Learning Texture Transformer Network for Image Super Resolution (Machine Learning 2021 Course)

Ghaith Mqawass [1] **Leonid Kulikov** [1] **Razan Dibo** [1]

Abstract

Image super-resolution (SR) is one of the vital image processing methods that upscale the resolution of an image and improve details in the field of computer vision. In the last two decades, significant progress has been made in the field of super-resolution, especially utilizing deep learning methods. This project is targeted to the replication, rigorous analysis, ablation study and enhancement of a Texture Transformer network for Image Super-Resolution TTSR. We minimized the network size in multiple different ways, checking the performance drop and found a balance. Also we injected dynamic convolutions that helped to significantly reduce artifacts on faces while also improving the details. Moreover, we mapped this network to a more complex task: Applying super-resolution with noisy data in order to check how stable the architecture is.

Github repo: https://github.com/theleokul/train-suite
Video presentation:
https://drive.google.com/file/d/1g2RRHqYdGptNbvBOIv6LPKDkyWhiRsUe/view?usp=sharing

1. Introduction

The concept of super-resolution was first used to improve the resolution of an optical system beyond the diffraction limit. In the past two decades, the concept of super-resolution (SR) is defined as the method of producing high-resolution (HR) images from a corresponding low-resolution (LR) image. Initially, this technique was classified as spatial resolution enhancement. The applications of super-resolution include computer graphics, medical imaging, security, and surveillance, which shows the importance of this topic in recent years. In the proposed paper, image super-resolution (SR) technique was applied to recover high resolution image with realistic textures from a low-resolution (LR) image. It depends on taking high-resolution images as references (Ref), so that relevant textures can be transferred to SR images. Moreover, authors in their work uses attention mechanisms to transfer high-resolution (HR) textures from Ref images. The outcome of their work was a novel Texture Transformer Network for Image Super-Resolution (TTSR), in which the LR and Ref images are formulated as queries and keys in a transformer, respectively. TTSR consists of four closely-related modules optimized for image generation tasks, including a learnable texture extractor by DNN, a relevance embedding module, a hard-attention module for texture transfer, and a soft attention module for texture synthesis. Such a design encourages joint feature learning across LR and Ref images, in which deep feature correspondences can be discovered by attention, and thus from a theoretical point of view, accurate texture features can be transferred. **The main contributions of this report are as follows**:

- Preparing datasets for training/evaluation
- Replicating paper results.
- Ablation study: Train networks with different scaling factors.
- Ablation study: Reducing the size of residual blocks (RBs)
- Enhancement: Applying dynamic convolutions.
- Mapping the network to a more complex task: denoising + super-resolution.
- Experimenting with different reference images.
- Finding weak spots
- Demonstrating some cool applications of the model.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Ghaith Mqawass <ghaith.mqawass@skoltech.ru
Leonid Kulikovleonid.kulikov@skoltech.ru
Razan Diborazan.dibo@skoltech.ru >.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

2. Related Work

In recent years, several works have been carried out in the scope of single image super-resolution (SISR) and reference-based image super-resolution (RefSR). SISR methods have achieved significant improvements over traditional non-learning based methods. Deep learning based methods in SISR treat this problem as a dense image regression task which learns an end-to-end image mapping function represented by a CNN between LR and HR images. A study carried by Dong et al. (Dong et al., 2015) is an example of using deep learning SISR methods. Authors used a convolutional neural network for super-resolution task. A three-layer CNN was used to represent the mapping function. The authors (Dong et al., 2015) further sped up the SR process by replacing the interpolated LR image with the original LR image where deconvolution takes place at the very last layer to enlarge the feature map. After that, attempts tried more deeper architectures using Residual Blocks to improve the quality of obtained SR images. Examples of these attempts can be found in the studies of (He et al., 2015) (Ledig et al., 2017). Furthermore, authors (Zhang et al., 2018) in their work added another step of improvement to residual block by adding channel attention. However all mentioned studies despite their improvements in this domain they used mean square error (MSE) or mean absolute error (MAE) as their objective function which ignores human perceptions. On the other hand, the studied SOTA paper (Yang et al., 2020) implements a more accurate objective function. This function consists of a linear combination 3 loss-components: **Reconstruction** loss, **adversarial** loss related to the WGAN network and **perceptual** loss.

3. Algorithms and Models

3.1. Texture Transformer Network

As a baseline model we used Texture Transformer Network from Image Super-Resolution (TTSR) (Yang et al., 2020) in x4 Super-Resolution task. It is a SOTA approach invented by Microsoft R&D department in 2020 that achieves significant improvement over previous SOTA solutions.

The main block of this network is a Texture Transformer is depicted on figure 1. It consists of several sub-blocks: Backbone, Learnable Texture Extractor, Relevance Embedding, Hard Attention and Soft Attention.

Backbone is a relatively small convolutional network which consists of several residual blocks with an additional global skip connection. Its purpose is to form expressive features from the low-resolution (LR) input.

The rest part of Texture Transformer is needed for relevant texture extraction from the reference (Ref) complement (hard-attention) and subsequent texture synthesis (soft-attention).

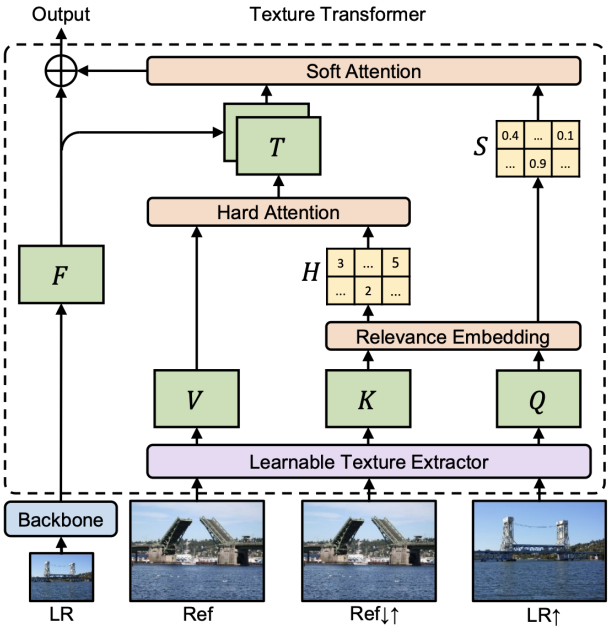

Figure 1. The proposed texture transformer. Q, K and V are the texture features extracted from an upsampled LR image, a sequentially down/upsampled Ref image, and an original Ref image, respectively. H and S indicate the hard/soft attention map, calculated from relevance embedding. F is the LR features extracted from a DNN backbone, and is further fused with the transferred texture features T for generating the SR output

Learnable Texture Extractor is an additional feature extractor, that pass inputs over VGG19 pretrained chunks. Different chunks represent different spatial scalings. In the original implementation three scalings are used: x1, x2, x4. The intuition behind the usage of different scalings is that it could be easier for network to extract features of different complexity from different spatial scalings of an image. But we will see in the Ablation study blocks that the amount of scalings can be reduced without significant performance drop.

Relevance embedding is a simple non-trainable function that allows to generate hard- and soft-attention maps. For that purpose, K (keys) and Q (queries) are unfolded into patches to get a vector for each spatial index. Then this vector representations of an upsampled LR and down-up sampled Ref are normalized and reduced by inner scalar product to form similarities (relevance) between patches. This relevance is further used to create the hard-attention map and the soft-attention map.

Weighted sum of different textures may result into blurry output. To avoid that, authors proposed to employ only the

most relevant texture using the hard-attention map, each element of which consists of the most relevant position (index) in the Ref image to the i-th position in the LR image. The hard-attention map is subsequently applied as index sampler to the Ref image to extract the most relevant patches.

To synthesize the new features based on the extracted textures from the Ref and features of the LR, soft-attention is used. i-th element of soft-attention map is a maximum relevance (not index as opposed to hard-attention map) corresponding to the i-th LR patch.

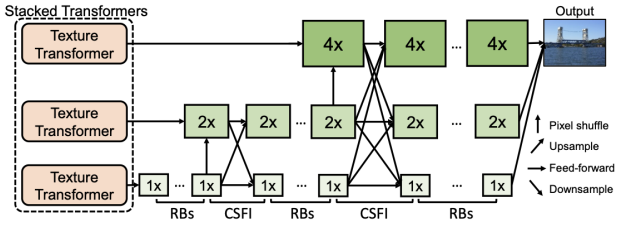

Figure 2. Architecture of stacking multiple texture transformers in a cross-scale way with the proposed cross-scale feature integration module (CSFI). RBs indicates a group of residual blocks. © [1]

Authors originally employ three scalings: x1, x2, x4. In order to fuse all this information into the network, additional residual blocks and so-called CSFI (Cross Scale Feature Integration) are used. CSFI allows to fuse features into each scale branch from other scale branches. Thus the final architecture is presented on figure 2.

One is highly encouraged to look into the original paper to see all the formulas (Yang et al., 2020) and intrinsic details. We consider that the given information is sufficient to understand the results in the next sections.

3.2. Dynamic convolutions

An idea of dynamic networks and dynamic parts of the networks seems to be perspective to study as it allows to use less layers for easy inputs and more for hard ones, thus the inference time is lower especially if simple examples are prevail and also they can provide better performance and additional regularization, which we'll discuss later in this section.

In simple terms dynamic convolution is a convolution that depends on an input. Specifically in (Verelst and Tuytelaars, 2020) the gating mechanism is applied. We construct an individual binary mask based on an input and apply the convolution operation only on the masked points i.e. only on those points that has 1 in the constructed binary mask. These discrete decisions in a mask, for every spatial location, are trained end-to-end using the Gumbel-Softmax trick.

We need the special Gumbel-Softmax trick for using this discrete spatial masks simply because we cannot do backpropagation on the function which has discrete output domain. Differentiation just doesn't work that way.

The Gumbel-Softmax trick turns soft decisions into hard decisions while enabling backpropagation, needed to optimize the weights of the mask unit.

Let's recall what a Gumbel distribution is. The random variable G is said to have a standard Gumbel distribution if $G = -log(-log(U))$ with $U \sim Uniform[0, 1]$. For us, its importance is a consequence that we can parametrize any discrete distribution in terms of Gumbel random variables by using the following fact:

Let X be a discrete random variable with $P(X = k) \propto \alpha_k$ random variable. And let $\{G_k\}_{k \leq K}$ be an i.i.d. sequence of standard Gumbel random variables. Then:

$$X = arg\max_k(log\alpha_k + G_k)$$

In other words, a recipe for sampling from a categorial or discrete distribution is: 1) draw Gumbel noise by just transforming uniform samples; 2) add it to $log\alpha_k$, which only has to be known up to a normalizing constant; and 3) take the value k that produces the maximum.

Unfortunately, the $arg\max$ operation that relates the Gumbel samples, the α_k's and the realizations of the discrete distribution is not continuous. One way of circumvent this is to relax the discrete set by considering random variables taking values in a larger set. To construct this relaxation we start by recognizing that 1) any discrete random variable can always be expressed as a one-hot vector (i.e, a vector filled zeros except for an index where the coordinate is one), by mapping the realization of the variable to the index of the non-zero entry of the vector, and 2) that the convex hull of the set of one-hot vector is the probability simplex:

$$\Delta^{K-1} = \{x \in \mathrm{R}_+^{\mathrm{K}}, \sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{x_k} = 1\}$$

Therefore, a natural way to extend (or 'relax') a discrete random variables is by allowing it to take values in the probability simplex. Both (Maddison et al., 2017) and (Jang et al., 2017) propose to consider the softmax map (indexed by a temperature parameter):

$$f_\tau(x)_k = \frac{exp(x_k)/\tau}{\sum_{k=1}^{K} exp(x_k/\tau)}$$

With this definition we can define (instead of the discrete valued random variable X the sequence of simplex-valued random variables:

$$X^\tau = (X_k^\tau)_k = \left(\frac{exp((log\alpha_k + G_k)/\tau)}{\sum_{k=1}^{K} exp((log\alpha_k + G_k)/\tau)} \right)_k$$

Gating decisions are binary, which makes it possible to strongly simplify the Gumbel-Softmax formulation. A soft-decision $m \in (\infty, \infty)$, outputted by a neural network, is converted to a probability α_1 indicating the probability that a pixel should be executed, using a sigmoid σ.

$$\alpha_1 = \sigma(m)$$

Then, the probability that a pixel is not executed is

$$\alpha_2 = 1 - \sigma(m)$$

Substituting α_1 and α_2 in the relaxed $arg\max$ operation, for the binary case of $K = 2$ and $k = 1$, makes it possible (Verelst and Tuytelaars, 2020) to reduce this to:

$$X_1^\tau = \sigma \left(\frac{m + G_1 - G_2}{\tau} \right)$$

We use a straight-through estimator, where hard samples are used during the forward pass and gradients are obtained from soft samples during the backward pass:

$$z = X_1^\tau > 0.5 \; (forward)$$
$$z = X_1^\tau \; (backward)$$

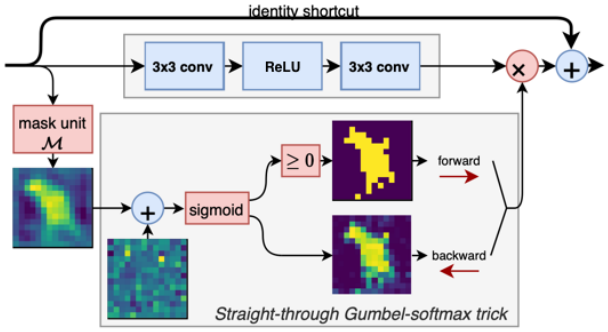

Figure 3. Training spatial execution masks using the Gumbel-Softmax trick. The mask unit generates a floating-point mask, after which the Gumbel-Softmax trick converts soft-decisions into hard-decisions and enables backpropagation for end-to-end learning

Having Gumbel-Softmax trick under the belt and it's relaxation to the binary case we are ready to dive into the details of the dynamic convolution operation that is presented on figure 3 in conjunction with a residual block.

Figure 3 visually explains all the details except the Mask unit $\mathcal{M}$. It is simply a combination of a convolution with one channel in the output and spatial average pooling. Therefore, it is a trainable layer that is corrected during backpropagation.

3.3. Losses

For a modern super-resolution usually three types of losses are employed: reconstruction, perceptual and adversarial. We used all of them in our work.

3.3.1. Reconstruction loss

Reconstruction loss measures the difference between pixels of the predicted image and the ground truth. There are different ways to aggregate that difference, in TTSR specifically $l1$ loss is used.

3.3.2. Perceptual loss

Perceptual loss helps to provide more visually pleasing results for a human eye. It is difficult to describe mathematical expression that matches human perception. Therefore, difference between features of a trained in advance deep neural network is used as a perceptual loss. Specifically, we used some VGG19 (Simonyan and Zisserman, 2015) chunks.

3.3.3. Transferal perceptual loss

Authors of TTSR also proposed to use one more perceptual (which is called transferal perceptual loss) between extracted reference textures and super-resolved VGG19 features to facilitate the network ability to pull appropriate parts of the reference.

3.3.4. Adversarial loss

We can call our TTSR network a generator that creates realistic to a human eye super-resolved images. In addition to the generator we can add a so-called discriminator, the entity that tries to distinguish the output of the generator from the high-resolution images. Thus, we can turn the training into the game between the generator and the discriminator where generator tries to fool the discriminator with artificially generated high-resolution (HR) images and the discriminator avoids mistakes. This is a "toy" description of a GAN (Goodfellow et al., 2014). Yet we consider that it is enough to grasp the subsequent sections. This adversarial setting can be encapsulated into an additional loss component.

As a discriminator we used the same network architecture as in TTSR. To be specific authors of TTSR employed Wasser-

stein GAN with Gradient Penalty (Gulrajani et al., 2017) (which we copied). Basically, this is just an extension to the original GAN framework that stabilize training.

3.3.5. Sparsity loss

Moreover, in later sections related to the dynamic convolutions we will employ one more loss component (5th to be exact), which is called Sparsity Loss (Verelst and Tuytelaars, 2020). Intuitively, it allows to constrain the number of desired activated pixels in the discrete mask units. It provides additional regularization which we will also employ in order to force the residual blocks to learn only some parts of the input. For example, the residual block a could effectively extract low frequency features from the sky which is usually located on the top of an image while the residual block b would effectively work with high frequency details on the grass or other high frequency textures. This naive constraints (duty separation between residual blocks) surprisingly allow us to get less artifacts, especially on faces, and improve the quality of details in general which is demonstrated in the sections related to the experiment with the dynamic convolutions.

3.4. Metrics

3.4.1. PSNR

In this report we will only use Peak Signal-to-Noise Ratio (PSNR) for performance evaluations. There is also a well-known metric Structure SIMilarity (SSIM), but because it usually correlates with PSNR in our experience we omit it for simplicity.

PSNR - 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.

$$PSNR(G, R) = 10log_{10}\left(\frac{MAX_G^2}{MSE(G, R)}\right)$$

where G is the ground truth image and R - reconstruction.

4. Experiments and Results

In this section all the results is listed of the proposed experiments. We used the same techniques and training hyperparameters as (Yang et al., 2020).

All the models were trained in two stages: pretraining for two epochs with reconstruction (rec) loss only and subsequent fine-tuning with the reconstruction, perceptual (per), transferal perceptual (tper) and adversarial (adv) loss components for 50 epochs. In experiments with the dynamic convolutions we also used sparsity (sparse) loss with no constraining of pixels amount in masks on pretraining stage and constraining to 70% on fine-tuning. Loss coefficients are denoted in table 1. Other hyperparameters are depicted in table 2. If not explicitly stated, these values were used.

Table 1. Loss coefficients

REC	PER	TPER	ADV	SPARSE
1.0	0.01	0.01	0.001	0.1

Table 2. Default hyperparameters

Parameter	Value
Optimizer: generator and discriminator	Adam
Adam: Beta$_1$	0.9
Adam: Beta$_2$	0.999
Adam: Epsilon	0.8
Adam: Discriminator: Beta$_1$	0.0
Adam: Discriminator: Beta$_2$	0.9
Adam: Discriminator: Epsilon	0.8
LR: MainNet (RBs and CSFIs)	1e-4
LR: LTE (Learnable Texture Extractor)	1e-5
LR: Discriminator	1e-4

For training Nvidia V100 and Nvidia P100 on the Google Colab were used. Each training iteration requires approximately 10 hours on V100 and 20 hours on P100.

As an evaluation metric we use maximum PSNR that were achieved throughout the training on a ready validation subset of CUFED (tra) (tes).

We also provided indexes for experiments to make this report consistent with our results on Google Drive (our) and github (git).

To reproduce all the experiments follow the guide on the README.md page of our github repository (git). We used standard libraries like torch, torchvision, pytorch-lightning, kornia, numpy and cv2.

4.1. Dataset preparation

To be able to compare the training results it was decided to take the same dataset that was used in (Yang et al., 2020). It's CUFED (tra) (tes). Moreover to our knowledge there aren't many datasets that also provide Ref images. All the information about CUFED and the download instructions you may find on our github (git). It's a relatively small dataset (125 validation images) that contains various scenes including but not limited: humans, cars, landscapes etc. The main advantage of CUFED is additional reference images similar to high-resolution image scene. They are used as an additional input together with x4 downsampled (by bicubic interpolation) low-resolution image. References are applied as a "texture stock" which NN can use to generate super-

resolved images of higher quality and finer detail.

Train-validation split is present in the raw data, therefore we hadn't need to split the data additionally.

Before passing the data to the NN, we normalized it to the range $[-1, 1]$. As augmentation, we have used random rotations (0, 90, 180 and 270 degrees) and random left-right, up-down flips. Random augmentations were applied only to the train data, validation part was used as is.

4.2. Original results replication (experiment 2)

We use experiment N notation to relate explanations here with out github (git).

First of all, author's github (tts) was thoroughly studied. The model with default settings was trained according to the instructions in the README.md file. The result presented in the paper wasn't achieved, but the difference gap was negligible. PSNR: 25.44 (ours) vs 25.53 (paper). Thus it was concluded that the result is reproducible.

Yet the original pipeline lacks of an automatic plotting capability and not so extensible as needed for dynamic convolutions part for example. Thus it was decided to transfer the network to ours pipeline which was semi-developed at the beginning, but it was a great moment to finish it. Also capability to launch experiments with configuration files was added. It might be considered as a substitution for an experiment with bursts (because we haven't manage to find an appropriate dataset). We copied the baseline model from the author's repository (tts).

In order to confirm the equivalence between ours and original pipelines we trained the model on both pipelines. Metrics and visual quality turned out to be approximately the same. PSNR: 25.61 (ours) vs 25.44 (theirs).

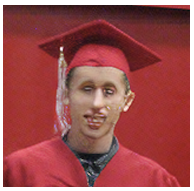

Figure 4. Example of artifacts on a face.

We have also implemented all the needed scripts to evaluate and predict the whole validation dataset to estimate visual quality with a single command. At this step we found out that quality on faces is a strong issue of this model as demonstrated on figure 4. Though Wasserstein GAN with gradient penalty (Gulrajani et al., 2017) was employed here, and it's a known effect of the presence of such artifacts when using GAN's.

4.3. Ablation study: lower number of scalings (experiment 3)

Two experiments were provided in that part. Firstly, we removed x2 scaling and adapted the code appropriately (experiment 3_1). We had also deleted some residual blocks that were in the corresponding part of the x1 scaling, because they basically extend the first portion of residual blocks that are applied only on the x1 scaling and their main purpose is for better mixing with the x2 scaling. Thus, the overall depth of the network was also slightly reduced. All the hard-attention and soft-attention mechanisms were completely removed from x2.

Figure 5. Left - original model; right - experiment 3_1.

In this first experiment we achieved PSNR = 25.54, which is on par with the original network, while the size was reduced by 2M of parameters. Original model has 7.3 million of parameters and this first version has only 5.3M. This is already a great result. Moreover visual quality didn't suffer drastically as shown on figure 5.

Figure 6. Left - experiment 3_1; right - experiment 3_2.

In the second experiment (3_2) we decided to remove CSFI blocks with x4 scaling, turn off fine-tuning of VGG19 texture extractors for x2 and x4 scalings, but keep the residual blocks on the x4 parallel, as it seems logical to preserve

Table 3. PSNR for experiment 4.

EXPERIMENT	PSNR
4.1	25.47
4.2	25.58

some transformations on the biggest spatial resolution to fix possible bugs after upsampling. All the hard-attention and soft-attention mechanisms were completely removed from x4. This resulted into the network with 4.7M parameters. We have achieved PSNR = 25.23. As shown on the figure 6 the detalization drop is clear.

Thus, we made a conclusion that x1 and x4 scalings are crucial for the details recovery while x2 (middle) scaling can be omitted. Indeed, it seems logical, because x1 scaling has the most expressive and abstract details (smallest spatial resolution), while x4 has the most little and fine details (biggest spatial resolution). On the other hand x2 is in between, so following this reasoning we believe that information from x2 scaling is redundant and don't provide the network additional information to significantly improve the output.

4.4. Ablation study: lower number of residual blocks (experiment 4)

As each training of a TTSR requires around 10 hours on nvidia V100 and 20 hours on nvidia P100 on the Google Colab, we were very accurate with the experiments to test here. In the original model $16 + 16 + 8 + 4$ number of blocks is used in four different parts of the network. First 16 residual blocks are in the backbone extractor that is used for feature extraction from LR. The rest three sets of residual blocks are associated with the blocks depicted on figure 2 in mix with CSFI blocks. It is reasonable to check the edge and the middle cases to get a feeling how the network behaves. Thus we'd chosen two sets of the residual blocks to test: $8 + 8 + 4 + 2$ (experiment 4_1) and $4 + 4 + 2 + 1$ (experiment 4_2).

In experiment 4_1 the network size turned out to be of 5.1M parameters. In 4_2 - 4.0M parameters. We got visually the same quality for both experiments comparable to the original model which is demonstrated on figure 7. The metrics are depicted on table 3.

Therefore we can make a conclusion that the model doesn't depend heavily on the number of the residual blocks and it's not the model's key factor that defines it's quality. This property can be very beneficial when deploying the network on the mobile devices, because not only the network size is reduced but also the depth, that makes it slightly faster on inference. Though we don't provide quantitative measurements on that matter.

Figure 7. Left - original model; middle - experiment 4_1; right - experiment 4_2.

4.5. Overall depth reduction

As two previous sections already imply depth reduction we conclude that the model is stable and doesn't drop the performance significantly while reducing the depth by the factor up to 1.8.

4.6. Analysis of the soft attention mechanism (experiment 6)

One major drawback was noted in the original implementation of texture transfer. Authors construct patches from the VGG19 features of reference and the low-resolution images and calculate Pearson correlations between those patches to find the most relevant patches in the reference to the low-resolution image and use them in the subsequent parts of the NN.

Despite the fact that instead of original image patches, VGG19 features are used in this process, it seems that this approach doesn't consider the nature of textures. Textures (we try to pull off) are usually of high frequency. Therefore it is desired to consider more complex relations than linear to calculate proper relevance.

We propose to apply the kernel trick in calculation of the corresponding scalar products between patches of the low-resolution image and the reference image. Yet the soft-attention (basically contains Pearson correlations) and hard-attention (most relevant indices in the reference) matrices are calculated from the matrices of very high dimensionality (more than 10000 rows and columns in our case) which makes it extremely expensive to add an additional kernelization.

Nevertheless, we decided to substitute the soft-attention block with the learnable contrast-aware channel attention module (Hui et al., 2019) to introduce additional capac-

ity to hopefully model more complex relations than linear (experiment 6_1).

As a result we observed that training process was optimal till the 20th epoch and after that transferal perceptual component in the loss started to grow steadily and divergence occurred.

Best achieved PSNR was 25.62. But the visual quality turned out to be worser than the original (even if we take checkpoints there divergence didn't start).

Figure 8. Left - original model; right - experiment 6_1

Therefore, we made a conclusion that this naive expansion of a capacity in the attention mechanism confuses the model during training which potentially leads to instability. We observed that instability. Smarter methods to overcome this issue may be consider in the future research.

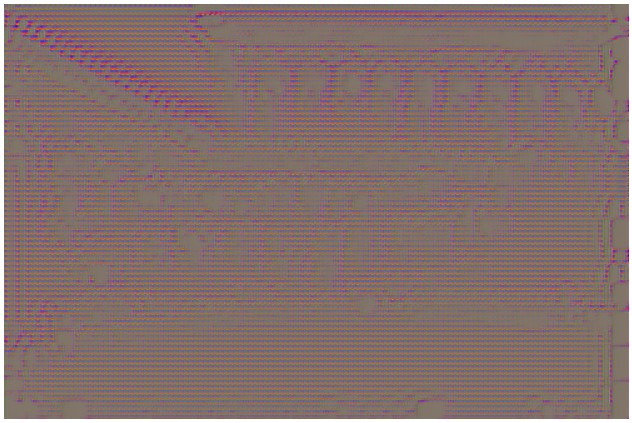

Figure 9. Example of the output after divergence in experiment 6_1

4.7. Enhancement: dynamic convolutions (experiment 8)

As we have already seen this network produces some visual artifacts that are especially noticeable on the objects with a complex patterns like a face. Therefore, we came up with an idea of an additional regularization in the convolutional layers that will help to reduce some artifacts. Specifically, dynamic convolutions come handy here. The intuition was provided in the Algorithms and Models section.

Figure 10. Left - original model ; right - with dynamic convolutions.

Figure 11. Left - original model ; right - with dynamic convolutions.

We achieved PSNR = 25.27. This is not the best result of this metric, but that is an expected effect, because PSNR is not the exact measurement of a human perception and sometimes tends to decrease even while the reconstruction becomes better to a human eye. Image quality became much better that is demonstrated on figures 10 11 12.

Yet this solution is not ideal. On most of the pictures from CUFED details are sharper and faces are better structured, but the dynamic convolution is not the "silver bullet". In too complex cases faces are still contain unpleasing artifacts. Complete elimination of such artifacts is yet to be discovered.

In the conclusion, the positive effect of the extension with the dynamic convolutions is obvious. The initial intuition

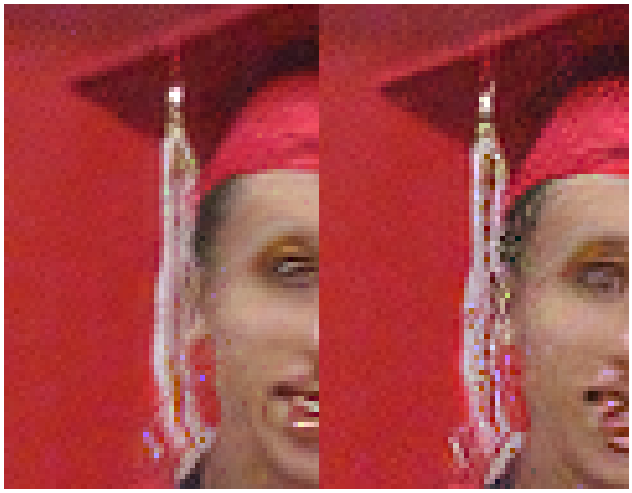

Figure 12. Left - original model ; right - with dynamic convolutions.

behind the usage of the dynamic convolutions - regularization of features learnt by the residual blocks is empirically confirmed.

4.8. Denoising + super-resolution (experiment 9)

It was interesting to check the stability of the TTSR in the complex environments. Specifically, we considered the environment there severe noise is present.

For that purpose we introduced random gaussian noise with the mean 0 and the sigma 0.25 in the training dataset and fixed noise with the same parameters in the validation dataset.

Figure 13. Left - noisy LR ; right - denoised SR.

As expected PSNR was decreased to 23.27. We have observed that with this noise model, the model copes perfectly. Almost complete noise elimination with a fair detail degradation as shown on figure 13.

Thus we make a conclusion that TTSR is robust and can be used in severe environments. For example, there light deficiency is present.

4.9. Trying different reference images (experiment 10)

Figure 14. From left to right: reference image - LR input - SR - Target

In all previous experiments a similar to LR image was used as a reference. In this experiment we used a totally different reference image to check the effect reference input has on the outcome of TTSR. We used a black image as reference to a LR input image. As we can see on figure 14 , the quality of the SR outcome didn't drop and is similar to the target.

5. Weak spots in TTSR

Generally, TTSR network performed well with regards to metrics. However, from some experiments conducted on the network we were able to find the following limitations and drawbacks:

- In our experiments we found that TTSR suffers from color bleeding when generating the SR image (some pixels are inconsistent to the neighborhood)
- From experiment 6, we found that attention schemes used in TTSR use correlation which may hinder extracting of non-linear textures.
- We have also observed that Ref doesn't play very important role and even using black image leads to approximately the same visual quality which is consistent with the experiment 6.

6. Applications

High-resolution imaging has recently attracted much attention in the field of medical imaging, and it is expected to lead to more accurate diagnosis, the advantages include obtaining a higher quality image from one that didn't exist or was lost, which could be useful in a variety of situations or even save lives in medical applications.

It takes time, money, and discomfort for the patient to obtain high-resolution (HR) clinically useful MR images. One way

to deal with this problem is to shorten the time it takes to acquire each image. These LR images should be post-processed to produce super resolved (SR) images with the same perceptual quality as the original.

Another application is the compression of computer network transfers. As a result, storage space and traffic are reduced.

Also in satellite images, which are used in various governmental applications, such as urbanization and monitoring the environment, where increasing the resolution of the image is an important preprocessing step that can improve the performance of various image processing tasks.

Nvidia's Deep Learning Super Sampling (DLSS) is an image upscaling technology that upscales images with the same quality as rendering the image natively in higher resolution but with less computation, making it faster and cheaper. For all of these tasks, our work represents a significant advancement.

7. Conclusion

In this project we have thoroughly studied the TTSR network, conducted multiple experiments and achieved several advancements. First of all, we have implemented our own training pipeline which supports configuration files for easier experimenting setup and plotting capability (with tensorboard). Secondly, we were able to reduce the complexity of the model without damaging quality of results in different ways (removing scalings and reducing the number of residual blocks). Thirdly, we enhanced quality of the model with the dynamic convolutions while reducing artifacts on complex patterns like faces. Fourthly, we found out that TTSR performs well even in conditions with severe noise.

In the conclusion, we conducted rigorous analysis and came up with critical weak points in this model. Another possible direction of research would be to eliminate the discussed disadvantages of soft-attention mechanism. This could help to improve the network and make it able to transfer more complex textures from the references.

References

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks, 2018.

Fuzhi Yang, Huan Yang, Jianlong Fu, Hongtao Lu, and Baining Guo. Learning texture transformer network for image super-resolution, 2020.

Thomas Verelst and Tinne Tuytelaars. Dynamic convolutions: Exploiting spatial sparsity for faster inference, 2020.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

Train cufed. URL https://drive.google.com/drive/folders/1hGHy36XcmSZ1LtARWmGL5OK1IUdWJi3I.

Test cufed. URL https://drive.google.com/file/d/1Fa1mopExA9YGG1RxrCZZn7QFTYXLx6ph/view.

Our results. URL https://drive.google.com/drive/folders/1m3XDS7lHJmz9uIY0TgAD1V-tX-oHVeyR?usp=sharing.

github. URL https://github.com/theleokul/train-suite.

Ttsr github. URL https://github.com/researchmm/TTSR.

Zheng Hui, Xinbo Gao, Yunchu Yang, and Xiumei Wang. Lightweight image super-resolution with information multi-distillation network. *Proceedings of the 27th ACM International Conference on Multimedia*, Oct 2019. doi: 10.1145/3343031.3351084. URL http://dx.doi.org/10.1145/3343031.3351084.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Ghaith Mqawass (30% of work)

- Reviewing literate on the topic (4 papers)
- Ablation study: Reducing size of residual blocks
- Experiment(10): trying different reference images
- Finding weak spots in TTSR model
- Preparing the Sections 1, 2, 4.9, 5, 7 in the report
- Preparing presentation

Leonid Kulikov (60% of work)

- Dataset preparation
- Replicating paper results and coding our own training pipeline
- Training of all the models
- Ablation study: reduced number of scalings
- Enhancement: analyzed attention mechanism and tried the Contrast-Aware attention block
- Enhancement: implemented dynamic convolutions
- Tested TTSR with noisy input
- Prepared Github repository
- Described the algorithms and experiments (except 4.9) in the report

Razan Dibo (10% of work)

- Reviewing literate on the topic
- Prepared the applications overview
- Prepared the script for the video recording
- Preparing presentation

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: None

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Separation is provided in advance by the authors of TTSR.

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: Each experiment was trained once for 52 epochs in total (2 pretraining epochs and 50 full).

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    ☐ Yes.
    ☐ No.
    ☑ Not applicable.

Students' comment: None

13. A description of the computing infrastructure used is included in the report.

- [x] Yes.
- [ ] No.
- [ ] Not applicable.

Students' comment: None