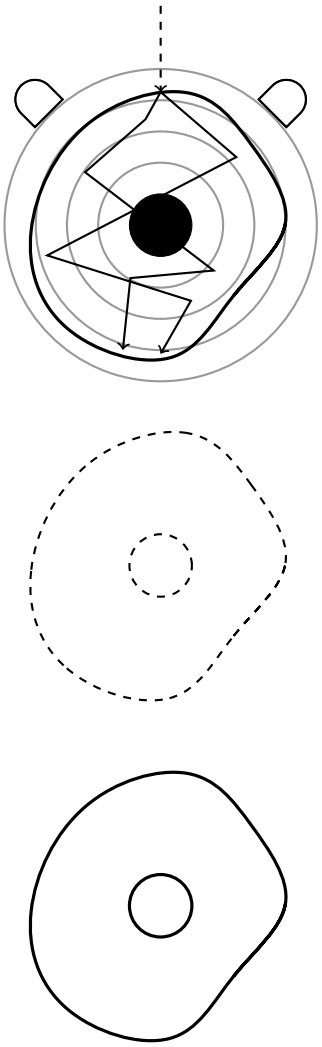
Deep Learning for Photoacoustic Imaging Reconstruction



Cristian Perez Jensen

Layout: typeset by the author using LaTeX.
Cover illustration: Cristian Perez Jensen – created with TikZ.

Deep Learning for Photoacoustic Imaging Reconstruction

Cristian Perez Jensen
12993301

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 900
1098 XH Amsterdam

Supervisor
Dr. Navchetan Awasthi

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 900
1098 XH Amsterdam

Semester 2, 2022–2023

Abstract

This thesis project focuses on addressing the challenges of photoacoustic imaging (PAI) reconstruction through the application of deep learning techniques. PAI is a non-invasive biomedical imaging modality that utilises the photoacoustic effect to generate images of biological tissue. However, PAI faces limitations such as limited acoustic detection and low optical propagation depth, resulting in poor image quality and shallow imaging depth. To overcome these challenges, this project formulates PAI reconstruction as an image-to-image translation task and explores the use of deep learning models.

The research evaluates the performance of various deep learning techniques in enhancing the quality of PAI reconstruction. The evaluation metrics include the structural similarity index (SSIM) and peak signal-to-noise ratio (PSNR), which measure image resemblance and perceptual similarity. The project aims to answer the research question of how recent advancements in generative deep learning can improve the quality of noisy deep PAI reconstruction in terms of SSIM, PSNR, and visual clarity.

In the image-to-image translation domain, two primary types of models are considered: U-nets and diffusion models. U-nets utilise encoding and decoding stages to map input images to output images, whilst diffusion models iteratively predict noise from a noisy image to generate representative images. The project explores variations in the implementation of U-nets, including simple convolutional layers, residual layers, and vision transformers. Additionally, the influence of different loss functions, including adversarial loss, mean squared error, SSIM, PSNR, and combinations thereof, is examined. The implementations of all models and loss functions are available at `https://github.com/cristianpjensen/thesis-pat-reconstruction`.

The data is obtained from thresholded images that were used for simulated PAI data on which the models are trained. These images come from the DRIVE and NNE datasets. Furthermore, an experimental dataset comprising four phantoms is introduced to validate the findings on non-simulated data, with the phantoms captured in water serving as ground truth for the images captured in a solution approximating tissue characteristics. The performance evaluation is conducted on both a simulated data test set and the experimental data. The evaluation shows promising results for deep learning to improve PAI in the form of more visible structure in the model outputs than inputs.

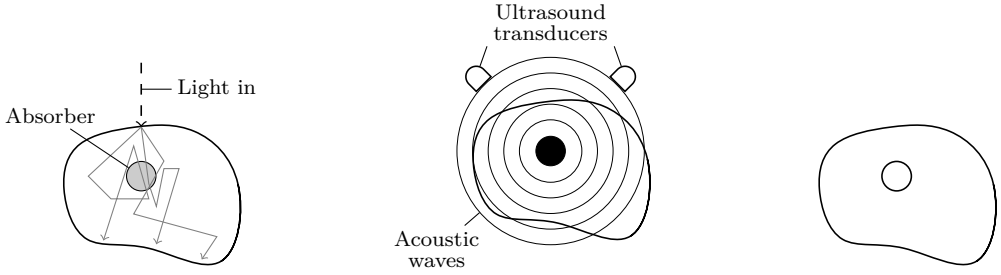
Contents

Chapter 1

Introduction

1.1 Photoacoustic imaging

Photoacoustic imaging (PAI) is a biomedical imaging modality that offers a less invasive alternative to procedures such as computed tomography (CT) and magnetic resonance imaging (MRI). Compared to other non-invasive modalities, PAI presents certain advantages, such as enhanced specificity compared to conventional ultrasound imaging, and greater penetration depth compared to other optical imaging techniques (Beard 2011, Attia et al. 2019). PAI operates as a hybrid modality by utilising the photoacoustic effect, a physical phenomenon that converts absorbed optical energy into acoustic energy (Wang & Yao 2016). The practical implementation of PAI involves the emission of short laser pulses onto biological tissue (Figure 1.1a). This process induces continuous heating and cooling of the absorber, leading



(a) Pulsed light excitation. (b) Temperate changes cause acoustic wave emission. (c) Image reconstruction.

Figure 1.1: Imaging principle of PAI.

to the generation of acoustic waves that propagate outside the tissue. These waves are subsequently captured by either a single or an array of ultrasound transducers (Figure 1.1b). The captured data is used to form an image of the tissue (Figure 1.1c) (Wang & Yao 2016).

PAI has two significant challenges that adversely impact the quality of reconstruction images (Choi et al. 2023):

1. Limited acoustic detection from the transducer results in the reconstructions having less information to work with.
2. Light has low optical propagation depth for effective acoustic conversion, making getting signals from deep tissue hard. As a consequence, the data becomes very superficial.

To address these problems, they can be formulated as an image-to-image translation task, where a poor construction is translated into a better one. Recently, experimentation with deep learning to solve these problems have emerged (Wang et al. 2020), improving the signal-to-noise ratio and imaging depth. This project builds on that research and uses recent developments in the image-to-image translation domain to build better models for PAI reconstruction.

1.2 Research statement

The evaluation of model performance in this research project will be based on two metrics: the structural similarity index (SSIM) (Wang et al. 2004) and the peak signal-to-noise ratio (PSNR). PSNR is a straightforward measure that quantifies the ratio between the maximum possible pixel value and the image's noise level (expressed as mean squared error, MSE). Thus, it provides a normalised measurement of MSE, with higher PSNR values indicating a closer image resemblance to the original. On the other hand, SSIM is a metric that assesses the perceptual similarity between two images based on the degradation of structural information. Notably, SSIM and PSNR differ in their sensitivity to different types of image degradations. For instance, PSNR exhibits higher sensitivity to additive Gaussian noise, while SSIM is more sensitive to JPEG compression. Nonetheless, they exhibit a close correlation for most forms of image degradation (Hore & Ziou 2010).

This project aims to assess the effectiveness of various deep learning techniques in addressing the challenges above of PAI reconstruction. Accordingly, this project seeks to answer the following research question: To what extent can recent ad-

vancements in generative deep learning enhance the quality of noisy deep PAI reconstruction in terms of SSIM, PSNR, and visual clarity?

1.3 Image-to-image translation

In the generative image-to-image translation domain, there are two primary types of models: U-nets (Ronneberger et al. 2015) and, more recently, diffusion models (Ho et al. 2020). U-nets operate by first encoding the input image to a latent space, serving as a more compact representation of the image that the model learns. Then, a decoder decodes the output image from this latent space. Conversely, diffusion models learn to model the underlying data distribution by iteratively predicting noise from a noisy image. During sampling, these models start from pure Gaussian noise and iteratively remove noise until an image representative of the training data's distribution emerges. Saharia et al. (2022) extended the original diffusion framework to enable image-to-image translation by conditioning on an input image, which is the framework implemented in this project.

Moreover, the building blocks of U-nets have many variations in their implementation. This project explores three types: a simple convolutional layer (Isola et al. 2018), residual layers (He et al. 2015,0, Xie et al. 2017), and vision transformers (Dosovitskiy et al. 2021). Instead of learning what the output should be, residual layers learn what should be changed about the input. These changes are usually close to a zero mapping, which makes residual layers easier to optimise (He et al. 2015). Thus, they should be an improvement on simple convolutional layers for this problem since U-nets can be challenging to optimise. Vision transformers partition the images into patches and utilise the attention mechanism (Vaswani et al. 2017) between the patches. In recent years, attention has found considerable success in diverse domains (Lin et al. 2021, Khan et al. 2022). Due to its success in these domains, this project will also explore how attention affects the performance of U-nets.

Lastly, this project explores the influence of variations in the loss functions on model performance. The first training objective that this project considers is a GAN loss function (ℓ_{GAN}), where the discriminator of Pix2Pix (Isola et al. 2018) is used. However, due to common convergence issues (Saxena & Cao 2023), ℓ_{GAN} might yield poor results. Consequently, it is compared to the following loss functions that do not involve an auxiliary discriminator: ℓ_{MSE}, ℓ_{SSIM}, ℓ_{PSNR}, and a combination of ℓ_{SSIM} and ℓ_{PSNR}. The reason for selecting SSIM and PSNR as loss functions is that these will be the metrics used to measure how good a model is. Thus optimising the models w.r.t. to them should produce good results.

1.4 Contributions

This project makes several contributions to improving the quality of PAI reconstruction. Firstly, a comprehensive comparison is conducted, evaluating various image-to-image translation models for PAI reconstruction. This analysis provides valuable insights into the strengths and weaknesses of different approaches, aiding in the selection of suitable models for PAI applications. In addition to the comparative analysis, this project introduces the application of conditional diffusion models to PAI reconstruction. Furthermore, this project evaluates various loss functions in the context of PAI. Namely, it finds that adversarial networks are not necessary for PAI reconstruction, making the training process easier. Finally, this research project improves upon simple models such as Pix2Pix by utilising more advanced models that further ease the training process. These models are, in some cases, in addition to being more performant, more lightweight in terms of their parameter count and floating point operations.

Chapter 2

U-net

U-nets map images to images by applying two stages (Ronneberger et al. 2015). The initial stage involves encoding the input image into a high-level representation. Notably, this encoding procedure occurs incrementally, allowing the model to acquire representations at various levels of detail. Namely, the early layers capture low-level knowledge, such as texture information, while the subsequent layers capture high-level knowledge. The encoding stage applies an alternation of encoders and max pooling layers, such that the outputs of the encoders are these intermediate representations of the input. The second stage decodes the high-level representation into an output image. Each decoder layer gets the previous decoding level's output as input and outputs a lower-level image. Additionally, the decoder layers get input from their corresponding encoder layer of the same level, establishing skip-connections. These skip-connections enable precise localisation, providing lower-level information that the decoder can combine with the higher-level information it got from its parent decoder (Ronneberger et al. 2015). Thus, It enables an easier flow of information about low-level details. An overview of the U-net architecture is provided in Figure 2.1.

There are two ways that U-net architectures differ from each other. Firstly, differences arise in implementing its basic blocks, *i.e.* encoder, decoder, and bottleneck blocks. The overall model gains enhanced expressive power by providing more expressive power in its individual blocks. Secondly, channel array C controls variations in the model's depth and width. The channel array C determines the number of channels of the data at each level of the U-net. Hence, the length of C determines the depth, and the channel amounts determine the width of the U-net. In this project, C is usually set to $\langle 64, 128, 256, 512, 512, 512, 512, 512 \rangle$, resulting in a $512 \times 1 \times 1$ image, assuming a 256×256 input image, as the output of the encoding stage, whereby every output pixel is a function of every input pixel. However, the

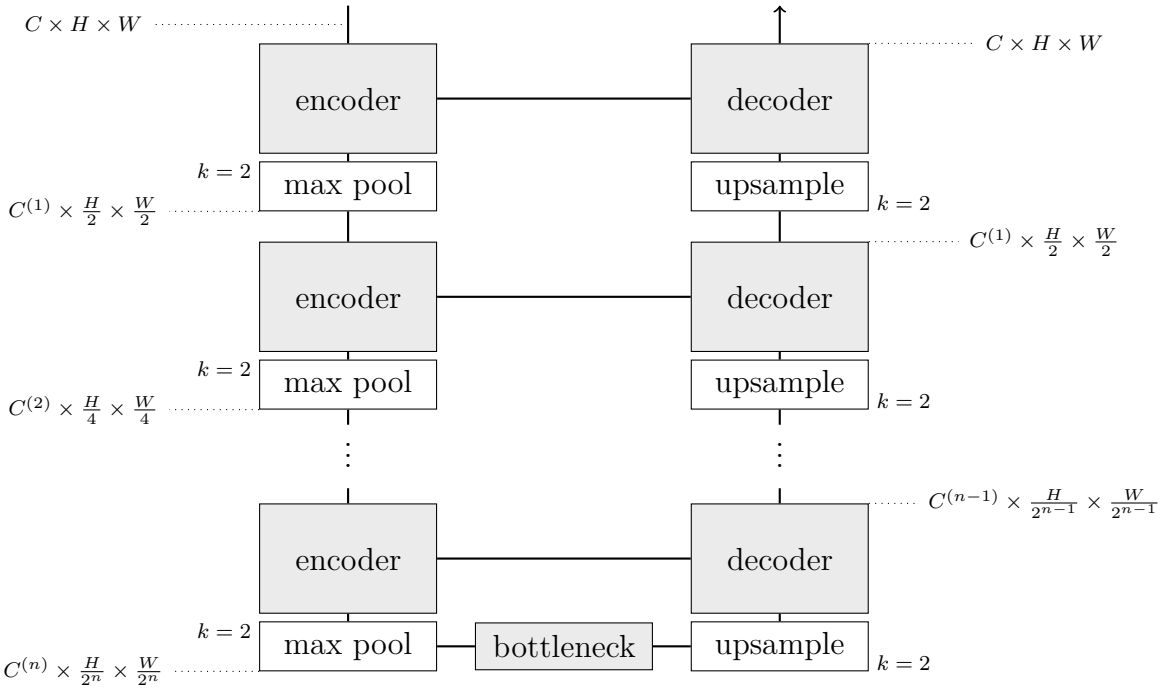
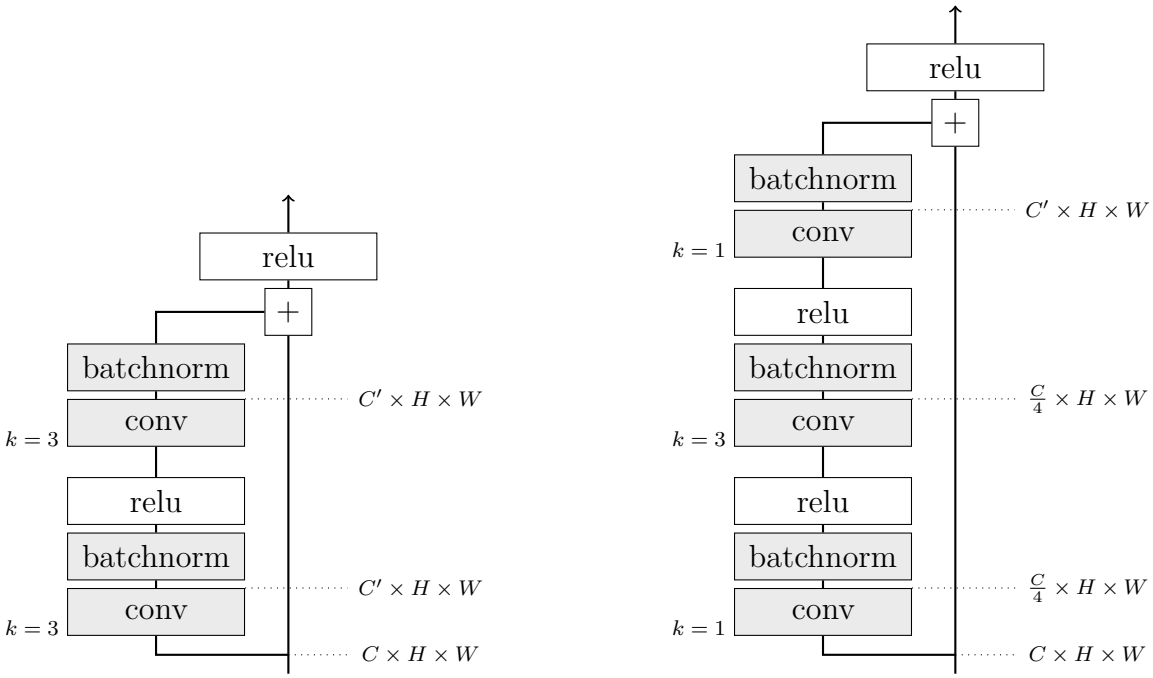


Figure 2.1: General U-net architecture. The skip-connections are implemented by concatenation. The encoder at the kth step takes as input $C^{(k-1)}$ channels and outputs $C^{(k)}$ channels, and the decoder at the kth step takes $2C^{(k)}$ channels as input and outputs $C^{(k-1)}$ channels.

parameter count of some models grows too fast with the depth and width, making a different value for C necessary in practice. Implementation details per model can be found in Appendix A.

Pix2Pix The first model considered in this project is the generator of the Pix2Pix (Isola et al. 2018) architecture, designed initially as a generative adversarial network (GAN). Pix2Pix uses a 4×4 convolutional layer as both encoder and decoder. A ReLU layer (Nair & Hinton 2010) precedes the convolutional layer, and a normalisation layer comes after it. In the bottom three decoder layers, 50% dropout layers simulate noise. This model acts as a baseline for more complex models due to its simplicity.



(a) ResNet-18 layer.

(b) ResNet-50 layer.

Figure 2.2: Residual layers of ResNet-18 and ResNet-50 (He et al. 2015).

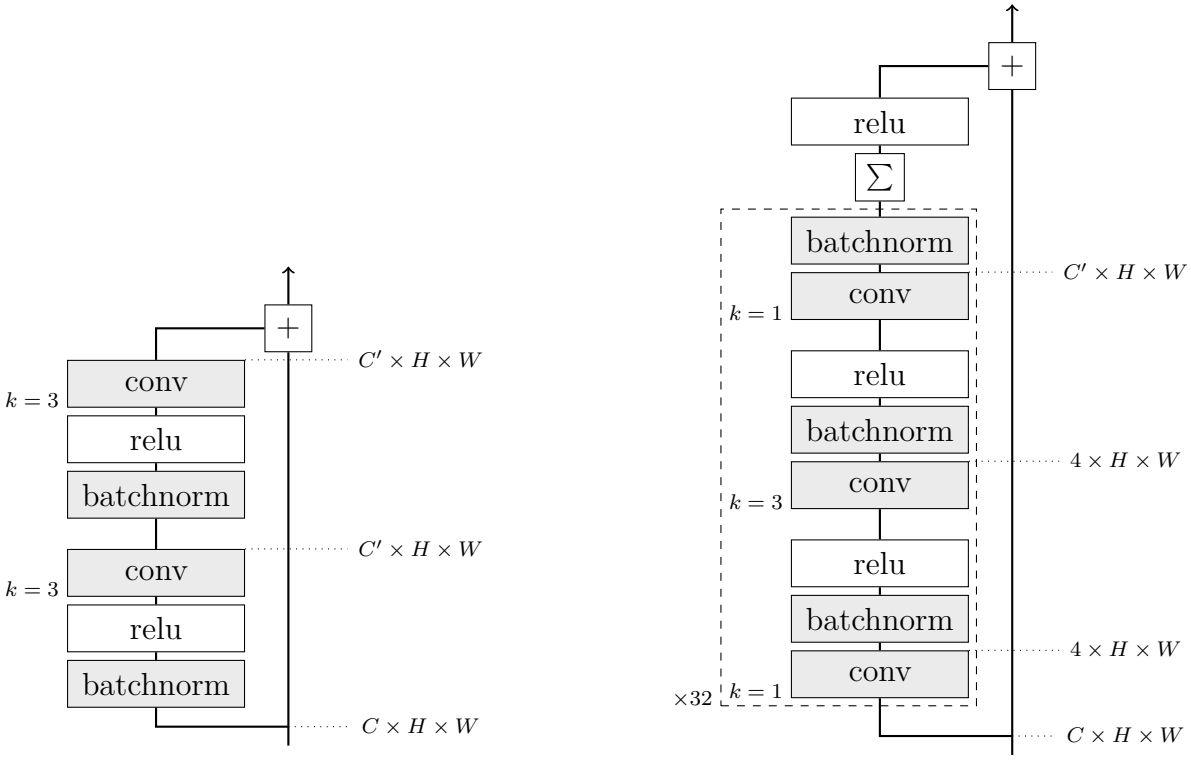
2.1 Residual blocks

Residual layers were introduced into deep convolutional neural networks because learning became harder for convolutional neural networks with normal convolutional layers as they got deeper (He et al. 2015). Let $f : \mathbb{R}^{C \times H \times W} \mapsto \mathbb{R}^{C' \times H \times W}$ be a learnable non-linear mapping consisting of convolutional and normalisation layers, and let α be an activation layer. Residual layers add their input to f:

$$r(\boldsymbol{x}) = \alpha(f(\boldsymbol{x}) + \boldsymbol{x}),$$

thus adding a direct reference of the input to the output. (This is similar to the skip-connections in U-nets with addition instead of concatenation.) The idea behind adding a skip-connection is that it is easier to optimise if an identity mapping is optimal, *i.e.* $r^\star(\boldsymbol{x}) = \boldsymbol{x}$, because it is easier to push $f(\boldsymbol{x})$ to be $\mathbf{0}$ than $\boldsymbol{x}$ due to its non-linearity (He et al. 2015). Thus, residual layers do not give models more capacity since f could become an identity mapping. However, it makes it easier for models to optimise w.r.t. to their training objective, because, intuitively, its task becomes what to change about the input, rather than what the output should

be. In practice, identity mappings are unlikely to be optimal, but it helps to precondition the problem. If the optimal function is closer to an identity mapping than a zero mapping, it is easier to find the optimal function with identity mapping than without one. He et al. (2015) found that the learned function f often has small responses when adding its input, suggesting that residual layers provide reasonable preconditioning.



(a) ResNet V2 layer.

(b) ResNeXt layer.

Figure 2.3: Residual layers of ResNet V2 (He et al. 2016) and ResNeXt (Xie et al. 2017).

Res18 and Res50 Two of the first models to utilise residual layers were ResNet-18 and ResNet-50 (Figure 2.2 depicts diagrams of the basic blocks' architectures). The learnable function f in Res18[1] consists of two convolutional layers with normalisation and the ReLU activation function. ResNet-50 increased the depth

[1]For all residual U-nets, the encoder and decoder blocks are replaced by its residual layer. The U-nets are named after the model that their residual layer comes from, *i.e.* Res18 U-net uses the residual layer of the ResNet-18 network. The layers themselves will be referenced to like Res18 for the layer used in Res18 U-net.

significantly, so the number of parameters became problematic. Thus, Res50 introduced a bottleneck to f, drastically decreasing the number of parameters (55.7M *vs.* 7.8M per Table 4.1 in the final U-net architecture).

ResV2 Later, He et al. (2016) developed ResNet V2 to ease optimisation further. By removing the activation layer after adding the skip-connection, a stack of layers that are all identity mappings, *i.e.* their learned functions are zero mappings, will also form an identity mapping. Furthermore, any layer connects to another in a residual fashion, *i.e.* they are an addition of the input and a weight layer:

$$r_L(\boldsymbol{x}_l) = \boldsymbol{x}_l + \sum_{i=l+1}^{L} f_i(\boldsymbol{x}_{i-1}),$$

where $\boldsymbol{x}_i = r_i(\boldsymbol{x_{i-1}})$. Also, the output of any layer is a function of the summation of the outputs of all preceding layers, which is in contrast to normal neural networks, where each layer's output is the product of its preceding layers (He et al. 2016). 2.3a depicts a diagram of the exact specification of ResV2.

ResNeXt Lastly, ResNeXt (Xie et al. 2017) introduced branches to residual layers. The idea is that the layer consists of many branches, each involving a very tight bottleneck. This project uses 32 branches with a bottleneck channel of 4, which was found to be optimal by Xie et al. (2017). The outputs of these branches are summed and used just like the convolutional layers in the other residual layers. Mathematically, ResNeXt layers can be formulated as

$$r(\boldsymbol{x}) = \alpha \left(\sum_{i=1}^{32} b_i(\boldsymbol{x}) \right) + \boldsymbol{x}.$$

Notice that, just like ResV2, ResNeXt uses the activation layer before the skip-connection to take advantage of the same benefits as ResV2. The advantage of this technique is that the branches have a small number of parameters (4.5M per Table 4.1). However, a disadvantage is that it is slow in practice due many convolutional operations.

2.2 Attention

Attention U-net Attention U-net is a model that utilises the attention mechanism in skip-connections, allowing the model to focus on the essential features from the lower-level layers (Oktay et al. 2018). This gives the model an advantage

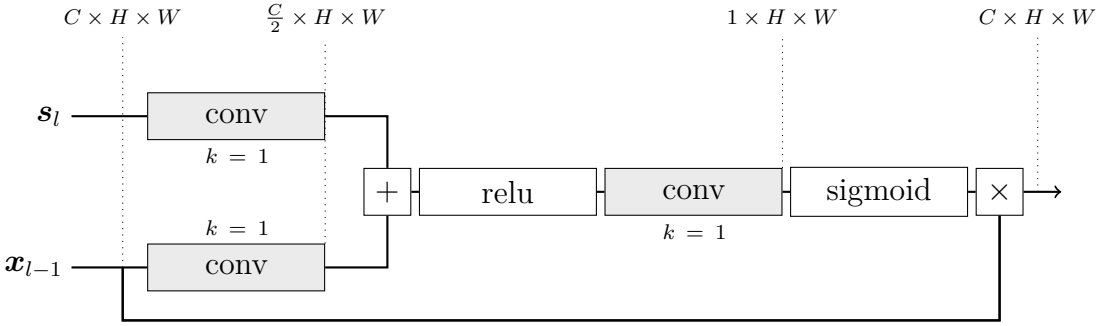


Figure 2.4: Diagram of the additive attention gate used in the skip-connections of the Attention U-net architecture. $\boldsymbol{x}_{l-1}$ is the previous decoder's output and $\boldsymbol{s}_l$ is the skip-connection from an encoder on the same level.

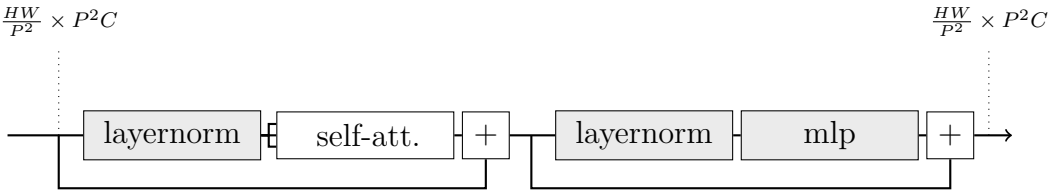


Figure 2.5: Architecture of a layer in a vision transformer with patch size P. It gets as input $\frac{HW}{P^2}$ flattened patches with P^2C dimensionality. The vision transformer consists of a linear projection of the patches, addition of a positional embedding, followed by 12 transformer layers as depicted in this figure. Afterward, it gets reshaped back to a $C \times H \times W$ image.

since it might be the case that low-level information can be discarded. Figure 2.4 shows an illustration of how the attention skip-connection module works. It does not use the traditional attention mechanism by considering all pixels, but it is implemented with 1×1 convolutions, which helps substantially with the parameter count, but sacrifices model expressiveness. The Pix2Pix generator is augmented with attention gates as skip-connections for this project.

Trans U-net Transformers have found considerable success in recent work. The transformer model that this project will use for comparison is Trans U-net (Chen et al. 2021), which introduces a vision transformer (ViT) (Dosovitskiy et al. 2021) in the bottleneck of the U-net. The ViT should give the model a better high-level representation of the input image, which should help the decoding phase. Initially, the input image is divided into patches of dimensionality $C \times P \times P$. These patches are flattened, resulting in $\frac{HW}{P^2}$ patches of size P^2C. Afterward, a linear layer is

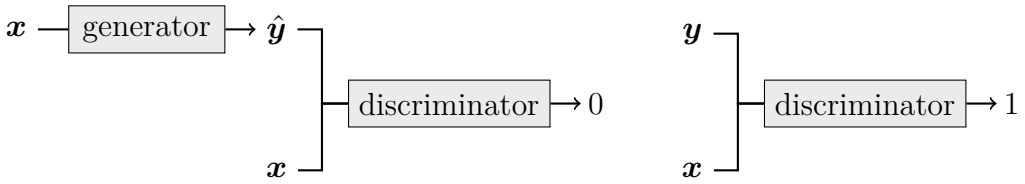


Figure 2.6: A discriminator involves learning to classify between fake $\langle \boldsymbol{x}, \hat{\boldsymbol{y}} \rangle$ and real $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ tuples.

applied to the flattened patches, followed by the addition of a learned positional embedding. Subsequently, a sequence of 12 transformer layers (Figure 2.5) is applied to these patches. Lastly, the patches are reshaped into a $C \times H \times W$ image. For this project, the Pix2Pix generator was augmented with a vision transformer in the bottleneck. Thus, we can compare the results with the Pix2Pix results to quantify the effect of the ViT directly.

2.3 Loss functions

The first loss function that this project explores is the loss function used by Pix2Pix (Isola et al. 2018). It is an adversarial loss function relying on an additional discriminator network d that classifies whether an input tuple is fake, *i.e.* generated by the generator, or real, *i.e.* part of the dataset (Figure 2.6). The generator must thus 'fool' the discriminator by minimising the binary cross entropy between the output of the discriminator on the generator's output and 1. To this, an ℓ_1 term is added:

$$\ell_{\text{GAN}} = \ell_{\text{BCE}}(d(\boldsymbol{x}, \hat{\boldsymbol{y}}), 1) + \lambda \ell_1(\hat{\boldsymbol{y}}, \boldsymbol{y}),$$

where λ is set to 50. The advantage of such a loss function is that it reduces the need for handcrafting a loss since the discriminator network will learn it, but the best objective might not be found, hurting performance.

The following considered loss functions are

$$\begin{aligned} \ell_{\text{MSE}} &= \text{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\ \ell_{\text{PSNR}} &= -\text{PSNR}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\ \ell_{\text{SSIM}} &= -\text{SSIM}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\ \ell_{\text{S+P}} &= \ell_{\text{PSNR}} + \ell_{\text{SSIM}}, \end{aligned}$$

which are based on the performance metrics on which the models are evaluated. Thus they might result in better evaluation scores.

Chapter 3

Diffusion models

The diffusion model is an emerging generative model that has seen considerable success in the image generation domain (Ho et al. 2020). This model has shown promise in domains such as text-to-image generation (Ramesh et al. 2022, Rombach et al. 2022). The core principle of diffusion models is the reversing of a forward diffusion process. The forward diffusion process is a Markov chain that iteratively adds a small amount of noise, resulting in pure Gaussian noise after many iterations. The primary objective of diffusion models is to learn how to reverse this process by predicting the added noise at each step of the Markov chain, which makes it possible to sample from the distribution of the training data by starting from pure Gaussian noise and iteratively removing noise until a sample emerges.

The diffusion process is parametrised by a noise schedule $\beta_1 < \beta_2 < \cdots < \beta_T$ that determines the amount of noise, β_t, added at each timestep t within the Markov chain. In order to compute the noisy image $\boldsymbol{y}_t$ at a given timestep of the Markov chain directly, we need to introduce the following variables. The amount of data left from the previous timestep is denoted as $\alpha_t = 1 - \beta_t$, while the amount of data left from the starting input is represented by $\gamma_t = \prod_{i=1}^{t} \alpha_i$. Now the noisy image $\boldsymbol{y}_t$ at timestep t can be computed directly:

$$\boldsymbol{y}_t \sim \mathcal{N}(\sqrt{\gamma_t}\boldsymbol{y}_0, (1-\gamma_t)\boldsymbol{I}) \tag{3.1}$$

$$\boldsymbol{y}_t = \sqrt{\gamma_t}\boldsymbol{y}_0 + \sqrt{1-\gamma_t}\boldsymbol{\epsilon}, \tag{3.2}$$

where $\boldsymbol{y}_0$ is the input to the forward process, and $\boldsymbol{\epsilon}$ is pure Gaussian noise. Directly computing $\boldsymbol{y}_t$ allows for efficient training because the algorithm does not need to go through timesteps 1 through t of the Markov chain.

There are various approaches to performing backward diffusion. The model could predict the starting point $\boldsymbol{y}_0$, the noise $\boldsymbol{\epsilon}_t$ added at timestep t, or the previous

point $\boldsymbol{y}_{t-1}$ in the Markov chain. The last option would yield suboptimal results, as the model would become too dependent on the training noise schedule, making it impossible to use two different noise schedules for training and inference. This enables faster sampling by using a Markov chain with fewer timesteps for inference. Predicting $\boldsymbol{y}_0$ or $\boldsymbol{\epsilon}_t$ is essentially the same task since one can be computed with the other by rearranging equation 3.2. However, Ho et al. (2020) found that noise prediction worked better. Thus that approach is used in this project.

The training loop (Algorithm 1) consists of sampling a data point $\boldsymbol{y}_0$ of the training set and uniformly sampling a timestep t, which are used to compute $\boldsymbol{y}_t$ with equation 3.2. The model takes $\boldsymbol{y}_t$ and γ_t as input and outputs a prediction of the noise $\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, \gamma_t)$. After many training steps, the model should be able to predict the noise accurately for any noise level γ_t. In order to sample from the diffusion model, Algorithm 2 is used, where noise is iteratively removed from initially pure Gaussian noise. The reverse diffusion process, *i.e.* predicting $\boldsymbol{y}_{t-1}$ from $\boldsymbol{y}_t$ and $\boldsymbol{y}_0$, can be computed as

$$\boldsymbol{y}_{t-1} \sim \mathcal{N}\left(\frac{\sqrt{\gamma_{t-1}}\beta_t}{1-\gamma_t}\boldsymbol{y}_0 + \frac{\sqrt{\alpha_t}(1-\gamma_{t-1})}{1-\gamma_t}\boldsymbol{y}_t, \beta_t \boldsymbol{I}\right),$$

where the mean can be simplified to be a function of $\boldsymbol{y}_t$ and the predicted noise:

$$\begin{aligned}
\boldsymbol{\mu}_\theta(\boldsymbol{y}_t, t) &= \frac{\sqrt{\gamma_{t-1}}\beta_t}{1-\gamma_t}\boldsymbol{y}_0 + \frac{\sqrt{\alpha_t}(1-\gamma_{t-1})}{1-\gamma_t}\boldsymbol{y}_t \\
&= \frac{1}{1-\gamma_t}\left(\sqrt{\gamma_{t-1}}\beta_t \boldsymbol{y}_0 + \sqrt{\alpha_t}(1-\gamma_{t-1})\boldsymbol{y}_t\right) \\
&= \frac{1}{1-\gamma_t}\left(\sqrt{\gamma_{t-1}}\beta_t \frac{1}{\sqrt{\gamma_t}}\left(\boldsymbol{y}_t - \sqrt{1-\gamma_t}\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, t)\right) + (\sqrt{\alpha_t} - \sqrt{\alpha_t}\gamma_{t-1})\boldsymbol{y}_t\right) \\
&= \frac{1}{1-\gamma_t}\left(\sqrt{\frac{\gamma_{t-1}}{\gamma_t}}\beta_t\left(\boldsymbol{y}_t - \sqrt{1-\gamma_t}\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, t)\right) + \left(\frac{\alpha_t}{\sqrt{\alpha_t}} - \frac{\gamma_t}{\sqrt{\alpha_t}}\right)\boldsymbol{y}_t\right) \\
&= \frac{1}{1-\gamma_t}\left(\frac{\beta_t}{\sqrt{\alpha_t}}\left(\boldsymbol{y}_t - \sqrt{1-\gamma_t}\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, t)\right) + \frac{\alpha_t - \gamma_t}{\sqrt{\alpha_t}}\boldsymbol{y}_t\right) \\
&= \frac{1}{1-\gamma_t}\left(\frac{\alpha_t + \beta_t - \gamma_t}{\sqrt{\alpha_t}}\boldsymbol{y}_t - \frac{\sqrt{1-\gamma_t}\beta_t}{\sqrt{\alpha_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, t)\right) \\
&= \frac{1}{\sqrt{\alpha_t}(1-\gamma_t)}\left((1-\gamma_t)\boldsymbol{y}_t - \sqrt{1-\gamma_t}\beta_t\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, t)\right) \\
&= \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{y}_t - \frac{\beta_t}{\sqrt{1-\gamma_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{y}_t, t)\right).
\end{aligned}$$

Doing this for every iteration of the noise schedule, starting from pure Gaussian noise, results in a sample of the data distribution, provided that the denoising model $\boldsymbol{\epsilon}_\theta$ has been trained appropriately.

Algorithm 1 Training a denoising model $\boldsymbol{\epsilon}_\theta$.

repeat
  $(\boldsymbol{x}, \boldsymbol{y}_0) \sim p(\boldsymbol{x}, \boldsymbol{y})$
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  $\ell \leftarrow \|\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, \sqrt{\gamma_t}\boldsymbol{y}_0 + \sqrt{1-\gamma_t}\boldsymbol{\epsilon}, \gamma_t) - \boldsymbol{\epsilon}\|_2^2$
until converged

Algorithm 2 Inference in T steps.

$\boldsymbol{y}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
for $t = T, \dots, 1$ **do**
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ if $t > 1$ else $\boldsymbol{z} \leftarrow \mathbf{0}$
  $\boldsymbol{y}_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{y}_t - \frac{\beta_t}{\sqrt{1-\gamma_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}, \boldsymbol{y}_t, t)\right)$
end for
return $\boldsymbol{y}_0$

Figure 3.1: Diffusion model algorithms for training and inference.

3.1 Conditioning

A problem with diffusion models, as they have been introduced thus far, is that they cannot condition on an input image, thus not making it possible to use for image-to-image translation. Saharia et al. (2022) introduced this by concatenating the input image to the noise image at each iteration of the reverse diffusion process. The concatenation of the input image proved to work well, getting excellent results in detailed image-to-image translation tasks (Saharia et al. 2022). This image conditioning diffusion framework is called Palette.

3.2 Impovements

Nichol & Dhariwal (2021) made two significant improvements to diffusion models that Saharia et al. (2022) did not incorporate in their implementation of Palette. Their first improvement concerns the noise schedule, which is usually a linear interpolation between two small values. However, this results in many steps mapping noise to noise, not adding any information, as seen in Figure 3.2. The consequence is that many timesteps do not contribute to sample quality. To address this problem, Nichol & Dhariwal (2021) constructed a different noise schedule in terms of γ_t:

$$\gamma_t = \frac{f(t)}{f(0)}, \; f(t) = \cos^2\left(\frac{\frac{t}{T} + s}{1+s} \cdot \frac{\pi}{2}\right),$$

where s is set to 0.008. Note that this can be converted to a β noise schedule by $\beta_t = 1 - \frac{\gamma_t}{\gamma_{t-1}}$. An advantage of this noise schedule is that it has the same amount of noise added relatively in all cosine noise schedules with different timesteps, which is not the case for linear noise schedules where the start and endpoint have to be carefully selected. The result of using the cosine noise schedule can be observed

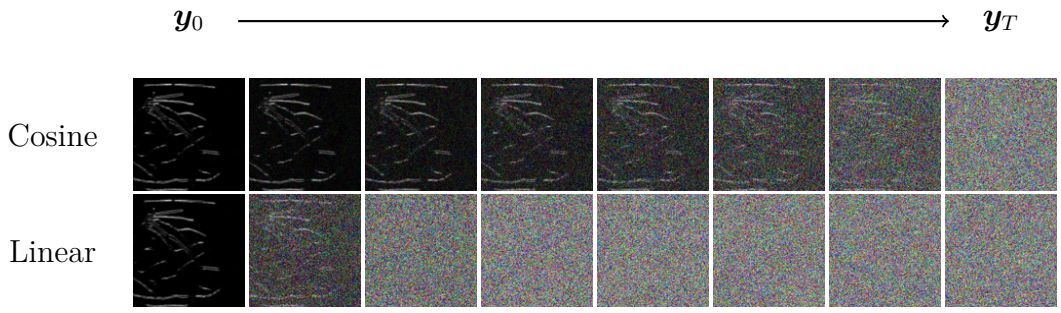


Figure 3.2: Cosine *vs.* linear β schedule. The cosine schedule is much faster at getting to adding actual information, while the linear schedule translates noise to noise for the majority of the timesteps.

in Figure 3.2, which shows that it is much faster at adding information in reverse diffusion.

Their second improvement concerns the variance in the reverse diffusion process. Initially, diffusion models use a fixed variance β_t (Ho et al. 2020, Saharia et al. 2022). However, this parameter can also be learned by a network $\Sigma_\theta(\boldsymbol{x}, \boldsymbol{y}_t, \gamma_t)$, which predicts the variance for every output dimension. Ho et al. (2020) also experimented with learning the variance, but they concluded that it did not offer any advantages. However, this was due to the problem that the variance has a very tight bound ($[\beta_t, \tilde{\beta}_t]$ with $\tilde{\beta}_t = \beta_t \cdot (1 - \gamma_{t-1})/(1 - \gamma_t)$), which makes it hard for Σ_θ to predict the exact variance. Thus, Nichol & Dhariwal (2021) propose to learn a logarithmic interpolation vector $\boldsymbol{v}$ between the lower and upper bound of the variance:

$$\Sigma_\theta(\boldsymbol{x}, \boldsymbol{y}_t, \gamma_t) = \exp(\boldsymbol{v} \log \beta_t + (1 - \boldsymbol{v}) \log \tilde{\beta}_t),$$

which makes it easier for the model to learn the variance.

In this project, the conditional diffusion model Palette (Saharia et al. 2022) is altered with the abovementioned improvements.

Chapter 4

Experiments and results

4.1 Metrics

This project seeks to answer the question of how much developments in generative deep learning improve the reconstruction of noisy deep PAI. In order to achieve this, a comparative analysis of various image-to-image translation models will be conducted. The models will be evaluated on the SSIM and PSNR metrics and visual inspections of model outputs. PSNR measures the ratio of the maximum possible signal, *i.e.* maximum possible pixel value, and the noise in the image, which is measured by the mean squared error (MSE) between the image $\hat{\boldsymbol{y}}$ and its perfect counterpart $\boldsymbol{y}$:

$$\mathrm{PSNR}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = 10 \log_{10} \left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y})} \right).$$

Hence, the PSNR metric offers a normalised metric of the quality of an image. SSIM is a metric that considers image degradation as a perceived change in structural information. The calculation of SSIM involves three distinct components, luminance (l), contrast (c), and structure (s) (Wang et al. 2004):

$$\mathrm{SSIM}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = l(\hat{\boldsymbol{y}}, \boldsymbol{y}) \cdot c(\hat{\boldsymbol{y}}, \boldsymbol{y}) \cdot s(\hat{\boldsymbol{y}}, \boldsymbol{y}),$$

where

$$l(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{2\mu_{\hat{\boldsymbol{y}}}\mu_{\boldsymbol{y}} + c_1}{\mu_{\hat{\boldsymbol{y}}}^2 + \mu_{\boldsymbol{y}}^2 + c_1}$$
$$c(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{2\sigma_{\hat{\boldsymbol{y}}}\sigma_{\boldsymbol{y}} + c_2}{\sigma_{\hat{\boldsymbol{y}}}^2 + \sigma_{\boldsymbol{y}}^2 + c_2}$$
$$s(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{\Sigma_{\hat{\boldsymbol{y}}\boldsymbol{y}} + c_3}{\sigma_{\hat{\boldsymbol{y}}}\sigma_{\boldsymbol{y}} + c_3}.$$

The fundamental difference between SSIM and PSNR is that they differ in their degree of sensitivity to image degradations. Specifically, PSNR has greater sensitivity to additive Gaussian noise, whereas SSIM exhibits higher sensitivity to JPEG compression. Nevertheless, the two metrics are closely related for most forms of image degradation (Hore & Ziou 2010).

Model	Parameters (M)	FLOPs (G)
Pix2Pix	54.4	6.1
Attention U-net	55.6	6.3
Res18 U-net	55.7	23.0
Res50 U-net	7.8	2.7
ResV2 U-net	55.7	23.1
ResNeXt U-net	4.5	4.1
Trans U-net	1 026.8	8.0
Palette	136.0	276.3*

Table 4.1: Model training configurations. $\star$: The FLOP count of Palette is counted per iteration of the reverse diffusion process, thus it must be multiplied by the amount of timesteps of the diffusion process.

4.2 Datasets

In order to evaluate the performance of the models in the context of PAI reconstruction, three datasets are used: the DRIVE dataset (Staal et al. 2004), the NNE dataset (Uhlirova et al. 2017), and an experimental dataset using phantoms from the DRIVE and NNE datasets. The DRIVE dataset comprises 48 images of segmented vessel images of the retina. Random transformations are applied to these images, so each image has five randomly transformed variants, resulting in

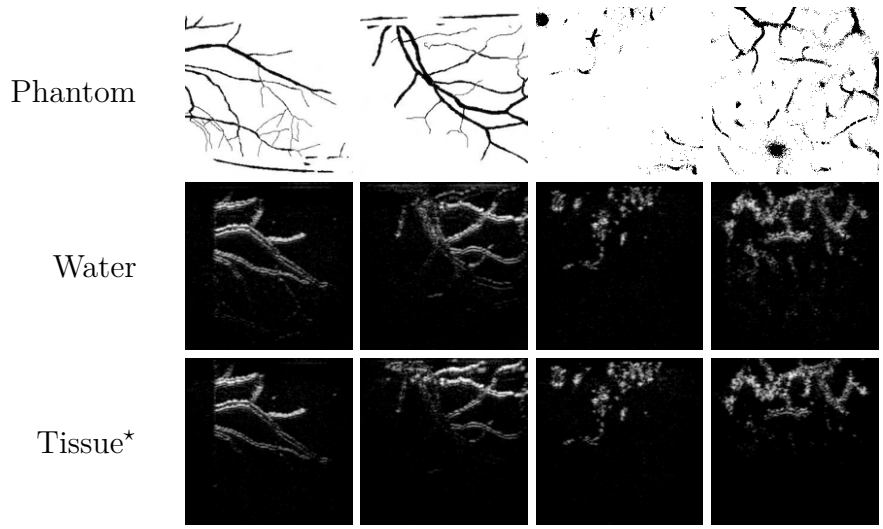


Figure 4.1: The top row depicts the phantoms used in experimentation, where the first two are taken from the DRIVE test set, while the remaining two are picked from the NNE test set. ⋆: The used solution is an approximation of the characteristics of tissue.

a dataset of 240 images. In contrast, The NNE dataset consists of 9531 images of 2-photon single-vessel measurements from the mouse SI cortex. Both datasets are passed into a k-wave algorithm (Treeby & Cox 2010) to simulate the photoacoustic effect that PAI relies on. 10, 20, 30, 40, and 50 dB white noise is added to the simulations to mimic background noise in the measurements. As the amount of white noise decreases, the amount of simulated noise in the images increases, leading to worse reconstructions. Furthermore, an experimental dataset comprising four phantoms is utilised to validate the findings on non-simulated data. In Figure 4.1, the phantoms and their respective PAI outputs can be observed. The phantoms captured in water serve as ground truth for the images captured in a solution approximating tissue characteristics.

In terms of dataset division, the DRIVE dataset is split into an 80/20 train-test ratio, with the last 20% of the train split acting validation data. Thus, 154 images are used for training, 38 for validation, and 48 for testing. Similarly, the NNE dataset is divided into a 67/33 train-test ratio, with the last 20% of the train split serving as validation data. Thus, 5084 images are used for training, 1271 for validation, and 3176 for testing. The only model with a different train-validation split is Palette, which uses 16 data points to validate the DRIVE and NNE datasets due to the high sampling time. All reported results are metrics on the test dataset, which is only used in the final stage to measure how well the trained models performed. Lastly, the models trained on the simulated DRIVE

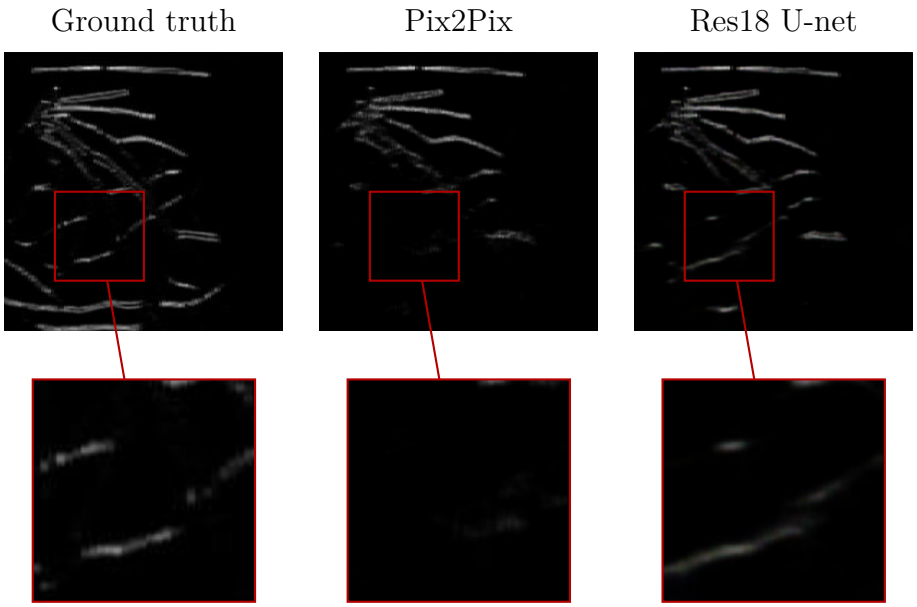


Figure 4.2: A visual comparison of an output of Pix2Pix and Res18 U-net with ℓ_{SSIM} on the 20 dB DRIVE dataset.

dataset are tested on the first two phantoms, while the NNE dataset models are tested on the remaining two phantoms.

4.3 Results

Simulated datasets The first problem this project sought to improve was the background noise making the data noisy, leading to imperfect reconstructions. As shown in Tables 4.2 and 4.3, all models significantly improve SSIM and PSNR on the simulated PAI datasets w.r.t. the initial input image. Of course, all models also perform better as the background noise decreases. Especially in high background noise levels, the models improved the PAI reconstructions remarkably. The most significant improvement is the SSIM metric in these high noise levels (0.213 *vs.* 0.775 on the DRIVE 10 dB dataset).

Another point is that simple models can be further improved using more complex building blocks, like residual layers. On the DRIVE 10 dB dataset, Pix2Pix got an SSIM score of 0.754, which was further improved to 0.775 by Res50 U-net. The significance of this result can be seen especially well by inspecting the output of Res18 U-net for the chosen data point in Figure 4.2. All U-nets with residual blocks improve upon the simpler Pix2Pix, suggesting that the optimal blocks are closer to

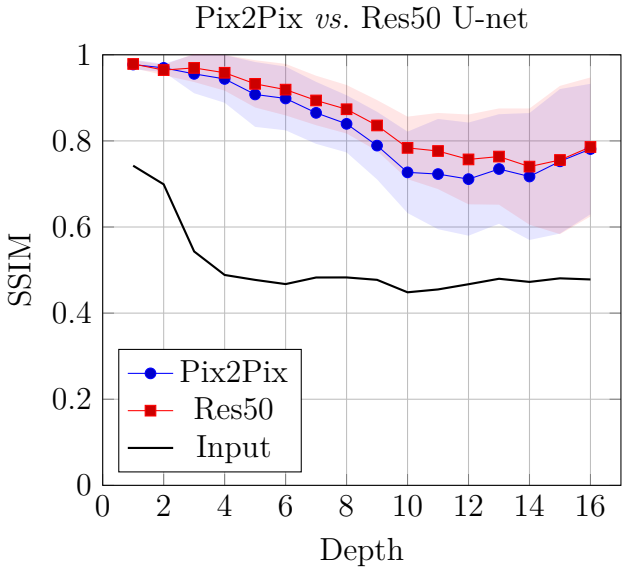


Figure 4.3: A comparison of the SSIM scores over depth on the 20 dB DRIVE dataset between Pix2Pix and the Res50 U-net with ℓ_{SSIM}, which is the best performing model for this dataset's noise level.

identity than zero mappings. However, Attention U-net does not perform better than Pix2Pix. Thus the attention gates in the skip-connections of its architecture are not important for PAI reconstruction. This could be due to Attention U-net originally being used for segmentation. Furthermore, Trans U-net does not improve the performance on the individual noise level DRIVE datasets much but does have a good performance on the NNE dataset and when trained on all noise levels of the DRIVE dataset. This is likely due to transformers needing a large amount of data to converge properly due to their large number of parameters (Hassani et al. 2022).

A visual comparison of the model outputs (Appendix D) shows that the residual models capture the output better than the simple Pix2Pix model. In Figure 4.2, we observe that Res18 U-net can capture a deep part of the output that Pix2Pix completely missed.

Overall, the best loss functions are the SSIM and PSNR loss functions. For most models, these give the best performance. Thus, an adversarial network is not necessary for PAI reconstruction. This result is significant because it allows for smoother and less problematic convergence, which solves many problems resulting from using an adversarial network.

The second problem this project wanted to improve was that PAI reconstruction

does not work well for deep imaging due to the low optical propagation depth of light. In order to compare how the models perform over depth, their outputs are partitioned vertically, and the SSIM is computed for the patches of the partition. As we go further down vertically in a PAI image, the depth increases. Thus by computing SSIM for these patches, we know how the SSIM score behaves over depth. In Appendix E, plots of the SSIM over depth for all models can be found. Two of them are shown in Figure 4.3, which compares the SSIM over depth between Pix2Pix and Res50 U-net with SSIM loss function on the DRIVE 20 dB dataset. This figure shows that the SSIM only increases slightly in the shallow parts, but as the image grows deeper, the SSIM increases markedly. The deepest parts (13–16) are not interesting due to this part being black in most data points.

The diffusion model Palette does not perform well on the PAI reconstruction task. This problem is likely due to it translating between two densities rather than translating in a more specific manner. This problem also grows more prominent as the number of timesteps in the diffusion process is increased. The performance is much worse than that of a U-net that directly translates one image to another, suggesting that the intermediary steps are unnecessary for highly detailed tasks like PAI reconstruction.

Experimental dataset The performance on the experimental dataset is not as good as that on the simulated datasets, which is not surprising when comparing the simulated images to the experimental images. The experimental images have high intensity at the shallow end, while the simulated data have the same intensity throughout the entire image. This is likely why the model with the highest SSIM score on the experimental data is Pix2Pix because it is the only model that captures the increased intensity at the top. Recall that one of the components of SSIM is luminance. So, we must resort to visual inspection to compare the models fairly based on captured structure.

Almost all models trained on a single noise level have artefacts in the form of colour or pixelated outputs. However, the models trained on all noise levels do not show any artefacts. Logically, these models perform better because it is impossible to know the noise level of experimental data. When comparing the different models trained on all noise levels in Figures B.6 and B.12, it is apparent that the residual U-nets and Trans U-net perform better than Pix2Pix and Attention U-net because they output more of the structure. However, it is not clear whether residual U-net or Trans U-net performs best. Nevertheless, the SSIM loss function performs best by a small margin because it captures the increased intensity best.

	10 dB		20 dB		30 dB		40 dB		50 dB		10–50 dB	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Input image	0.213	22.715	0.504	24.705	0.826	25.790	0.832	25.810	0.833	25.824	0.642	24.969
Pix2Pix												
ℓ_{GAN}	0.754	24.085	*0.829*	26.563	0.897	30.201	0.912	31.396	0.906	31.044	*0.831*	*28.298*
ℓ_{MSE}	0.662	*24.396*	0.812	26.584	0.879	30.077	0.903	30.604	0.906	31.141	0.810	26.852
ℓ_{PSNR}	0.747	24.243	0.828	26.746	*0.902*	*31.091*	0.911	*31.591*	*0.914*	*32.302*	0.814	27.695
ℓ_{SSIM}	0.754	18.168	0.822	20.092	0.896	29.596	*0.912*	20.314	0.893	29.933	0.827	27.785
$\ell_{\text{S+P}}$	*0.760*	24.019	0.828	*26.963*	0.893	29.673	0.909	30.490	0.888	29.489	0.825	28.136
Att. U-net												
ℓ_{GAN}	0.751	23.961	0.814	26.379	0.892	30.579	0.902	28.953	0.908	31.467	0.819	27.532
ℓ_{MSE}	0.685	24.195	0.812	26.332	0.894	30.370	0.912	31.323	0.912	*31.821*	0.820	*28.459*
ℓ_{PSNR}	0.761	*24.213*	0.826	*26.910*	*0.904*	*30.864*	*0.916*	*31.724*	0.901	30.014	0.826	27.888
ℓ_{SSIM}	0.752	19.445	*0.831*	24.963	0.896	26.480	0.887	24.402	*0.918*	30.145	*0.836*	27.972
$\ell_{\text{S+P}}$	*0.765*	24.086	0.830	26.626	0.903	30.790	0.893	30.430	0.913	30.972	0.829	28.112
Res18 U-net												
ℓ_{GAN}	0.762	24.377	0.840	27.198	0.915	31.924	0.923	33.244	0.929	*33.719*	0.861	30.320
ℓ_{MSE}	0.757	24.088	0.832	27.014	0.915	32.281	0.928	33.245	0.923	32.975	0.853	30.181
ℓ_{PSNR}	0.741	*24.438*	0.839	*27.585*	0.921	32.213	0.929	33.792	0.926	33.550	0.862	30.295
ℓ_{SSIM}	0.762	24.120	*0.850*	27.473	0.922	32.043	*0.934*	33.376	**0.934**	33.600	*0.867*	29.993
$\ell_{\text{S+P}}$	*0.764*	24.319	0.847	27.431	**0.926**	*32.480*	0.933	**34.040**	0.933	33.535	0.861	*30.337*
Res50 U-net												
ℓ_{GAN}	0.761	24.044	0.836	26.987	0.919	32.094	0.924	*33.430*	0.909	32.039	0.860	30.045
ℓ_{MSE}	0.732	24.120	0.831	27.299	0.915	*32.165*	0.924	32.843	0.929	33.172	0.847	29.451
ℓ_{PSNR}	0.750	24.296	0.834	27.223	0.916	32.094	0.928	33.417	0.927	*33.235*	0.856	*30.233*
ℓ_{SSIM}	**0.775**	24.248	**0.854**	*27.377*	0.918	31.796	*0.929*	33.045	*0.930*	33.127	0.863	29.919
$\ell_{\text{S+P}}$	0.770	*24.342*	0.840	27.059	*0.920*	32.079	0.921	32.416	0.923	32.912	*0.865*	29.976
ResV2 U-net												
ℓ_{GAN}	0.763	24.291	0.844	27.507	0.917	32.404	0.924	33.472	0.927	33.878	0.863	30.169
ℓ_{MSE}	0.716	**24.468**	0.833	27.209	0.920	32.347	0.932	33.568	0.932	33.669	0.858	30.061
ℓ_{PSNR}	0.757	24.379	0.836	**27.608**	0.920	**32.504**	0.932	33.772	0.933	**34.073**	0.862	30.327
ℓ_{SSIM}	*0.772*	24.205	*0.847*	27.354	*0.923*	31.910	0.932	33.432	0.933	33.468	*0.867*	30.027
$\ell_{\text{S+P}}$	0.766	24.363	0.846	27.503	0.922	32.079	**0.934**	*33.914*	*0.933*	33.971	0.864	*30.353*
ResNeXt U-net												
ℓ_{GAN}	0.755	24.045	0.835	27.050	0.912	*31.614*	0.920	32.538	0.916	31.937	0.853	29.299
ℓ_{MSE}	0.721	*24.309*	0.833	27.140	0.912	31.140	0.924	*33.455*	*0.929*	*33.488*	0.849	29.275
ℓ_{PSNR}	0.751	24.182	0.839	27.264	0.911	31.276	0.919	32.224	0.927	32.642	0.855	30.110
ℓ_{SSIM}	*0.772*	24.221	*0.850*	27.324	*0.913*	31.167	*0.925*	32.774	0.927	32.218	0.839	14.955
$\ell_{\text{S+P}}$	0.766	24.201	0.842	*27.348*	0.912	31.597	0.902	30.875	0.920	32.249	*0.862*	*30.256*
Trans U-net												
ℓ_{GAN}	0.750	23.783	0.832	27.138	0.893	31.559	0.922	33.085	0.897	31.932	0.865	30.314
ℓ_{MSE}	0.661	*24.040*	0.812	26.472	0.898	30.668	0.917	33.179	0.920	33.145	0.827	28.599
ℓ_{PSNR}	0.724	22.499	0.815	27.191	0.916	*32.166*	0.930	*33.879*	*0.931*	*33.480*	0.838	29.746
ℓ_{SSIM}	*0.760*	23.791	0.836	27.019	0.911	31.129	0.917	32.165	0.912	31.037	**0.877**	**30.372**
$\ell_{\text{S+P}}$	0.757	24.030	*0.843*	*27.391*	*0.917*	32.098	*0.930*	33.469	0.930	33.412	0.872	30.227
Palette	0.703	23.974	0.168	19.879	0.806	32.015	0.676	32.400	0.541	31.630		

Table 4.2: Performance metrics of all models with their various loss functions for the DRIVE dataset. The maximum per model is italicised, while the global maximum per noise level is bolded.

	10 dB		20 dB		30 dB		40 dB		50 dB		10–50 dB	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Input image	0.536	28.841	0.811	31.458	0.937	32.274	0.938	32.313	0.938	32.331	0.832	31.443
Pix2Pix												
ℓ_{GAN}	0.909	30.561	*0.922*	*32.020*	*0.958*	*35.826*	0.943	*34.201*	0.932	32.769	*0.831*	*28.298*
ℓ_{MSE}	0.790	30.233	0.913	31.172	0.923	32.582	0.917	31.239	0.949	33.722	0.810	26.852
ℓ_{PSNR}	0.846	29.855	0.900	31.978	0.929	31.895	0.928	31.856	*0.951*	*35.102*	0.814	27.695
ℓ_{SSIM}	0.899	19.051	0.914	19.115	0.944	19.234	*0.945*	23.724	0.929	19.192	0.827	27.785
$\ell_{\text{S+P}}$	*0.916*	*31.157*	0.918	31.609	0.939	32.918	0.936	32.605	0.936	32.531	0.825	28.136
Att. U-net												
ℓ_{GAN}	0.899	29.519	0.928	*33.089*	*0.942*	27.852	0.932	32.569	0.942	33.850	0.752	21.951
ℓ_{MSE}	0.897	29.127	0.890	30.058	0.934	32.448	0.942	33.155	*0.956*	*35.171*	0.938	36.317
ℓ_{PSNR}	0.897	*30.861*	*0.929*	32.780	0.940	32.805	*0.954*	*35.075*	0.946	33.435	0.939	*38.435*
ℓ_{SSIM}	*0.909*	30.537	0.866	19.149	0.937	31.102	0.949	27.419	0.941	29.086	0.951	35.442
$\ell_{\text{S+P}}$	0.904	30.119	0.914	29.789	0.939	*34.046*	0.941	33.457	0.945	33.983	*0.955*	37.199
Res18 U-net												
ℓ_{GAN}	0.905	30.916	0.945	34.125	0.959	36.241	0.966	38.267	0.962	36.893	0.861	30.320
ℓ_{MSE}	0.864	31.416	0.943	*34.976*	0.960	36.561	0.955	35.164	0.962	37.084	0.853	30.181
ℓ_{PSNR}	0.867	30.683	0.945	34.568	0.968	37.930	0.971	38.603	0.962	36.272	0.862	30.295
ℓ_{SSIM}	**0.926**	31.638	*0.955*	34.936	0.965	37.423	0.944	35.023	*0.971*	*38.370*	*0.867*	29.993
$\ell_{\text{S+P}}$	0.923	*31.698*	0.942	34.146	*0.977*	*40.214*	*0.976*	*39.755*	0.934	32.610	0.861	*30.337*
Res50 U-net												
ℓ_{GAN}	0.910	31.134	0.951	34.872	0.970	38.541	0.952	34.900	0.954	36.005	0.860	30.045
ℓ_{MSE}	0.867	30.680	0.943	*35.808*	0.968	39.056	0.951	35.692	0.976	*40.377*	0.847	29.451
ℓ_{PSNR}	0.875	30.539	0.946	34.830	0.955	35.390	0.944	33.400	0.957	39.534	0.856	*30.233*
ℓ_{SSIM}	*0.920*	31.192	0.950	34.408	*0.977*	*39.401*	0.917	34.437	0.961	36.335	0.863	29.919
$\ell_{\text{S+P}}$	0.920	*31.357*	*0.951*	35.756	0.965	37.592	*0.959*	*36.828*	*0.977*	39.534	*0.865*	29.976
ResV2 U-net												
ℓ_{GAN}	0.919	31.601	0.954	35.213	0.969	38.022	0.976	40.729	*0.982*	*42.232*	0.863	30.169
ℓ_{MSE}	0.865	31.696	0.942	34.934	0.964	37.910	0.939	33.266	0.971	39.177	0.858	30.061
ℓ_{PSNR}	0.819	**31.773**	0.930	33.690	*0.972*	*38.861*	0.975	*40.973*	0.955	35.849	0.862	30.327
ℓ_{SSIM}	0.921	31.270	0.953	34.786	0.966	38.106	*0.978*	40.680	0.975	39.328	*0.867*	30.027
$\ell_{\text{S+P}}$	*0.923*	31.762	**0.958**	**36.040**	0.971	38.682	0.961	37.730	0.964	37.417	0.864	*30.353*
ResNeXt U-net												
ℓ_{GAN}	0.912	30.519	*0.955*	*35.333*	0.966	37.388	0.970	38.724	0.971	38.761	0.853	29.299
ℓ_{MSE}	0.861	30.461	0.944	34.478	0.971	*38.650*	0.973	*39.971*	0.978	*41.413*	0.849	29.275
ℓ_{PSNR}	0.890	30.390	0.945	33.946	0.965	37.167	0.962	36.648	0.968	38.657	0.855	30.110
ℓ_{SSIM}	*0.920*	30.945	0.950	34.100	*0.973*	37.476	*0.981*	39.721	0.969	38.246	0.839	14.955
$\ell_{\text{S+P}}$	0.919	*31.536*	0.953	34.926	0.971	38.345	0.954	35.729	*0.981*	40.512	*0.862*	*30.256*
Trans U-net												
ℓ_{GAN}	0.914	31.138	0.954	35.070	0.946	34.340	0.930	32.205	0.936	32.795	0.960	38.357
ℓ_{MSE}	0.830	31.363	0.923	35.103	0.961	39.457	0.932	32.195	0.895	29.043	0.935	38.218
ℓ_{PSNR}	0.863	31.541	0.948	35.339	0.978	41.047	**0.984**	**42.995**	**0.983**	**43.806**	0.949	39.953
ℓ_{SSIM}	*0.923*	31.431	*0.958*	35.524	0.943	33.949	0.967	36.858	0.948	33.919	**0.965**	39.385
$\ell_{\text{S+P}}$	0.920	*31.584*	0.958	*35.731*	**0.979**	**41.184**	0.951	35.320	0.977	41.628	0.963	**40.057**
Palette	0.770	28.758	0.159	15.503	0.270	22.123	0.214	27.675	0.564	29.825		

Table 4.3: Performance metrics of all models with their various loss functions for the NNE dataset. The maximum per model is italicised, while the global maximum per noise level is bolded.

Chapter 5

Discussion

In this project, solutions to the poor performance of PAI reconstruction in noisy environments and deep tissue were sought. For this, image-to-image translation models were used that take as input a poor PAI reconstruction and output a good one. Many models were compared, including variations in the loss function. An adversarial learned loss function was compared to simpler functions optimising the SSIM and PSNR scores. Apart from the diffusion model, all models performed very well on the PAI reconstruction task. They improved significantly upon the performance of the input image on both problems.

Three main types of building blocks of the U-net were compared, namely simple convolutional layers, residual layers, and attention layers. It was found that the residual layers outperformed the simple convolutional layers considerably, increasing the SSIM scores by 0.02–0.13 on simulated PAI data. Notably, the more complex models resulted in their outputs capturing more structure of their ground truth. This is likely because the optimal layers are closer to identity mappings than zero mappings for this specific task. Attention layers were also explored in the form of Trans U-net. This project found that Trans U-net requires a large amount of data because it performed acceptably on the DRIVE dataset while performing very well on the NNE dataset, which consists of significantly more data.

Lastly, this project used experimental data to evaluate how the models work on real-world data. It was found that the improvements made by the complex models are visually less pronounced when compared with the simulated data model outputs, but there are minor improvements to be found.

In conclusion, the answer to the research question is that developments in image-to-image translation models can improve the SSIM and PSNR scores of noisy deep PAI reconstruction. The most significant improvements are made in

the visual inspections when evaluating the models on experimental data. However, there is still room for even more potential improvements.

5.1 Future work

The conditional diffusion model Palette did not work well, but this is likely due to it translating from distribution to distribution rather than from specific inputs to specific outputs. To amplify the effect of the input on the outcome, future work could use classifier-free guidance (Ho & Salimans 2022), where two models are trained jointly, *i.e.* a conditional and an unconditional one. It involves running both models for every iteration and computing the difference in outputs between them. The conditioning on the input image must cause this difference. Thus we amplify this difference to steer the model toward the wanted output. This is also used by text-to-image models such as Stable Diffusion (Rombach et al. 2022) to amplify the conditioning on the input text.

Furthermore, variations of the adversarial network can be explored. In this project, the discriminator of Pix2Pix was solely used. However, more complex discriminators could possibly outperform the simple one used in this work. It could perhaps even outperform the SSIM and PSNR loss functions.

References

Attia, A. B. E., Balasundaram, G., Moothanchery, M., Dinish, U., Bi, R., Ntziachristos, V. & Olivo, M. (2019), 'A review of clinical photoacoustic imaging: Current and future trends', *Photoacoustics* **16**, 100144.

Beard, P. (2011), 'Biomedical photoacoustic imaging', *Interface focus* **1**(4), 602–631.

Chen, J., Lu, Y., Yu, Q., Luo, X., Adeli, E., Wang, Y., Lu, L., Yuille, A. L. & Zhou, Y. (2021), 'Transunet: Transformers make strong encoders for medical image segmentation'.

Choi, W., Park, B., Choi, S., Oh, D., Kim, J. & Kim, C. (2023), 'Recent advances in contrast-enhanced photoacoustic imaging: Overcoming the physical and practical challenges', *Chemical Reviews* .

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. (2021), 'An image is worth 16x16 words: Transformers for image recognition at scale'.

Hassani, A., Walton, S., Shah, N., Abuduweili, A., Li, J. & Shi, H. (2022), 'Escaping the big data paradigm with compact transformers'.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), 'Deep residual learning for image recognition'.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), 'Identity mappings in deep residual networks'.

Ho, J., Jain, A. & Abbeel, P. (2020), 'Denoising diffusion probabilistic models'.

Ho, J. & Salimans, T. (2022), 'Classifier-free diffusion guidance'.

Hore, A. & Ziou, D. (2010), Image quality metrics: Psnr vs. ssim, *in* '2010 20th international conference on pattern recognition', IEEE, pp. 2366–2369.

Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. (2018), 'Image-to-image translation with conditional adversarial networks'.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S. & Shah, M. (2022), 'Transformers in vision: A survey', *ACM Computing Surveys* **54**(10s), 1–41.

Lin, T., Wang, Y., Liu, X. & Qiu, X. (2021), 'A survey of transformers'.

Nair, V. & Hinton, G. E. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of the 27th international conference on machine learning (ICML-10)', pp. 807–814.

Nichol, A. & Dhariwal, P. (2021), 'Improved denoising diffusion probabilistic models'.

Oktay, O., Schlemper, J., Folgoc, L. L., Lee, M., Heinrich, M., Misawa, K., Mori, K., McDonagh, S., Hammerla, N. Y., Kainz, B. et al. (2018), 'Attention u-net: Learning where to look for the pancreas', *arXiv preprint arXiv:1804.03999* .

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C. & Chen, M. (2022), 'Hierarchical text-conditional image generation with clip latents', *arXiv preprint arXiv:2204.06125* .

Rombach, R., Blattmann, A., Lorenz, D., Esser, P. & Ommer, B. (2022), 'High-resolution image synthesis with latent diffusion models'.

Ronneberger, O., Fischer, P. & Brox, T. (2015), U-net: Convolutional networks for biomedical image segmentation, *in* 'Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18', Springer, pp. 234–241.

Saharia, C., Chan, W., Chang, H., Lee, C. A., Ho, J., Salimans, T., Fleet, D. J. & Norouzi, M. (2022), 'Palette: Image-to-image diffusion models'.

Saxena, D. & Cao, J. (2023), 'Generative adversarial networks (gans survey): Challenges, solutions, and future directions'.

Staal, J., Abràmoff, M. D., Niemeijer, M., Viergever, M. A. & Van Ginneken, B. (2004), 'Ridge-based vessel segmentation in color images of the retina', *IEEE transactions on medical imaging* **23**(4), 501–509.

Treeby, B. E. & Cox, B. T. (2010), 'k-wave: Matlab toolbox for the simulation and reconstruction of photoacoustic wave fields', *Journal of biomedical optics* **15**(2), 021314–021314.

Uhlirova, H., Tian, P., Kılıç, K., Thunemann, M., Sridhar, V. B., Bartsch, H., Dale, A. M., Devor, A. & Saisan, P. A. (2017), 'Neurovascular network explorer 2.0: a database of 2-photon single-vessel diameter measurements from mouse si cortex in response to optogenetic stimulation', *Frontiers in neuroinformatics* p. 4.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017), 'Attention is all you need', *Advances in neural information processing systems* **30**.

Wang, G., Ye, J. C. & De Man, B. (2020), 'Deep learning for tomographic image reconstruction', *Nature Machine Intelligence* **2**(12), 737–748.

Wang, L. V. & Yao, J. (2016), 'A practical guide to photoacoustic tomography in the life sciences', *Nature methods* **13**(8), 627–638.

Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. (2004), 'Image quality assessment: from error visibility to structural similarity', *IEEE transactions on image processing* **13**(4), 600–612.

Xie, S., Girshick, R., Dollár, P., Tu, Z. & He, K. (2017), 'Aggregated residual transformations for deep neural networks'.

Appendix A

Implementation details

Training details All models are trained with a batch-size of 8. The U-net models are trained for 6K training steps on the noise level datasets and 30K training steps on the dataset containing all noise levels 10–50 dB. The diffusion model requires more training steps due to their inherently harder task, so they are trained for 100K steps. A standard Adam optimizer is used with the U-nets using a fixed 2e−4 learning rate. Consistent with (Saharia et al. 2022), the diffusion model uses a fixed 1e−4 learning rate with a 10K linear learning rate warmup schedule and 0.9999 EMA. The checkpoint to test on is chosen by maximising the SSIM score on the validation data.

Hyperparameters Most U-net models in this project use the channel array $C = \langle 64, 128, 256, 512, 512, 512, 512, 512 \rangle$. But, to mitigate an explosion of learnable parameters, Trans U-net uses $C = \langle 64, 128, 128, 256, 256 \rangle$. During training, the diffusion model uses a cosine noise schedule with 1000 timesteps, while during inference, it uses a cosine noise schedule with 100 timesteps.

Appendix B

Experimental results

The following tables and figures summarise the results on the experimental datasets. Tables B.1 and B.2 are categorised by which noise level the models are trained on, where the maximum SSIM and PSNR are bolded. The figures contain the outputs of one experimental datapoint used to visually compare the models.

Training data →	10 dB		20 dB		30 dB		40 dB		50 dB		10–50 dB	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Input image	0.683	23.880	0.683	23.880	0.683	23.880	0.683	23.880	0.683	23.880	0.683	23.880
Pix2Pix												
ℓ_{GAN}	0.585	19.130	0.645	20.927	0.680	23.006	0.650	22.221	0.586	21.913	0.683	24.330
ℓ_{MSE}	0.596	19.029	0.680	22.017	0.693	24.131	0.666	23.778	**0.705**	23.669	0.659	22.399
ℓ_{PSNR}	0.548	19.083	0.693	22.825	0.691	24.183	0.681	23.851	0.645	23.910	0.672	22.918
ℓ_{SSIM}	0.576	16.079	0.654	17.887	0.673	18.688	0.636	21.814	0.648	23.103	0.653	22.691
$\ell_{\text{S+P}}$	0.595	19.453	0.676	22.898	0.688	24.306	0.686	24.094	0.604	23.395	0.658	22.548
Att. U-net												
ℓ_{GAN}	0.584	19.635	0.628	20.905	0.673	23.255	0.640	21.486	0.677	21.332	0.666	23.678
ℓ_{MSE}	0.612	20.451	0.643	22.398	0.676	23.142	0.661	23.014	0.654	23.091	0.657	22.543
ℓ_{PSNR}	0.582	20.116	0.648	22.520	0.667	22.675	0.586	22.500	0.624	22.428	0.653	22.433
ℓ_{SSIM}	0.583	18.983	0.647	20.444	0.673	21.832	0.579	21.740	0.651	21.441	0.670	23.584
$\ell_{\text{S+P}}$	0.593	19.799	0.663	23.047	0.666	23.006	0.654	23.022	0.650	22.646	0.667	23.309
Res18 U-net												
ℓ_{GAN}	0.593	20.852	0.655	23.768	0.679	24.152	0.598	22.900	0.612	23.513	0.671	23.728
ℓ_{MSE}	0.625	21.160	0.641	22.128	0.670	23.812	0.615	22.798	0.624	23.156	0.658	23.454
ℓ_{PSNR}	0.622	21.364	0.666	23.733	0.658	23.859	0.567	22.634	0.590	22.910	0.673	23.576
ℓ_{SSIM}	0.609	21.601	0.664	23.945	0.683	24.660	0.600	23.221	0.576	22.957	0.672	23.942
$\ell_{\text{S+P}}$	0.608	21.392	0.673	23.836	0.681	24.077	0.570	22.605	0.625	23.232	0.677	23.835
Res50 U-net												
ℓ_{GAN}	0.538	17.976	0.604	21.527	0.675	24.307	0.651	23.379	0.640	23.548	0.649	23.122
ℓ_{MSE}	0.564	18.898	0.628	21.895	0.692	23.886	0.655	23.784	0.679	24.159	0.690	23.657
ℓ_{PSNR}	0.593	20.474	0.655	23.011	0.673	23.863	0.638	23.393	0.628	23.286	0.679	24.027
ℓ_{SSIM}	0.596	20.568	0.646	23.291	0.672	24.303	0.622	23.335	0.672	23.828	0.667	24.310
$\ell_{\text{S+P}}$	0.536	17.698	0.629	22.897	0.680	23.860	0.604	23.181	0.608	23.235	0.670	24.041
ResV2 U-net												
ℓ_{GAN}	0.534	17.527	0.623	23.268	0.686	24.181	0.628	22.417	0.624	23.199	0.676	23.455
ℓ_{MSE}	0.588	19.787	0.640	23.231	0.692	24.341	0.653	23.700	0.623	23.954	0.677	23.181
ℓ_{PSNR}	0.617	21.905	0.649	23.838	0.674	23.708	0.616	23.355	0.613	22.993	0.690	23.421
ℓ_{SSIM}	0.572	19.756	0.659	23.911	0.697	24.555	0.628	23.824	0.634	23.696	0.690	24.546
$\ell_{\text{S+P}}$	0.566	19.408	0.662	23.902	0.685	23.660	0.611	23.530	0.627	23.500	0.673	23.507
ResNeXt U-net												
ℓ_{GAN}	0.555	17.131	0.670	22.749	0.674	23.971	0.599	23.013	0.618	17.960	0.679	23.846
ℓ_{MSE}	0.544	18.700	0.632	22.088	0.702	23.856	0.677	23.690	0.650	22.594	0.695	23.898
ℓ_{PSNR}	0.492	17.647	0.669	23.593	0.685	24.036	0.606	23.337	0.674	23.850	0.681	23.470
ℓ_{SSIM}	0.528	13.065	0.613	13.836	0.675	14.484	0.591	22.177	0.596	22.625	0.663	14.551
$\ell_{\text{S+P}}$	0.566	18.351	0.632	23.167	0.687	24.111	0.591	23.244	0.631	23.681	0.699	24.507
Trans U-net												
ℓ_{GAN}	0.594	19.763	0.639	22.875	0.678	23.846	0.627	22.689	0.656	22.836	0.675	23.531
ℓ_{MSE}	0.574	19.677	0.665	22.742	0.653	23.124	0.642	22.857	0.698	23.342	0.692	23.164
ℓ_{PSNR}	0.572	19.853	0.647	22.652	0.663	23.040	0.620	22.324	0.635	22.949	0.685	22.805
ℓ_{SSIM}	0.594	19.158	0.669	23.168	0.636	23.457	0.638	23.454	0.652	23.181	0.662	23.342
$\ell_{\text{S+P}}$	0.550	19.066	0.666	23.461	0.678	23.505	0.621	22.660	0.572	22.066	0.694	23.187
Palette	0.622	21.538	0.634	23.793	0.672	**24.739**	0.556	22.451	0.575	23.033		

Table B.1: Experimental performance metrics of all models with their various loss function trained on the DRIVE dataset.

Training data →	10 dB		20 dB		30 dB		40 dB		50 dB		10–50 dB	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Input image	0.733	26.569	0.733	26.569	0.733	26.569	0.733	26.569	0.733	26.569	0.733	26.569
Pix2Pix												
ℓ_{GAN}	0.630	20.461	0.713	20.830	0.732	23.766	0.727	24.616	0.674	24.730	0.655	24.474
ℓ_{MSE}	0.707	23.001	0.727	23.642	**0.756**	25.545	0.697	24.084	0.731	25.441	0.533	25.328
ℓ_{PSNR}	0.685	21.654	0.734	25.276	0.729	23.269	0.710	24.047	0.724	24.821	0.686	23.473
ℓ_{SSIM}	0.679	18.965	0.687	17.167	0.715	18.128	0.699	18.098	0.597	19.049	0.717	25.009
$\ell_{\text{S+P}}$	0.638	22.417	0.735	23.524	0.740	25.566	0.735	25.236	0.740	25.099	0.734	26.534
Att. U-net												
ℓ_{GAN}	0.657	20.943	0.675	22.302	0.719	24.036	0.614	23.238	0.710	24.724	0.557	18.761
ℓ_{MSE}	0.677	24.240	0.719	25.627	0.737	24.962	0.711	25.165	0.721	24.624	0.639	25.092
ℓ_{PSNR}	0.699	23.789	0.719	25.389	0.685	24.279	0.694	24.997	0.679	24.988	0.586	24.624
ℓ_{SSIM}	0.673	22.811	0.702	17.972	0.695	15.192	0.659	21.719	0.708	23.248	0.741	23.621
$\ell_{\text{S+P}}$	0.652	21.634	0.720	24.597	0.715	24.323	0.654	23.356	0.725	25.116	0.677	24.707
Res18 U-net												
ℓ_{GAN}	0.641	21.240	0.719	25.800	0.722	25.583	0.698	25.242	0.698	25.266	0.465	24.754
ℓ_{MSE}	0.695	22.135	0.735	25.501	0.720	25.345	0.615	24.112	0.728	25.704	0.746	**26.871**
ℓ_{PSNR}	0.684	22.209	0.736	25.776	0.645	25.050	0.692	25.517	0.649	24.693	0.745	26.286
ℓ_{SSIM}	0.702	22.221	0.736	25.398	0.733	25.999	0.623	24.487	0.655	24.879	0.711	24.693
$\ell_{\text{S+P}}$	0.659	21.984	0.680	24.669	0.670	24.695	0.655	25.395	0.641	24.831	0.716	26.656
Res50 U-net												
ℓ_{GAN}	0.641	20.395	0.719	25.456	0.681	25.190	0.701	25.450	0.702	25.414	0.734	26.833
ℓ_{MSE}	0.653	21.694	0.754	25.197	0.696	25.317	0.718	26.083	0.700	25.871	0.548	26.141
ℓ_{PSNR}	0.669	21.324	0.742	25.622	0.710	25.429	0.707	26.104	0.664	25.568	0.714	26.764
ℓ_{SSIM}	0.705	22.933	0.735	25.730	0.696	25.238	0.679	25.553	0.695	25.430	0.740	26.436
$\ell_{\text{S+P}}$	0.703	24.082	0.745	25.667	0.720	26.166	0.694	25.541	0.638	25.003	0.752	26.807
ResV2 U-net												
ℓ_{GAN}	0.666	21.679	0.717	25.773	0.674	25.478	0.655	25.114	0.660	25.539	0.712	25.720
ℓ_{MSE}	0.684	22.824	0.706	25.634	0.689	25.476	0.602	24.336	0.641	24.968	0.715	24.941
ℓ_{PSNR}	0.645	20.620	0.731	25.790	0.726	26.084	0.666	25.290	0.653	24.102	0.713	25.678
ℓ_{SSIM}	0.698	23.369	0.753	25.848	0.672	25.153	0.653	25.342	0.609	24.328	0.745	25.509
$\ell_{\text{S+P}}$	0.702	23.398	0.699	25.155	0.689	25.329	0.619	24.644	0.618	24.409	0.684	25.853
ResNeXt U-net												
ℓ_{GAN}	0.660	16.857	0.725	23.594	0.737	26.481	0.584	18.070	0.687	25.070	0.623	24.378
ℓ_{MSE}	0.656	20.990	0.752	24.368	0.689	24.683	0.675	25.643	0.640	24.741	0.713	26.466
ℓ_{PSNR}	0.665	21.341	0.735	26.082	0.733	25.925	0.718	25.927	0.687	25.469	0.729	26.506
ℓ_{SSIM}	0.656	17.067	0.699	16.289	0.652	16.019	0.627	14.174	0.594	16.274	0.662	17.387
$\ell_{\text{S+P}}$	0.677	21.644	0.736	25.311	0.736	26.009	0.659	25.383	0.675	25.608	0.729	26.496
Trans U-net												
ℓ_{GAN}	0.672	21.986	0.724	25.309	0.669	24.281	0.722	25.843	0.725	26.029	0.704	25.546
ℓ_{MSE}	0.652	20.919	0.736	25.413	0.553	21.086	0.687	25.566	0.673	25.193	0.700	25.852
ℓ_{PSNR}	0.626	21.633	0.712	25.069	0.665	24.831	0.647	24.444	0.696	25.540	0.696	25.272
ℓ_{SSIM}	0.690	22.370	0.747	25.896	0.666	25.066	0.639	24.579	0.654	24.702	0.642	24.730
$\ell_{\text{S+P}}$	0.665	21.793	0.702	25.205	0.679	25.263	0.613	23.987	0.625	24.353	0.690	25.154
Palette	0.673	23.175	0.400	18.419	0.543	22.735	0.445	23.194	0.584	23.676		

Table B.2: Experimental performance metrics of all models with their various loss function trained on the NNE dataset.

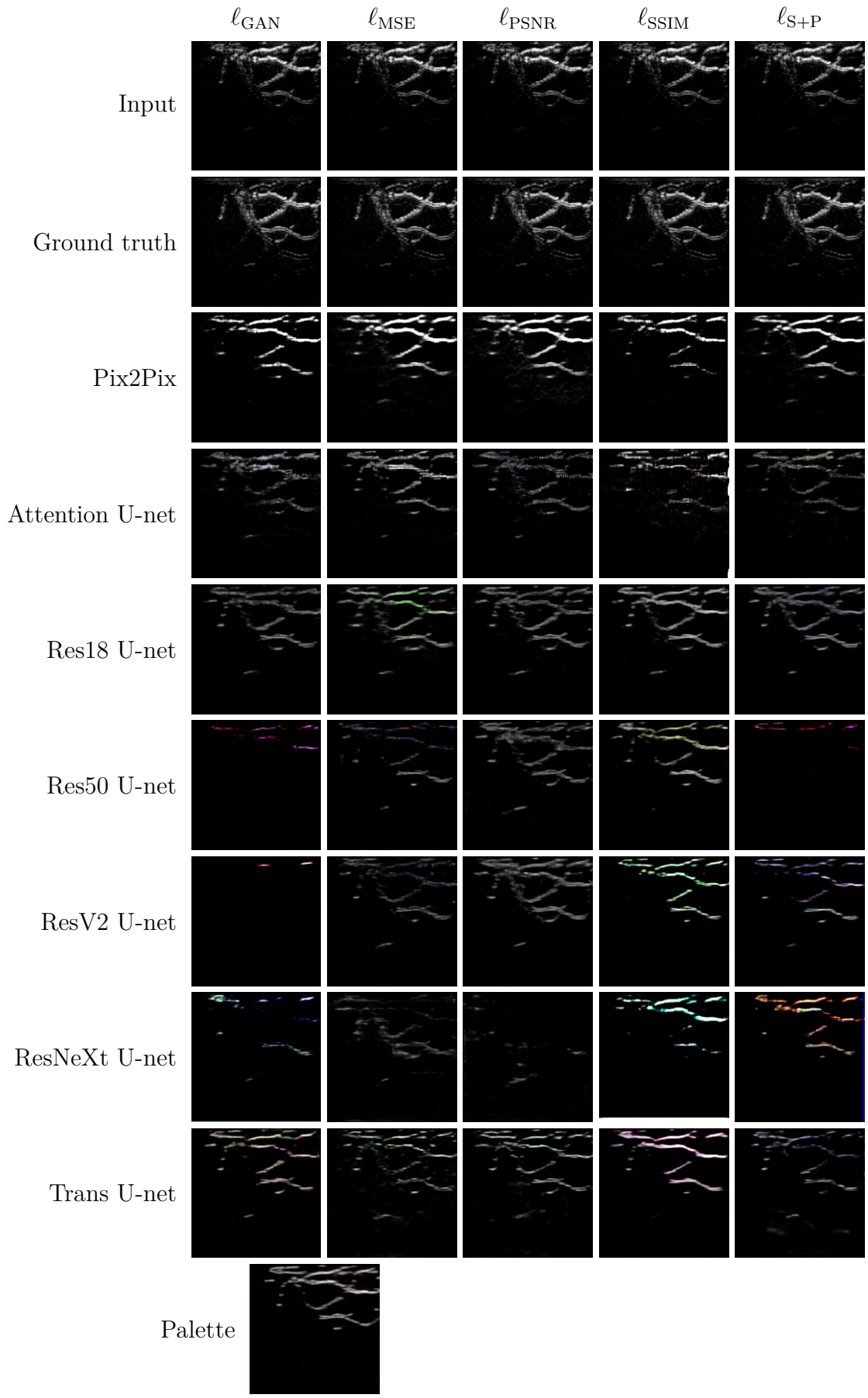


Figure B.1: A visual inspection of the outputs of the models trained and tested on the Experimental DRIVE 10 dB dataset with their various loss functions.

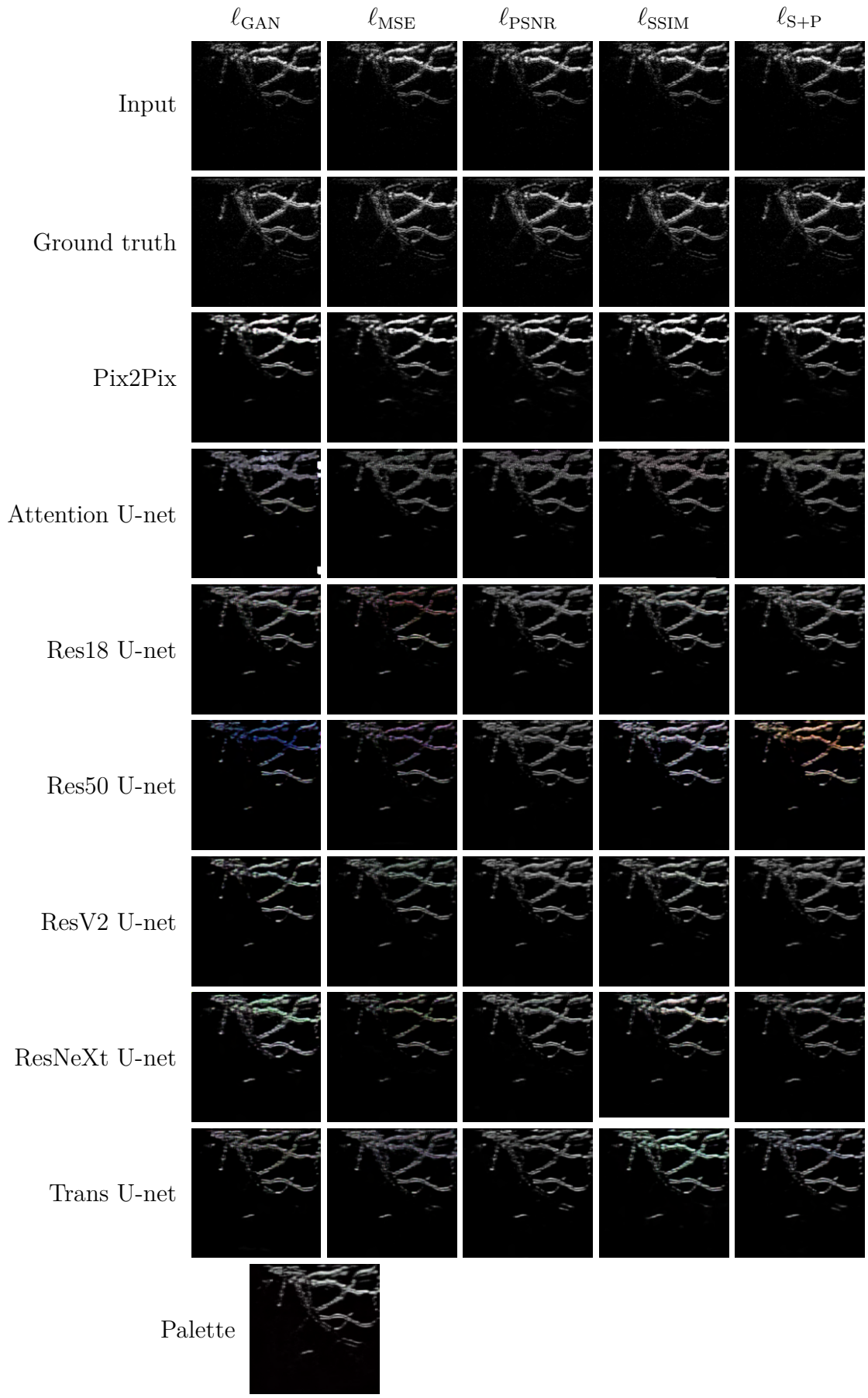


Figure B.2: A visual inspection of the outputs of the models trained and tested on the Experimental DRIVE 20 dB dataset with their various loss functions.

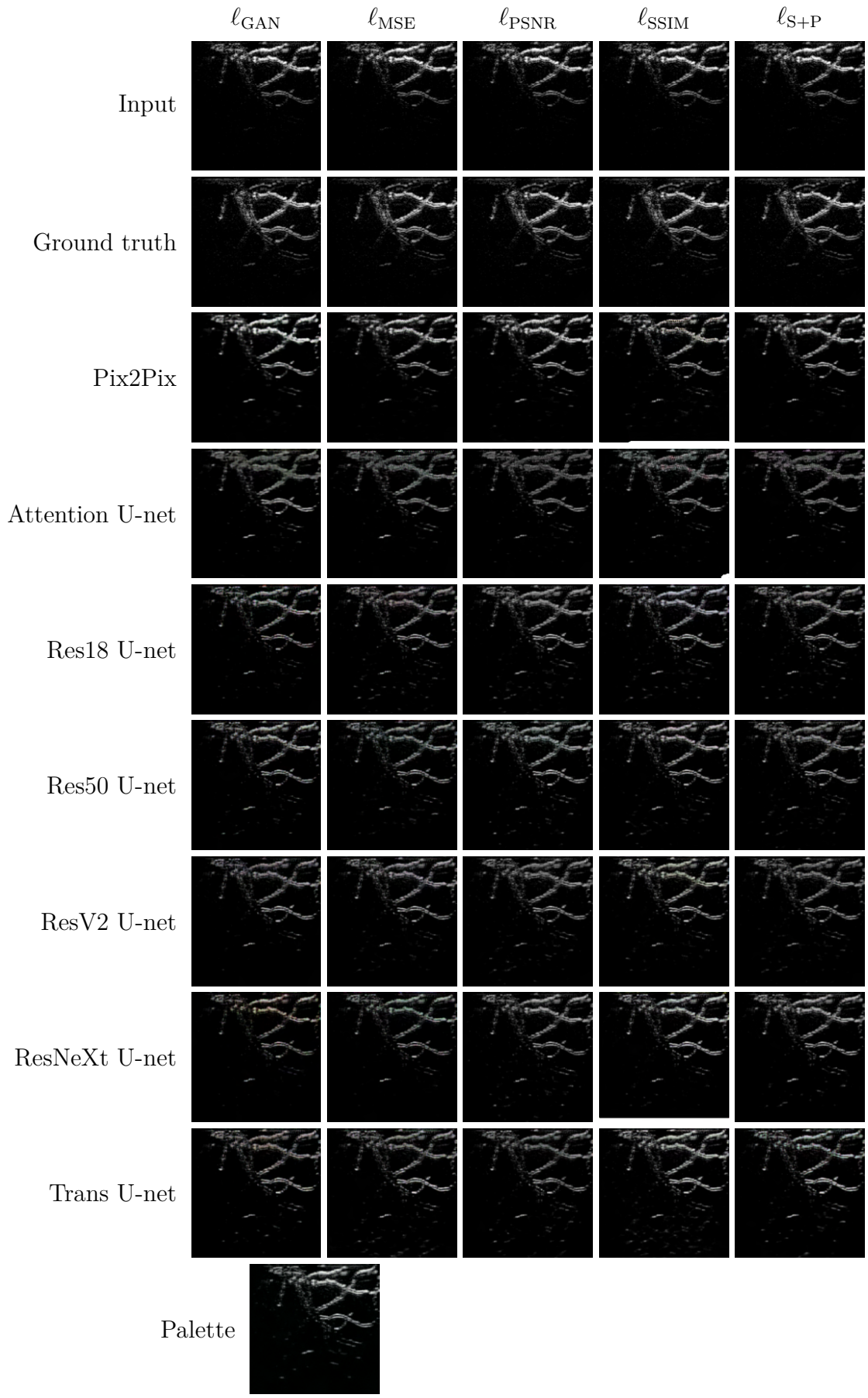


Figure B.3: A visual inspection of the outputs of the models trained and tested on the Experimental DRIVE 30 dB dataset with their various loss functions.

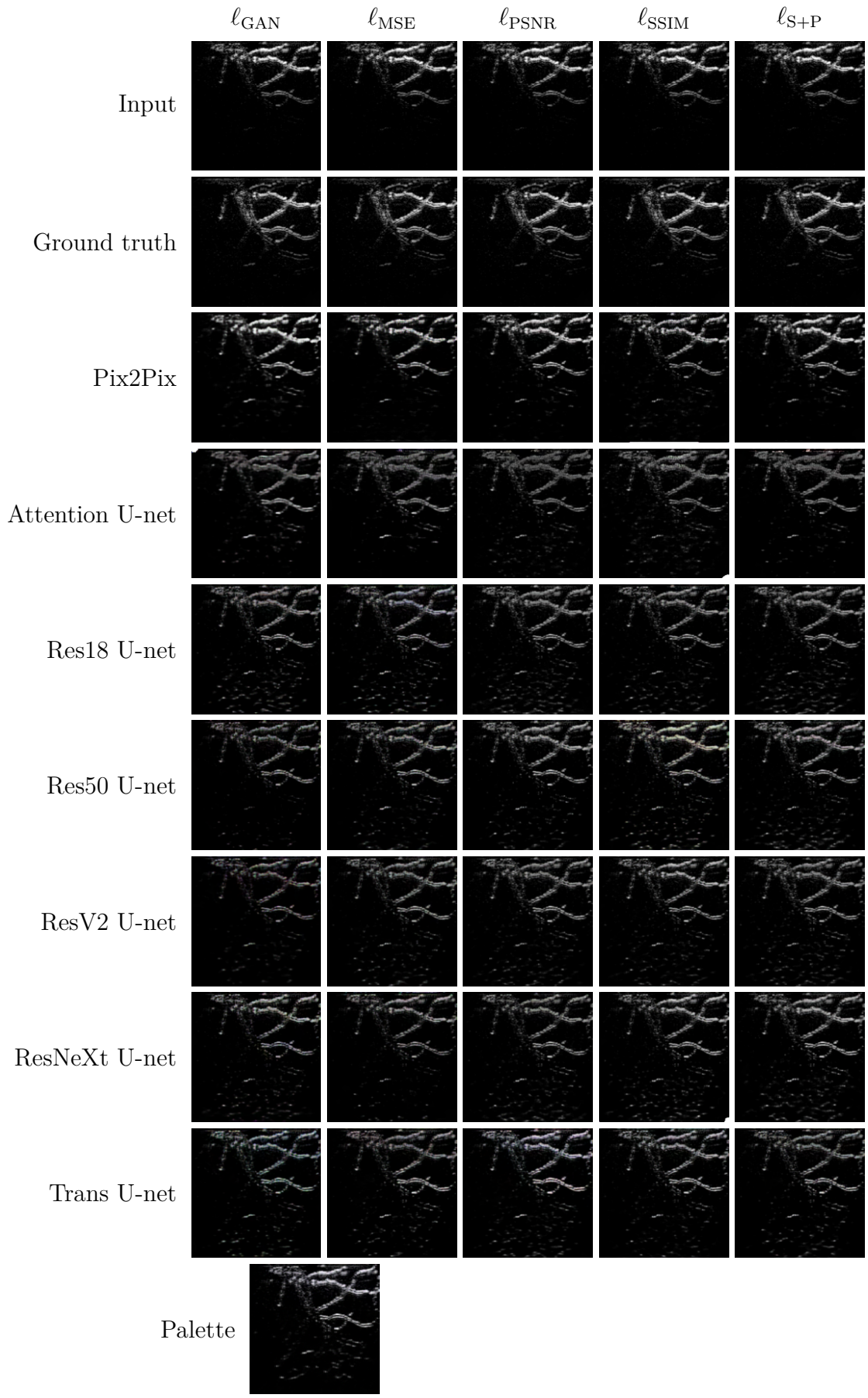


Figure B.4: A visual inspection of the outputs of the models trained and tested on the Experimental DRIVE 40 dB dataset with their various loss functions.

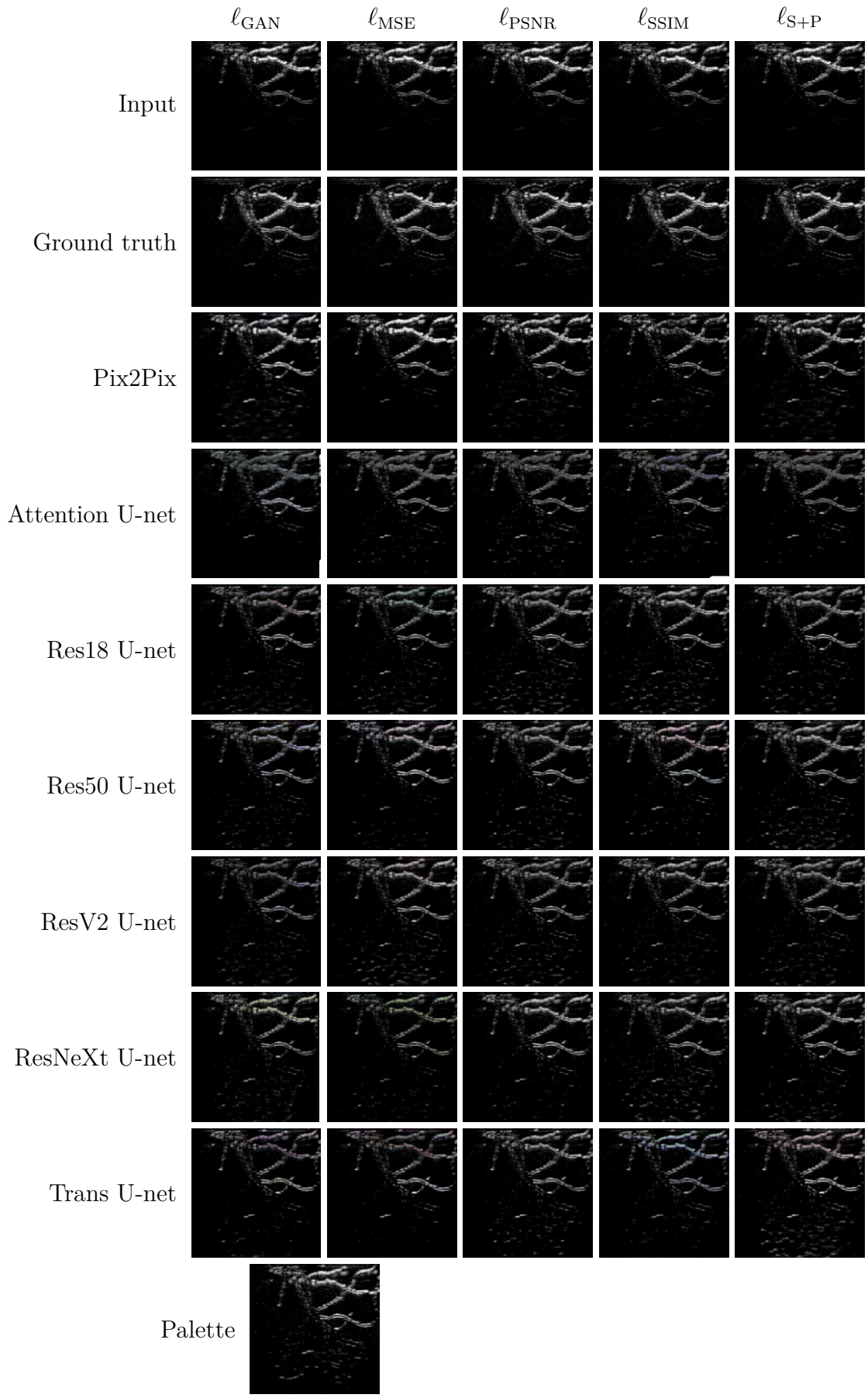


Figure B.5: A visual inspection of the outputs of the models trained and tested on the Experimental DRIVE 50 dB dataset with their various loss functions.

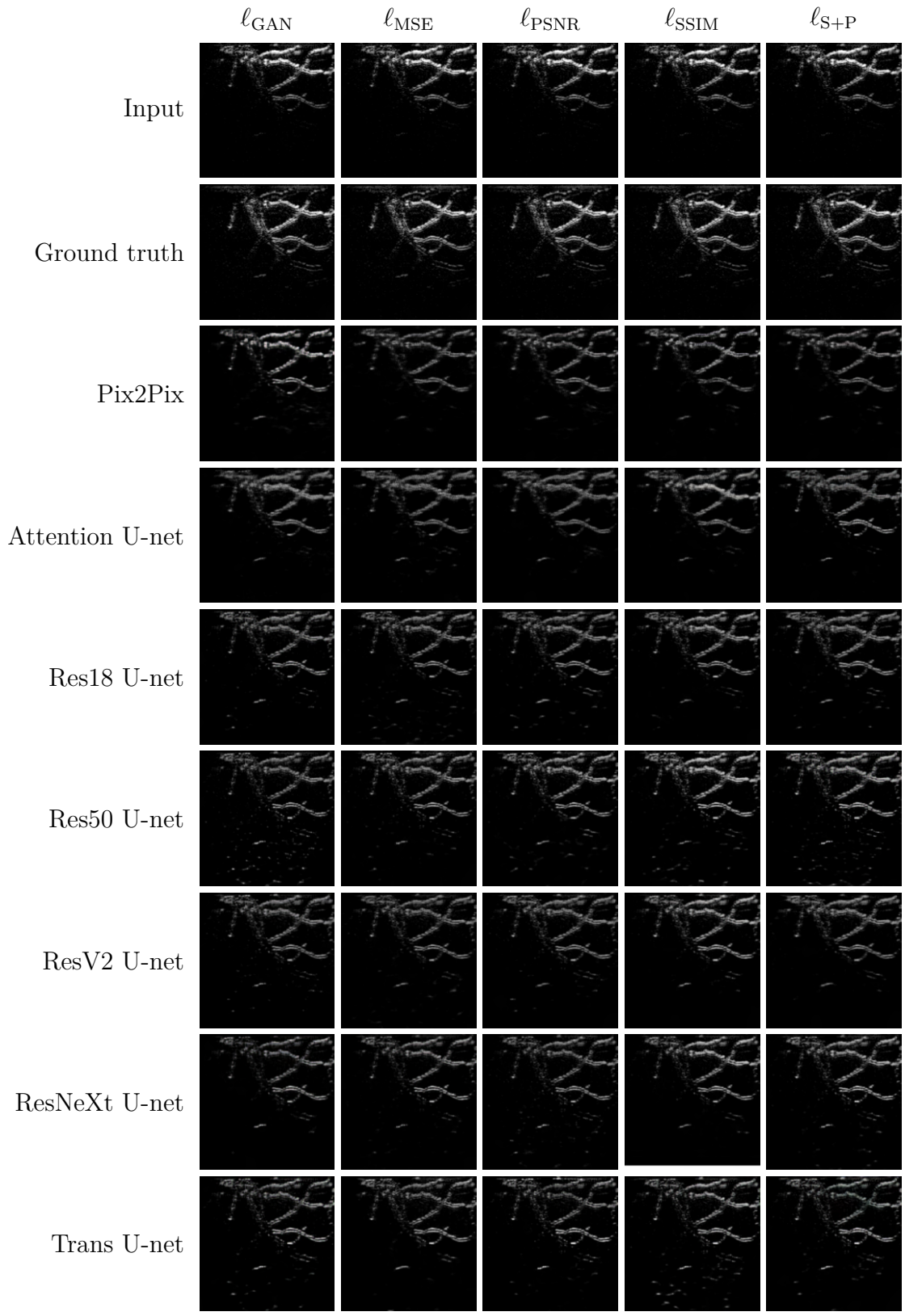


Figure B.6: A visual inspection of the outputs of the models trained and tested on the Experimental DRIVE 10–50 dB dataset with their various loss functions.

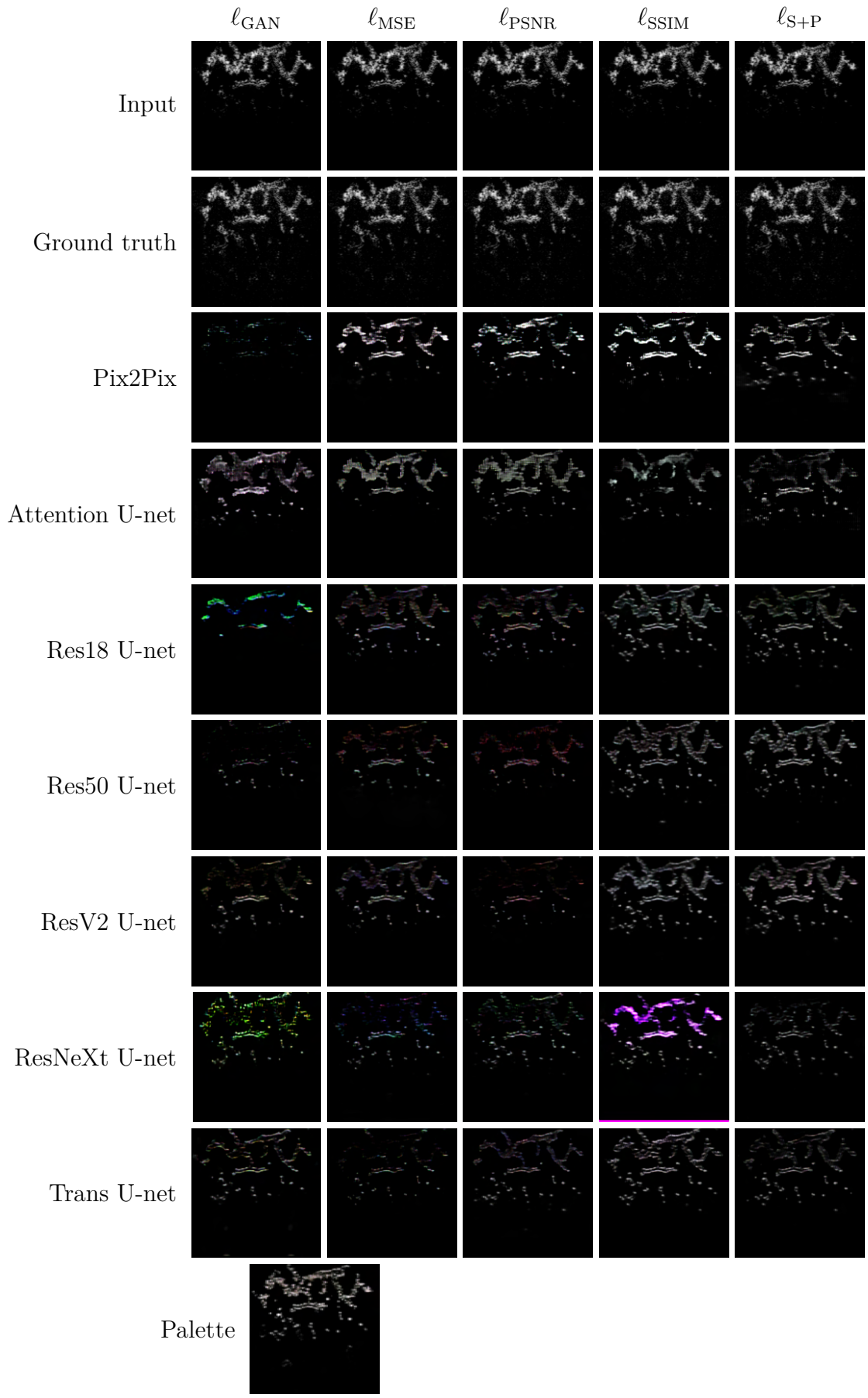


Figure B.7: A visual inspection of the outputs of the models trained and tested on the Experimental NNE 10 dB dataset with their various loss functions.

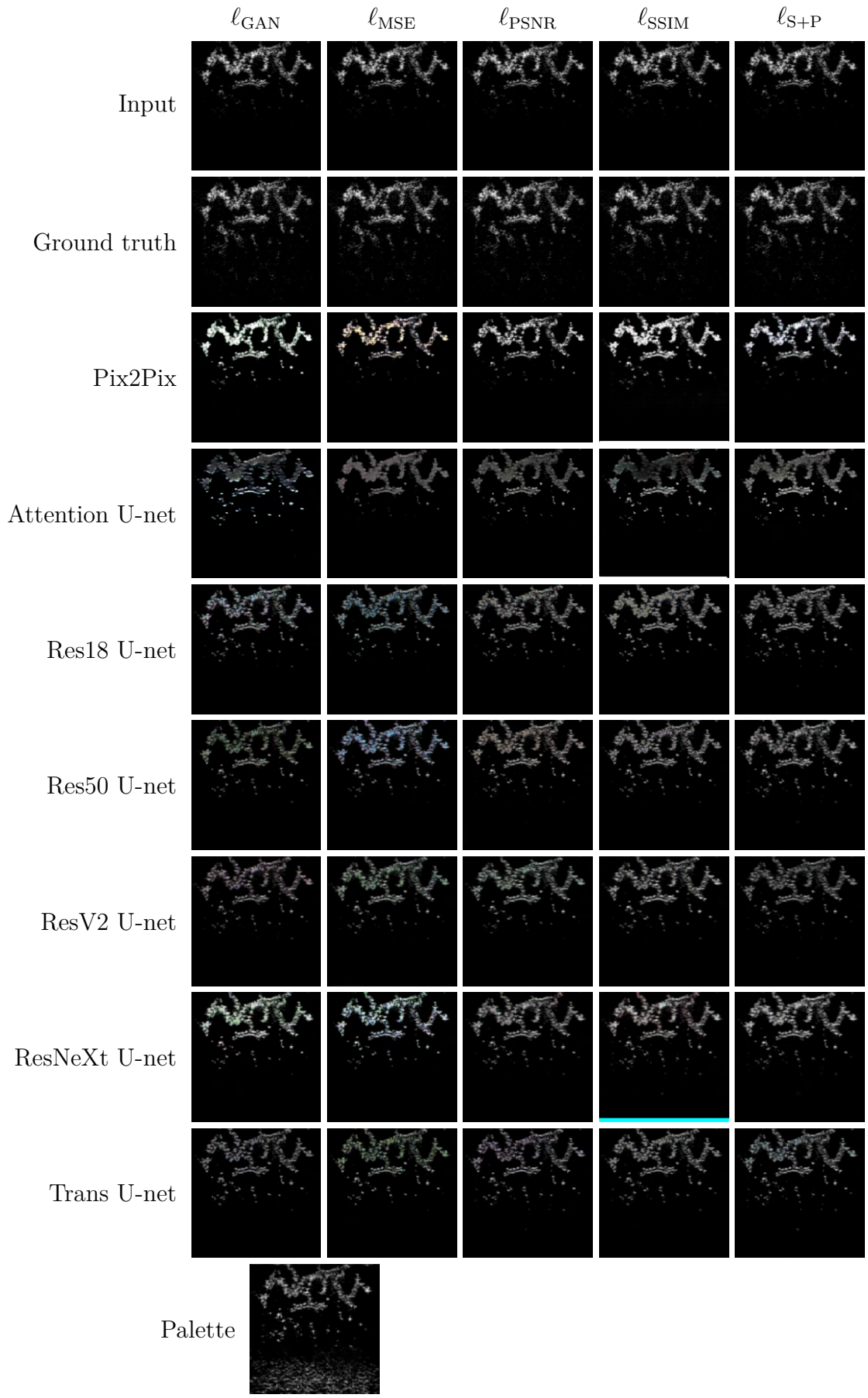


Figure B.8: A visual inspection of the outputs of the models trained and tested on the Experimental NNE 20 dB dataset with their various loss functions.

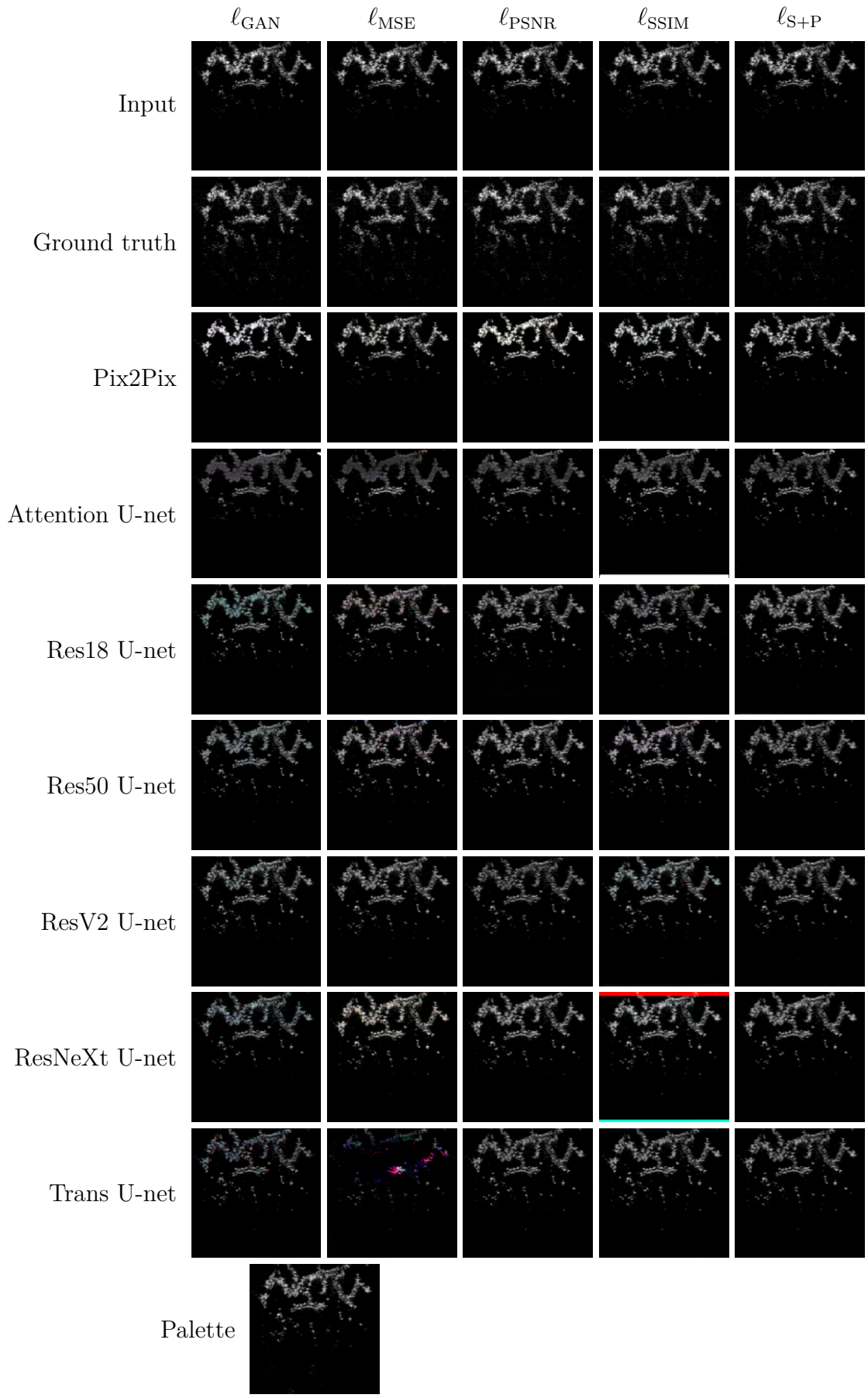


Figure B.9: A visual inspection of the outputs of the models trained and tested on the Experimental NNE 30 dB dataset with their various loss functions.

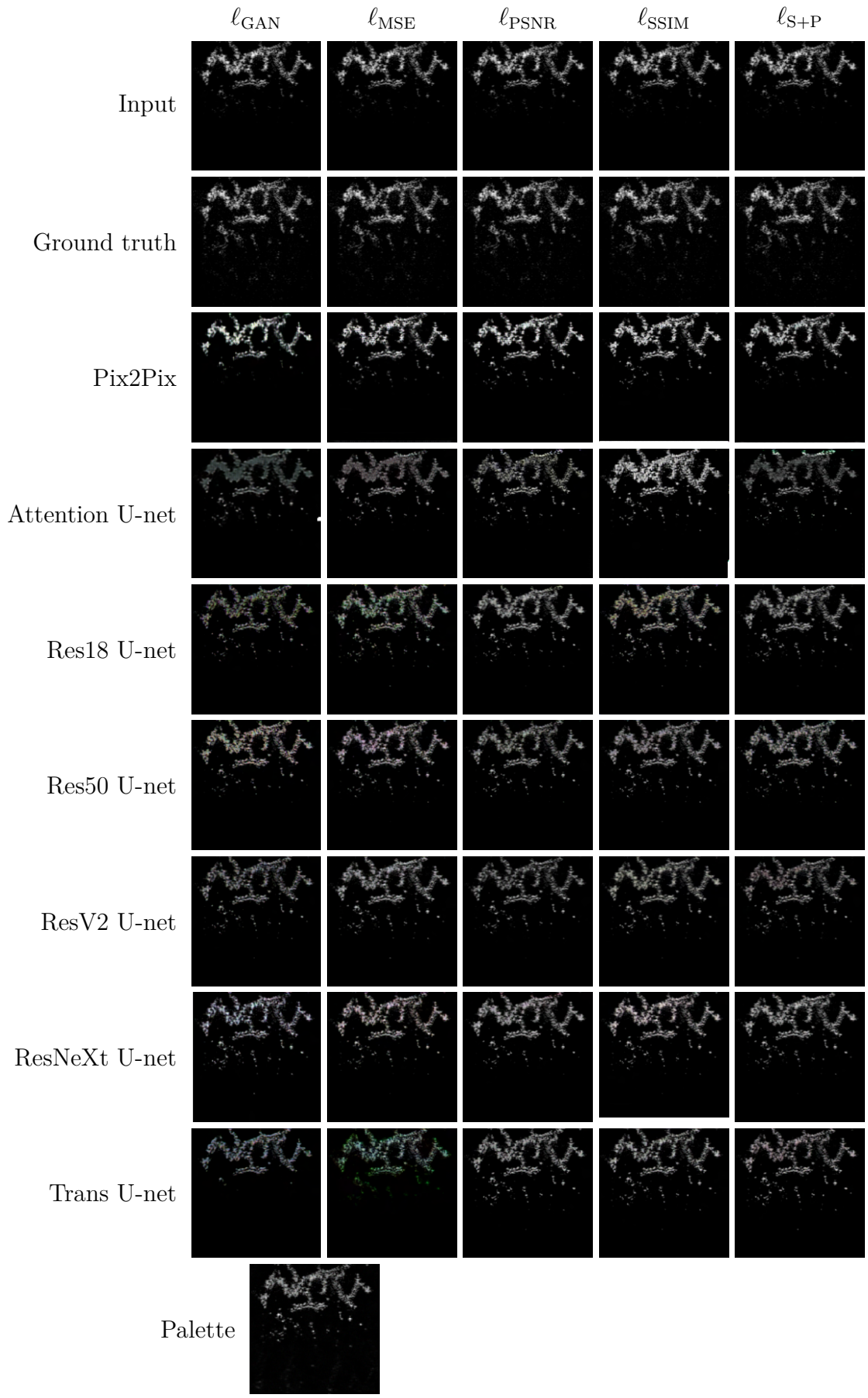


Figure B.10: A visual inspection of the outputs of the models trained and tested on the Experimental NNE 40 dB dataset with their various loss functions.

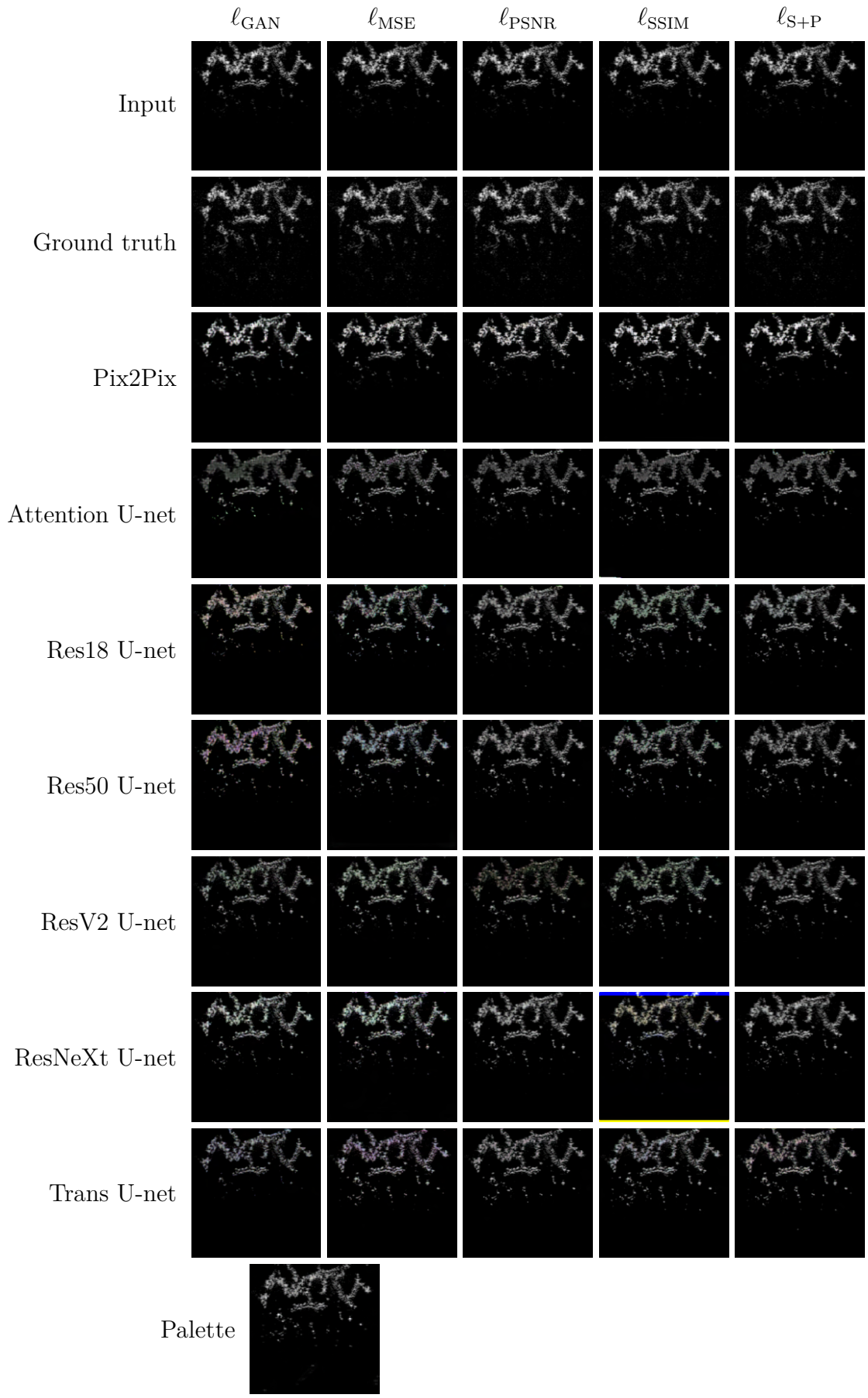


Figure B.11: A visual inspection of the outputs of the models trained and tested on the Experimental NNE 50 dB dataset with their various loss functions.

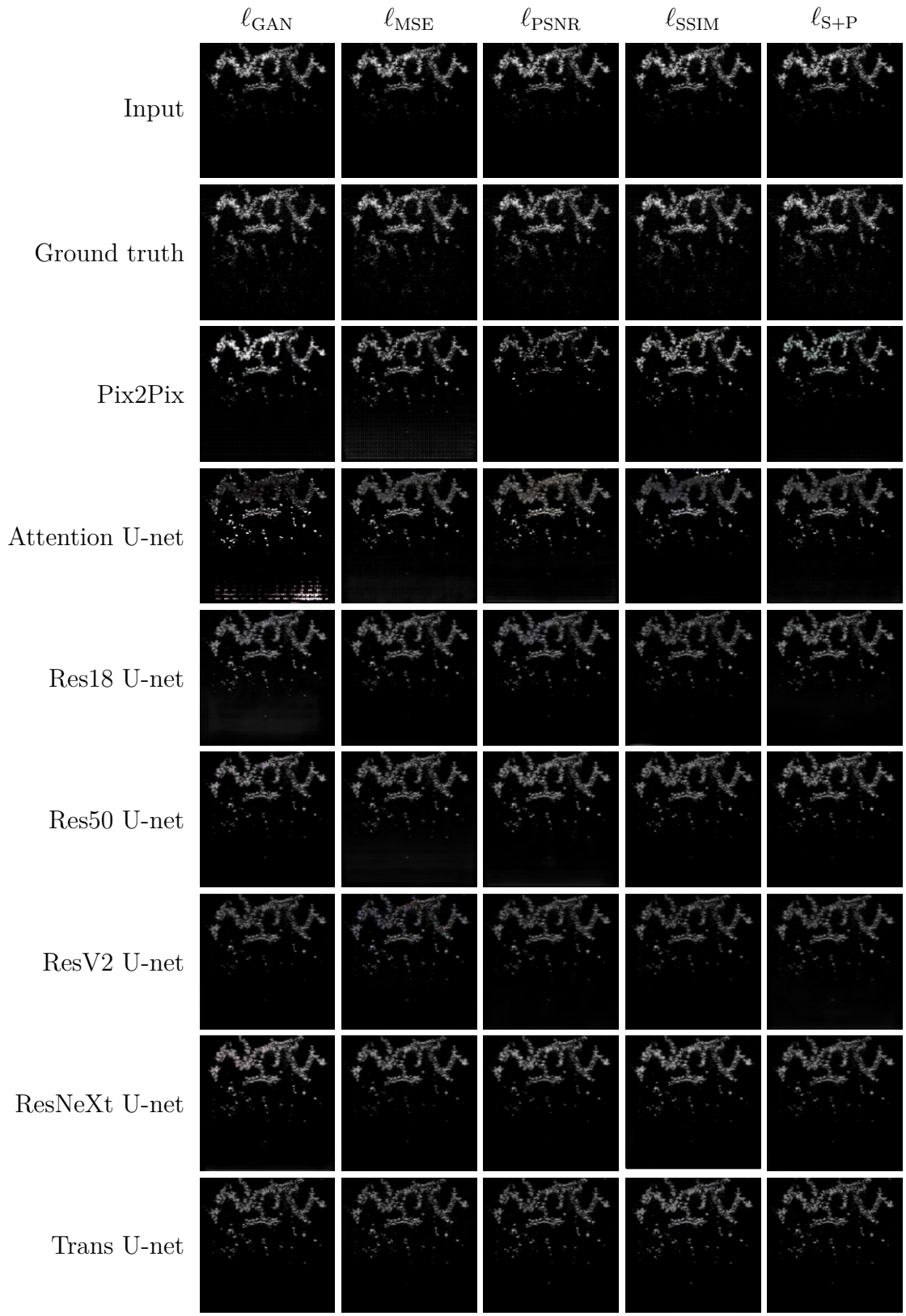


Figure B.12: A visual inspection of the outputs of the models trained and tested on the Experimental NNE 10–50 dB dataset with their various loss functions.

Appendix C

Peformance per noise level for models trained on all noise levels

The following tables depict the performance of the models trained on noise levels 10–50 dB per noise level. This provides a comparison per noise level for the models trained per noise level *vs*. all noise levels.

Testing data →	10 dB		20 dB		30 dB		40 dB		50 dB	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Input image	0.213	22.715	0.504	24.705	0.826	25.790	0.832	25.810	0.833	25.824
Pix2Pix										
ℓ_{GAN}	0.722	24.555	0.812	27.026	*0.871*	*29.754*	*0.876*	*30.071*	*0.876*	*30.083*
ℓ_{MSE}	*0.729*	24.551	0.799	26.310	0.838	27.736	0.841	27.828	0.841	27.835
ℓ_{PSNR}	0.697	*24.746*	0.803	26.931	0.854	28.845	0.858	28.980	0.858	28.975
ℓ_{SSIM}	0.721	24.651	*0.818*	26.999	0.864	29.025	0.866	29.126	0.866	29.125
$\ell_{\text{S+P}}$	0.702	24.692	0.809	*27.190*	0.869	29.510	0.872	29.647	0.872	29.640
Att. U-net										
ℓ_{GAN}	0.713	24.602	0.809	26.731	0.855	28.676	0.858	28.816	0.858	28.835
ℓ_{MSE}	0.659	24.634	0.796	*27.272*	*0.878*	*30.031*	*0.883*	*30.185*	*0.883*	*30.170*
ℓ_{PSNR}	0.716	*24.697*	0.811	27.023	0.865	29.139	0.868	29.294	0.868	29.286
ℓ_{SSIM}	*0.733*	24.653	*0.825*	27.066	0.873	29.280	0.875	29.423	0.875	29.437
$\ell_{\text{S+P}}$	0.723	24.661	0.822	27.210	0.867	29.526	0.866	29.576	0.866	29.588
Res18 U-net										
ℓ_{GAN}	0.746	24.700	0.833	27.730	0.904	32.405	0.912	*33.402*	0.912	*33.362*
ℓ_{MSE}	0.714	24.801	0.824	27.790	0.904	32.179	0.913	33.062	0.913	33.073
ℓ_{PSNR}	0.731	*24.831*	0.832	27.797	0.910	32.321	0.918	33.263	0.919	33.265
ℓ_{SSIM}	*0.760*	24.729	*0.837*	27.633	0.908	32.035	0.915	32.733	0.916	32.835
$\ell_{\text{S+P}}$	0.726	24.790	0.832	*27.867*	*0.911*	*32.446*	*0.919*	33.284	*0.919*	33.297
Res50 U-net										
ℓ_{GAN}	0.739	24.589	0.826	27.629	0.909	32.134	0.914	32.923	0.914	32.948
ℓ_{MSE}	0.703	24.677	0.812	27.625	0.902	31.278	0.908	31.825	0.908	31.849
ℓ_{PSNR}	0.714	*24.703*	0.826	*27.751*	0.907	*32.208*	*0.916*	*33.243*	*0.916*	*33.262*
ℓ_{SSIM}	0.744	24.413	0.834	27.402	0.908	32.025	0.914	32.846	0.915	32.909
$\ell_{\text{S+P}}$	*0.751*	24.576	*0.836*	27.493	*0.910*	32.153	0.915	32.841	0.914	32.816
ResV2 U-net										
ℓ_{GAN}	0.751	24.776	0.832	27.743	0.908	32.308	0.913	33.012	0.912	33.005
ℓ_{MSE}	0.734	**24.907**	0.827	27.882	0.903	31.917	0.913	32.785	0.913	32.813
ℓ_{PSNR}	0.733	24.843	0.830	**27.965**	*0.910*	32.354	*0.918*	33.236	*0.918*	33.238
ℓ_{SSIM}	*0.759*	24.749	*0.837*	27.782	0.907	31.852	0.916	32.836	0.916	32.915
$\ell_{\text{S+P}}$	0.746	24.866	0.833	27.913	0.907	*32.357*	0.917	*33.310*	0.917	*33.319*
ResNeXt U-net										
ℓ_{GAN}	*0.749*	24.369	*0.829*	27.454	0.891	31.063	0.899	31.781	0.899	31.826
ℓ_{MSE}	0.730	*24.741*	0.828	*27.727*	0.893	30.964	0.897	31.442	0.898	31.501
ℓ_{PSNR}	0.706	24.581	0.821	27.575	0.910	32.157	0.919	33.108	0.919	33.127
ℓ_{SSIM}	0.714	14.662	0.813	14.900	0.882	15.056	0.893	15.078	0.893	15.079
$\ell_{\text{S+P}}$	0.726	24.629	0.828	27.441	*0.913*	*32.277*	*0.922*	*33.441*	*0.923*	*33.492*
Trans U-net										
ℓ_{GAN}	0.757	24.495	0.830	27.482	0.908	**32.587**	0.915	**33.498**	0.915	33.509
ℓ_{MSE}	0.684	24.290	0.788	26.770	0.881	30.302	0.890	30.829	0.890	30.803
ℓ_{PSNR}	0.651	24.353	0.818	27.617	0.902	31.699	0.911	32.522	0.911	32.536
ℓ_{SSIM}	**0.772**	24.678	**0.850**	*27.889*	**0.915**	32.328	**0.925**	33.452	**0.925**	**33.513**
$\ell_{\text{S+P}}$	0.767	*24.714*	0.843	27.634	0.912	32.465	0.918	33.170	0.918	33.150

Table C.1: Performance metrics per noise level of the models trained on all 10–50 dB noise levels of the DRIVE dataset.

Testing data →	10 dB		20 dB		30 dB		40 dB		50 dB	
	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR
Input image	0.536	28.841	0.811	31.458	0.937	32.274	0.938	32.313	0.938	32.331
Pix2Pix										
ℓ_{GAN}	0.860	30.601	0.825	33.405	0.789	34.071	0.789	34.075	0.789	34.094
ℓ_{MSE}	0.864	*32.428*	0.930	*35.963*	0.945	*38.123*	0.944	*38.197*	0.944	*38.211*
ℓ_{PSNR}	0.845	29.317	0.898	30.444	0.899	30.322	0.899	30.289	0.899	30.296
ℓ_{SSIM}	*0.921*	32.138	*0.952*	35.523	*0.966*	37.696	*0.966*	37.824	*0.966*	37.866
$\ell_{\text{S+P}}$	0.897	30.086	0.924	32.659	0.929	33.475	0.929	33.510	0.929	33.525
Att. U-net										
ℓ_{GAN}	0.554	26.246	0.749	23.807	0.821	20.168	0.819	19.775	0.819	19.760
ℓ_{MSE}	0.872	32.434	0.939	35.935	0.960	37.756	0.960	37.738	0.960	37.723
ℓ_{PSNR}	0.853	**32.864**	0.934	*37.131*	*0.969*	*40.642*	*0.970*	*40.771*	*0.970*	*40.768*
ℓ_{SSIM}	0.920	31.965	0.950	35.072	0.961	36.674	0.961	36.748	0.961	36.753
$\ell_{\text{S+P}}$	*0.922*	32.651	*0.955*	36.605	0.967	38.934	0.966	38.895	0.966	38.908
Res18 U-net										
ℓ_{GAN}	0.917	31.947	*0.948*	35.495	0.959	37.827	0.962	38.516	0.962	38.552
ℓ_{MSE}	0.805	*32.691*	0.925	*36.283*	0.956	38.457	0.957	38.720	0.958	38.725
ℓ_{PSNR}	0.839	32.500	0.938	36.146	*0.966*	*38.615*	*0.967*	*38.913*	*0.967*	*38.928*
ℓ_{SSIM}	*0.920*	32.053	0.947	34.848	0.957	36.427	0.958	36.641	0.959	36.666
$\ell_{\text{S+P}}$	0.914	32.096	0.948	35.153	0.960	36.844	0.961	37.095	0.961	37.117
Res50 U-net										
ℓ_{GAN}	0.916	32.375	0.953	36.273	0.973	39.494	0.978	40.538	0.978	40.612
ℓ_{MSE}	0.862	32.750	0.922	36.571	0.948	39.116	0.948	39.366	0.948	39.385
ℓ_{PSNR}	0.872	32.382	0.930	36.480	0.925	38.596	0.905	38.502	0.905	38.492
ℓ_{SSIM}	**0.924**	32.643	0.957	36.605	0.975	39.944	0.979	40.910	0.979	40.966
$\ell_{\text{S+P}}$	0.919	*32.811*	*0.958*	*36.947*	*0.977*	*40.770*	*0.981*	*42.060*	*0.981*	*42.136*
ResV2 U-net										
ℓ_{GAN}	0.909	32.208	0.942	35.984	0.968	39.381	0.968	39.588	0.968	39.614
ℓ_{MSE}	0.857	32.500	0.932	*36.492*	*0.968*	*39.972*	*0.971*	*40.618*	0.970	*40.656*
ℓ_{PSNR}	0.837	*32.513*	0.941	36.286	0.968	38.771	0.969	39.277	0.969	39.317
ℓ_{SSIM}	*0.919*	32.107	*0.951*	35.560	0.968	38.230	0.970	38.954	*0.971*	38.978
$\ell_{\text{S+P}}$	0.914	32.512	0.949	36.077	0.965	38.725	0.967	39.270	0.967	39.301
ResNeXt U-net										
ℓ_{GAN}	0.913	31.361	0.930	31.329	0.972	37.020	0.975	38.527	0.975	38.536
ℓ_{MSE}	0.825	32.674	0.920	36.450	0.971	40.238	0.973	40.608	0.973	40.672
ℓ_{PSNR}	0.829	31.986	0.913	34.610	0.955	36.946	0.954	36.877	0.954	36.893
ℓ_{SSIM}	*0.913*	17.905	0.947	18.017	0.967	18.064	0.971	18.070	0.971	18.070
$\ell_{\text{S+P}}$	0.913	*32.691*	*0.957*	*36.849*	*0.977*	*40.723*	*0.980*	*42.040*	*0.980*	*42.143*
Trans U-net										
ℓ_{GAN}	0.917	32.244	0.955	36.760	0.975	40.615	0.975	41.076	0.975	41.089
ℓ_{MSE}	0.865	32.683	0.926	36.918	0.960	40.300	0.961	40.586	0.961	40.600
ℓ_{PSNR}	0.860	32.495	0.942	**37.371**	0.978	42.438	0.982	43.684	0.982	43.778
ℓ_{SSIM}	*0.919*	32.323	**0.958**	36.922	0.980	41.578	**0.985**	43.001	**0.985**	43.102
$\ell_{\text{S+P}}$	0.910	*32.696*	0.958	37.305	**0.981**	**42.723**	0.983	**43.737**	0.983	**43.825**

Table C.2: Performance metrics per noise level of the models trained on all 10–50 dB noise levels of the NNE dataset.

Appendix D

Output images

The following figures depict the outputs of all models for one test image of the DRIVE and NNE datasets.

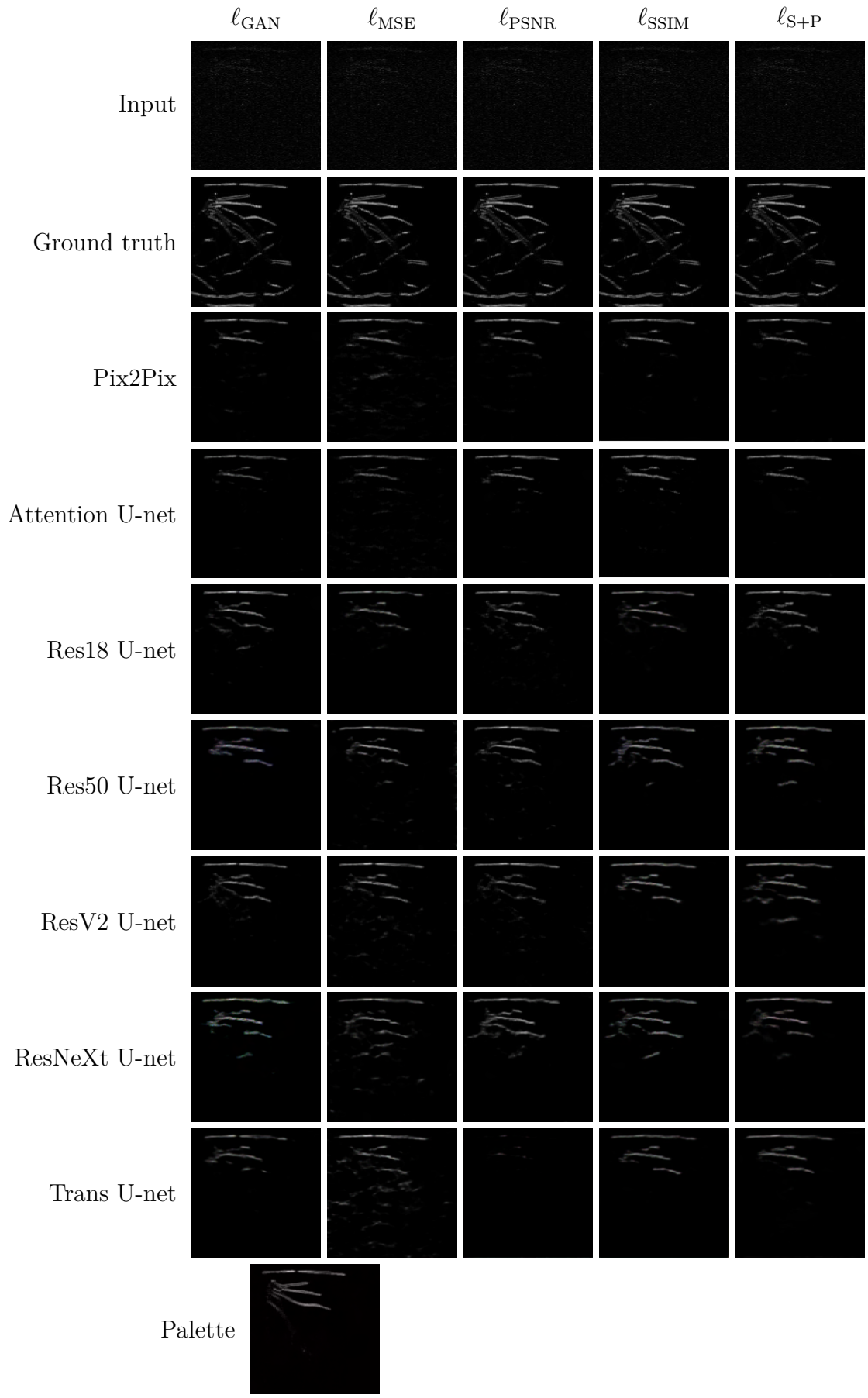


Figure D.1: A visual inspection of the outputs of the models trained and tested on the DRIVE 10 dB dataset with their various loss functions.

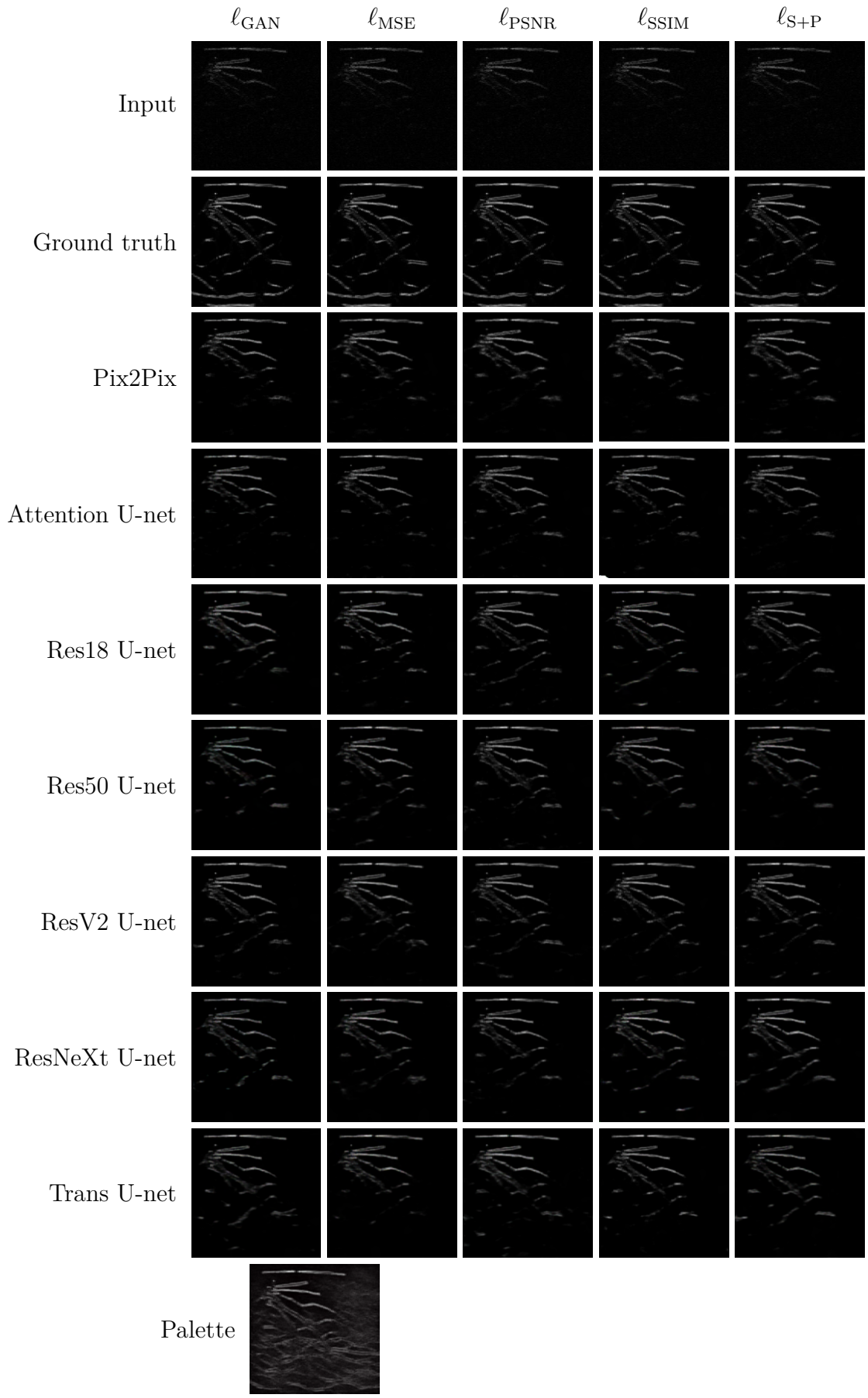


Figure D.2: A visual inspection of the outputs of the models trained and tested on the DRIVE 20 dB dataset with their various loss functions.

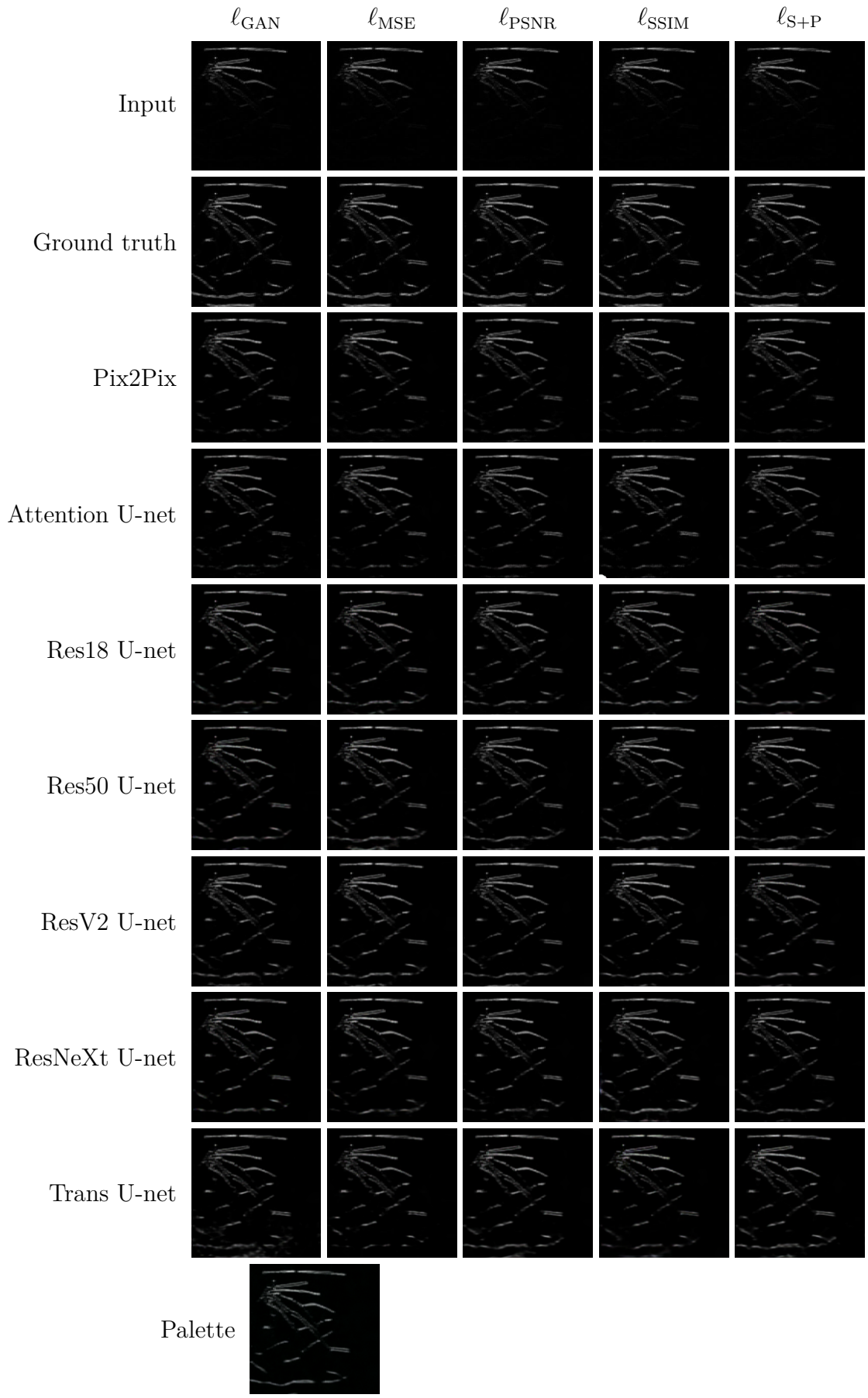


Figure D.3: A visual inspection of the outputs of the models trained and tested on the DRIVE 30 dB dataset with their various loss functions.

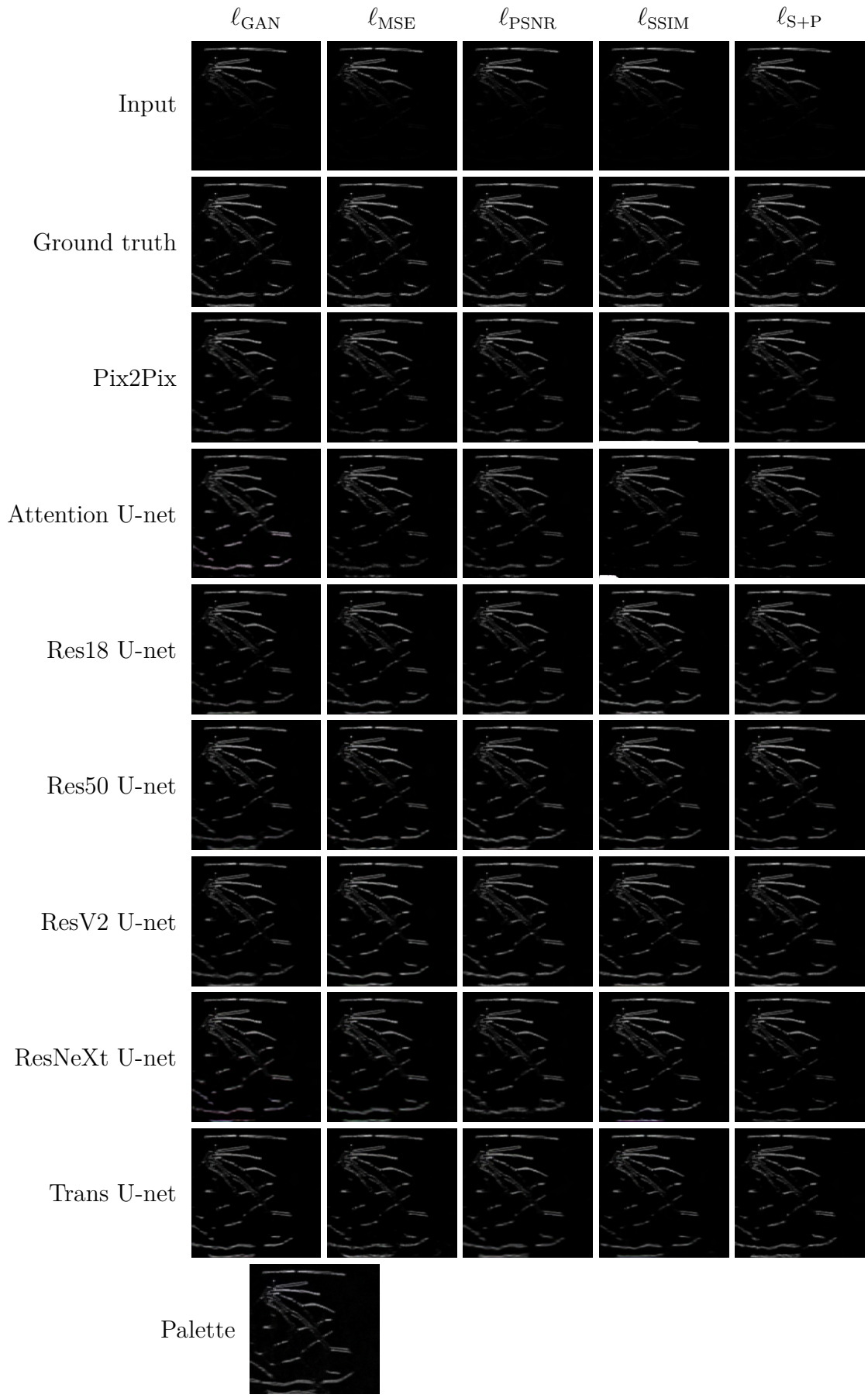


Figure D.4: A visual inspection of the outputs of the models trained and tested on the DRIVE 40 dB dataset with their various loss functions.

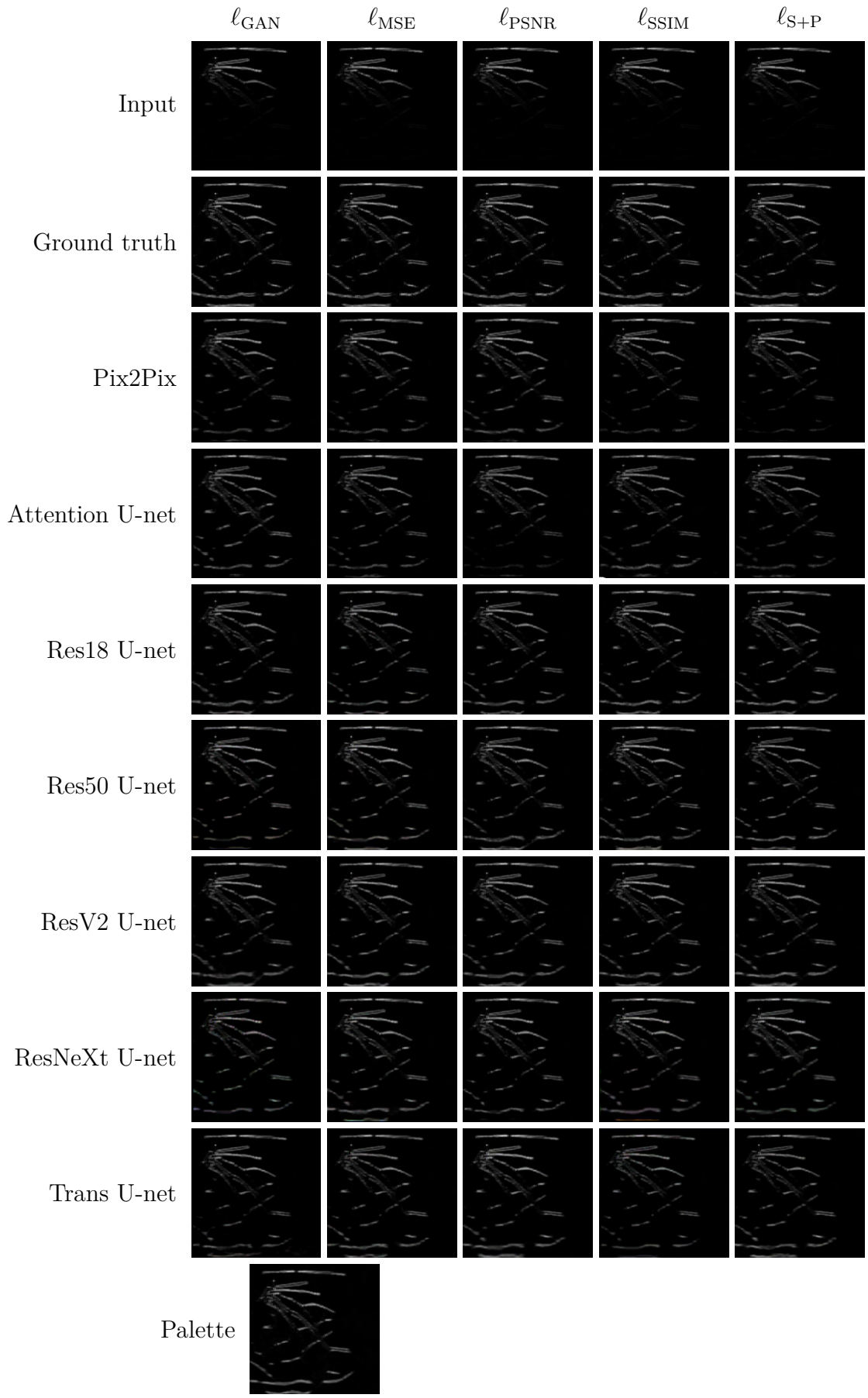


Figure D.5: A visual inspection of the outputs of the models trained and tested on the DRIVE 50 dB dataset with their various loss functions.

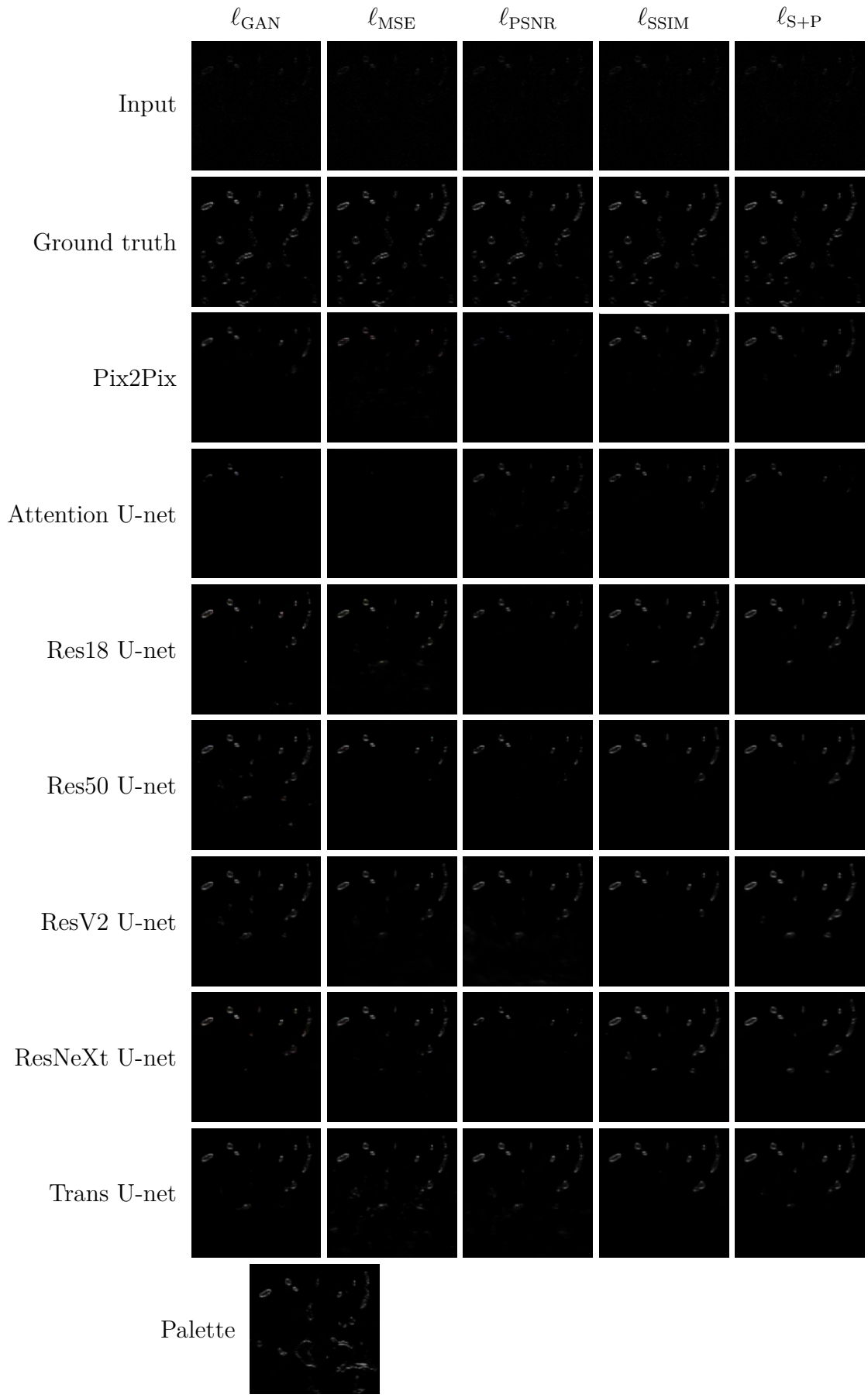


Figure D.6: A visual inspection of the outputs of the models trained and tested on the NNE 10 dB dataset with their various loss functions.

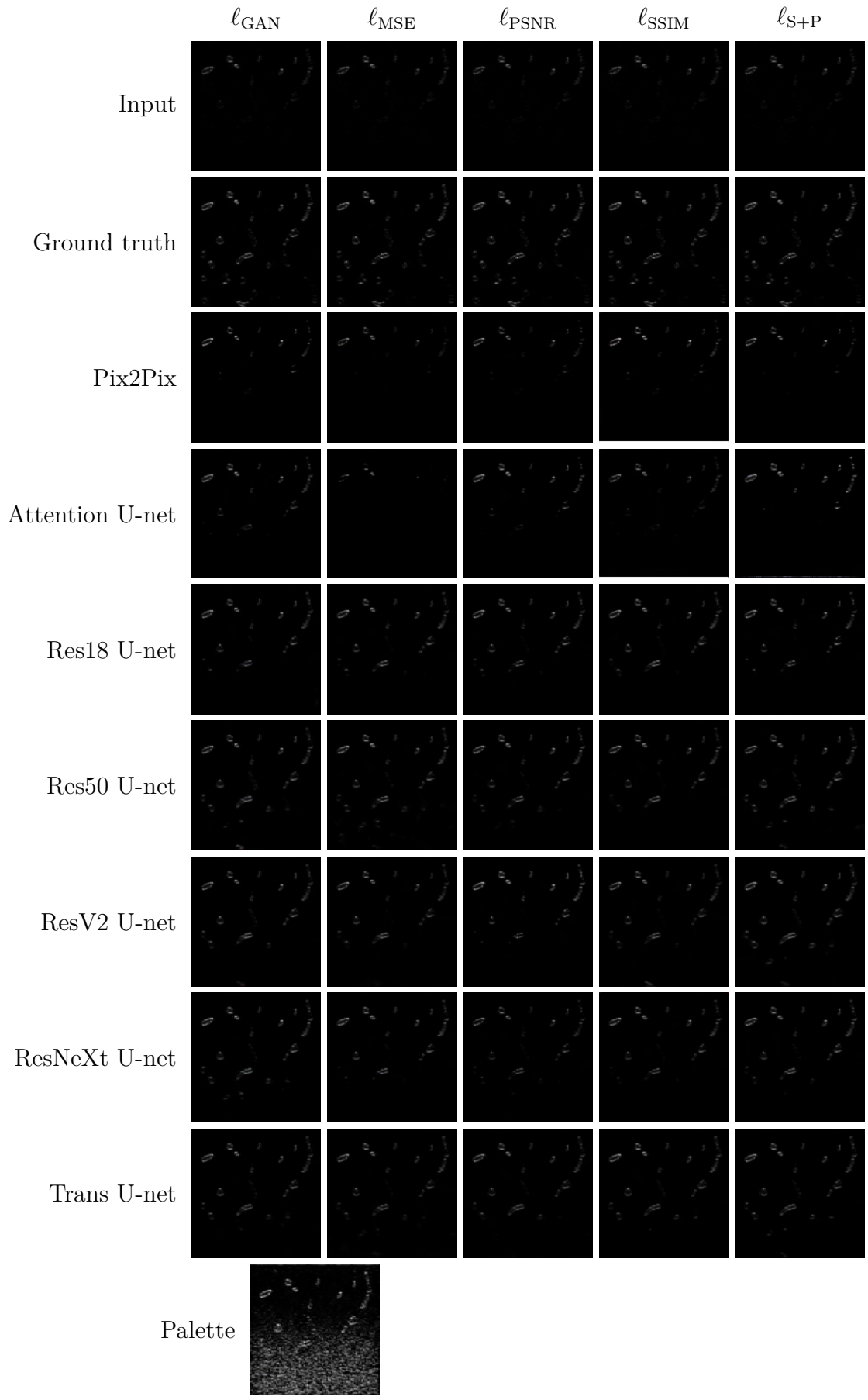


Figure D.7: A visual inspection of the outputs of the models trained and tested on the NNE 20 dB dataset with their various loss functions.

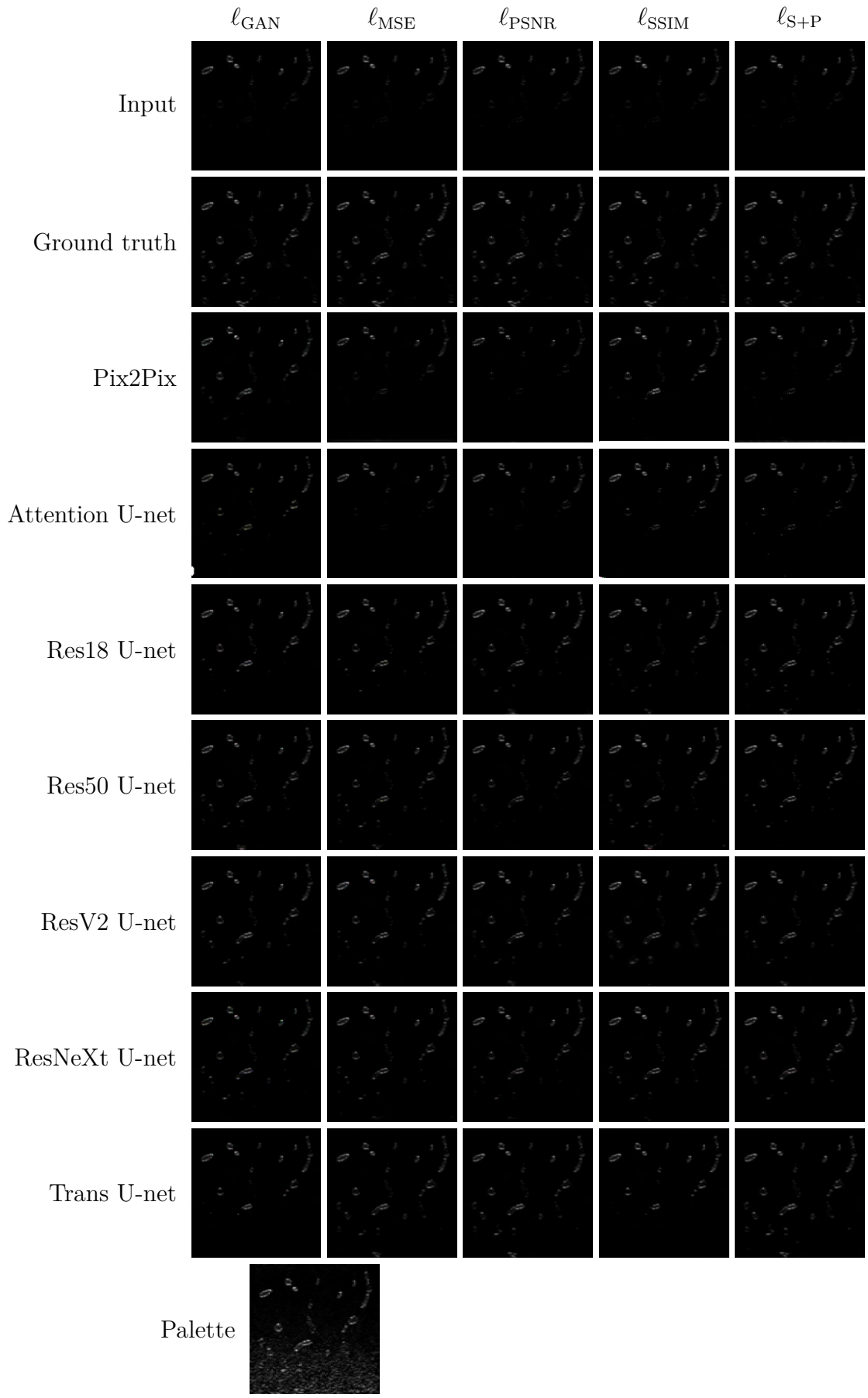


Figure D.8: A visual inspection of the outputs of the models trained and tested on the NNE 30 dB dataset with their various loss functions.

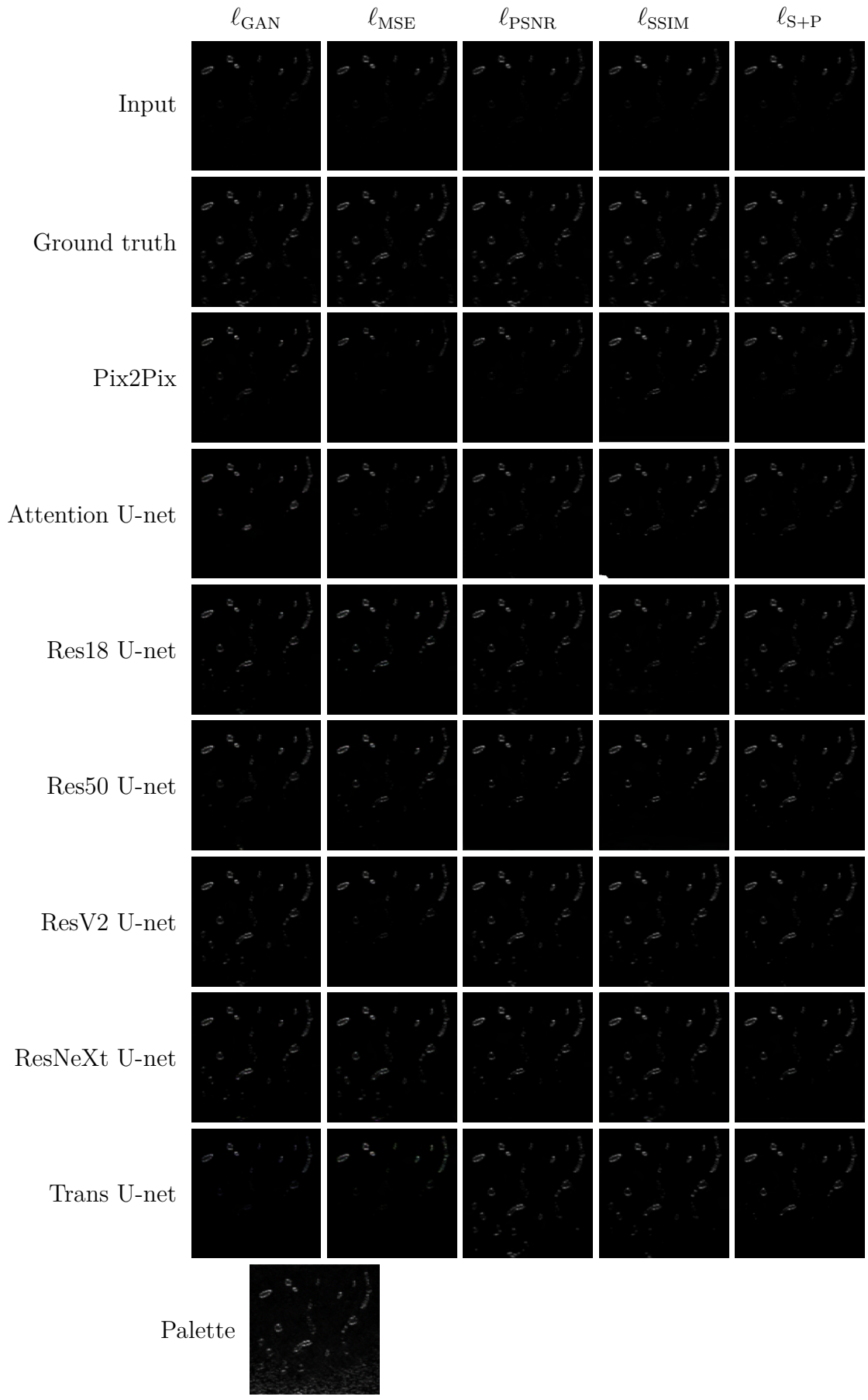


Figure D.9: A visual inspection of the outputs of the models trained and tested on the NNE 40 dB dataset with their various loss functions.

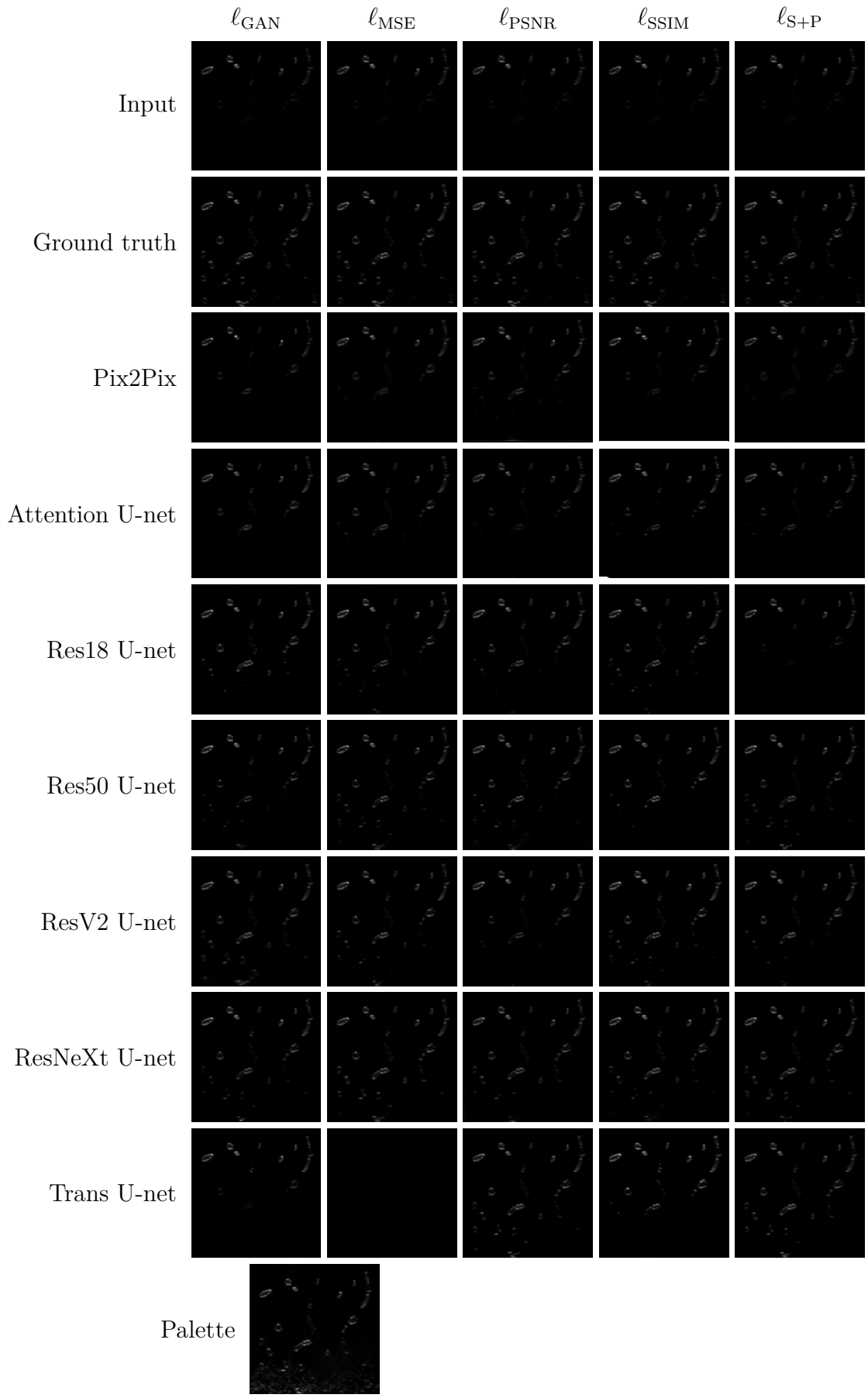


Figure D.10: A visual inspection of the outputs of the models trained and tested on the NNE 50 dB dataset with their various loss functions.

Appendix E

SSIM over depth

The following plots depict the relationship between SSIM and depth for every model, averaged over all test data points. The depth was taken by equally partitioning the output images vertically.

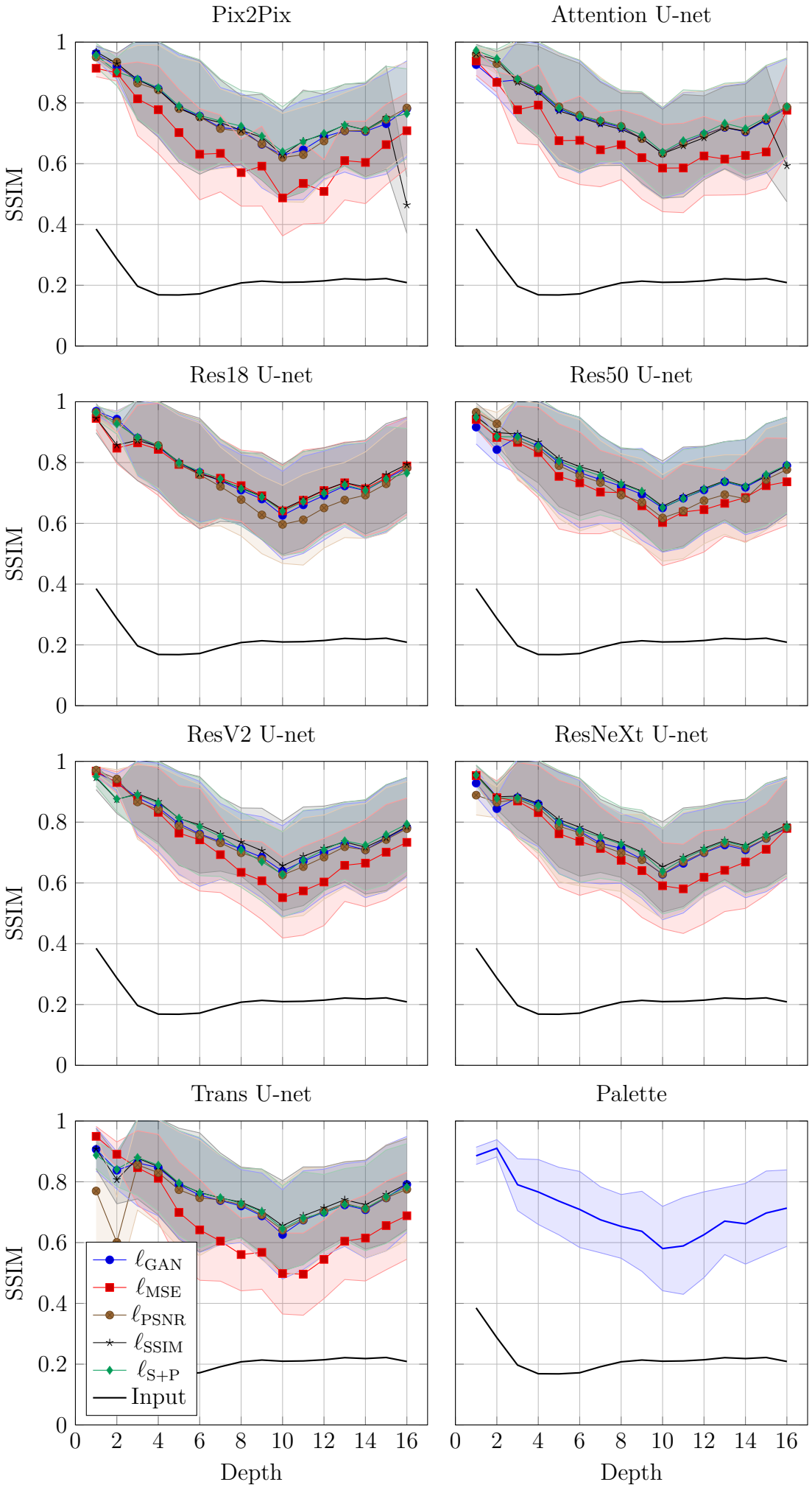


Figure E.1: SSIM score over depth of the models trained and tested on the DRIVE 10 dB dataset with their various loss functions.

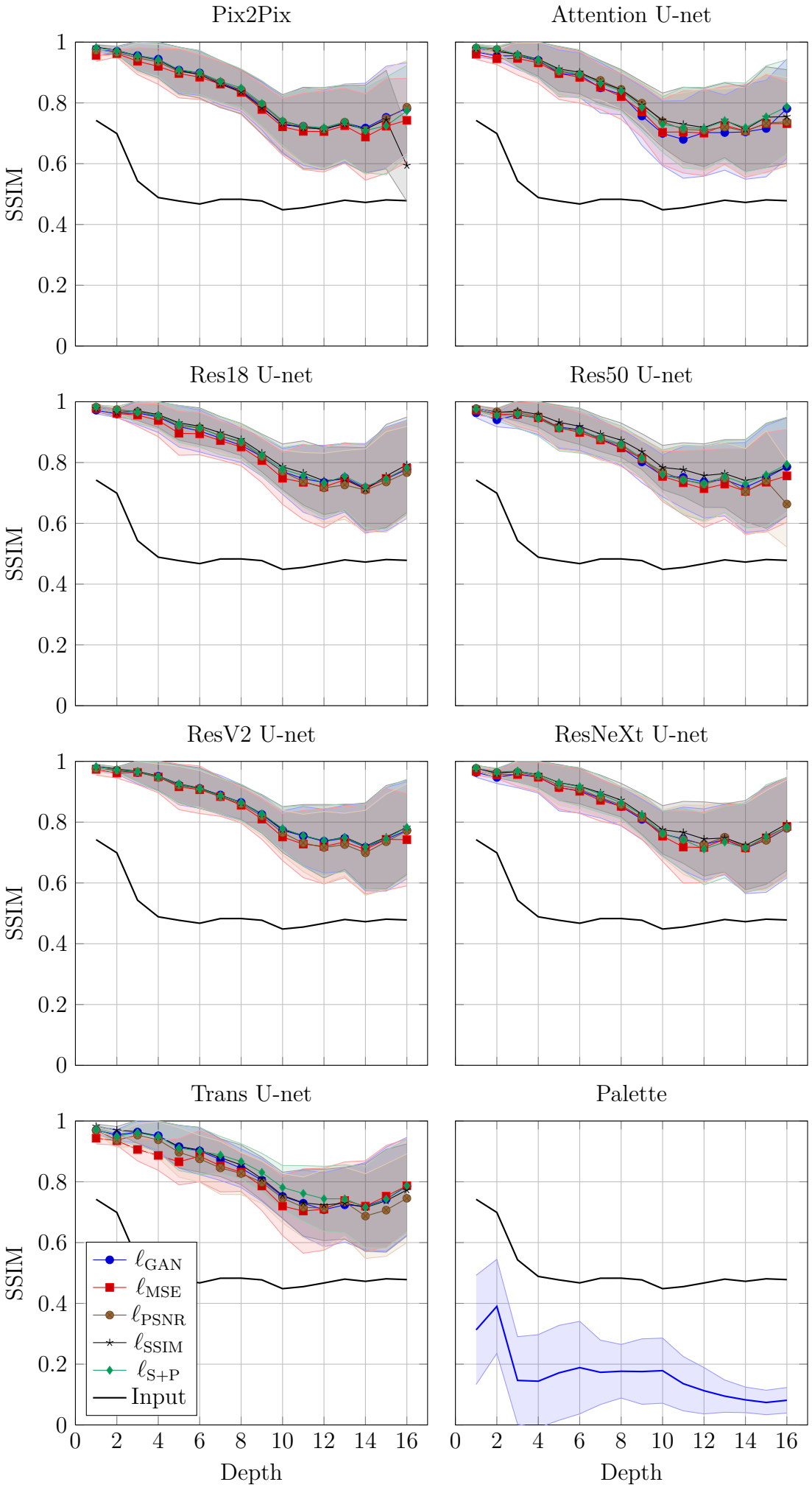


Figure E.2: SSIM score over depth of the models trained and tested on the DRIVE 20 dB dataset with their various loss functions.

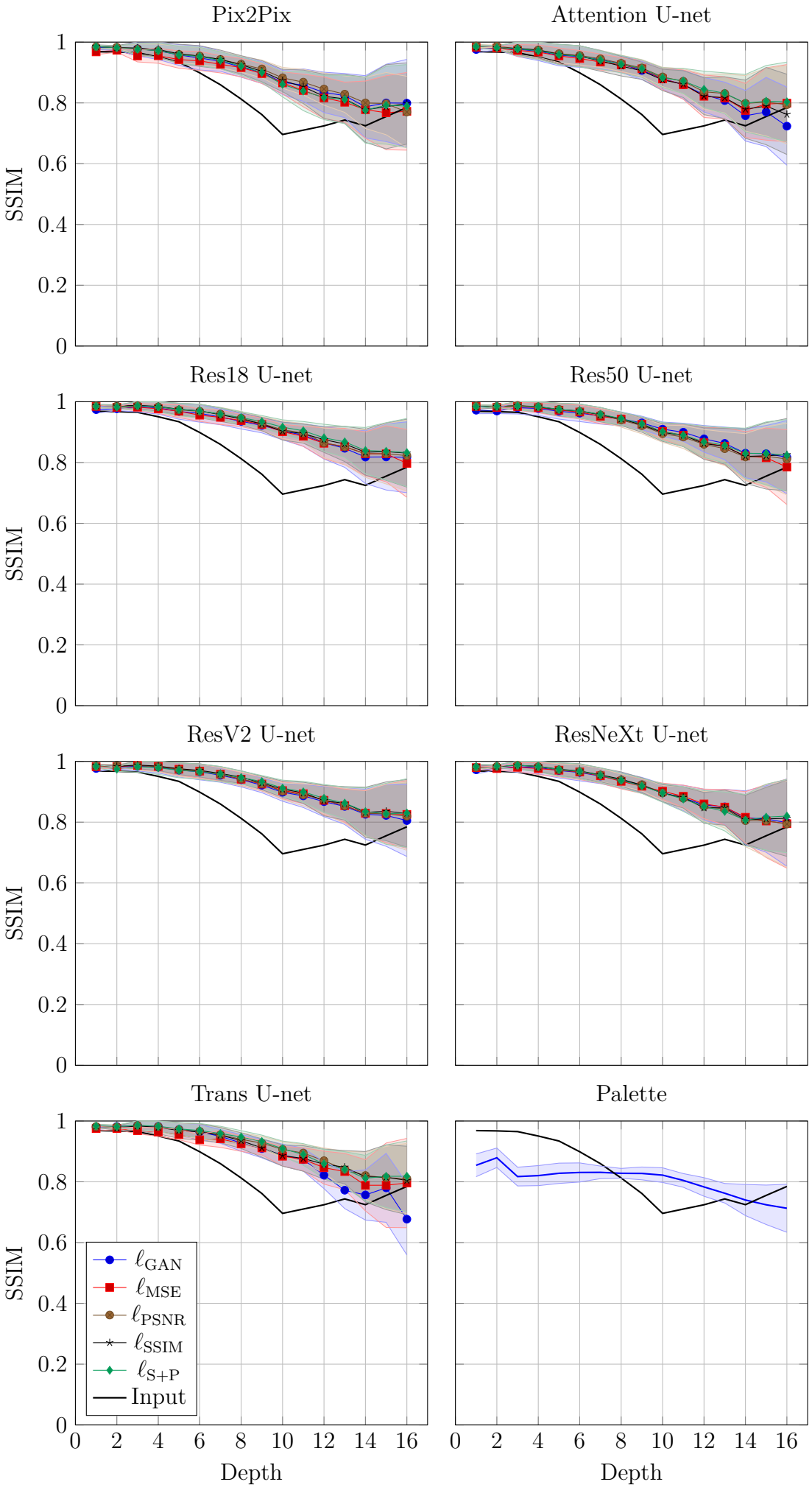


Figure E.3: SSIM score over depth of the models trained and tested on the DRIVE 30 dB dataset with their various loss functions.

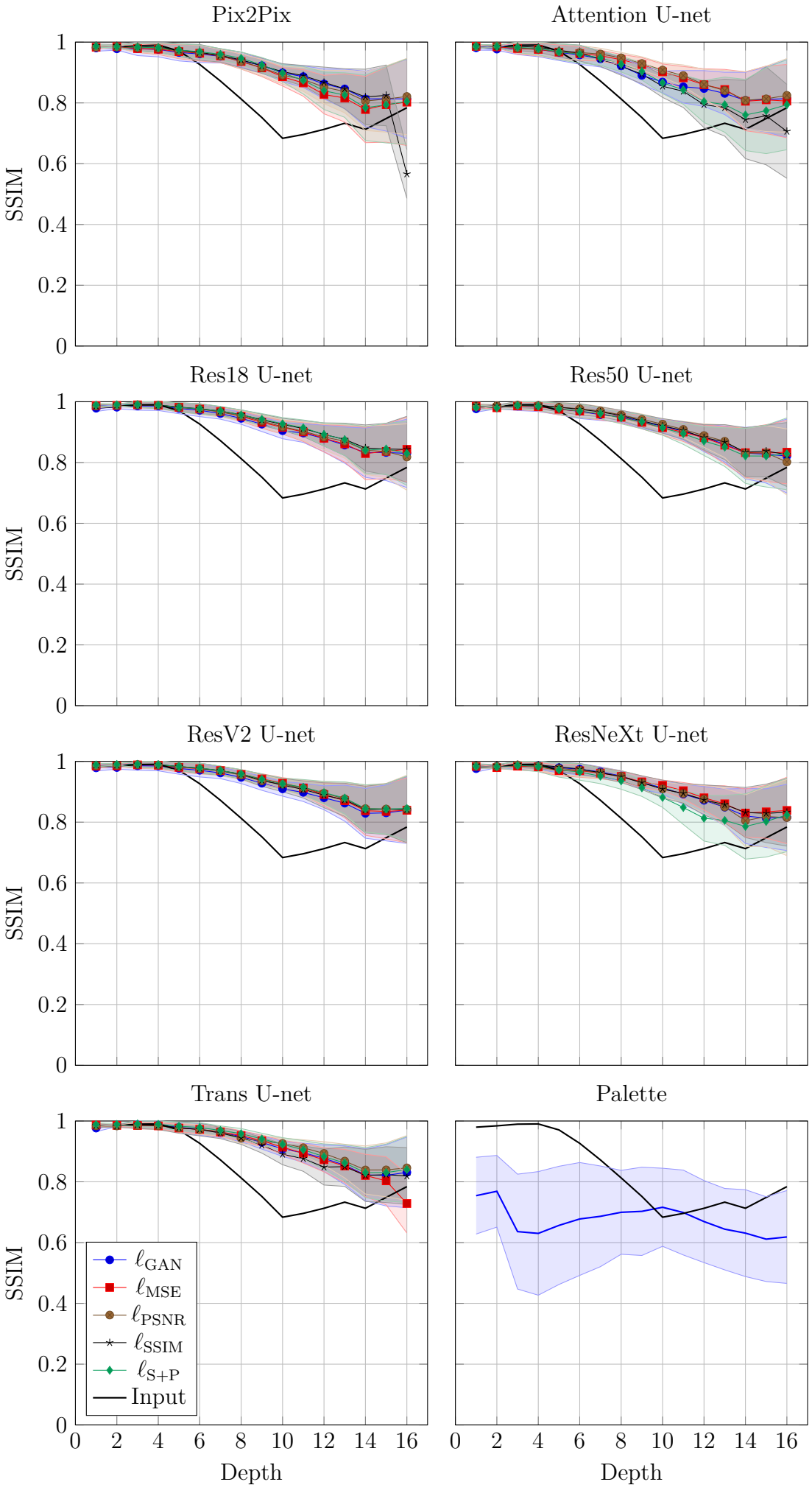


Figure E.4: SSIM score over depth of the models trained and tested on the DRIVE 40 dB dataset with their various loss functions.

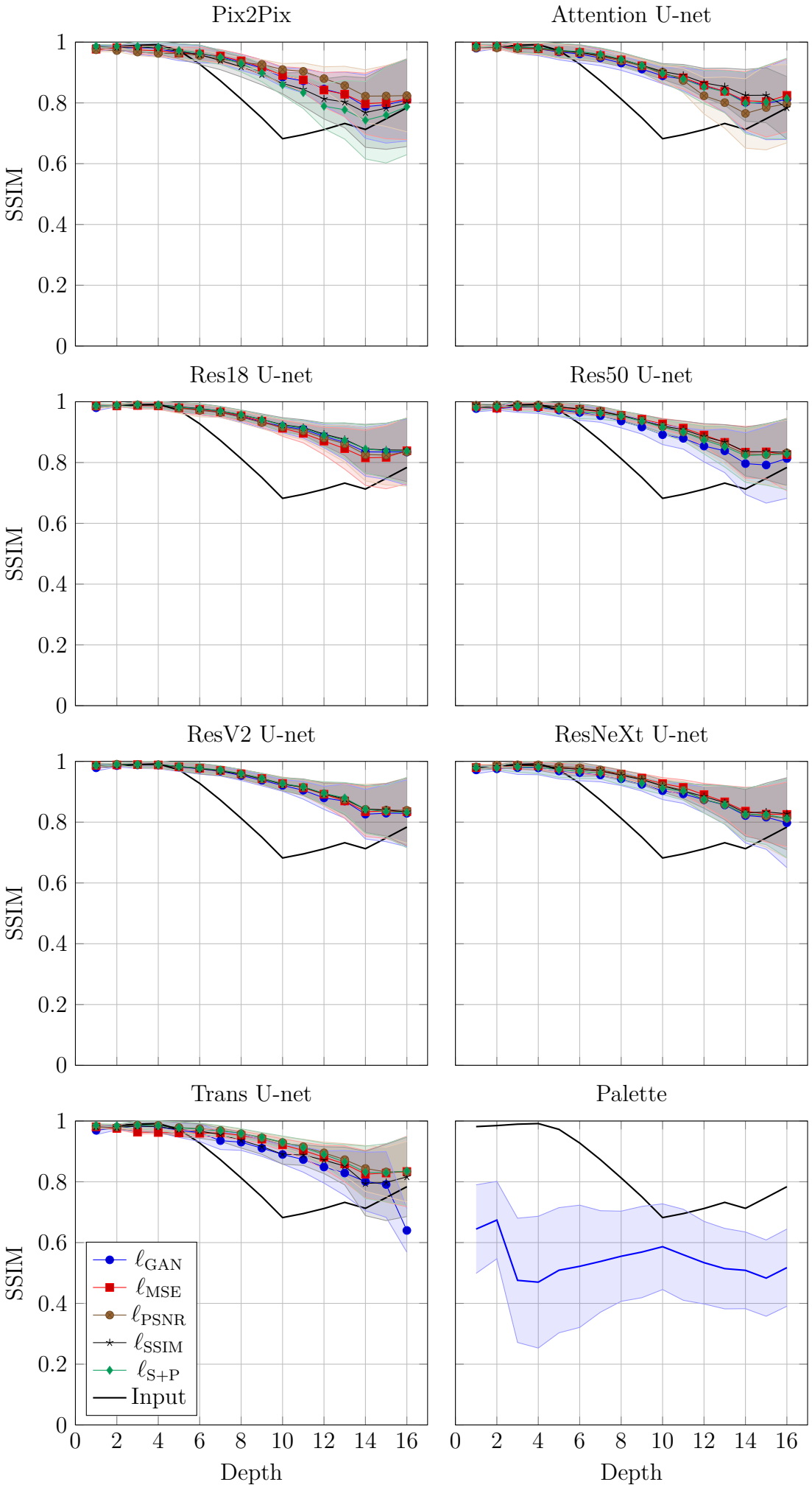


Figure E.5: SSIM score over depth of the models trained and tested on the DRIVE 50 dB dataset with their various loss functions.

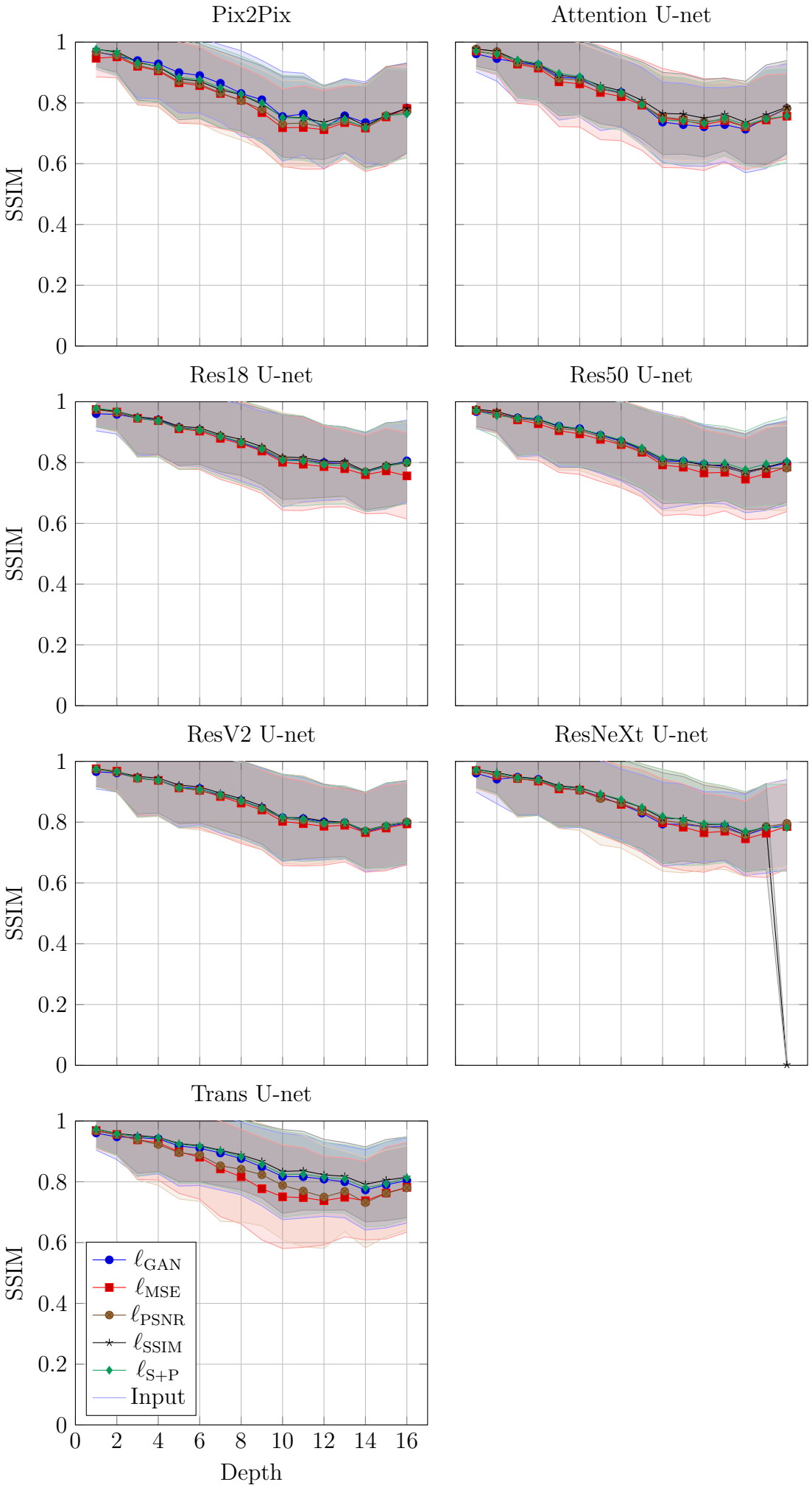


Figure E.6: SSIM score over depth of the models trained and tested on the DRIVE 10–50 dB dataset with their various loss functions.

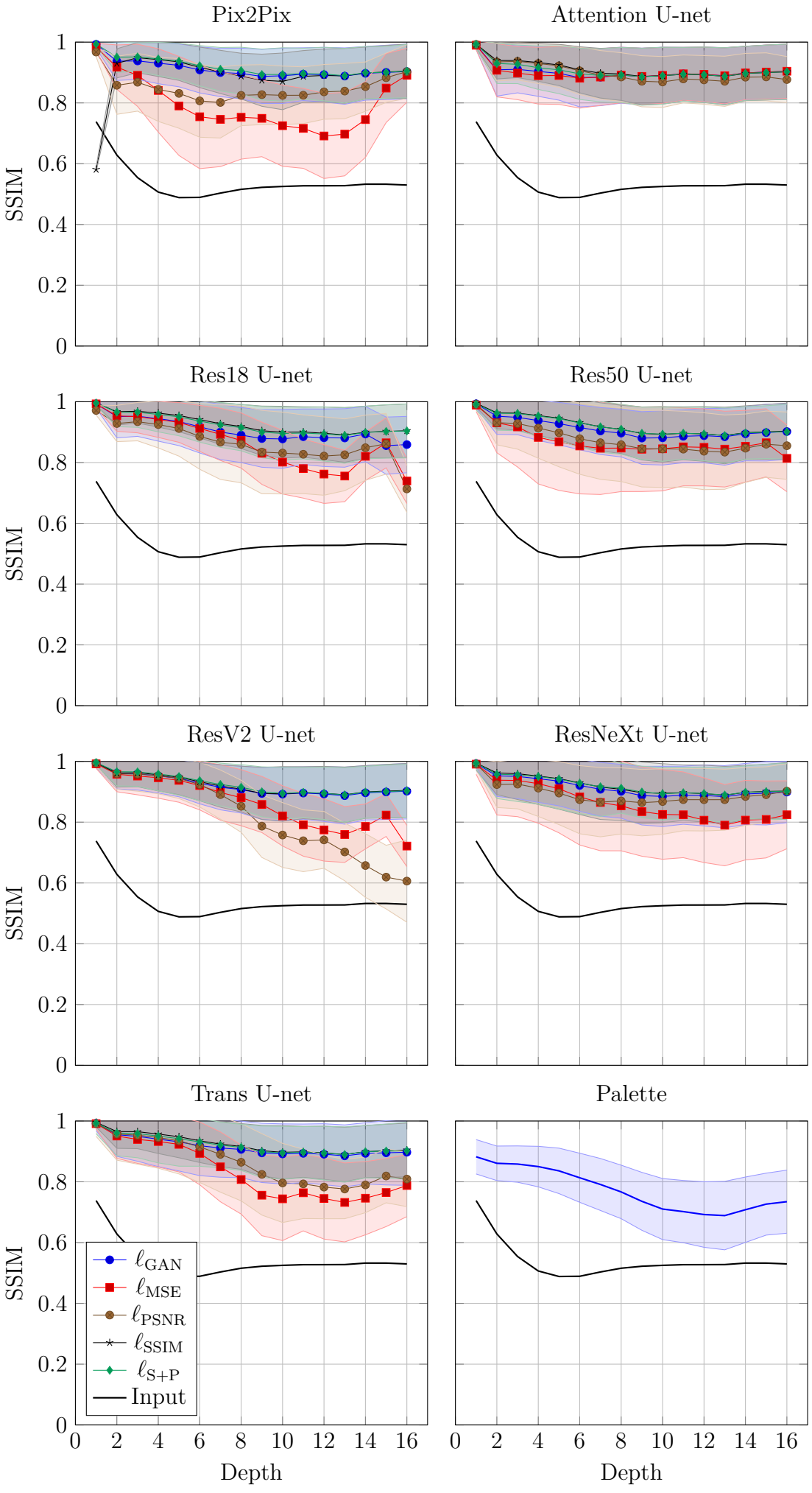


Figure E.7: SSIM score over depth of the models trained and tested on the NNE 10 dB dataset with their various loss functions.

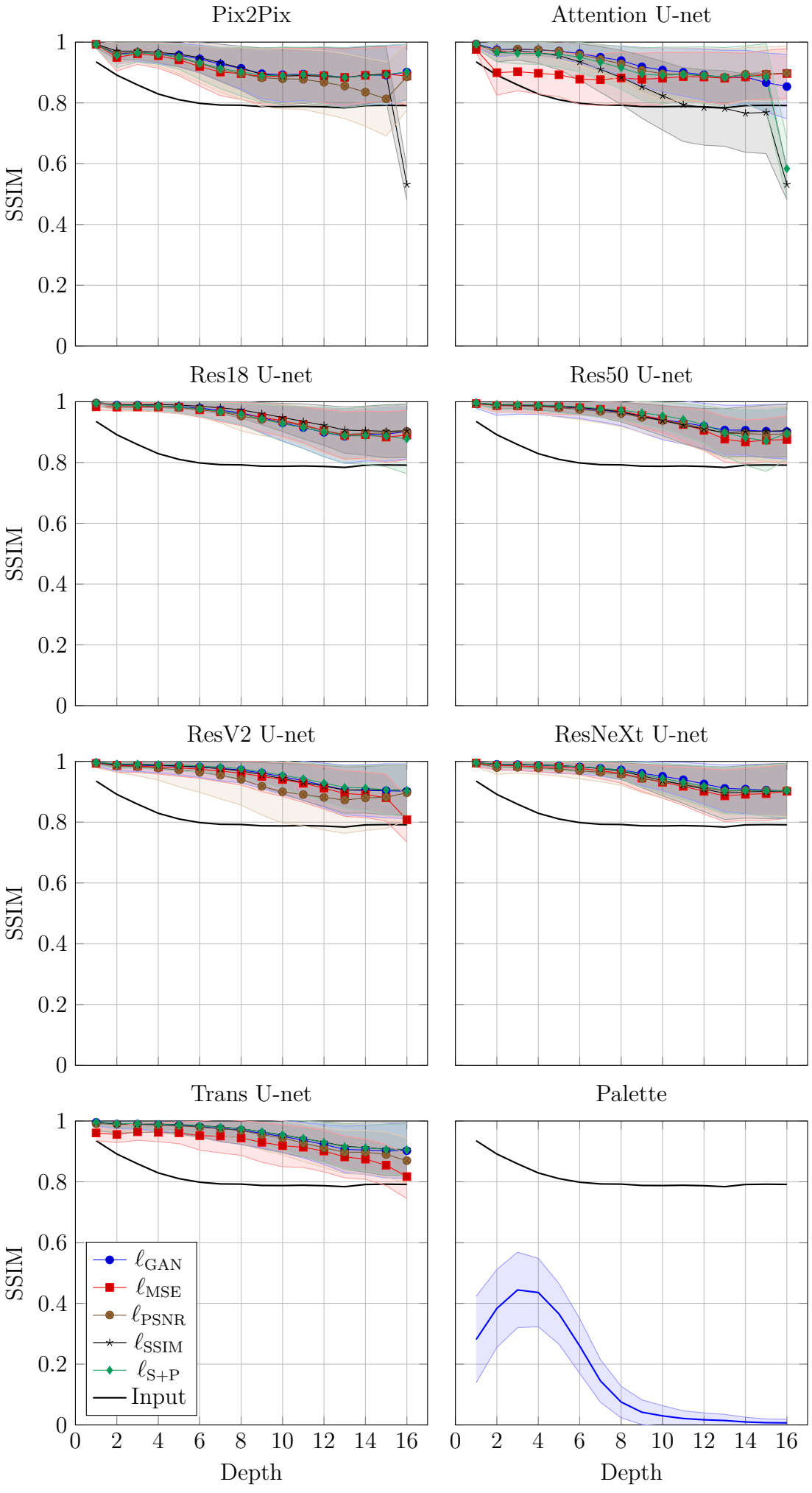


Figure E.8: SSIM score over depth of the models trained and tested on the NNE 20 dB dataset with their various loss functions.

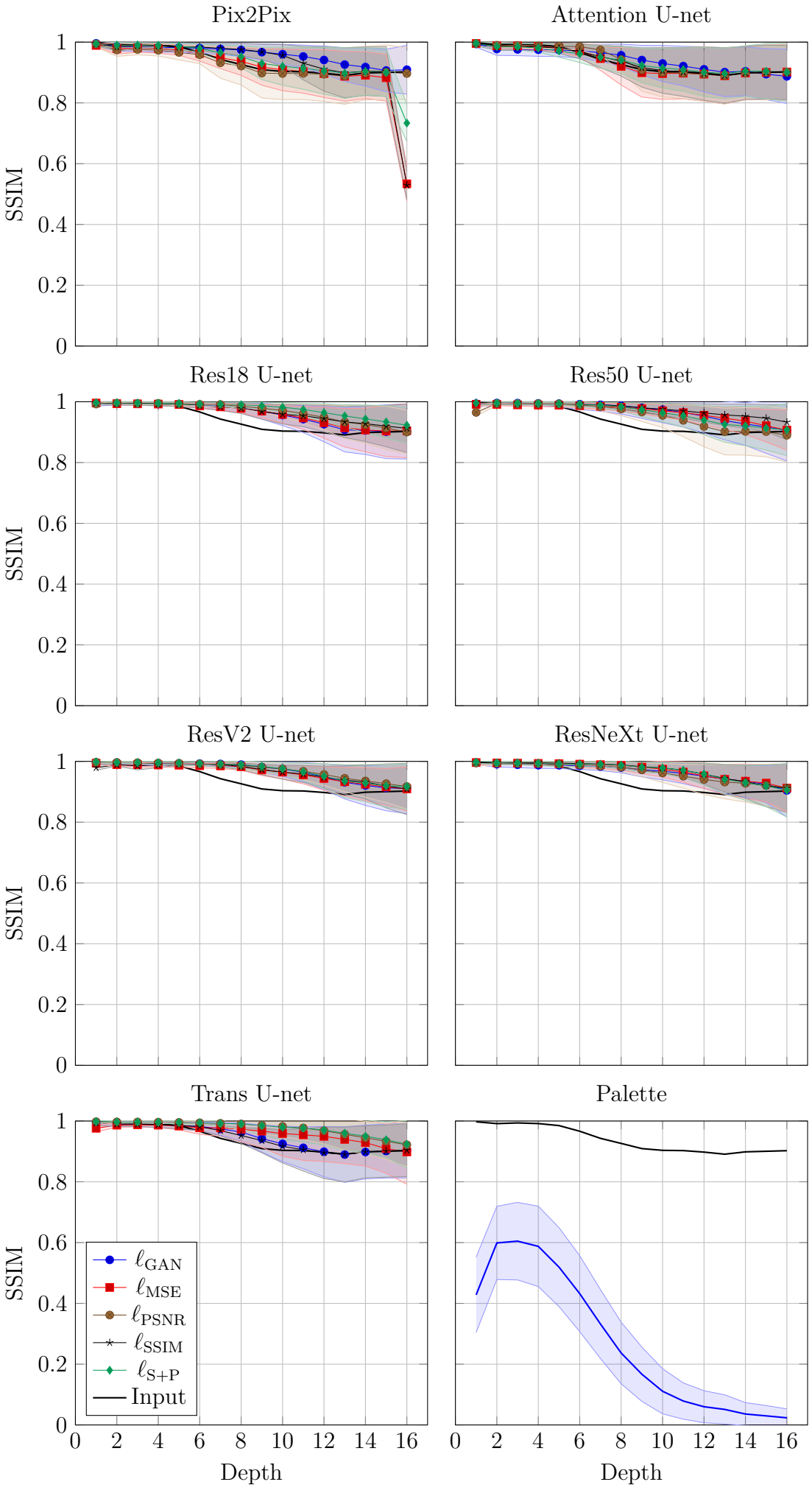


Figure E.9: SSIM score over depth of the models trained and tested on the NNE 30 dB dataset with their various loss functions.

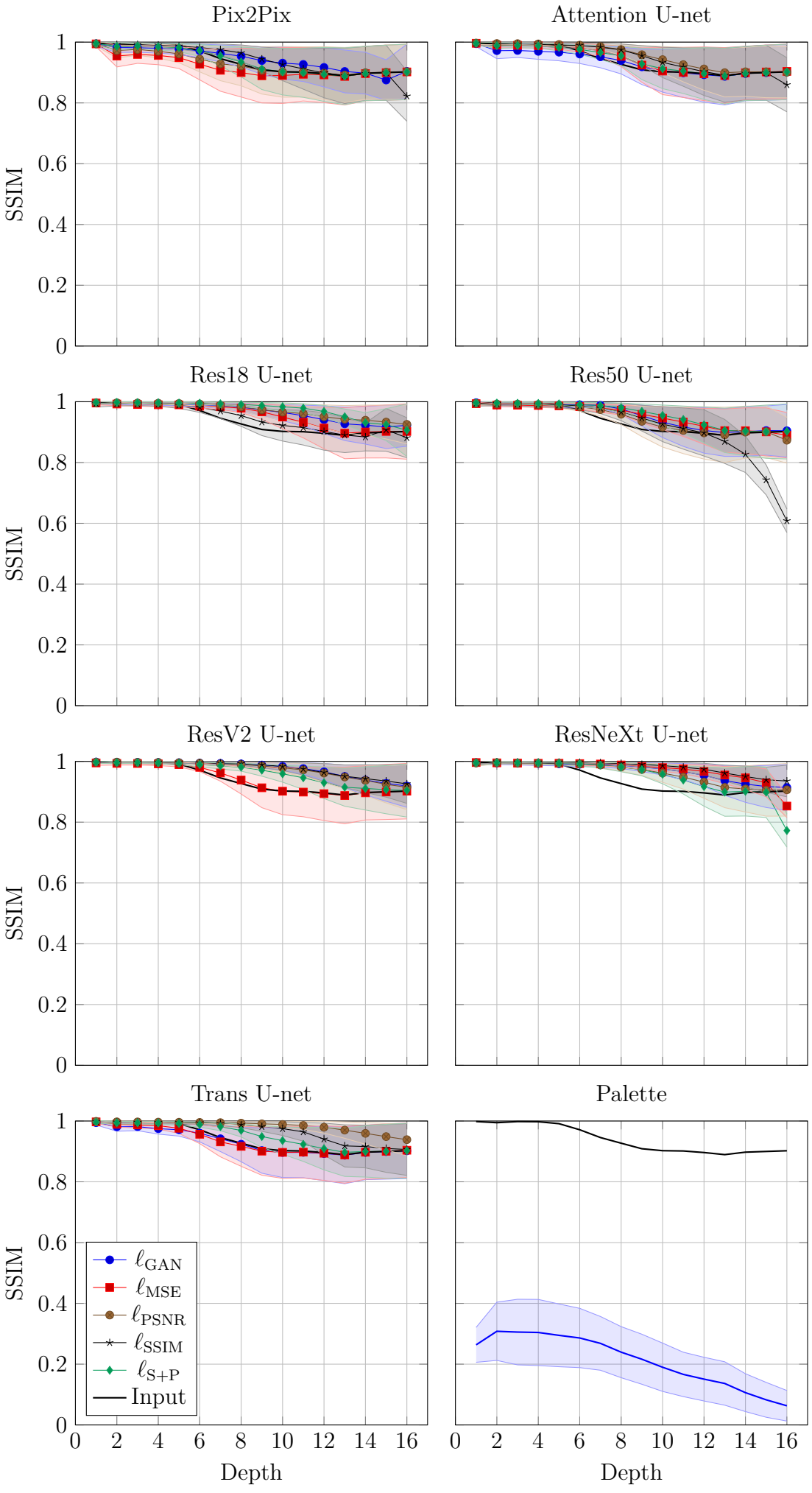


Figure E.10: SSIM score over depth of the models trained and tested on the NNE 40 dB dataset with their various loss functions.

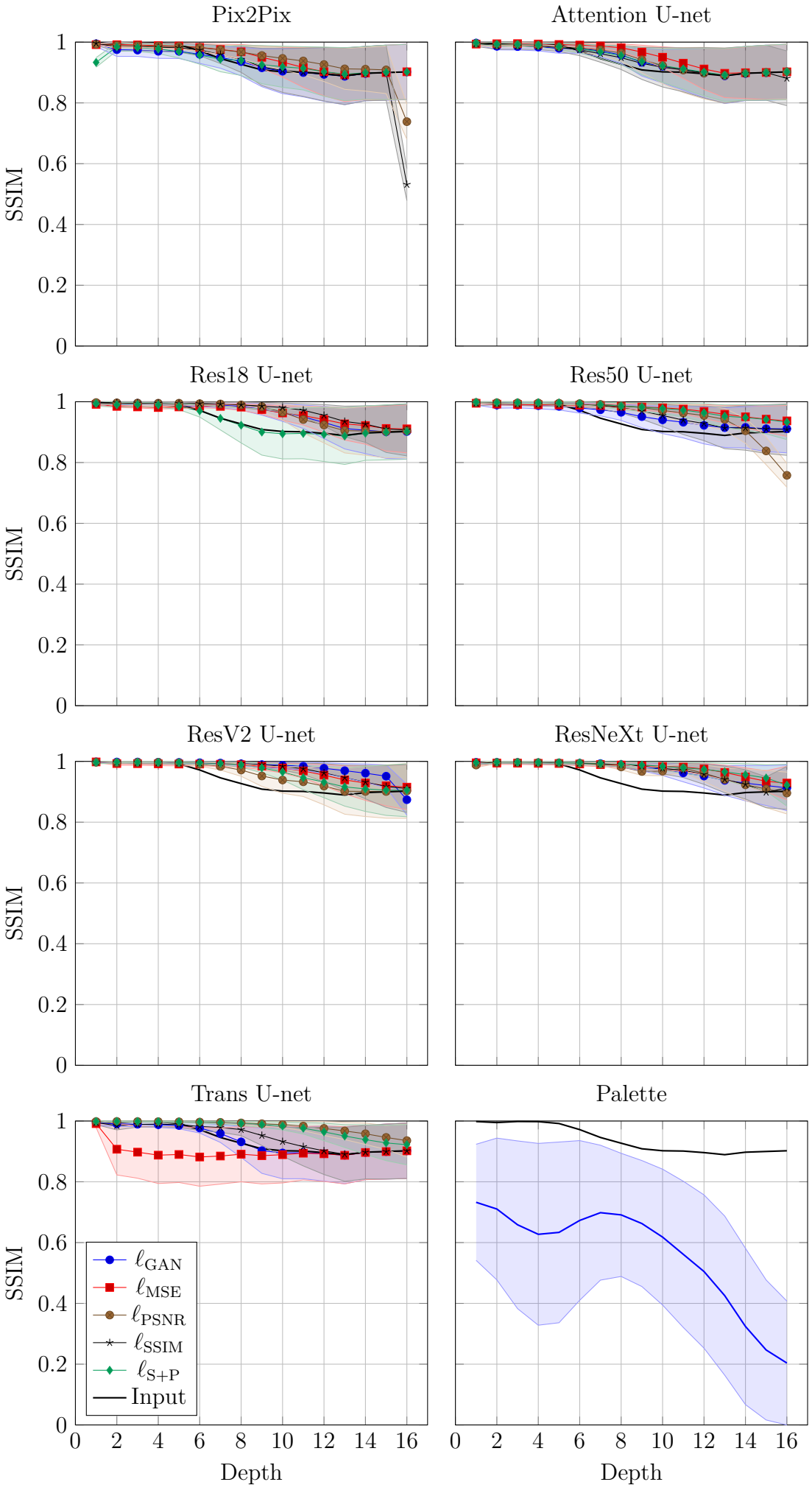


Figure E.11: SSIM score over depth of the models trained and tested on the NNE 50 dB dataset with their various loss functions.

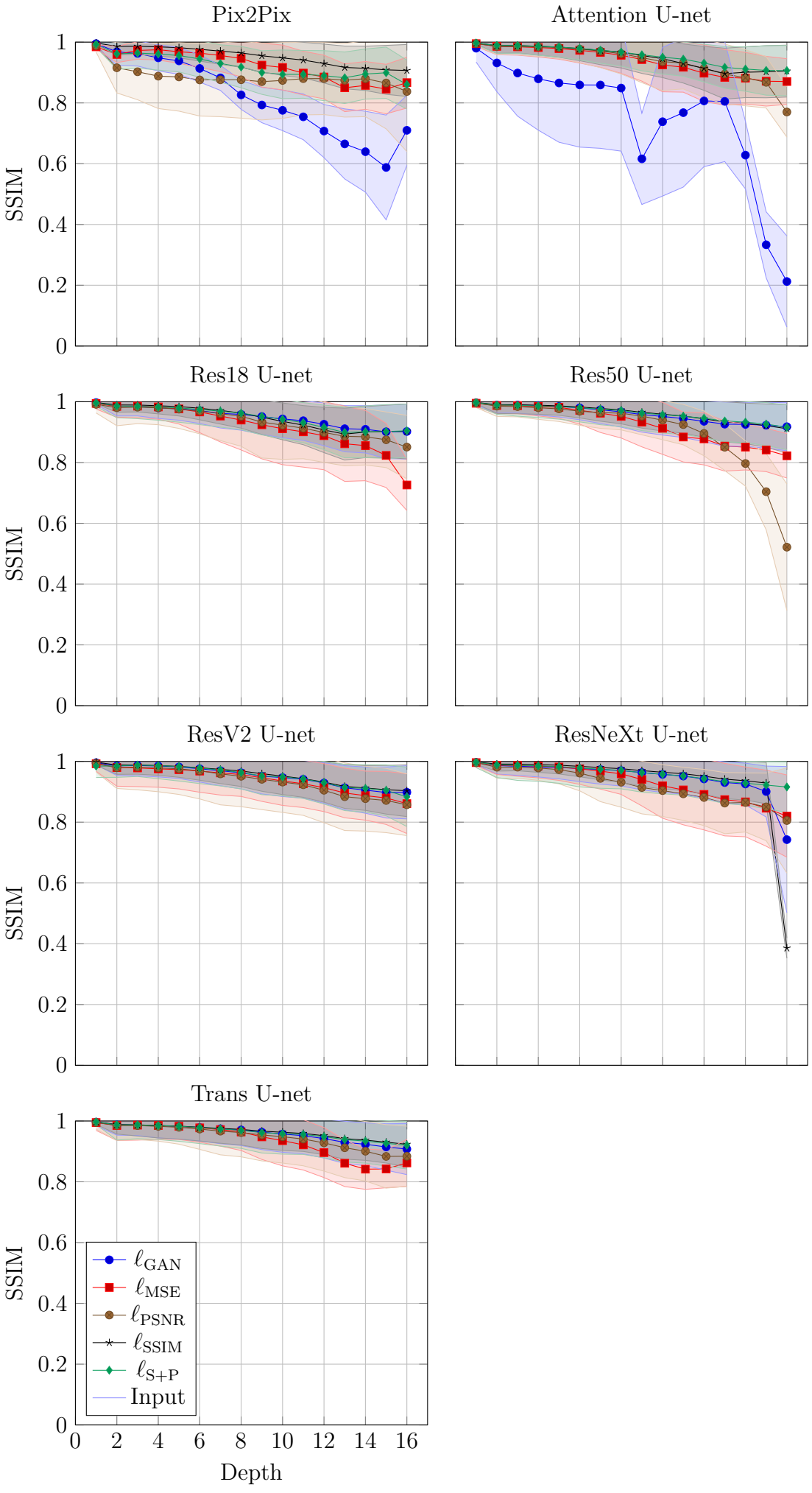


Figure E.12: SSIM score over depth of the models trained and tested on the NNE 10–50 dB dataset with their various loss functions.