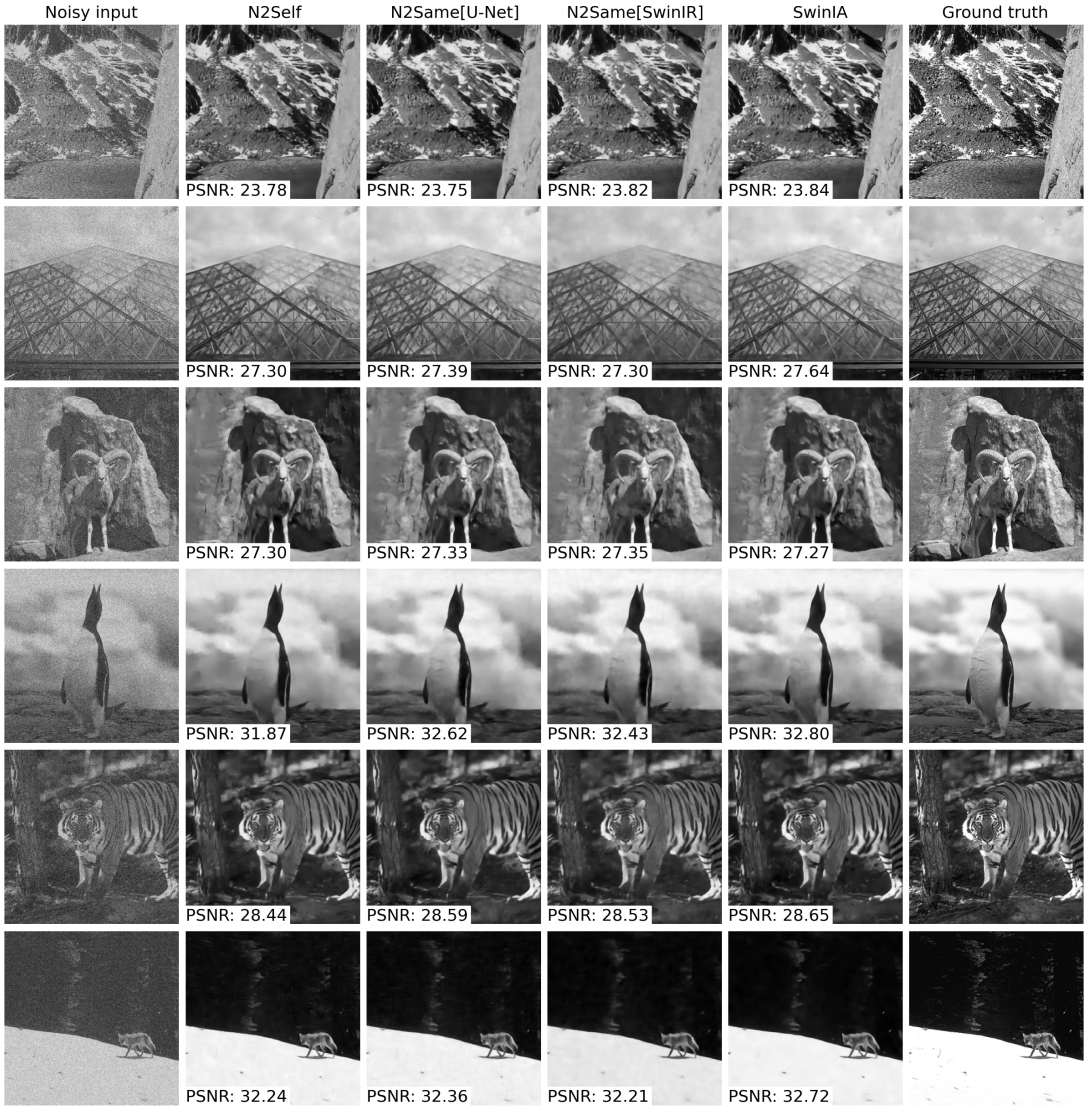



Figure 32. Denoising examples on BSD68 dataset [9] of grayscale natural images with Gaussian noise ($\mu = 0, \sigma = 25$). There are six rows with example images and six columns with noisy input, denoising results from Noise2Self (N2Self) [39], Noise2Same (N2Same) with U-Net [52] and SwinIR [69] as backbones, SwinIA, and ground truth images. Each denoising result has a caption with PSNR (in dB) for this image [70].


Figure 33. Denoising examples on Set12 dataset of grayscale natural images with Gaussian noise ($\mu = 0, \sigma = 25$). There are six rows with example images and six columns with noisy input, denoising results from Noise2Self (N2Self) [39], Noise2Same (N2Same) with U-Net [52] and SwinIR [69] as backbones, SwinIA, and ground truth images. Each denoising result has a caption with PSNR (in dB) for this image [70].

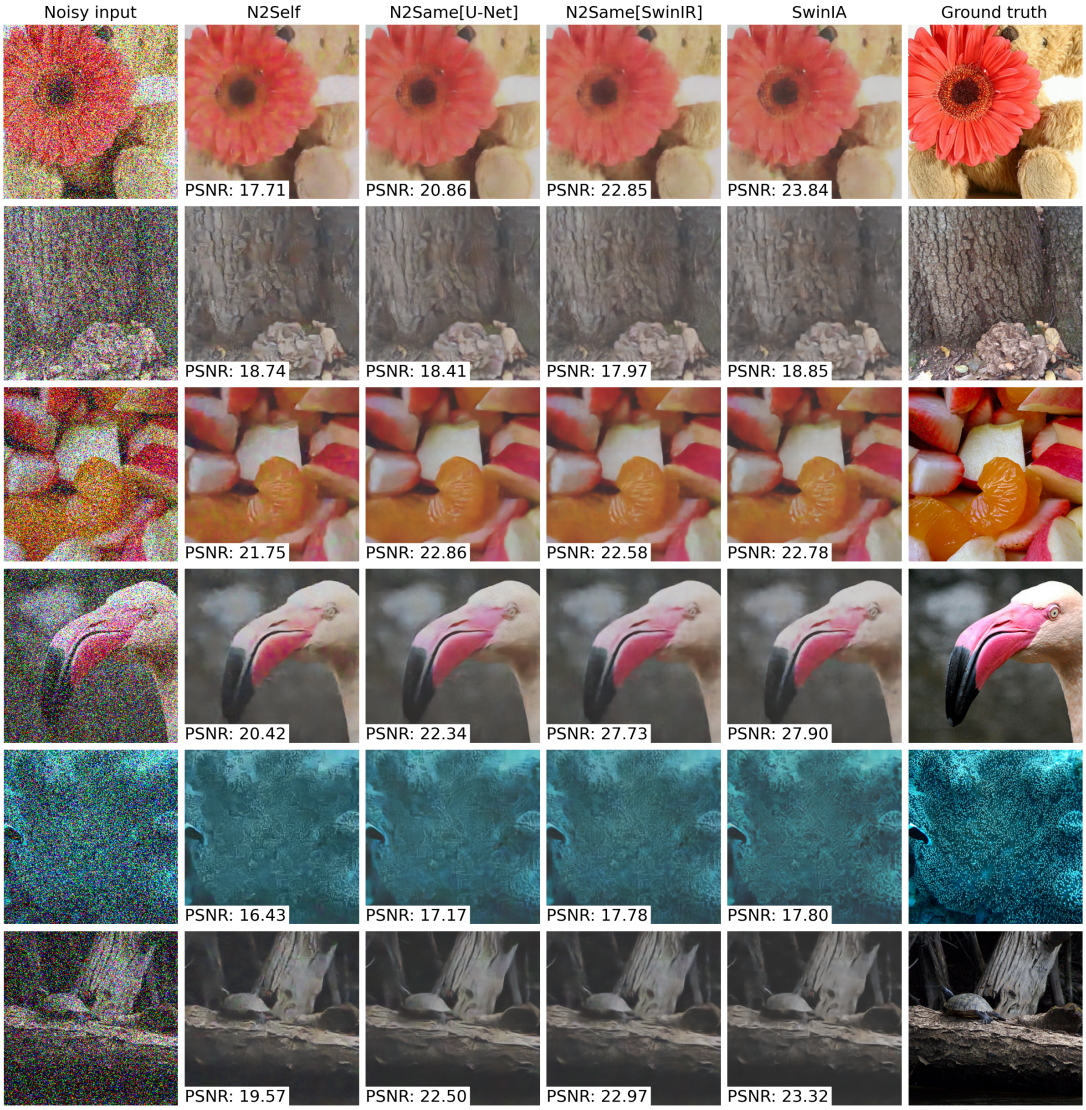



Figure 34. Denoising examples on ImageNet denoising dataset [52] of sRGB natural images with mixture of Poisson noise with $\lambda = 30$, zero-mean Gaussian noise with $\sigma = 60$, and Bernoulli noise with $p = 0.2$. There are six rows with example images and six columns with noisy input, denoising results from Noise2Self (N2Self) [39], Noise2Same (N2Same) with U-Net [52] and SwinIR [69] as backbones, SwinIA, and ground truth images. Each denoising result has a caption with PSNR (in dB) for this image [70].

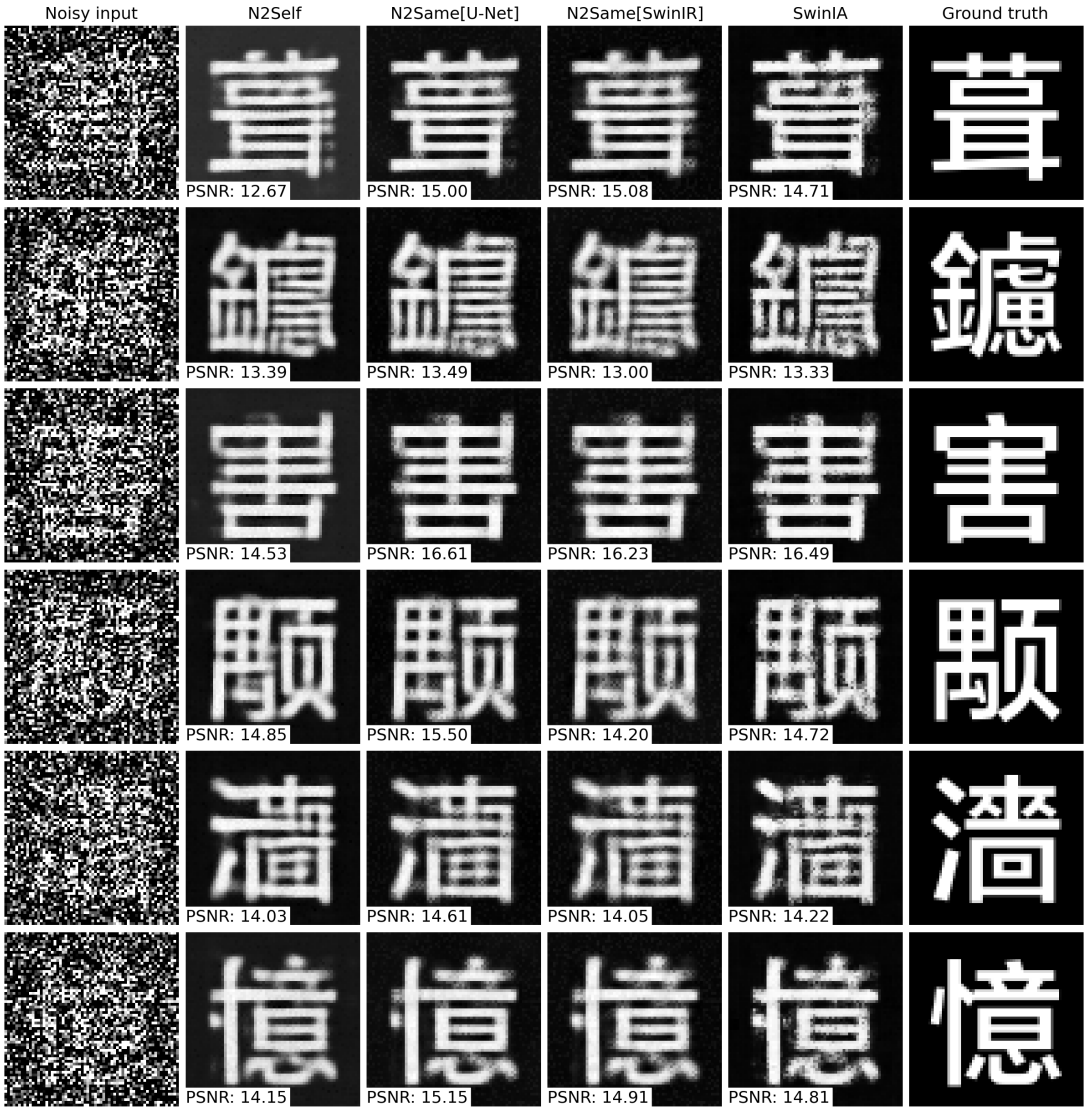



Figure 35. Denoising examples on HànZì denoising dataset [52] of grayscale handwritten Chinese characters with mixture of zero-mean Gaussian noise with $\sigma = 0.7$ and Bernoulli noise with $p = 0.5$. There are six rows with example images and six columns with noisy input, denoising results from Noise2Self (N2Self) [39], Noise2Same (N2Same) with U-Net [52] and SwinIR [69] as backbones, SwinIA, and ground truth images. Each denoising result has a caption with PSNR (in dB) for this image [70].

# II. Resources and complexity

| Dataset | N2Same[U-Net] | | N2Same[SwinIR] | | SwinIA | |
|---|---|---|---|---|---|---|
| | TT (h) | AIT (ms) | TT (h) | AIT (ms) | TT (h) | AIT (ms) |
| Synthetic (sRGB) [42, 54, 66] | 4 | 26 | 20 | 3263 | 15 | 941 |
| Synthetic (grayscale) [66] | 2 | 12 | 19 | 869 | 13 | 605 |
| ImageNet [52] | 1 | 14 | 18 | 2601 | 15 | 765 |
| HànZì [39] | 1 | 5 | 13 | 30 | 14 | 42 |
| Microscopy [42, 54, 66] | 1.5 | 20 | 4 | 7025 | 5 | 845 |

Table 10. Comparison of training time (TT) in hours, and average inference time (AIT) on the test set in milliseconds of Noise2Same [52] and SwinIA (ours) on various datasets [69, 70].

| Criterion | N2Same[U-Net] | N2Same[SwinIR] | SwinIA |
|---|---|---|---|
| Number of trainable parameters | 5.564M | 4.610M | 2.369M |
| FLOPs per grayscale image $64 \times 64$ | 5.001G | 18.978G | 10.117G |

Table 11. Comparison of the numbers of parameters and floating point operations per second (FLOPS) between Noise2Same [52] with U-Net and SwinIR backbones, and SwinIA (ours). The FLOPS are documented for inference time, therefore this number doubles when training in Noise2Same mode with two forward passes [69, 70].

# III. Licence

## Non-exclusive licence to reproduce thesis and make thesis public

I, **Pavel Chizhov**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   **Self-supervised image denoising using transformers**,
   (title of thesis)

   supervised by Mikhail Papkov.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Pavel Chizhov
***09/05/2023***