Continual-level Super-Resolution GAN based

Eliran Elisha
email: eliranelisha@mail.tau.ac.il

Harel Keller
email: harelkeller@mail.tau.ac.il

Department of Electrical Engineering, Tel-Aviv university, Israel

October 17, 2022

Abstract

Transitions between different levels of resolution are done discretely, which requires a specially-trained network for each transition between levels. Training each network for each level takes a lot of resources and time. The goal is to build one network which would allow a continuous transition between the different levels with a small number of parameters. The idea is to take a block of AdaFM and improve it by adding a GAN architecture to basis model. The results for transitions between the different resolutions are satisfactory in both numerical and visual aspects.

1 Introduction

The real world consists of physical processes which are continuous processes. The digital world, though, is a discrete world which discretizes all physical processes. The discretization process is common in the vast majority of the fields. Fields such as: signal processing; image processing; image restoration, and the list goes on. As you can see in the block diagram, there are always transitions from continuous to discrete.

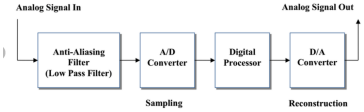

Figure 1: Description of the digitization process of physical signal reconstruction

The discretization process causes challenges in the field of image restoration, that is because deep models cannot be easily generalized to data of continuous and unseen levels. Therefore, we need to train a network for each level. But we must keep in mind, this requires several models which take resources and time. There have been attempts to train one network for all levels of resolutions, but this wound not being entirely practical since a network requires a lot of datasets with different resolutions. This network contains tens of thousands - if not millions of parameters! - and its accuracy at some of the

resolution levels is insufficient. The requirement for the network to make the transitions continuously has 2 main requirements: Firstly, applying models with mismatched restoration levels. This tends to produce either over-sharpened, or over-smoothed images. Secondly, in industrial and commercial scenarios (e.g., human-interactive software). It is often necessary to consecutively modulate the restoration strength/effect to meet different requirements.

In this work, we will implement the AdaFM block (explained in the related work section). We add GAN architecture for basis network. That is to see if the GAN architecture will improve performance in transitions from resolution to resolution; mainly from the opposite direction (X4 to X3). Because in this direction, it is necessary to restore pixels.

2 Related Work

We based on the article "Modulating Image Restoration with Continual Levels via Adaptive Feature Modification Layers". In this article, we'll use the principle of the AdaFM block. The AdaFM block approach stems from the observation that filters among networks of different restoration levels are similar at patterns while varying on scales and variances. Furthermore, the model outputs could change continuously by modulating the statistics of features/filters. The proposed framework is built upon a novel Adaptive Feature Modification (AdaFM) layer that modifies the middle-layer features with depth-wise convolution filters. In practice, we will first train a standard restoration CNN for the start level, and then insert AdaFM layers and optimize it to the end level. After the training stage, we will fix the CNN parameters and interpolate the filters of AdaFM layers according to testing restoration level. By tuning a controlling coefficient (ranging from 0 to 1), we can interactively and consecutively manipulate the restoration results/effects (note that we only need to train the CNN and AdaFM layers once - no further training is required in the test time). To ensure the output quality, we demonstrate that the model with AdaFM layers achieves comparable performance to the single-level image restoration network in both start and end level.
Besides, we also examine the properties of AdaFM layers: complexity, range and direction - providing a detailed instruction on the usage of the proposed method. Notably, the added AdaFM layers contribute to less than 4% parameters of the CNN model, yet the program achieves excellent modulation performance.

3 Data

For this project we used the "DIV2K" dataset: this dataset contains 1000 high-resolution (2K) images separated to train, validate and test parts. DIV2K is commonly used in image restoration tasks and divided into train set with 800 high definition high and resolution images; validation set with 100 high definition and high resolution images, and test set with 100 high definition and high resolution images.

The dataset was subbed into patches of 480x480 sized images and augmented by horizontal flipping and 90-degree rotations on-the-fly. In the test and compare parts, we also used dataset called "Set5" which is very common for testing and evaluating image restoration. The LR (Low-Resolution) data were sampled from these datasets via Bicubic image creator script in Matlab.

4 Method

4.1 Problem formulation

The problem of consecutive modulation of restoration levels can be formulated as follows: Suppose we have a "start" restoration level – L_a and an "end" restoration level – L_b. The objective is to construct a deep network to handle images with arbitrary degradation level

$$L_c(L_a \leq L_c \leq L_b)$$

Our solution pipeline consists of two stages – model training and modulation testing. In model training, we train a basic model and an adaptive model that could deal with level L_a and L_b, respectively. While in modulation testing, we propose a new network that can realize arbitrary restoration effects between level L_a and L_b by modulating certain hyper-parameters.

4.2 Adaptive Feature Modification

Inspired by the observations written above, we propose a continual modulation method by introducing an Adaptive Feature Modification layer and the corresponding modulating strategy. Our aim is to add another layer to manipulate the statistics of the filters, so they could be adapted to another restoration level. We can add a depth-wise convolution layer (or a group convolution layer with the group number equal to the number of feature maps) after each convolution layer and before the activation function (e.g., ReLU). We name the added layer as the Adaptive Feature Modification layer, which is formulated as

$$AdaFM(x_i) = g_i * x_i + b_i, 0 < i \leq N$$

where x_i is the input feature map and N is the number of feature maps. g_i and b_i are the corresponding filter and bias, respectively. It is worth noting that g_i depends on the degradation level of input images.

4.3 ESRGAN

The main architecture of the ESRGAN is the same as the SRGAN with some modifications. ESRGAN has Residual in Residual Dense Block (RRDB), which combines multi-level residual network and dense connection without Batch Normalization. Besides using standard discriminator, ESRGAN uses the relativistic GAN, which tries to predict the probability that the real image is relatively more realistic than a fake image.

$$L_D^{Ra} = -E_{x_r}[log(D_{Ra}(x_r, x_f))] - E_{x_f}[log(1 - D_{Ra}(x_f, x_r))]$$

$$L_G^{Ra} = -E_{x_r}[log(1 - D_{Ra}(x_r, x_f))] - E_{x_f}[log(D_{Ra}(x_f, x_r))]$$

4.4 ELGAN

We changed the base network (which was a simple convolution network) from the original article to the ESRGAN network. In addition to the training process, we will teach the network to produce images with the same resolution as the original. We'll add the AdaFM block that will allow a continuous transition between the different resolution levels to the ESRGAN network. The hope of integrating a GAN architecture in our case ESRGAN will visually improve the transition between the resolutions mainly by moving back for example from X3 to X2 that it is necessary to restore pixels. Since the network learns to produce images like the original, it will be easier for it to reproduce pixels, and thus improve the resolution. It is clear that in metric terms, the

PSNR will be lower than the PSNR of the AdaFM because the new network (ESRGAN) adds new details; therefore there is a difference between the original image and the image that the network released. Resulted of this, we need to use a new metric which will better represent the improvement that the network has added visually; hence we will use non-numerical metrics apart for a network that has been trained for this (for example: LPIPS).

4.5 Model Training

In this subsection, we discuss how to utilize the proposed AdaFM layer for model training. The entire model consists of a basic network and the AdaFM layers. First, we train the basic network N^a_{bas}, which can be any standard CNN model, for the start restoration level L_a. Then we insert AdaFM layers to N^a_{bas} and form the AdaFM-Net N_{ada}. By fixing the parameters of N^a_{bas}, we optimize the parameters of AdaFM layers on the end level L_b. Experiments demonstrate that by only finetuning the AdaFM layers, the model N^b_{ada} could achieve comparable performance with a basic model N^b_{bas} trained from scratch on level L_b. As the AdaFM-Net is optimized from L_a to L_b, we name this process as adaptation, and use adaptation accuracy to denote its performance. Specifically, we can use the PSNR distance between PSNR of N^b_{ada} and N^b_{bas} bas as the measurement of adaptation accuracy. There are three factors that affect the adaptation accuracy: filter size, direction, and range.

- For filter size, a larger filter size or more parameters will lead to better adaptation accuracy. We try filter size from 1×1 to 7×7. From convergence curves shown in figure 2, we find that 3×3 performs much better than 1×1 while 7×7 is only comparable to 5×5. Further increasing the filter size will not continuously improve the performance.
- For direction, different restoration levels have different degrees of difficulty for the same network. Which begs the question: should we modulate the model from an easy level to a hard level, or the opposite direction? Experimentally, we found that from easy to hard is a better choice.
- For range, the smaller of the range/gap $\|L_b - L_a\|$ the better the adaptation accuracy. For example, in super resolution problem, transferring the filters from ×2 to ×3 is easier than from ×2 to ×4. In Section 4, we conduct numerous experiments to choose the best range for super-resolution.

4.6 Modulation testing

After the training process, we discuss how to modulate the AdaFM layers according to degradation level at test time. As the features remain the same after convolution with an identity filter, we initialize AdaFM layers with identity filters I and zero biases, which is regarded as the start point of AdaFM layers. We can linearly interpolate the parameters of AdaFM layers as

$$g^*_i = I + \lambda(g_i - I), b^*_i = \lambda b_i, 0 < i \leq N$$

Where g^*_i , b^*_i are the filter and bias of the interpolated AdaFM layers,$\lambda(0 \leq \lambda \leq 1)$ is the interpolation, coefficient is determined by the degradation level of input image. After adding the interpolated AdaFM layers back to the basic network Na bas, we can get the AdaFM-Net N_c Ada for a middle level $L_c(L_a \leq L_c \leq L_b)$. The effects of changing the coefficient λ from 0 to 1 are shown in Figure 4,

where the output effects change continuously along with λ. Interestingly, we find that the interpolated network could fairly deal with any restoration level L_c between level L_a and L_b by adjusting the coefficient λ, which behaves like a strength controller in traditional methods. Experimentally, we found that the relationship between the coefficient λ and restoration level L_c can be formulated/approximated as a polynomial function:

$$\lambda = f(L_c) = \Sigma_{j=0}^{M} \omega_j L_c^j$$

where M is the order and $\{w_j\}_0^M$ are coefficients. To fit this polynomial function, we need to determine at least M points $\{L_c^i, \lambda^i\}_{i=0}^M$. Specially, the start point is $\{L_c^0 = L_a, \lambda^0 = 0\}$ and the end point is $\{\lambda^M = 1, L_c^M = L_b\}$ Furthermore, we require a test set with degraded images and ground truth to measure the adaptation accuracy. For a middle level L_c^i, we use the test images of level L_c^i as inputs. By adjusting the coefficient λ, the AdaFM-Net could generate a series of outputs. We select the λ which achieves the highest PSNR (evaluated on the test set) as the best coefficient, recorded as λ^i for L_c^i. It is worth noticing that the modulation process and curve fitting require no additional training. Extensive experiments show that the fitting curve varies a lot with ranges and problems. As an alternative choice, we can also use the piece-wise linear function for approximation. Actually, when the range is small enough, the relationship between λ and L_c is almost linear. We can train a set of AdaFM-Nets on middle levels $\{L_c^i\}$. For a given level $L_c (L_c^i < L_c < L_c^{i+1})$, we can use the coefficient

$$\lambda = (L_c - L_c^i)/(L_c^{i+1} - L_c^i)$$

to interpolate the AdaFM-Nets between L_c^i and L_c^{i+1}. This strategy requires training and storing more AdaFM-Nets on middle levels, but the adaptation accuracy is comparably higher due to the small range.

5 Experiments

5.1 AdaFM SR training

Following AdaFM (and SRResNet) recommendation with a little modification, the train batches size stated to 16 and the HR patch size is 96 × 96. For this part we used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and L1 as the loss function. For model training, the initial learning rate value was $1 \times e^{-4}$, and then decayed by a factor 10 after $5 \times e^4$ iterations. All models are built in PyTorch library and executed with NVIDIA 1650 (Laptop) and the "Colab Pro+" GPU (may be changed due to how busy the servers are).

5.2 ELGAN implementation and training

For this task, we took ESRGAN network and integrated with the AdaFM block and build our network, ELGAN. We used Adam optimizer as our GAN optimizer. We used the same learning rate and kept all AdaFM settings as mentioned above.

5.3 Evaluation of Model Training

In this section, we evaluate our proposed method on Super-Resolution tasks. This section contains our AdaFM PSNR and visual results compared to the AdaFM PSNR results from the paper, ELGAN PSNR and visual results, comparing LPIPS Results.

In the beginning, we will determine the best filter size (K - Kernel size) and the number of the iterations to converges.

the original article trained the model and tested the PSNR along the iterations for each $K = 1, 3, 5, 7$. We did the same, trained and tested the model for each $K = 1, 3, 5, 7$ as written in the paper.

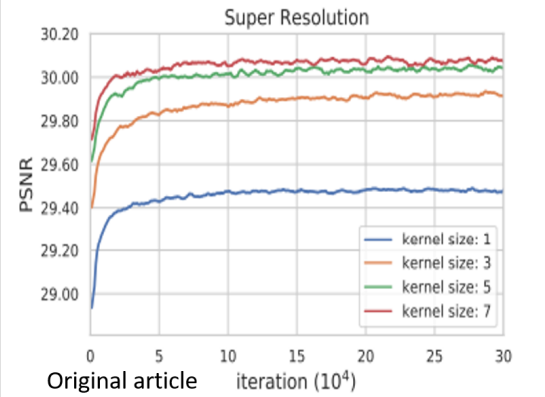

Figure 2: PSNR over trains iterations for each Kernel size - original article results.

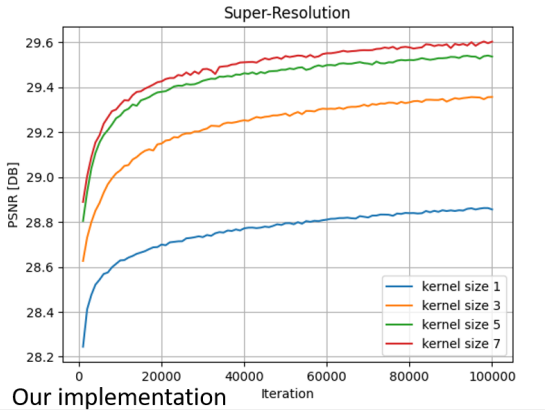

Figure 3: PSNR over trains iterations for each Kernel size - our results.

As we can see, we got approximately the same number of iteration to converges - around $1 \times e^5$, and the best K size is 7. There is a minor difference in the results between $K = 7$ and $K = 5$ so the article chose $K = 5$, so we did this as well.

SR task from Bicubic x4 to x1: As we saw, the best K for this task is 5, after training the model $1 \times e^5$ times, we saw converges and after that we tested the model on the DIV2K testset and Set5, to compare our results with the article original results. The results will be used as indicator of implementation goodness and will be used as baseline for comparing with our ELGAN results. Results of Super-Resolution task x4 to x1 while $\lambda = 1$:

PSNR(dB)	DIV2K	Set5
1x1	29.89	31.42
3x3	30.20	31.88
5x5	30.28	32.00
7x7	30.30	32.03
Baseline	30.37	32.13

Table 1: original AdaFM article PSNR results of adaptation with different kernel sizes

PSNR(dB)	DIV2K	Set5
1x1	28.86	30.66
3x3	29.35	30.90
5x5	29.54	30.67
7x7	29.6	30.72
Baseline	29.38	31.03

Table 2: AdaFM implementation PSNR results of adaptation with different kernel sizes

As we can see, the original AdaFM results were a little bit higher than ours. We took this into account, because maybe they worked with higher-resolution images or with minor changes from the parameters they wrote in the article.

Visual results: For AdaFM net with Super-Resolution task x4 to x1, there is the visual results while changing λ between 0 to 1:

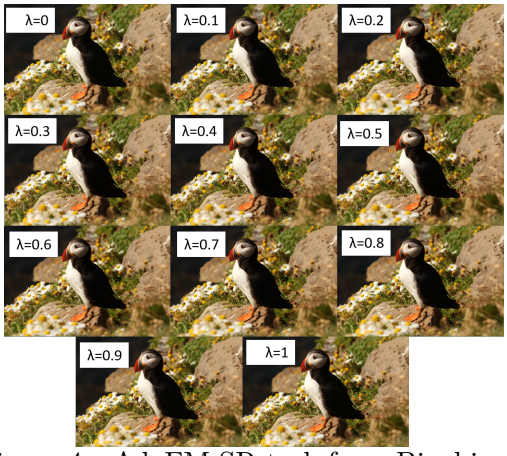

Figure 4: AdaFM SR task from Bicubic x4 to x1 for $\lambda = 0 : 0.1 : 1$

The best result achieved for $\lambda = 1$:

Figure 5: x4 image (the input image) vs SR result with $\lambda = 1$ vs High-Resolution image

There is a visual improvement by using the AdaFM net, the best visual result achieved for $\lambda = 1$ as expected.

ELGAN results: ELGAN net trained for 5e4 while AdaFM parameters stayed the same, with Kernel size of 5. At first, the model been tested with the same test-sets as AdaFM results. results of Super-Resolution task x4 to x1 while $\lambda = 1$:

PSNR(dB)	DIV2K	Set5
1x1	27.52	27.23
3x3	27.6	27.73
5x5	27.66	27.7
7x7	27.67	27.79

Table 3: ELGAN PSNR results of adaptation with different kernel sizes

As observed, ELGAN PSNR results are lower than AdaFM PSNR results. This occurred because the GAN added new details to the image, so the result image may look greater but it's more different than the original image.

Visual results while changing λ between 0 to 1:

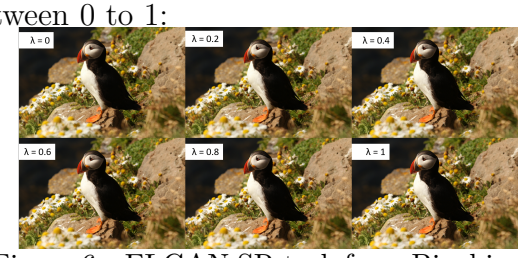

Figure 6: ELGAN SR task from Bicubic x4 to x1 for $\lambda = 0 : 0.2 : 1$

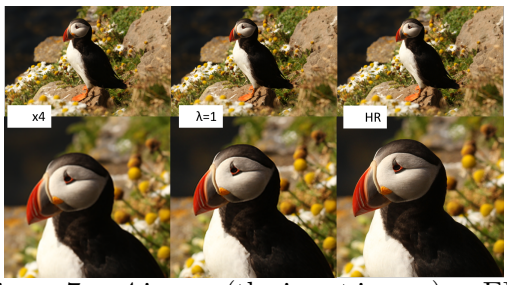

Figure 7: x4 image (the input image) vs ELGAN result with $\lambda = 1$ vs High-Resolution image

There is a visual improvement by using the ELGAN net. The best visual result achieved for $\lambda = 1$ like in the AdaFM implementation result.

Results of Super-Resolution task x3 to x1 while λ changed between 0 to 1:

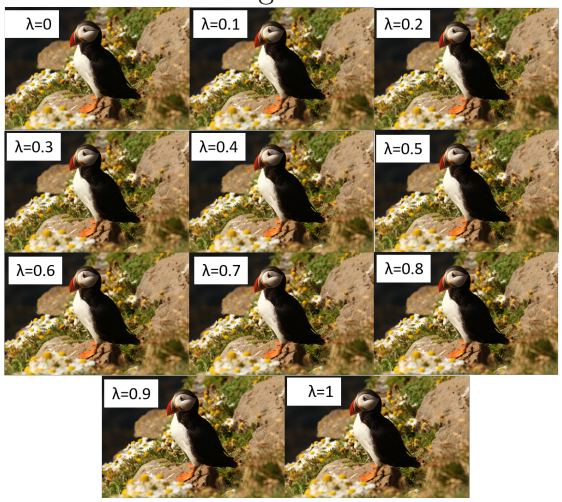

Figure 8: ELGAN SR task from Bicubic x3 to x1 for $\lambda = 0 : 0.1 : 1$

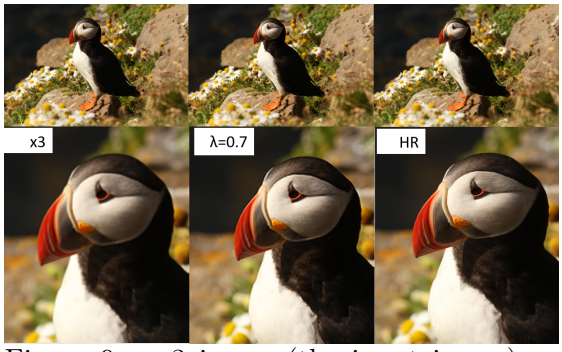

Figure 9: x3 image (the input image) vs SR result with $\lambda = 0.7$ vs High-Resolution image

There is a visual improvement by using the ELGAN net. The best visual result achieved for $\lambda = 0.7$, like in the AdaFM article's predicted result. When we compare both of the models, we can see some interesting things: As written above, the PSNR of ELGAN is lower than our AdaFM implementation PSNR. But in the visual results we can see a little improvement. Visual examination between ELGAN and AdaFM implementation:

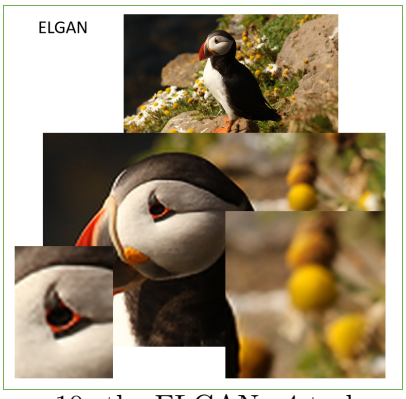

Figure 10: the ELGAN x4 task result

Figure 11: the AdaFM x4 task result

From observing both images, it can be seen that ELGAN image looks sharper and more realistic.

LPIPS is a NN based metric for evaluation a distance between images. As the distance goes lower - the images have more of the same features.

With usage of Alex net as the loss function, the average on the DIV2K test dataset results:

ELGAN LPIPS:	0.3206
AdaFM LPIPS:	0.2918
ELGAN better:	12 of 100 images

Table 4: LPIPS result with Alex net as the loss function

With usage of VGG net as the loss function: Average on the DIV2K test dataset:

ELGAN LPIPS:	0.329
AdaFM LPIPS:	0.328
ELGAN better:	52 of 100 images

Table 5: LPIPS result with VGG net as the loss function

The tables shows that the average LPIPS results of AdaFM net are better than ELGAN net. But with VGG as loss function of LPIPS, the ELGAN net shows better results in 52%.

For individual investigation, we took an image with the highest score difference between AdaFM implementation to ELGAN from the testset, image 770.

LPIPS	AdaFM	ELGAN	diff:
Alex net	0.405	0.505	0.1
VGG net	0.403	0.422	-0.019

Table 6: LPIPS score for image 770, with VGG and Alex net as the loss function

This image gets higher score between

AdaFM implementation to ELGAN in Alex net as loss function. Despite that, in VGG it's the opposite.

Visual results:

Figure 12: the ELGAN x4 task result on image 0x770

Figure 13: the AdaFM x4 task result on image 0x770

As observed, AdaFM implementation results look better.

LPIPS inserted to the project as a numeric evaluator for our SR tasks. We want to find a correlation between LPIPS score to the best visualize image we achieved by changing coefficient λ. For that, we built a table with LPIPS score for each λ when we know that the best visualize results are given by $\lambda(0.9 \leq \lambda \leq 1.0)$ for x4 task and given by $\lambda(0.6 \leq \lambda \leq 0.8)$ for x3 task.

On a single image 0x727.png (The Pengulin) from the testset, we created a table to investigate and finding the correlation between LPIPS score to the visual result.

x4 task	ELGAN		ADAFM	
λ	Alex	VGG	Alex	VGG
0.0	0.172	0.277	0.172	0.277
0.1	0.170	0.270	0.166	0.256
0.2	0.166	0.261	0.164	0.249
0.3	0.162	0.252	0.163	0.243
0.4	0.159	0.249	0.163	0.237
0.5	0.156	0.242	0.163	0.234
0.6	0.152	0.237	0.166	0.233
0.7	0.146	0.235	0.162	0.236
0.8	0.141	0.232	0.160	0.234
0.9	0.139	0.232	0.158	0.235
1.0	0.140	0.233	0.159	0.237

Table 7: LPIPS score for image 727 for x4 task with VGG and Alex net as the loss function for AdaFM and ELGAN for each $\lambda = 0 : 0.1 : 1$

x3 task	ELGAN		ADAFM	
λ	Alex	VGG	Alex	VGG
0.0	0.131	0.226	0.135	0.234
0.1	0.125	0.227	0.128	0.230
0.2	0.116	0.213	0.128	0.232
0.3	0.110	0.210	0.129	0.227
0.4	0.108	0.202	0.131	0.219
0.5	0.109	0.202	0.133	0.216
0.6	0.113	0.200	0.136	0.210
0.7	0.123	0.208	0.140	0.216
0.8	0.134	0.217	0.146	0.228
0.9	0.148	0.230	0.154	0.237
1.0	0.161	0.242	0.164	0.241

Table 8: LPIPS score for image 727 for x3 task with VGG and Alex net as the loss function for AdaFM and ELGAN for each $\lambda = 0 : 0.1 : 1$

Light Cyan metnion the lowest LPIPS score (best score) of each λ, we seeing a strong correlation between λ and the visual result.

6 Conclusion

We were able to reproduce the AdaFM model of the original article in a good approximation as can be seen from the results. According to the results of the implementation of the AdaFM model with the resources available to us, we were able to reach a good approximation to the same results according to the metrics used in the article. Our innovation, is that when the AdaFM block in the original article was adapted only to basic networks which are simple convolutional networks. We tried to adapt the AdaFM block to basic networks with a GAN architecture (for example the ESRGAN network) so that in test time, the model can be adapted to any restoration level directly adjusting the AdaFM layers without an additional training stage. As can be seen from the results that the base network with the GAN configuration known as ELGAN was not superior to the AdaFM network. While according to the PSNR metric, it is clear that there will be gaps because the ELGAN network adds details to the image and therefore will not be the same as the original image. According to the LPIPS metric, although the ELGAN network is not superior to the AdaFM network, it is very close to it in terms of results. That is, the ELGAN network does not manage to produce better results than the AdaFM network, but it maintains the principles and capabilities of the AdaFM layer for base networks of the type GAN. We are suggesting a little optimization to the proccess - run both AdaFM and ELGAN on tested image. The result with best LPIPS score will be the output of the system. **Acknowledgements,** Idan Ofer Huawei company who accompanied and guided us, Tel Aviv University and the course staff of Deep Learning for supply an infrastructure.

7 appendix

ELGAN code	Link
AdaFM paper	Link
AdaFM github	Link
ESRGAN paper	Link
ESRGAN github	Link
LPIPS github	Link
DIV2K Dataset	Link
Set5 Dataset	Link