An Improved Texture Transformer Network for Ref-based Image Super-Resolution

09118228 Feihong Shen, 09118229 Boyu Zhang, 09118230 Hanyang Lu

Abstract

Image super-resolution, which aims to recover high resolution (HR) image from original low resolution (LR) image , has been widely used in many areas. SOTA methods like SRNTT and TTSR utilize an extra high resolution reference image (Ref) to help super-resolve the LR image that shares similar features. However, their work still have some liminations, HR image is hard to recovered when large scale feature matches between LR image and Ref image. To address these problems, we modify the structure of TTSR and proposed a new Auto-Adaptive-Algorithm(TTSRA). Results show that our method achieves quite improvement when large scale feature matches between LR and Ref image.

1 Introduction

Single image super resolution (SISR) has encountered a bottleneck due to the loss of information in low resolution image, reference based super resolution (RefSR) has been proved to have the potential to restore high-resolution details when giving high resolution reference images with similar content to LR inputs. However, the quality of restored HR image is affected by the similarity of LR and HR-Ref, when Ref image is less similar, the quality of restored HR image decreases significantly, so it is necessary to find an algorithm to calculate similarity of LR and Ref images. Existing method has made some progress in finding different scale features similarity, but some large scale feature similarities is hard to be notified. In this report we proposed a new Auto-Adaptive-Algorithm module to find large scale feature similarity and transfer features between LR and Ref more efficient, Besides, we modified the texture feature extracter module and loss functions in TTSR to improve the training speed of our model.

2 Previous Work

In this section we take a brief review of previous work on reference based image super resolution which are the most relevant to our work.

2.1 CrossNet

Classic RefSR approaches like Relevance Embedding and Patch Matching usually adopt a straight-forward method to transfer texture between which may result unsatisfied images, Based on existing RefSR methods, Zheng et al. proposed CrossNet which firstly adopted a cross-scale-feature-transfer module to transfer features between different scale LR and RefHR images.

2.2 SRNTT

Zhang et al. modified the cross-scale-feature-transfer in their SRNTT method. SRNTT adopts the ideas of local texture matching which could handle long distance dependency. It also "fuse" Ref texture to the final output, but we conduct it in the multi-scale feature space through a DNN model,

which enables the learning of complicated transfer process from references with scaling, rotation, or even non-rigid deformations.

2.3 TTSR

TTSR is the SOTA method on RefSR, it proposed a Learnable-Texture-Extraction module to extract different scale features from LR and HR-Ref images. TTSR consists of three different scales of Texture Transformer. The input of Texture Transformer is a low resolution image to be inpainted and a high resolution image as a reference. With a series of up-sampling and down-sampling, we can get their feature matrix in a same scale, further we can find the most similiar texture in the ref image and find the probality matrix, which represents the similarity between the texture in the ref image and the low resolution image. After symphasising the features and adding them back to the low resolution image, the original image can be amplified to high resoltion.

3 Methodology

3.1 Problems with TTSR

TTSR is a effective texture recovery model and performs better in many data sets than all previous RefSR models. But there are still many defects with TTSR. First of all, TTSR's adversarial loss is not easy to control. After many experiments, we found that the WGAN-GP loss curve always presents a trend of fluctuation rather than decline. The fluctuate loss curve is shown in Figure 1.

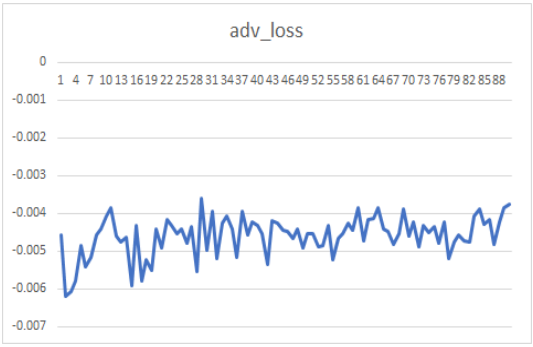


Figure 1: TTSR has fluctuate adversarial loss curve

On the other hand, TTSR only focuses on how to extract texture features without considering the size and continuity of the texture. But in real life, the textures we see are not necessarily the same size. That also means we actually pay more attention to those large pieces of texture that represent the areas that we most want to restore. Let's take a sea landscape as an example. In Figure 2, we can see that the TTSR performs a preliminary restoration of the LR. For the ship and the sea, we can see that the ship in the TTSR image is similar to the ship in the HR image, but the sea surface texture recovery is not satisfactory.

3.2 Our Methods

3.2.1 A New Adversarial Loss

As the loss curve of TTSR is not easy to control, we decide to try to replace the loss of adversarialism. In the original paper, WGAN-GP loss considers both discriminator and generator to ensure that gradient explosion will not occur.

However, in the TTSR model, the curve fluctuation of WGAN-GP Loss is serious. Therefore, we try to use ordinary GAN Loss to carry out the experiment. Finally, the experiment results show that such substitution is helpful to accelerate the convergence speed.

(a) LR (b) TTSR

(c) HR

Figure 2: TTSR does not focus on the size of textures

3.2.2 A New Learnable Texture Extractor

Learnable texture extractor determines the efficiency of texture extraction and controls the output of the entire TTSR. So having a good Learnable texture extractor is critical.

The Learnable texture extractor in TTSR uses VGG19 to construct feature extraction layers. Its structure is shown in Table 1.

Table 1: Feature extraction layers in VGG19

ID	Layer Name
0	Conv(3,64), ReLU
1	Conv(64,64), ReLU
2	Pool(2×2)
3	Conv(64,128), ReLU
4	Conv(128,128), ReLU
5	Pool(2×2)
6	Conv(128, 256), ReLU

After consulting some papers and blogs, we found several models are more efficient than VGG19 in extracting features: Inception v3, VGG19 BN, ResNet152 and DenseNet161. We tried to migrate all these models, but in the end only VGG19 BN and ResNet152 could perfectly integrate with TTSR. Considering the time cost of training, we finally chose to use VGG19 BN as the new learnable texture extractor. And the new learnable texture extractor's structure is shown in Table 2.

3.2.3 A New Auto Adaptive Algorithm

In the last section of the report, we noted that TTSR only focuses on how to extract texture features without considering the size and continuity of the texture. In order to solve this problem, we propose a new adaptive algorithm. In essence, we enhance the S matrix in the soft attention mechanism adaptively, making it easier for large texture areas to be restored to the final result, and ensuring that this enhancement does not affect the overall image. The whole algorithm is shown in Algorithm 1.

Table 2: New learnable texture extractor

ID	Layer Name
0	Conv(3,64), ReLU
1	BatchNorm2d(64)
2	Conv(64,64), ReLU
3	BatchNorm2d(64)
4	Pool(2×2)
5	Conv(64,128), ReLU
6	BatchNorm2d(128)
7	Conv(128,128), ReLU
8	BatchNorm2d(128)
9	Pool(2×2)
10	Conv(128, 256), ReLU
11	BatchNorm2d(256)

Algorithm 1 Auto Adaptive Algorithm

Input: Matrix S
Output: Matrix S

$m = med(S)$
for s in S **do**
  if $s >= m \; and \; neighborhood(s) >= m$ **then**
    $s = \frac{1-m}{log(2-m)} * log(s - m + 1) + m$
  if $s > max(S)$ **then**
    $s = max(S)$
return S

Step1: We first get a threshold value that is selected from all elements of the S matrix. After many attempts we finally chose the median of the S matrix as the threshold.

Step2: According to the selected threshold, we divide the elements of S matrix into two parts that are greater than and less than the threshold.

Step3: In the element greater than the threshold, we walk through each element and check whether the current element's eight neighborhoods are all greater than the threshold. If they are, then we reinforce this element.

Step4: We check whether the current element is greater than the maximum value of S matrix.If it is, change the value of the current element to the maximum value.

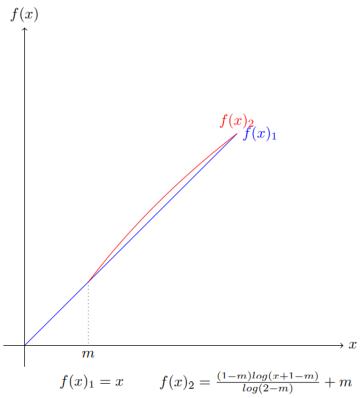


Figure 3: Mapping comparison

Intuitively, our algorithm improves the confidence of areas that are more likely to be large continuous textures, while also ensuring that the confidence is not too high to affect the overall image. As can be seen from Figure 3, when x is greater than the threshold, we increase the confidence of x. The curve is a nice improvement on the original linear mapping.

Figure 4 shows two S matrixs, both of which have dimensions of 40×40. When we take the median as the threshold, areas greater than the threshold are marked red. In this image, we can clearly see that the red areas are clustered regularly into large areas, which means that the model has successfully found a large continuous texture in the reference image. So when we increase confidence in these large areas, it means that we restore large areas of texture more efficiently.

However, a significant problem is that when we enhance the local texture, it will affect the overall restoration. This problem maybe serious in pictures of faces and minor in landscapes. Therefore, the final result needs to be determined according to the PSNR and SSIM of all test set.

(a) Matrix 1: m=0.59961 (b) Matrix 2: m=0.66561

Figure 4: S Matrix

4 Experiments

4.1 Task Assignment

Our project is finely assigned as shown in Table 3, and divided into the following several parts:

Data preprocessing: Collect dataset like CUFED5, Sun80, Urban100, select the HR and ref from the dataset. Load the data in the correct form and convert the HR sample into LR by downsampling.

Feature extractor network implementation: Write LTE to intercept the reel layer from VGG19 network and soft attention mechanism.

Main network implementation: Write CSFI and connect all network parts.

Loss functions implementation: Reconstruction loss, Adversarial loss, Perceptual loss.

Auto adaptive algorithm implementation: Enhancing S matrix in the soft attention mechanism adaptively.

Changing adversarial loss function: Changing adversarial loss to speed up convergence.

New learnable texture extractor implementation: Changing LTE structure to extract features more efficiently.

4.2 Implementation of TTSR

The implementation of TTSR can refer to an open source code. But because the library version is not compatible with the environment, we need to rewrite some critical parts.

First is the data processing. Due to the system problems, we changed the number of workers in reading. The default number of workers is required to keep the code running.

On the other hand, we wrote learnable texture extractor and packaged it into a library. Since the number of channels in part of VGG19 of the open source code is not compatible with the data set, we

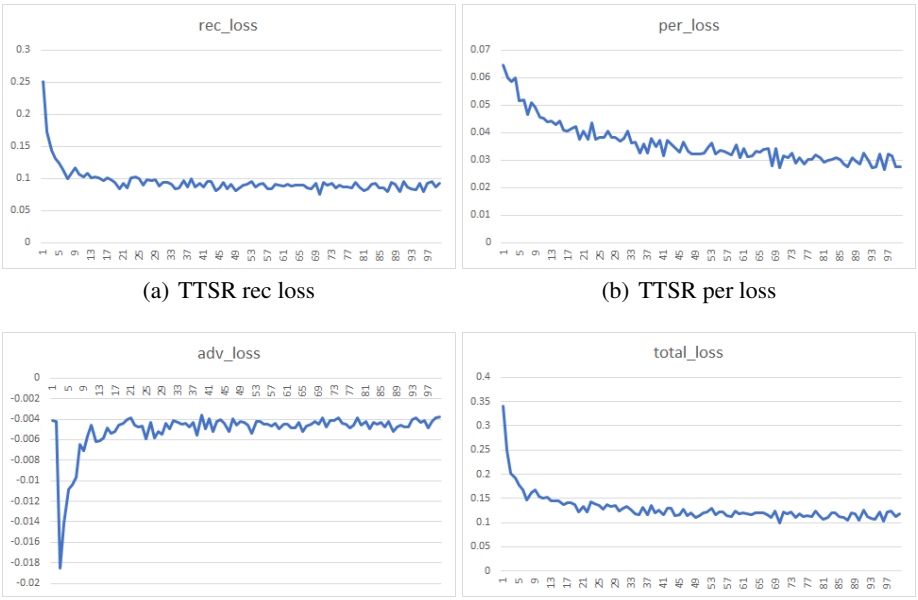
Table 3: Task Assignment

Name	Task	Percentage	
Boyu Zhang	Loss functions implementation	10%	35%
	Auto adaptive algorithm implementation	20%	
	Changing adversarial loss function	5%	
Feihong Shen	Feature extractor network implementation	15%	30%
	New learnable texture extractor implementation	15%	
Hanyang Lu	Data preprocessing	15%	30%
	Main network implementation	15%	
All	Deploy server, debug important parameters.	5%	5%

cut out the feature extraction layer of VGG19 by ourselves, with using 64,128,256 channels as output respectively.

To evaluate our implementation, we train and test our model on the recently proposed RefSR dataset, CUFED5. The training set in CUFED5 contains 11,871 pairs, each pair consisting of an input image and a reference image. There are 126 testing images in CUFED5 testing set, each accompanied by 4 reference images with different similarity levels.

As for tranning the code, the loss weight, learning rate and other super parameters are all consistent with the original paper. After more than 90 hours of training, we got a set of loss curves. The loss function includes reconstruction loss, adversarial loss and perceptual loss. Figure 5 shows the loss curves of the three types of losses and the total loss curves.



(a) TTSR rec loss (b) TTSR per loss

(c) TTSR adv loss (d) TTSR total loss

Figure 5: TTSR loss

In the total loss, reconstruction loss occupies a dominant position due to its large weight. So even if the adversarial loss is always fluctuating, it can not affect the convergence of the model. We can also see that all the three kinds of losses except adversarial loss show a downward trend, which is consistent with our previous analysis and the description in this paper. So preliminarily, our implementation of TTSR is successful.

Then we tested the actual performance of TTSR model trained by ourselves. We selected two pictures of Captain Jack, one for reference and the other for low resolution picture. The final result is shown in Figure 6. Captain Jack's body was mostly restored. Texture Details such as mast and sky are also well represented. The only drawback is that Captain Jack's face does not recover properly.

(a) LR (b) Ref (c) TTSR

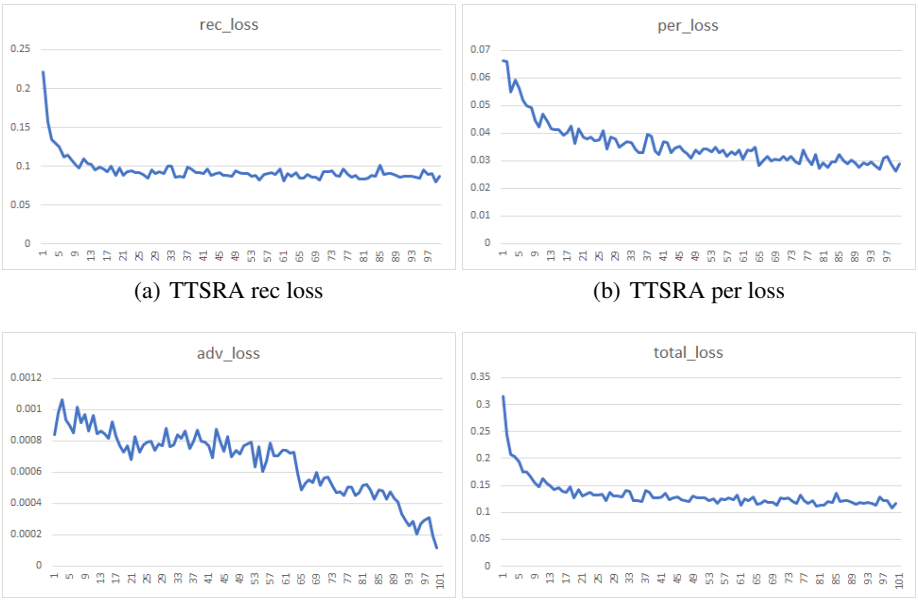
Figure 6: TTSR performance

4.3 Implementation of TTSRA

Since each change required a retraining of the model and the training time was more than 90 hours, we decided to deploy our program to a server for training. After trying with various platforms, we finally chose Kaggle. Training on Kaggle has proven to be more than eight times faster than personal GPU.

First, we took 800 pictures in CUFED5 data set as the demo for the experiment. We changed adversarial loss and learnable texture extractor and experimented with these new alters. Not surprisingly, we found that the convergence speed of the network was improved during the training.

Next, we trained all the data sets on the basis of the three innovations, and finally got new loss curve as shown in Figure 7.



(a) TTSRA rec loss (b) TTSRA per loss

(c) TTSRA adv loss (d) TTSRA total loss

Figure 7: TTSRA loss

The curve trend of adversarial loss is ideal very much, which is in sharp contrast to the previous fluctuation trend. At the same time, the total loss curve of the TTSRA also tends to be stable faster than the original one. The other two curves show little change while their overall trend is also

downward. Due to GPU and time constraints, we could only train 100 epochs. Ideally, more epochs should be trained to further study the stability of the training.

4.4 Evaluation

For the evaluation of the representation of TTSRA, we selected a landscape that appeared in the previous section. In this landscape, we have analyzed that TTSR model in original paper is not satisfactory for its texture restoration. So, this is an apt opportunity to inspect the new TTSR model's performance.

In Figure 8, we compare the TTSR with TTSRA in the same landscape. The results shown are satisfactory.

On the whole, there are dramatic differences between Figure 8(c) and Figure 8(d). Figure 8(d) depicts the texture more meticulously, and the edges and curves are more obvious, while the edges of Figure 8(c) seem to have some blurring that effect on the overall vision.

In terms of details, Figure 8(f) also has a finer texture filling than Figure 8(e). For this landscape, what we want to restore most is the sea and the sky, so the texture of the sea is extremely significant. In Figure 8(f), you can see that the waves on the sea surface take on a more detailed appearance, which is our ideal texture restoration.

(a) LR (b) HR

(c) TTSR (d) TTSRA

(e) TTSR (f) TTSRA

Figure 8: Texture recovery comparison

To evaluate the effectiveness of our method, we compare our model with other state-of-the-art RefSR methods. These methods include TTSR, CrossNet, SRNTT among which TTSR has achieved state-of-the-art performance on both PSNR and SSIM.

Table 4: PSNR/SSIM comparison

Model	**PSNR**	**SSIM**
CrossNet	25.48	0.764
SRNTT	25.61	0.764
TTSR	25.40	0.760
TTSRA	25.31	0.760

In Table 4, our TTSRA gets a pair of normal PSNR and SSMI. TTSRA and TTSR have similar performance. TTSRA not achieve the highest PSNR and SSMI may because there are a large number of images in the test set with a small part of the detailed texture. TTSRA has a good recovery ability for large continuous textures, but it does not handle the detailed texture properly, such as the faces.

5 Conclusion and Futhur Work

In this report we modify the TTSR structure and propose a new Auto-Adaptive-Algorithm to handle problems caused by existing Super-Resolution algorithms. Our method, find the similarity of Experiments demonstrate our method achieve quite improvement when large scale feature matches between Ref and LR images. We also modify the texture extractor module proprosed by TTSR to make our model easier to train, Their are still liminations in our new model, when paying to much attention on large scale feature, sometimes small scale feature is also affected, the model may recognize wrong texture similarities and transfer wrong textures in some area, which causes unsatisfied results. What's more, current Super-Resolution models always cause some artifacts like gap effects and irregular noises in some materials like face and complex patterns because wrong texture matches or no texture matches, existing methods like denoising are used to to ease these effects, but the result often make small textures disappear.

In the future, we will try to modify our model structure to find more accurate similarities between different scale features.