EE541 Final Project Report

Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network(SRGAN and EDSRGAN TWO METHODS)

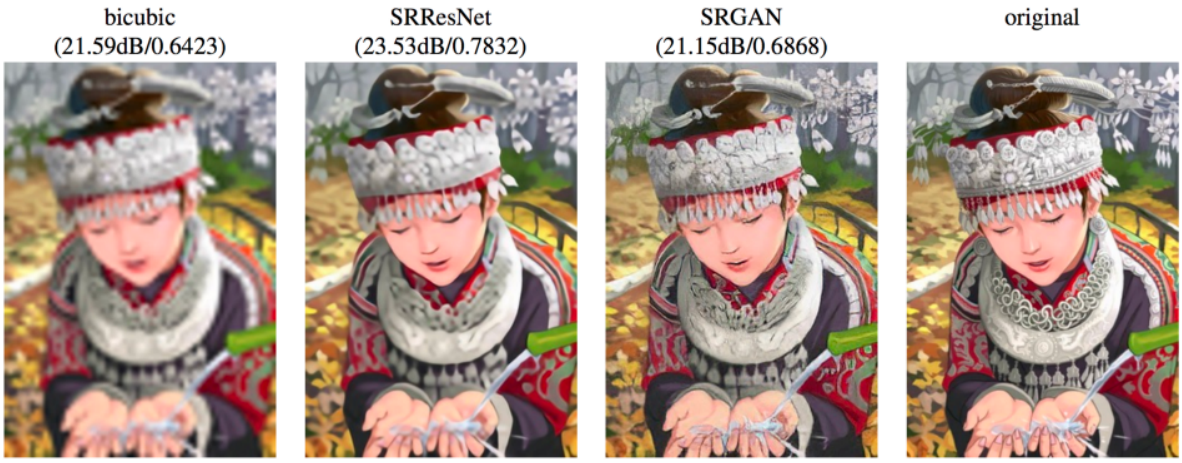


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Group Member: Jiayu Guo and Ziyang Zhou

05/07/2022

1. Abstract (SRGAN)

Super-resolution (Christian Ledig) is the task of constructing a higher-resolution image from a lower-resolution image. While this task has traditionally be

en approached with non-neural methods such as bilinear and bicubic upsampling, neural networks offer an opportunity for significant improvements. We implement a Super-Resolution Generative Adversarial Network (SRGAN), we infer photo-realistic natural images for 4× upscaling factors. To achieve this, we propose a perceptual loss function which consists of an adversarial loss and a content loss. The adversarial loss pushes our solution to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images. In addition, we use a content loss motivated by perceptual similarity instead of similarity in pixel space. Our deep residual network is able to recover photo-realistic textures from heavily downsampled images on public benchmarks. An extensive mean-opinion-score (MOS) test shows hugely significant gains in perceptual quality using SRGAN. The MOS scores obtained with SRGAN are closer to those of the original high-resolution images than to those obtained with any state-of-the-art method.

2. Introduction (SRGAN)

The highly challenging task of estimating a high-resolution (HR) image from its low-resolution (LR) counterpart is referred to as super-resolution (SR). SR received substantial attention from within the computer vision research community and has a wide range of applications. In this work, we propose a super-resolution generative adversarial network (SRGAN) for which we employ a deep residual network (ResNet) with skip-connection and diverge from MSE as the sole optimization target. Different from previous works, we define a novel perceptual loss using high-level feature maps of the VGG network combined with a discriminator that encourages solutions perceptually hard to distinguish from the HR reference images. An example photo-realistic image that was super-resolved with a 4× upscaling factor is shown in Figure 1.

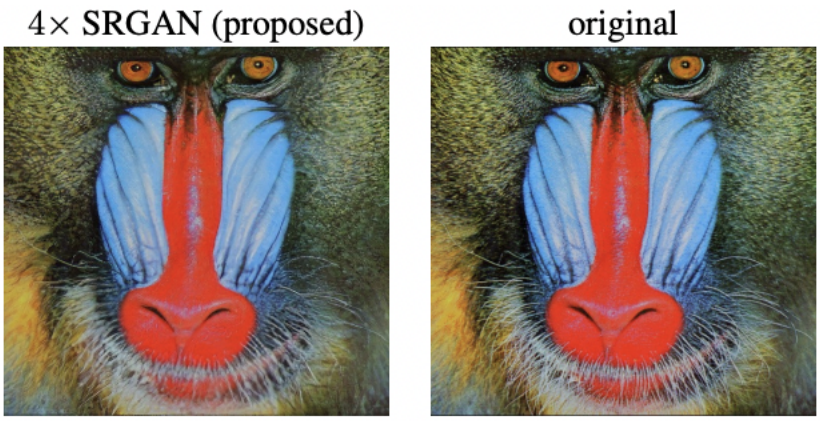


Figure 1: Super-resolved image (left) is almost indistinguishable from original (right). [4× upscaling]

3. Related Work(SRGAN)

1. Generative Adversarial Network

Generative Adversarial Networks (GANs) are a class of neural networks th at are used to generate images that are indistinguishable from the origi nal ones. They were first coined by I. Goodfellow in the paper titled ” Generative Adversarial Networks”

Generative Adversarial Network has two components:

- **Generator Network:** Generates fake images based on a noise vector.
- **Discriminator Network:** Distinguishes between ”real” from the da taset and ”fake” images generated by the generator.

Generative model G captures the data distribution, and a discriminative model D estimates the probability that a sample came from the training data rather t han G. The two models are trained simultaneously. G generators examples, along with the real examples, are given to D to be classified as real or fake. D is then updated to discriminate fake or real examples better in the next round an d the training procedure for G is to maximize the probability of D making a mi stake, as figure2 shows.

$$\max_D \max_G = \mathrm{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathrm{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

In this way, these two models, G, and D are like two players competing against each other. D and G play the following two-player minimax game with value function V(G, D). Usually, we will use convolution neural networks (CNNs) to train the generator and discriminator models. The key to GANs' success is the idea of an adversarial loss that forces the generated images to be indistinguishable from real images. This loss is particularly powerful for image generation tasks

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathrm{E}_{y\sim p_{data}(y)}[logD_Y(y)] + \mathrm{E}_{x\sim p_{data}(x)}[log(1 - D_Y(G(x)))] \quad (2)$$

The general idea behind this formulation is that it allows one to train a generative model G with the goal of fooling a differentiable discriminator D that is trained to distinguish super-resolved images from real images. With this approach, our generator can learn to create solutions that are highly similar to real images and thus difficult to classify by D. This encourages perceptually superior solutions residing in the subspace, the manifold, of natural images. This is in contrast to SR solutions obtained by minimizing pixel-wise error measurements, such as the MSE.

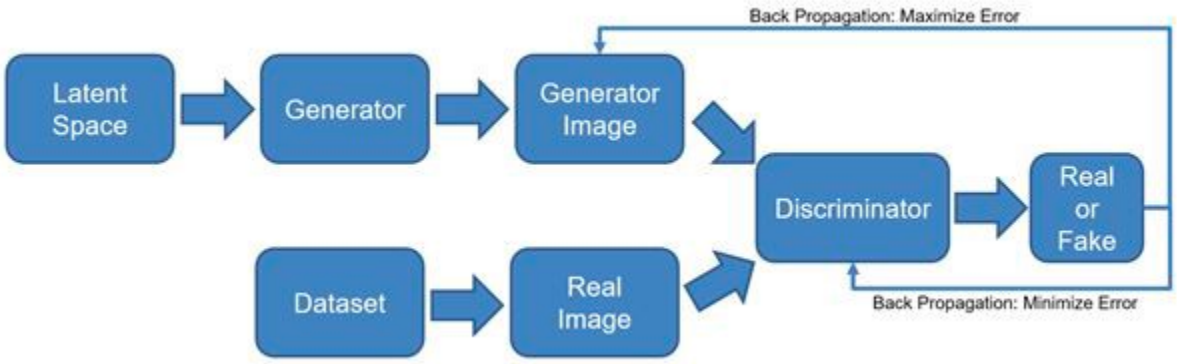


Figure 2: Example of GAN architecture from https://medium.com/@Packt_Pub/inside-the-generative-adversarial-networks-gan-architecture-2435afbd6b3b

4. SRGANs Architecture

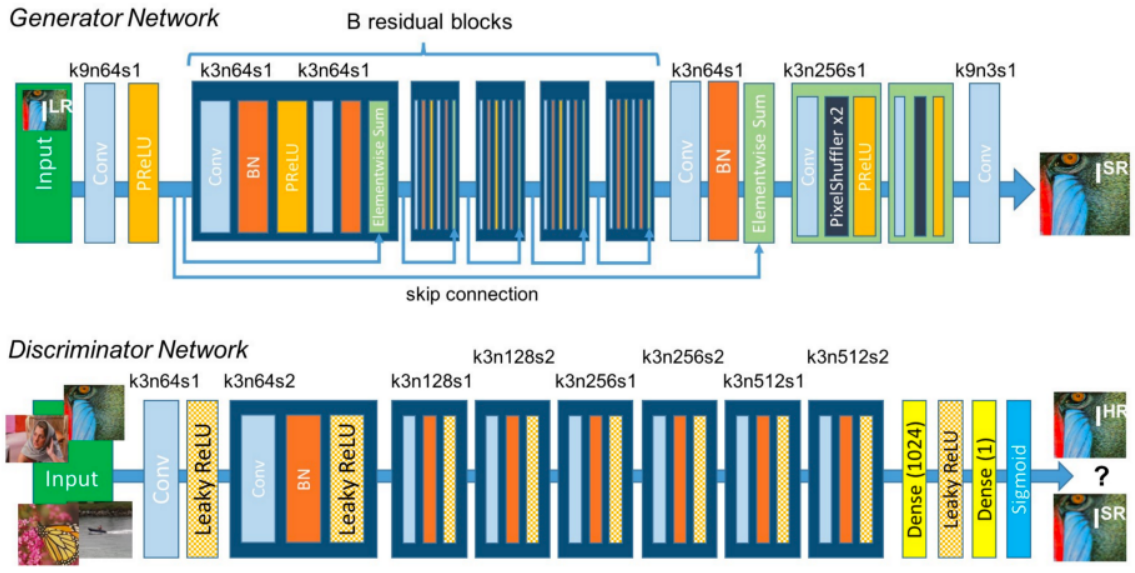


Figure 4: Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

- We set a new state of the art for image SR with high upscaling factors (4×) as measured by PSNR and structural similarity (SSIM) with our 16 blocks deep ResNet optimized for MSE.
- We propose SRGAN which is a GAN-based network optimized for a new perceptual loss. Here we replace the MSE-based content loss with a loss calculated on feature maps of the VGG network, which are more invariant to changes in pixel space.
- We confirm with an extensive mean opinion score (MOS) test on images from three public benchmark datasets that SRGAN is the new state of the art, by a large margin, for the estimation of photo-realistic SR images with high upscaling factors (4×).

SRGAN further improves the results of SRResNet by fine-tuning its weights so that it can generate high-frequency details in the generated image. This is done by training the model in a GAN using the *Perceptual* loss function.

4. 1 . Perceptual loss function

- The definition of our perceptual loss function l SR is criti cal for the performance of our generator network. We formula te the perceptual loss as the weighted sum of a content loss (l SR X) and an adversarial loss component as

$$\ell^{SR} = \underbrace{\underbrace{\ell_X^{SR}}_{\text{Content loss}} + \underbrace{10^{-3}\ell_{Gen}^{SR}}_{\text{Adversarial loss}}}_{\text{Perceptual loss (for VGG based content losses)}}$$

4. 2 Content loss function

- Content Loss: compares deep features extracted from SR and HR imag es with a pre-trained VGG network. With Φi,j we indicate the feat ure map obtained by the j-th convolution (after activation) before the i-th max-pooling layer within the VGG19 network

$$\ell_{VGG/i,j}^{SR} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{x=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

Here Wi,j, and Hi,j describe the dimensions of the respective feat ure maps within the VGG network.

4. 3 Adversarial Loss

- Adversarial Loss: The GAN discriminator D is optimized for discrim inating SR from HR images whereas the generator is optimized for g enerating more realistic SR images in order to fool the discrimina tor.

$$\ell_{Gen}^{SR} = \sum_{n=1}^{N} -log D_{\theta_D}(G_{\theta_G}((I^{LR}))$$

5. Training SRGANs

In this project, I have trained the model on the DIV2K dataset which contains high-quality (2K resolution) images and a corresponding downgraded images data set for image restoration tasks. Citation: (Persson; Persson; Persson; Persson; Persson; Persson; Persson; Persson; Different model and loss function Compared)

DIV2K dataset has many downgrading factors which are stated below: bicubic_x2, bicubic_x3, bicubic_x4, bicubic_x8, unknown_x2, unknown_x3, unknown_x4, realistic_mild_x4, realistic_difficult_x4, realistic_wild_x4, I have used bicubic_x4 for model training.

The DIV2K dataset is divided into:

Train data: First 800 high definition images and corresponding low-resolution images with a particular downgrading factor.

Validation data: 100 high definition images with corresponding low-resolution images as validation data

We use Data Augmentation, which is a technique to get different variety of the same image like flipped images, rotated images, cropped images, etc.

It is a way of getting a different variety of data especially when training data is very less in number as we have DIV2k data which has only 800 training images.

There are only 800 images for training which is very few to train the model so we will do data augmentation on train data to get more kinds of different varieties of images.

Augmentation is very important to get more generalized results from the trained model. Because by training on augmented data model can learn to vide variety of features so augmentation gives more generalized result.

Random Crop: We will crop a random part of an image with size 96 X 96 from High-Resolution Image and the corresponding patch from Low-resolution Image.

If the scaling factor is 4 and if we crop 96 X 96 patch from HR image then Corresponding patch size from Low-Resolution image would be 24 X 24 (96/4 = 24.

tf.random.normal is less than 0.5 then we do left_right flip, otherwise we won't flip images.

Random Rotate: In this operation, we will rotate LR and HR images by 90 degrees multiple times. We will generate one random number using

Now that we have successfully completed the construction of the SRGAN architecture, we can proceed to train the model. Store the generator model and the discriminator model in their respective models. Define the VGG model for the interpretation of the perpetual loss that we will use for this model. Create checkpoints and define both the optimizers for the generator and discriminator networks. And the following are each single step contain how many parameters for g

enerator and discriminator.

Train data steps:

1. Random crop
2. Random Rotate
3. Random Flip
4. Assign batch size (16 for train data in our case)
5. Repeat (infinite for train data)
6. Prefetch (for a faster training process)

```
Model: "Generator"
_________________________________________________________________
 Layer (type)                                Output Shape
=================================================================
 input_1 (InputLayer)                        [(None, None, None, 3)]

 lambda (Lambda)                             (None, None, None, 3)

 conv2d_block (Conv2DBlock)                  (None, None, None, 64)

 conv2d_block_1 (Conv2DBlock)                (None, None, None, 64)

 rrd_block (RRDBlock)                        (None, None, None, 64)

 rrd_block_1 (RRDBlock)                      (None, None, None, 64)

 rrd_block_2 (RRDBlock)                      (None, None, None, 64)

 rrd_block_3 (RRDBlock)                      (None, None, None, 64)

 pixel_shuffle_up_sampling (PixelShuffleUpSa  (None, None, None, 64)
 mpling)

 pixel_shuffle_up_sampling_1 (PixelShuffleUp  (None, None, None, 64)
 Sampling)

 conv2d_block_40 (Conv2DBlock)               (None, None, None, 64)

 conv2d_block_41 (Conv2DBlock)               (None, None, None, 3)
```



```
 activation (Activation)                     (None, None, None, 3)

 lambda_3 (Lambda)                           (None, None, None, 3)

=================================================================
Total params: 1,344,643
Trainable params: 1,341,571
Non-trainable params: 3,072
```

```
discriminator = Discriminator()
discriminator.summary(100)
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                                Output Shape
=================================================================
 input_2 (InputLayer)                        [(None, 128, 128, 3)]

 lambda_4 (Lambda)                           (None, 128, 128, 3)

 conv2d_42 (Conv2D)                          (None, 128, 128, 32)

 leaky_re_lu (LeakyReLU)                     (None, 128, 128, 32)

 conv2d_43 (Conv2D)                          (None, 64, 64, 32)

 batch_normalization_36 (BatchNormalization)  (None, 64, 64, 32)

 leaky_re_lu_1 (LeakyReLU)                   (None, 64, 64, 32)

 conv2d_44 (Conv2D)                          (None, 64, 64, 64)

 batch_normalization_37 (BatchNormalization)  (None, 64, 64, 64)

 leaky_re_lu_2 (LeakyReLU)                   (None, 64, 64, 64)

 conv2d_45 (Conv2D)                          (None, 32, 32, 64)

 batch_normalization_38 (BatchNormalization)  (None, 32, 32, 64)

 leaky_re_lu_3 (LeakyReLU)                   (None, 32, 32, 64)

 conv2d_46 (Conv2D)                          (None, 32, 32, 128)

 batch_normalization_39 (BatchNormalization)  (None, 32, 32, 128)

 leaky_re_lu_4 (LeakyReLU)                   (None, 32, 32, 128)

 conv2d_47 (Conv2D)                          (None, 16, 16, 128)

 batch_normalization_40 (BatchNormalization)  (None, 16, 16, 128)

 leaky_re_lu_5 (LeakyReLU)                   (None, 16, 16, 128)

 conv2d_48 (Conv2D)                          (None, 16, 16, 256)

 batch_normalization_41 (BatchNormalization)  (None, 16, 16, 256)

 leaky_re_lu_6 (LeakyReLU)                   (None, 16, 16, 256)
```

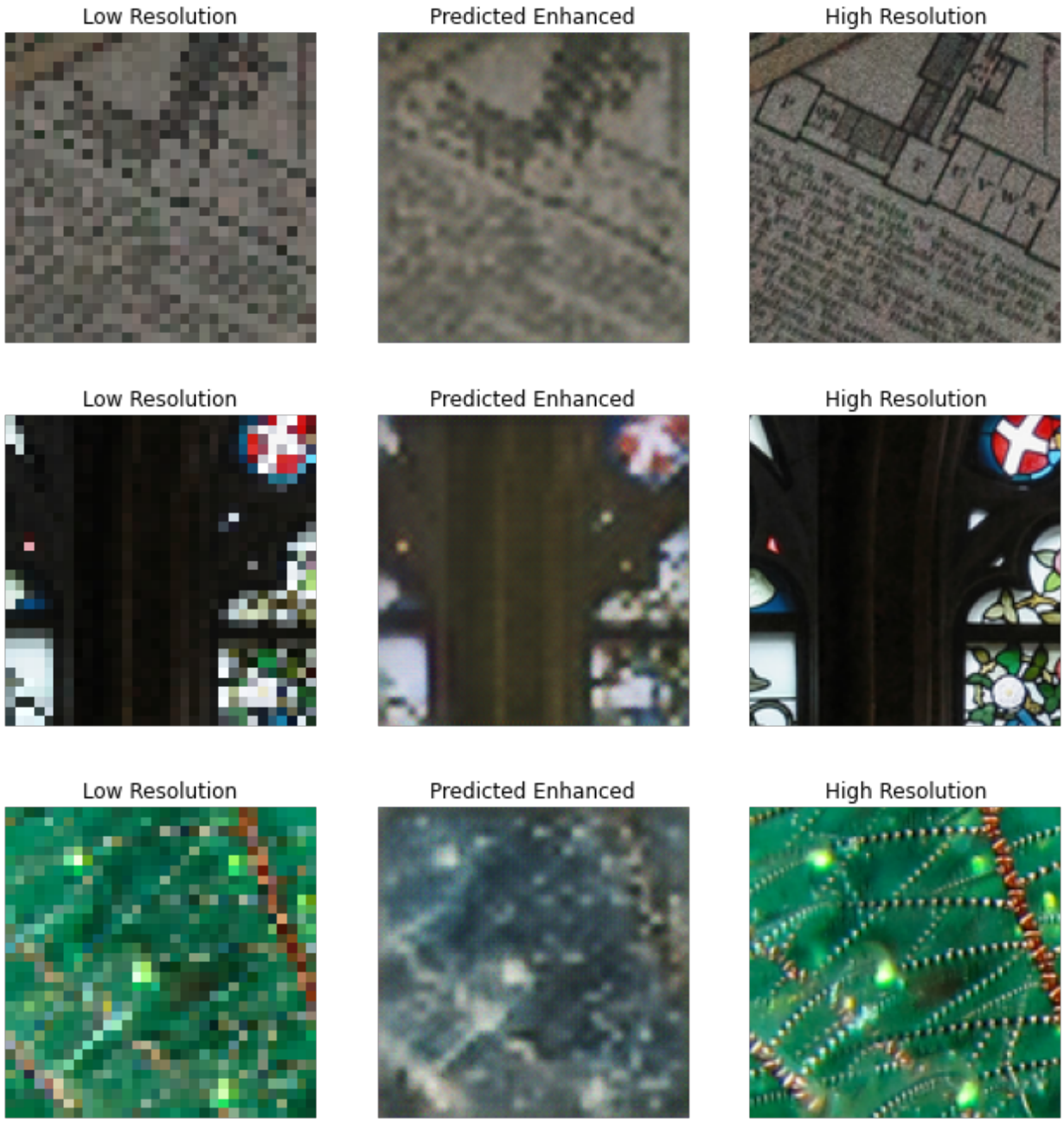


```
 conv2d_49 (Conv2D)                          (None, 8, 8, 256)

 batch_normalization_42 (BatchNormalization)  (None, 8, 8, 256)

 leaky_re_lu_7 (LeakyReLU)                   (None, 8, 8, 256)

 flatten (Flatten)                           (None, 16384)

 dense (Dense)                               (None, 1024)
```

5. Conclusion (SRGAN)

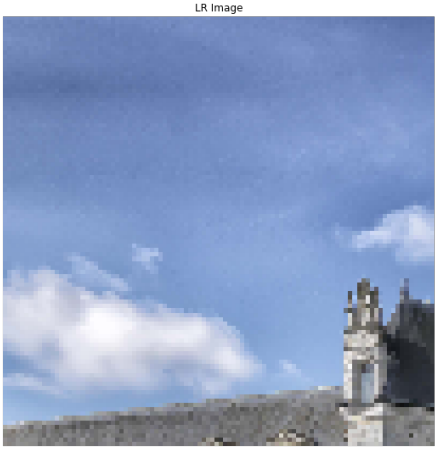
The following are the training result output:

From left to right are low-Resolution X4 and middle is our SRGAN predicted pic ture and the right side is High-Resolution pictures, the pictures are after pr eprocessing, use the cropped pictures.



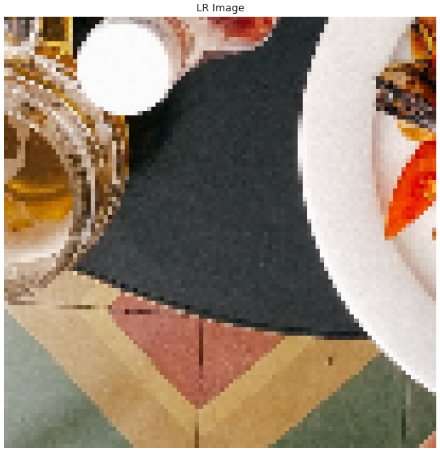
And the following are test data sets output: Left is Low-Resolution pictures a nd right is Super Resolution, obviously we can see the improvement of the pict ures.

LR Image

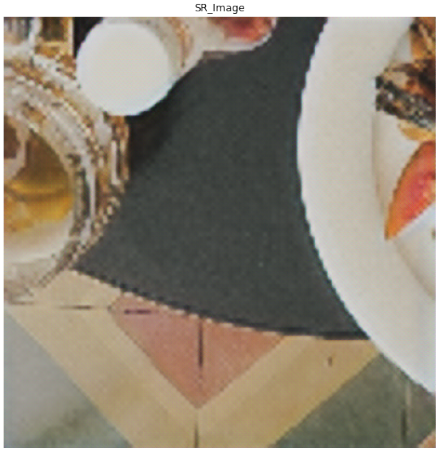


SR_Image

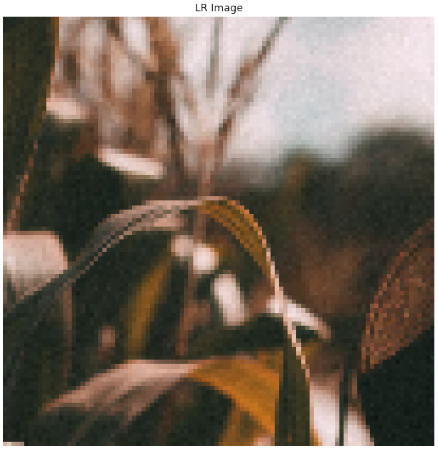
LR Image



SR_Image



LR Image



SR_Image

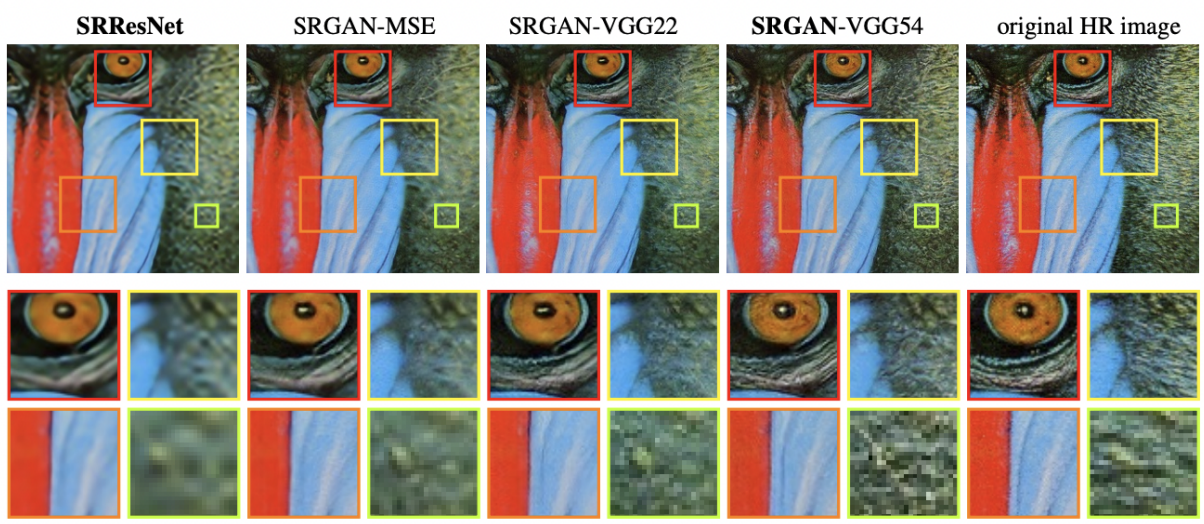


Figure 6: **SRResNet** (left: a,b), SRGAN-MSE (middle left: c,d), SRGAN-VGG2.2 (middle: e,f) and **SRGAN**-VGG54 (middle right: g,h) reconstruction results and corresponding reference HR image (right: i,j). [4× upscaling]

(“Different model and loss function Compared”)

We have described a deep residual network SRResNet that sets a new state of th e art on public benchmark datasets We have highlighted some limitations of thi s PSNR- focused image super-resolution and introduced SRGAN, which augments th e content loss function with an adversarial loss by training a GAN. We have co nfirmed that SRGAN reconstructions for large upscaling factors (4×) are, by a considerable margin, more photo-realistic than reconstructions obtained with s tate-ofthe-art reference methods.

Now it’s time to implement an EDSR research paper. Enhanced Deep Residual Networks for Single Image Super-Resolution

7. Abstract (EDSRGAN)

We develop an enhanced deep super-resolution network (EDSR) with performance e xceeding those of current state-of-the-art SR methods. The significant perform ance improvement of our model is due to optimization by removing unnecessary m odules in conventional residual networks. The performance is further improved by expanding the model size while we stabilize the training procedure. We also propose a new multi-scale deep super-resolution system (MDSR) and training met hod, which can reconstruct high-resolution images of different upscaling facto rs in a single model.

8. Introduction (EDSRGAN)

Image super-resolution (SR) problem, particularly single image super-resolution (SISR), has gained increasing research attention for decades. SISR aims to reconstruct a high-resolution image I^SR from a single low-resolution image I^LR. Generally, the relationship between I^LR and the original high-resolution image I^HR can vary depending on the situation. Many studies assume that I^LR is a bicubic downsampled version of I^HR, but other degrading factors such as blur, decimation, or noise can also be considered for practical applications.

We evaluate our models on the standard benchmark datasets and on a 2017 year provided DIV2K dataset. The proposed single- and multi-scale super-resolution networks show state-of-the-art performances on all datasets in terms of PSNR and SSIM.

9. EDSRGAN Architecture

The General Architecture of the proposed EDSR network is as follows.

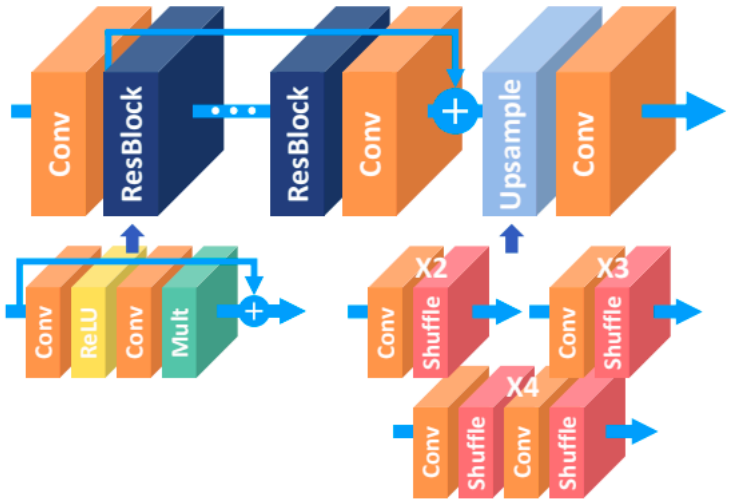
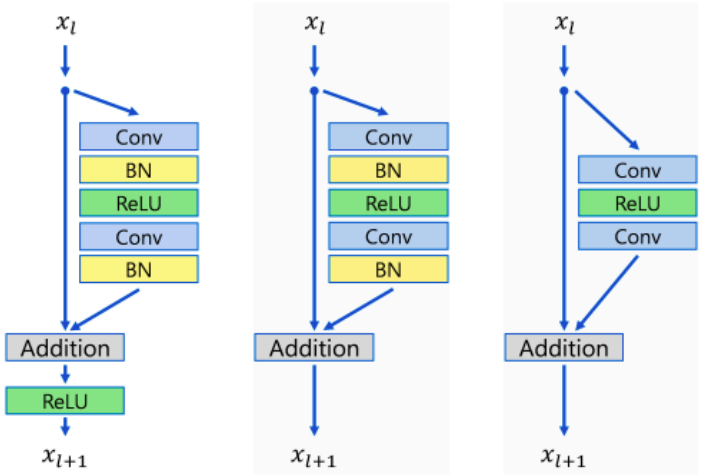


Figure 3: The architecture of the proposed single-scale SR network (EDSR).

In EDSR they proposed different architecture of ResBlock which more efficient to train the model.



(a) Original (b) SRResNet (c) Proposed

Figure 2: Comparison of residual blocks in original ResNet, SRResNet, and ours.

Proposed networks. We remove the batch normalization layers from our network a s Nah et al presented in their image deblurring work. Since batch normalizatio n layers normalize the features, they get rid of range flexibility from networ ks by normalizing the features, it is better to remove them.

Furthermore, GPU memory usage is also sufficiently reduced since the batch nor malization layers consume the same amount of memory as the preceding convoluti onal layers. this baseline model without a batch normalization layer saves app roximately 40% of memory usage during training, compared to SRResNet. Conseque ntly, we can build up a larger model that has better performance than the conv entional ResNet structure under limited computational resources.

Considered following is the complete Architecture of the EDSR model:

1. Normalize Input by subtracting DIV2K RGB mean
2. Conv2d layer with 64 filters and kernel size=3
3. Resblock (In our model 8 ResBlocks)
4. Conv2d
5. Add (ResBlock output and original Input)
6. Upsampling (Conv2d → Pixel Shuffle)

10. Training (EDSRGAN)

To train the EDSR model as per the research paper we have to train a model for 300000 steps and evaluate the model at every 1000 steps on the first 10 valida

tion set images, So to fulfill this requirement I have made a whole training pipeline class to train the model as per requirement and also we can restore the latest checkpoint anytime so that we can resume training anytime.

11. Conclusion (EDSRGAN)

1. DIV2k Validation set Results (x4 vs x4 sr)

In this paper, we proposed an enhanced super-resolution algorithm. By removing unnecessary modules from conventional ResNet architecture, we achieve improved

results while making our model compact. We also employ residual scaling techniques to stably train large models. Our proposed single-scale model surpasses current models and achieves state-of-the-art performance. Furthermore, we develop a multi-scale super-resolution network to reduce the model size and training time. With scale-dependent modules and a shared main network, our multi-scale model can effectively deal with various scales of super-resolution in a unified framework. While the multi-scale model remains compact compared with a set of single-scale models, it shows comparable performance to the single-scale SR model.

12. Scope of Improvement (Future work)

1. As of now, we have trained EDSR and SRGAN model on only the div2k dataset with bicubic_x4.
2. For further Improvement, we can train model wide variety of data with mixed downgrading factors so that our model can predict any real-world images.

Reference

1. https://arxiv.org/pdf/1707.02921.pdf
2. https://arxiv.org/pdf/1808.08718.pdf
3. https://github.com/krasserm/super-resolution
4. https://www.appliedaicourse.com/

Works Cited

1. Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, ´ Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, We nzhe Shi. *Photo-Realistic Single Image Super-Resolution Using a Generative Adv ersarial Network*. n.d. <https://arxiv.org/pdf/1609.04802.pdf>.

2."Different model and loss function Compared." 19 November 2019. 9 May 2022. <https: //arxiv.org/pdf/1609.04802.pdf>.

3. *https://www.youtube.com/watch?v=qwYO1XRdADI&list=PLhhyoLH6IjfwIp8bZnzX8QR30TRcHO8V a&index=14&ab_channel=AladdinPersson*. n.d.

4. Persson, Aladdin, SRGAN implementation from scratch