Image Denoising Using LambdaNetworks

Loran Cheplanov, 206899700 Roi Ben-Gigi, 308132729

loran@mail.tau.ac.il roibengigi@mail.tau.ac.il

Abstract

In our work we dealt with the popular task of image denoising and tried to cope with it through adaptation of lambda layers. “Lambda layers” is a new approach of self-attention mechanism that bypasses the expensive memory usage requirements of attention maps and makes an efficient use of RAM. It was used as a main part in our model for accomplishing our goal and achieved success in terms of memory, PSNR and SSIM compared to more classical models that demand much more memory and epochs to train.

1. Introduction

The Lambda-layers was supposed recently by Bello [1] as a self-attention mechanism that reduces memory requirements and training time opposed to classical self-attention approaches. While even a local self-attention is a high demanding task for a standard computer, the lambda-layers find the solution by fitting each local area of the picture a linear transformation. It contains “information” about all the pixels in that local area and is actually can be thought as the “attention map” in the relevant area of the picture. This transformation needs to be computed only once and is applied on every pixel in the local image. The following figure demonstrates this idea:

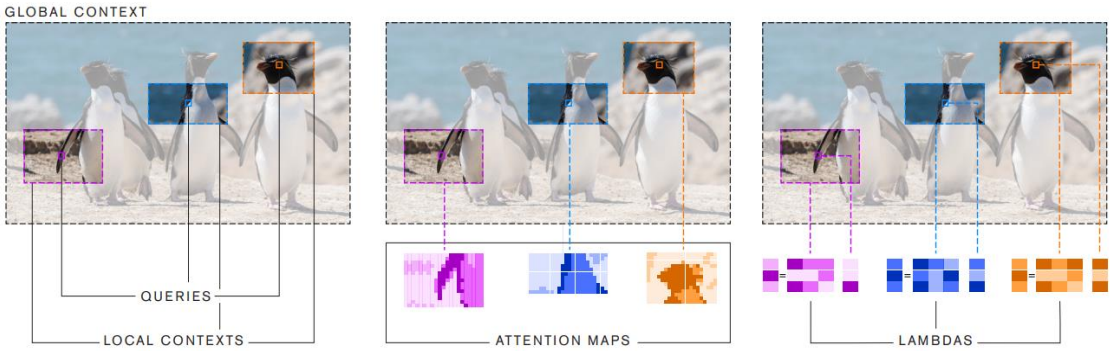

Figure 1: Comparison between Attention and Lambda layers. Left – a definition of a query (pixel) and a local context (the processing image). Middle – the original self-attention process that creates for each pixel a different attention map according to its relatives. Right – the Lambda function that creates an attention map for the whole local image. This single attention map is applied on each of the pixels in the local context and creates a deeper representation image.

In our net we used the lambda-layer as a main component to create a new and deep representation of each pixel, which served us for the mission of image denoising. This new representation was run then through some convolutional layers to a a final denoised picture.

Our models’ structure design is inspired by an auto-encoders’ structure that consists of a decoder and an encoder. The total net can be thought of as made of two parts: a lambda-layer that is built from 3 convolutional-layers (including the latent vector), and 2 more convolutional-layers taking the place of a decoder. This idea is enhanced and explained more in the methods part.

2. Related work

As we already mentioned our work is mainly based on the paper "LambdaNetworks: Modeling long-range Interactions without Attention" where the author used a "Lambda layer" as a self-attention tool in images. While Transformers that are the traditional way of modeling interactions usually require high memory and running time demands, the Lambda network is much more compact in that sense (see [1]).

The original goal of the LambdaNetworks focused on the task of classification. The work achieved good results firstly on the accuracy stats, and secondly on the net parameters amount. This gave us the inspiration to try the LambdaNetworks as an encoder on our task of image denoising, as each pixel is highly related to its surrounding.

We found that there is other use in linear layers in image tasks that also saves the need of attention maps, but still takes into account the spatial information (Katharopoulos et al., 2020; Choromanski et al., 2020 [2,3]). This concept indeed offers a scalable remedy for high memory usage but fails to model internal data structure.

As far as we checked there is no paper trying to use the Lambda-layer on the mission of image-denoising.

3. Data

The data we used to examine our model is the CIFAR-10. It consists of 32x32 color images in 10 different classes. There are a total of 60000 images in the dataset: 50000 for training and 10000 for test. In addition, in order to visualize our results with better resolution and for gaining another sight of results from a different "dataset", a few images were downloaded from the web and resized in our model from their high resolution to 512x512.

3.1. Clean Images

Each picture that entered our lambda-based net or the compared models was first scaled to values between [0,1], while the output was also scaled and matched to these range of values. Beside that we tried some more scaling options, yet found this scaling yields the best results.

3.2. Noised Images

There are some types of noises that we used to evaluate the performances of the models: Gaussian (σ = 0.1, 0.2, 0.3), Salt & Pepper and Speckle. These noises were added to the Clean images in different constellations that will be explained later in the paper. A figure of the general model:

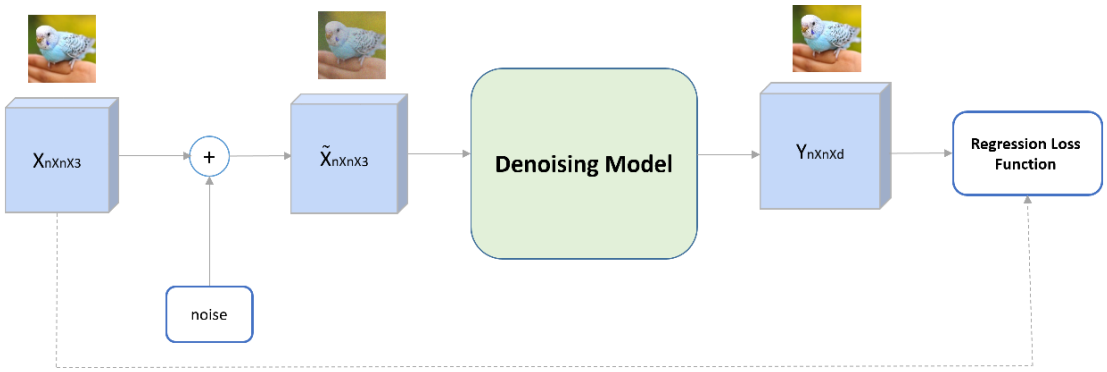

Figure 2: General image denoising training model. An image combines with an additive noise, passes through a model to achieve a denoised image. The loss function usually depends on the output models' image and the original noise-free image.

4. Methods

There are a lot of techniques to deal with image denoising. A prior one is by training an efficient Autoencoder. Generally, the picture first gets a new representation in lower dimension after going through layers of the encoder. After that the new representation enters into the decoder to get a new picture. The goal here is minimizing a loss function, usually MSE between the new picture that the autoencoder yields and the original image.

In our case we actually took a Lambda-net to maintain the role of an encoder for its ability to make a "self-attention" at a low price. That was done by enlarging the number of channels of the tensor V (see architecture in figure 5) and gaining a wide representation for each pixel. Then, we concatenated it with a designed decoder to input a denoised image.

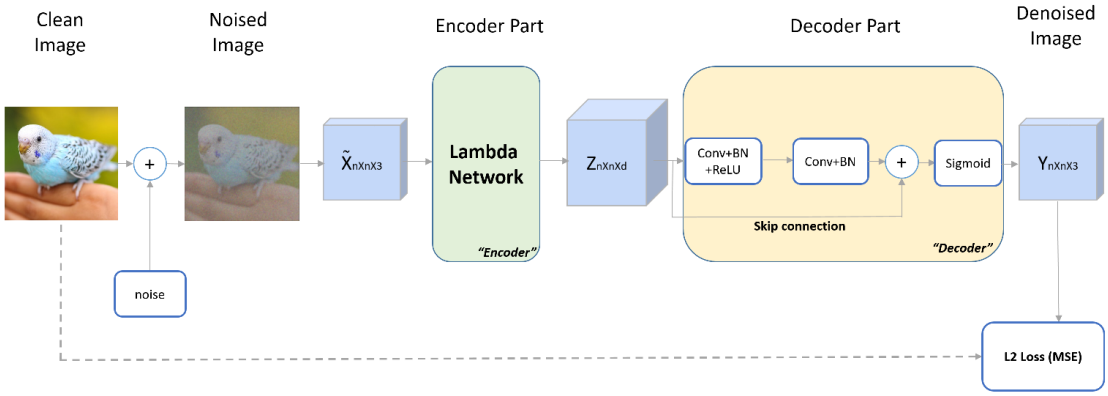

Figure 3: Our net scheme. The noisy image goes through a lambda-network, achieves a new and deeper representation and then passes through a decoder to finally obtain a denoised image.

In order that the self-attention would be local and more relevant to the pixels, the lambda net is not applied on the whole image but on local windows just like is done in regular convolutional networks. The size of that window is denoted by m and is one of the hyper-parameters that were checked in the training stage. For simplicity, we will demonstrate the explanations on the case that m=n.

<u>The connection between the encoder part and self-attention:</u>

In the notation of self-attention, every pixel in the input image is thought as a "query". That means that we have in total $n * n$ queries. To get more meaningful queries we enlarge their dimension by inserting them into a convolutional layer. At the same layer we also receive our "keys" and "values" matrices by inserting the image into a convolutional layer.

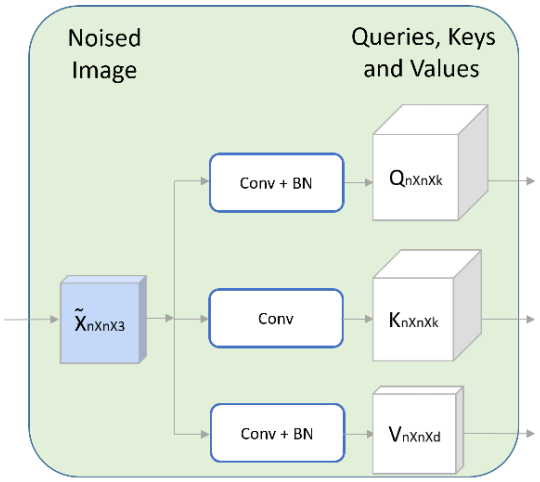

Figure 4: The 3 matrices of queries, keys and values that are created from the noised image in the first layer. The queries and keys matrices have 'k' channels, the values matrix has 'd' channels.

Simultaneously, a positional embedding vector is created for each pair of a queried pixel and other pixel in the image. In total there are $n * n$ relations. The purpose is to grant each pixel the information of its location in the image in relation to the surroundings. It is a crucial part for the net to work appropriately.

The second layer generates from the positional embedding, keys and the values matrices 2 linear functions: content Lambdas and positional Lambdas. The difference between the classical self-attention and our encoder (Lambda Network) appears in the creation of the attention maps – in the "Transformer" they are created from multiplication between the queries and the keys while in our model they are created by multiplication between the keys (K) and values (V). In that way we make a huge reduction of models' parameters usage – attention maps size $k * d$ instead of $n * n$.

Then as the third layer, the lambdas functions are applied to the queries, yielding outputs for each of the lambda types and finally the outputs are summed into a merged matrix Z which functions as our "latent vector". This matrix is a deep representation of the pixels that contains a local information deduced from the self-attention mechanism.

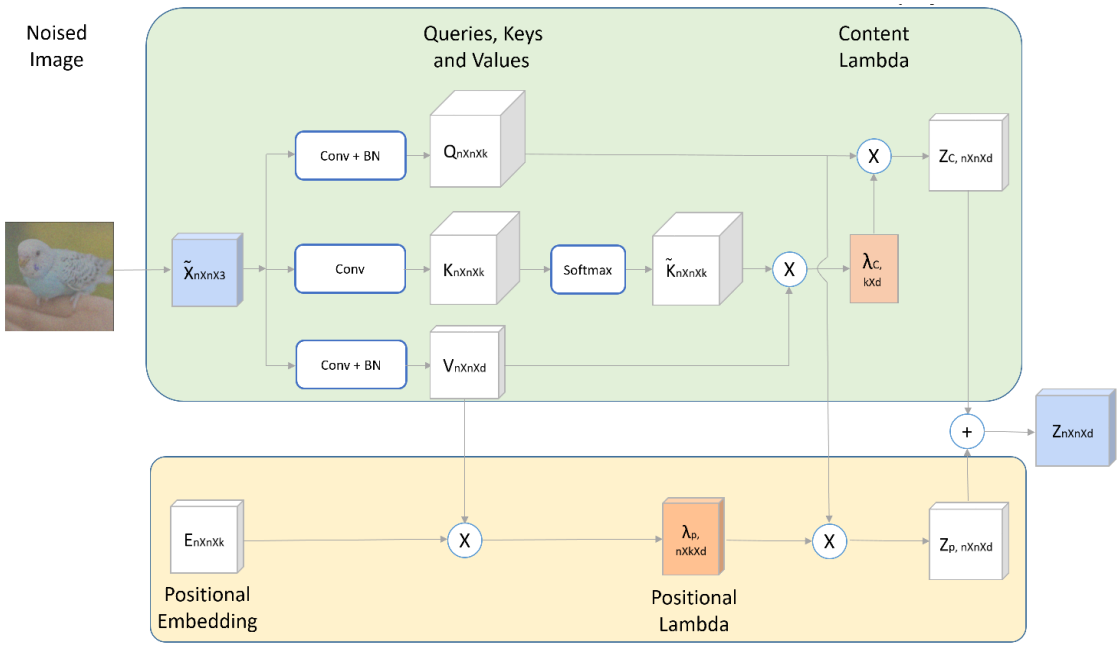

Figure 5: Insight into the lambda's part. First, the queries, keys and values are created based on the noised image. Then, the content Lambda is produced while in parallel the positional Lambda is created from a combination with the positional embedding and the values matrix. Finally, the Lambda functions are applied separately on the queries and finally summed up to a new representation of the noisy image that is self-attentioned.

The decoder part:

In our net we took the Z matrix (the output of the lambda net) and treated it as a bottleneck of an auto-encoder. We added a decoder that consists of 2 additional convolutional layers to receive the original images' dimensions ($n * n * 3$).

This decoder we took was selected to yield the best results after some experiments with changing the hyper-parameters, and we also added a "skip-connection" feature which was found to improve the results.

At the end of the decoder we use a sigmoid activation function. The sigmoid values ranges between [0,1] just as the original images values, which is the reason this activation function was taken.

The structure of our decoder is described in the following figure:

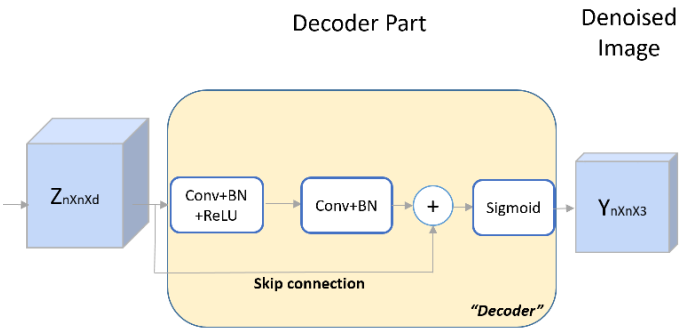

Figure 6: Models' decoder part

Results type of measurement:

At the test set we checked 3 main measurements: PSNR, SSIM and RAM/models' number of parameters for the noised image against the respective values at the denoised image:

1. The **PSNR** or the Peak signal-to-noise ratio is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Higher PSNR generally indicates that the image is of higher quality.

$$\mathrm{PSNR} = 10\mathrm{Log}_{10}\frac{\arg\max(Image)^2}{\mathrm{MSE}}$$

2. The structural similarity index measure (**SSIM**) is a method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. SSIM is used for measuring the similarity between two images. higher SSIM generally indicates that the image is more loyal to its origin.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

   Where x and y are the given images, μ_x the average of x, μ_y the average of y, σ^2_x the variance of x, σ^2_y the variance of y, σ_{xy} the covariance of x and y.

3. **RAM, model parameters** - for each net that we ran, the amount of parameters that they concluded was checked. This measure reflects the memory demand of the net and its running time. Hence, we definitely see the model parameters amount as a crucial consideration for choosing which model is preferred.

The compared models:

Our model was compared to two main models: DnCNN and denoising auto-encoder. It is important to say that we also made an optimization stage on hyper-parameters of the compared models, so they yield their best results.

The net depth, which usually has a more effect on running time, stayed six for all the combinations of hyper parameters we tried in our lambda-based net. As for, we compared the performance of our net with auto-encoders containing 6 layers, so our net is less depth then the nets we compared with and can be fairly compared in a sense of running time.

We mentioned for each experiment how the hyper-parameters affect the number of parameters in the net, which has more influence on memory demands, less on running time.

Therefore, we indeed find the number of parameters a good measure for a better model to use, along the equality of the denoising.

The compared model structures:

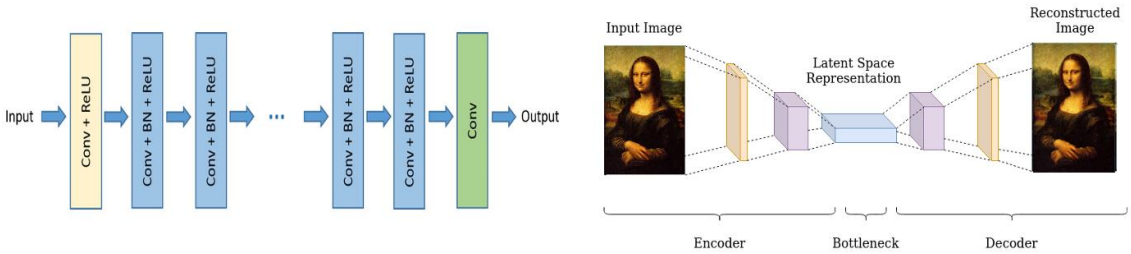

Figure 7: The comparable models' schemes. Left – DnCNN, built from 17 layers in total. Right – a denoising auto-encoder, built from

5. Experiments

5.1. Optimizations

In order to choose our best model, optimizations were made in some aspects:

Architecture

In the encoder, the basic structure of the LambdaNetwork was preserved while the encoder was examined with some designed types. At first a comparison was made between implementation of fully-connected layers and convolutional layers, which resulted in better scores with the convolutional layers. A combination with skip connection was checked and approved to gain even better scores. The final design of the encoder part (Figure 6) includes 2 convolutional layers that are combined with batch normalization, ReLU and a skip-connection from the input to the output (before applying the final Sigmoid activation function).

Loss Function

A comparison was made between using 2 loss functions - MSE loss and BCE loss, which achieved similar results. The MSE loss is frequently used for regression problems, due to its convexity and the fact that absolute minimum is promised in most optimization techniques under convexity. The MSE loss was the final choice, its formula is:

$$\mathrm{MSE} = \frac{1}{M \times N} \sum_{i=1}^{N} \sum_{j=1}^{M} \left[I(i,j) - I'(i,j) \right]^2$$

Where M, N are the dimensions of the image, I and I' are the images being compared.

Hyper-Parameters

As has been done in classical optimizations, the hyper-parameters of the model were tuned in order to achieve the best results. The optimization was done on the learning rate, encoders parameters (output channels, m and u – which are detailed in the appendix), weight decay, optimization functions (Adam, SGD, AdaMax), activation functions (ReLU, LeakyReLU, ELU, Sigmoid, Tanh).

As was mentioned before, the hyper-parameters optimization was made also on the compared models DnCNN and denoising auto-encoder.

5.2. Final Model Training Settings

The analysis of the model was made with consideration of the trade-off between the memory usage (models' number of parameters) and the denoising scores (PSNR and SSIM). Therefore, some scenarios were tested and there are some hyper-parameters that will be mentioned as "dynamic parameters" that can be chosen depending on the need – gaining higher denoising score or less memory usage. The basic hyper-parameters that are shared with all of the scenarios are shown in the table below.

Mini-batch size	256
Image size	32X32X3
k	16
Epochs number	50
Optimization function	Adam
Learning rate	0.001
Learning rate decay	0.99 (each epoch)
Weight decay	0.00001
Criterion	MSE Loss
Library	Pytroch
Workspace	Google Colab (GPU)

Table 1: Our models' basic parameters

5.3. Tested Scenarios and Results

Firstly, the influence of the decoders' number of output channels was studied in aspects of denoising scores and model parameters that are used (the SSIM score remained around 0.88 and is not mentioned in the graph below). The scenario was tested on a Gaussian noise (σ = 0.1), with model hyper-parameters u=1 and m=23.

As can be seen from Figure 8, the PSNR score grows especially in the beginning and reaches a threshold around output channels of 128, while the growth in the models number of parameters has an exponential behavior. Our recommendation is to pick output channels between 16 to 128 for achieving optimal trade-off. For the further scenarios an output channels (d) of 16 was chosen.

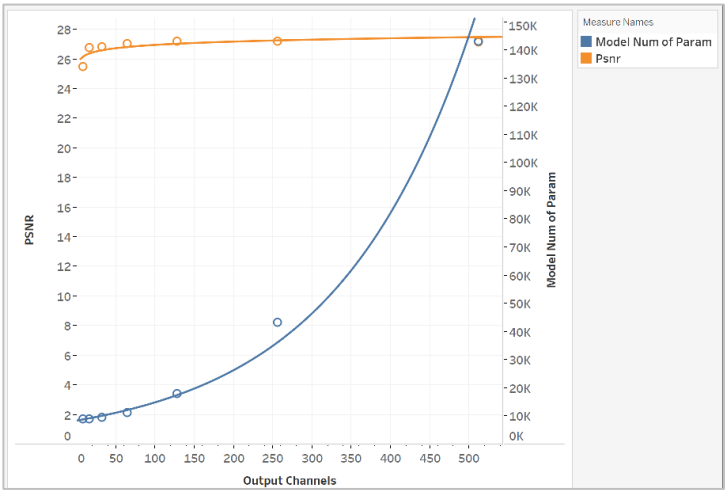

Figure 8: Relation between PSNR, models' number of parameters and the encoders' output channels of our model

Secondly, a check of the amount of epochs to converge a minimum MSE loss was checked and resulted around 50 epochs. The model can be trained more than 50 epochs, but the results difference is negligible.

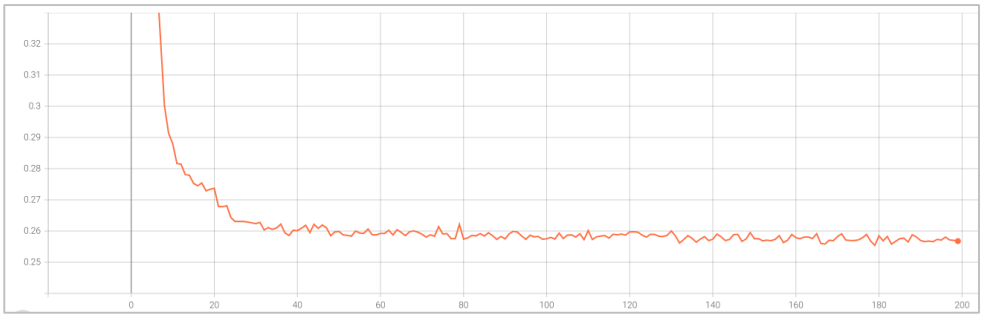

Figure 9: Convergence graph of our model

Thirdly, the impact of different noise types was analyzed on our and the comparable models. An analyzation of the hyper-parameters u and m was made in order to find the combination that reduces even more the use of model parameters, in an optimal trade-off with denoising scores. These parameters are shown only for Gaussian noise (σ = 0.1) for demonstrating an additional option of RAM reduction. The final test results which are based on the CIFAR10 test dataset are shown in the table below:

Trained Noise Type	Model Name	Epochs Number to Converge	u	m	PSNR	SSIM	Model Parameters
Gaussian (σ=0.1)	LambdaNet	50	1	23	26.88	0.88	9k
	LambdaNet	50	4	7	26.71	0.88	4k
	Denoising Auto-Encoder	300	X	X	26.23	0.87	47k
	DnCNN	200	X	X	26.7	0.81	558k
Gaussian (σ=0.2)	LambdaNet	50	1	23	23.47	0.78	9k
	Denoising Auto-Encoder	300	X	X	24.38	0.82	47k
	DnCNN	200	X	X	22.79	0.81	558k
Gaussian (σ=0.3)	LambdaNet	50	1	23	21.67	0.71	9k
	Denoising Auto-Encoder	300	X	X	21.97	0.71	47k
	DnCNN	200	X	X	22.75	0.76	558k
Speckle	LambdaNet	50	1	23	24.8	0.84	9k
	Denoising Auto-Encoder	300	X	X	24.38	0.82	47k
	DnCNN	200	X	X	26.41	0.89	558k
Salt & Pepper	LambdaNet	50	1	23	31.18	0.96	9k
	Denoising Auto-Encoder	300	X	X	26.28	0.88	47k
	DnCNN	200	X	X	35.66	0.99	558k

Table 2: Impact of different noise types the models performances.

As can be seen from the table above, our model wins without a doubt in aspects of memory and epochs number that it takes to converge in comparison to the other models. In terms of the various applied noise types, the best received denoising scores are changing:

- For a Gaussian noise (σ = 0.1), **our model** is the best (PSNR=26.88, SSIM=0.88), the other models have similar scores.
- For a Gaussian noise (σ = 0.2), the **denoising auto-encoder** is the best (PSNR=24.38, SSIM=0.82), **our model** is not far behind.
- For a Gaussian noise (σ = 0.3), **DnCNN** is the best (PSNR=22.75, SSIM=0.76)
- For a Speckle noise, **DnCNN** is the best (PSNR=26.41, SSIM=0.89)
- For a Salt & Pepper noise, **DnCNN** has an astounding scores (PSNR=35.66, SSIM=0.99), while **our model** has successful results too (PSNR=31.18, SSIM=0.96)

In Figure 10, a visualization comparison was made between the models. In this testing part we used several random images from the web that were resized from high-resolution images to 512x512. It can be seen that a matching occurs between the numerical scores and the visualized denoised images. In general, it seems that DnCNN tends to hard-smooth the images. That can sometimes serve as an advantage, but in many cases it can constitute as a problem since some of the data loses this way.

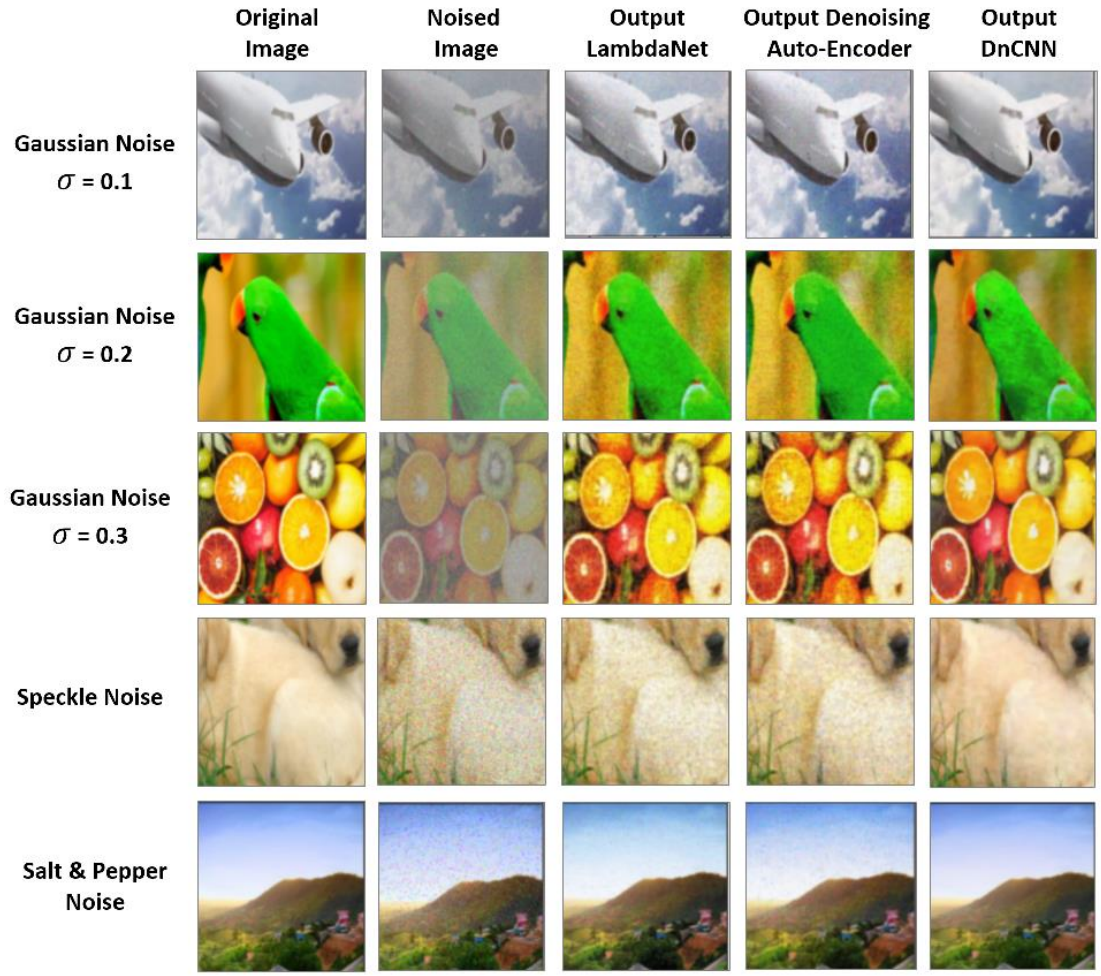

Figure 10: Visualization of models performance. The images were resized from higher dimensions to 512x512 and a closer view was captured for enabling to distinguish the differences between them.

6. Conclusion and Further Work

A successful proof of concept was achieved – beyond of the mission of classification, the Lambda-model can also serve for the mission of image denoising. Surprisingly, the PSNR and SSIM scores of our model are pretty high with respect to the fact that the net was designed in advance with the goal of receiving a memory usage reduction, which usually has a trade off with the performances. Furthermore, the achieved memory usage reduction is significant in comparison to our 2 other tested models. As also can be seen, our model converges very fast to good results in the training, which can be useful for additional trainings on other scenarios and noise types.

In order to make our model even better, there are some existing vectors to delve in: a further optimization can be made on the 'k' parameter, usage of layer normalization instead of batch normalization and increasing the batch size, architecture adjustment with transfer learning, training and testing on other noises and adding in the decoder part another skip connection between the first and the second convolutional layers.

7. Appendix

The code content and the PowerPoint presentation are attached in google drive (that was shared with barakhadad@mail.tau.ac.il) and in the following GitHub address:

https://github.com/loran100/Image-Denoising-Using-LambdaNetworks

List of Lambdas' net hyper-parameters that were tuned during the training process:

- m - size of the kernels in the lambda process, i.e the window of pixels that are "attentioning" each other
- u – intra-depth. It provides the keys and values a deeper representation that can be used later for creating an attention map for the whole image [1]

- d - channels size of the output matrix that comes out from the Lambda net and actually encodes local information for each pixel. The parameter is designed within the values matrix.

References:

[1] Irwan Bello. "LambdaNetworks: Modeling long-range Interactions without Attention". Virtual only, 2021.
[2] Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In Proc. Int. Conf. on Machine Learning (ICML), Virtual only, July 2020.
[3] Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. In Int. Conf. on Learning Representations (ICLR), Virtual only, 2021.
[4] Code of Lambda-net: https://github.com/leaderj1001/LambdaNetworks
[5] Code of DAE that is adjusted to the task of Image denoising: https://gist.github.com/bigsnarfdude/dde651f6e06f266b48bc3750ac730f80
[6] Implementation of DCNN: https://github.com/cszn/DnCNN