A Comparison of Image Super-Resolution Systems

Hualong Qiao
21111956
h3qiao@uwaterloo.ca

Abstract

This project develops an image super-resolution system to enhance low-quality images using deep learning algorithms. I implement the Super-Resolution Convolutional Neural Network (SRCNN) from scratch and integrate pre-trained models of Enhanced Deep Residual Networks (EDSR) and Enhanced Super-Resolution Generative Adversarial Network (ESRGAN). Using the DIV2K dataset augmented with high-resolution anime-style images, I evaluate and compare these models, focusing particularly on their performance with anime-style and cartoon images. I used Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) as key metrics for quantitative assessment, complemented by qualitative visual comparisons. The goal of this project is to compare the performance of various deep learning algorithms in image SR field, which demonstrated the potential of deep learning in image enhancement. I propose to extend the system to automatically select the optimal result based on quantitative metrics as a future work, which allows the system to have a more adaptive and efficient super-resolution application.

1 Problem Definition

Image super-resolution (SR) is a popular area in CV(computer vision) that aims to improve the resolution of low-quality images[1]. SR has wide range of applications, from photography to security cameras, satellite imaging to old document restoration. Traditional algorithms often lack comprehension of complex images or images from different contexts. Recent developments in super-resolution algorithms provide possibilities to resolve these issues, making it possible to achieve high-quality image enhancement[2].

This project focuses on implementing SRCNN and comparing the performance of these three SR models:

1. Super-Resolution Convolutional Neural Network (SRCNN)
2. Enhanced Deep Super-Resolution Network (EDSR)
3. Enhanced Super-Resolution Generative Adversarial Network (ESRGAN)

Each of these models represents a different approach to the SR problem, from the SRCNN that was easy to implement and allow us to train the model right away to the more advanced EDSR and GAN-based ESRGAN. In this project, I am going to compare the differences of their performance, particularly on anime-style and cartoon images which are used intensively for training the models. Anime-style and cartoon style images also present unique challenges due to their stylized nature[2, 3, 4, 5].

2 Dataset Construction

The dataset I chose to be used as training sets and validation sets are:

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

- **DIV2K dataset:** This dataset consists of 800 high resolution images for training, I used 700 for training and 100 images for validation.
- **Anime-style images from Kaggle:** Approximately 100,000 training images are used to capture the unique stylistic elements of anime. Since the dataset is too big, only used 20,000 of them for training and 1000 images for validation.
- **Cartoon images from Kaggle:** This includes around 800 Pokémon cartoon-style images, providing a diverse range of cartoon features, also 100 of them was used as validation set.

Our dataset construction process involves the following steps:

1. **High-resolution images:** I started with high resolution images of size 512x512 from our dataset.
2. **Down-scaling:** These images are down-scaled to 128x128 using bi-cubic interpolation to create low-resolution versions, this reduces the image 16 times than the original images.
3. **Training:** The model is trained using the low resolution (128x128) images, learn from them and then output images which will be compared with the original high resolution (512x512) images as targets.

This approach allows us to simulate real-world scenarios where I start with a low resolution image, and try to recover its high resolution versions. The use of different image styles ensures that our models can perform well across different domains.

3 Algorithm Design

For this task, I've implemented Super-Resolution Convolutional Neural Network (SRCNN) using three layers neural networks. Also running two other algorithms: Enhanced Deep Super-Resolution Network (EDSR), and Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) with pre-trained model. In terms of complexity, ESRGAN is among all the most complex one and will have the best performance among all three algorithms if trained with the same data.

3.1 SRCNN

For this project, I implemented the Super-Resolution Convolutional Neural Network (SRCNN) introduced by Dong et al. [2]. SRCNN was one of the first CNN-based approaches to image super-resolution and consists of three convolutional layers:

1. Patch extraction: Dividing original images into small pieces and extract patches.
2. Non-linear mapping: Each of the small patches are passed through transformation aimed to improve their resolution.
3. Reconstruction: Combines the high-resolution transformed images into the final image.

The SRCNN architecture can be represented as:

$$F(Y) = W_3 * \sigma(W_2 * \sigma(W_1 * Y + B_1) + B_2) + B_3 \tag{1}$$

where W_i and B_i are weights and biases, σ is the activation function, and $*$ denotes convolution.

My implementation follows this structure:

Algorithm 1 SRCNN Implementation (Initialization)

```
model ← train_srcnn()
test_data ← load_test_data()
psnr, ssim ← evaluate_srcnn(model, test_data)
Print PSNR and SSIM scores
```

Algorithm 2 SRCNN Implementation (SRCNN Part 2)

```
function SRCNN_MODEL
    Define Sequential model
    Add Conv2D layer (64 filters, 9x9 kernel, ReLU, same padding)
    Add Conv2D layer (32 filters, 1x1 kernel, ReLU, same padding)
    Add Conv2D layer (1 filter, 5x5 kernel, same padding)
    return model
end function

function TRAIN_SRCNN
    Load and preprocess training data
    model ← srcnn_model()
    Compile model with Adam optimizer and MSE loss
    Train model on low-res and high-res image pairs
    return model
end function

function APPLY_SRCNN(image, model)
    Preprocess input image
    sr_image ← model.predict(image)
    Postprocess output image
    return sr_image
end function

function EVALUATE_SRCNN(model, test_data)
    for all image in test_data['lr_images'] do
        sr_image ← apply_srcnn(image, model)
        Calculate PSNR and SSIM between sr_image and corresponding hr_image
    end for
    return average PSNR, average SSIM
end function
```

Despite its simplicity, SRCNN demonstrated significant improvements over traditional SR methods. Here, I use it as the grounding point for comparison.

3.2 EDSR

For this project, I used the pre-trained model for Enhanced Deep Super-Resolution Network (EDSR) proposed by Lim et al. [4]. The reason why EDSR algorithm is performing better than other ones is because EDSR is to combine residual networks with expanded model size, this will result in a visible improvement in its performance.

3.2.1 What is Residual Scaling

It is a technique used to stabilize the neuro-network during deep learning. The input is combined with residual function applied on the input to generate output. The reason why we use this is that we want to learn a residual function which is to add a small correction to the input when it goes through the neuro net. But the problem is sometimes the residual value is too large, which affects the output, in this case it will affect the output image quality.

3.2.2 How is EDSR Implemented

Here we introduce how of EDSR is better than the CNNs or other similar algorithms:

- **Removing batch normalization layers:** The authors discovered that removing batch normalization will not only reduce memory usage but also will help to improve models performance [4]. Batch normalization can some time normalize the features too much, it might be useful in other tasks but not really useful in computer vision (CV) tasks as it will make the network less flexible according to the author. [6].

- **Using residual scaling:** To solve the issue of large feature values accumulating in deep networks, EDSR introduces a scaling factor within the residual blocks. This helps to stabilize the training process by preventing the residual value to be too big. [4].

- **Expanding model size:** EDSR can allocate the saved memory to increase the number of feature maps when we eliminate the batch normalization layer, this helps to boost the model's overall performance [4].

- **Single-scale architecture:** Unlike previous models that involve in using multi-scale (not multi layer) architectures, EDSR employs a single-scale model, which proves to be more efficient and effective [7].

The core of EDSR is its residual block, defined as:

$$F_{RB}(x) = x + 0.1 * (W_2 * \text{ReLU}(W_1 * x))[4] \tag{2}$$

where W_1 and W_2 are convolutional layers, and 0.1 is the residual scaling factor.

Each residual block in EDSR learns to add a small, refined correction to its input. These corrections accumulate through the network, steadily transforming the low-resolution input into a high-resolution output. The residual scaling factor is then multiplied with the residual value from input, this will ensure these "corrections" remain small and not dominating input, it helps to less instability during training by stop residual value from being too big.

In my experiments using the pre-trained EDSR model, it showed comprehensive improvements over previous methods, especially for large scale factors (3x and 4x). This made it a valuable component in my SR system and a good point of comparison with other models.

EDSR demonstrated comprehensive improvements over previous methods, especially for large scale factors (4x), making it a milestone in the development of super-resolution algorithms [5].

The residual block is implemented as pseudocode shown below:

Algorithm 3 EDSR Implementation

```
function EDSR_MODEL
    Define main network with residual blocks
    Remove batch normalization layers
    Add residual scaling (factor 0.1) to each residual block
    Add final upsampling module
    return model
end function

function RES_BLOCK(x)
    res ← Conv2D(x, filters=256, kernel_size=3)
    res ← ReLU(res)
    res ← Conv2D(res, filters=256, kernel_size=3)
    return x + res * 0.1
end function

function TRAIN_EDSR
    Load and preprocess training data
    model ← edsr_model()
    Compile model with Adam optimizer and L1 loss
    Train model on low-res and high-res image pairs
    return model
end function
```

Algorithm 4 EDSR Implementation (Part 2)

```
function APPLY_EDSR(image, model)
    Preprocess input image
    sr_image ← model.predict(image)
    Postprocess output image
    return sr_image
end function

function EVALUATE_EDSR(model, test_data)
    for all image in test_data['lr_images'] do
        sr_image ← apply_edsr(image, model)
        Calculate PSNR and SSIM between sr_image and corresponding hr_image
    end for
    return average PSNR, average SSIM
end function

model ← train_edsr()
test_data ← load_test_data()
psnr, ssim ← evaluate_edsr(model, test_data)
Print PSNR and SSIM scores
```

3.3 ESRGAN

3.3.1 What is A GAN?

Generative Adversarial network(GAN), which is different from SRCNN, is an unsupervised learning algorithm that aims to generate unseen examples to mimic the training datasets. GANs consist of a pair of neural networks: Generator and Discriminator. In our SR context, generator encodes a deep CNN, takes low resolution images, approximate the true distribution of training datasets, and transform them into high resolution images. Discriminator is then a neural-network responsible for telling the differences between real images and generated ones.

3.3.2 Real-ESRGAN

The model we picked for this paradigm is Real-ESRGAN, it was inspired by ESRGAN developed by Wang et al. [5]. One of the improvement it has over other algorithms is to apply real world degradation to high resolution source images in order to generate the down-scales version. In this way, algorithm could learn how to generate more realistic, visually pleasing high resolution images. ESRGAN has several improvements to be listed:

- **Residual-in-Residual Dense Block (RRDB):** This is the fundamental building unit in ESRGAN. It combines the concept of residual learning and dense connections, which allows layers to not only receive input from previous layers, but also from earlier layers, which helps to create a powerful block that helps to create high resolution images [5].
- **Removal of all batch normalization layers:** Similar to EDSR, ESRGAN removes batch normalization layers to increase the network's flexibility and reduce computational complexity [5].
- **Use of Relativistic average GAN (RaGAN):** Instead of having the discriminator predict the absolute probability of an image being real, it predicts the probability of an image being more realistic compared to a fake image. This approach leads to more stable training and improved visual quality [8].
- **Improvement of the perceptual loss:** Different from traditional loss function that use metrics such as SSIM and PSNR to calculate how good the up-scaling is, in ESRGAN perceptual loss was calculated using features before activation in the VGG network, providing stronger supervision and resulting in better perceptual quality [7].
- **High-Order Degradation Modeling:** Real-ESRGAN applies multiple degradation processes such as blur, resizing (bi-cubic and bi-linear), noise addition (with gaussian noise,

poisson noise, color and gray noise), and JPEG compression in high-order to simulate real-world complex degradations [9].

The RRDB block in ESRGAN is defined as:

$$F_{RRDB}(x) = x + \beta * (D_3(D_2(D_1(x)))) \tag{3}$$

where D_i are dense blocks and β is a residual scaling factor.

In simpler terms, Real-ESRGAN works as follows:

1. The generator network takes a low-resolution image and enhances it through a series of RRDB blocks.
2. Each RRDB block allows for complex feature extraction and transformation, which allows it to add finer details to the image.
3. The discriminator network, designed as a U-Net with spectral normalization, tries to distinguish between the generated high-resolution images and real high-resolution images.
4. The generator is trained to produce images that are realistic enough to bypass the discriminator, leading to more realistic and higher quality outputs.

Real-ESRGAN's use of adversarial training allows it to generate images that are perceptually more realistic, often outperforming previous methods in terms of visual quality, even if not always in terms of pixel-wise accuracy metrics like PSNR.

The core Real-ESRGAN implementation is as follows:

Algorithm 5 Real-ESRGAN high level Implementation

```
function HIGH_ORDER_DEGRADATION(image, order=2)
    for i = 1 to order do
        image ← apply_blur(image)
        image ← resize(image)
        image ← add_noise(image)
        image ← jpeg_compression(image)
    end for
    image ← apply_sinc_filter(image)
    return image
end function

function TRAIN_REAL_ESRGAN
    Load training data
    model ← real_esrgan_model()
    for each iteration do
        hr_image ← random_crop(training_image)
        lr_image ← high_order_degradation(hr_image)
        sr_image ← model.generator(lr_image)
        Update generator with L1, perceptual, and GAN losses
        Update discriminator
    end for
    return model
end function

function REAL_ESRGAN_MODEL
    Define generator network (same as ESRGAN)
    Define U-Net discriminator with spectral normalization
    return model
end function
```

Algorithm 6 Real-ESRGAN high level Implementation (part 2)

```
function APPLY_REAL_ESRGAN(image, model)
    sr_image ← model.generator(image)
    return sr_image
end function

model ← train_real_esrgan()
test_images ← load_test_images()
for each image in test_images do
    sr_image ← apply_real_esrgan(image, model)
    Save or display sr_image
end for
```

4 Experiments

My experimental setup involves:

1. Training SRCNN from scratch on our dataset.
2. Fine-tuning pre-trained EDSR and ESRGAN models with our datasets.
3. Using the Adam optimizer with a learning rate of 1e-4, set to be halved every 200,000 iterations.
4. Training each model for a fixed number of epochs or when convergence criteria is met.
5. Evaluating the models on separate test sets for anime-style images and cartoon style images.

4.1 Training Details

My training process for this project could be described in the following steps:

1. First start with high-resolution images from the dataset.
2. These images are down-scaled using bi-cubic interpolation to create low-resolution versions.
3. Several models (SRCNN, EDSR, and ESRGAN) are then trained to map these bi-cubic down-scaled images to the original high-resolution images.
4. During training, I used patches of size 128x128 as the bi-cubic down-scaled images and corresponding 512x512 patches from the original high-resolution images (for 4x upscaling).
5. SRCNN was trained without using pre-trained models, while EDSR and ESRGAN was fine tuning based on pre-trained models with data used for SRCNN.

4.2 Testing Details

The testing process could be described in the following steps:

1. Used the down scaled images with biscuits as mentioned above.
2. Upscale the images using three different algorithms and compare the result with source images.

This approach allows me to simulate a real-world scenario where I start with a low resolution images and try to recover its high resolution ones. By using SRCNN as a baseline method, I can now compare the performance of all three models: SRCNN, EDSR, and ESRGAN. The comparison will tell me how much improvement each model improved upon the baseline, which helps us to visually understand the advantages of each model.

5 Evaluation

I evaluated the models using two primary metrics:

- Peak Signal-to-Noise Ratio (PSNR)
- Structural Similarity Index (SSIM)

PSNR measures the ratio between the maximum possible power of a signal and the power of corrupting noise [10]. It is defined as:

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{4}$$

where MAX_I is the maximum possible pixel value of the image and MSE is the mean squared error between the images.

SSIM is used for measuring the similarity between two images [11]. It considers changes in structural information, luminance, and contrast:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5}$$

where μ_x, μ_y are the average of x and y, σ_x^2, σ_y^2 are the variance of x and y, σ_{xy} is the covariance of x and y, and c_1, c_2 are variables to stabilize the division.

5.1 Quantitative Results

The experiments show the following results:

5.1.1 Performance on Cartoon Dataset

Model	Avg PSNR	Avg SSIM
SRCNN	25.40	0.7303
EDSR	24.63	0.7221
Real-ESRGAN	26.29	0.8097

Table 1: Performance comparison on Cartoon Dataset

The results in Table 1 show that Real-ESRGAN outperforms both SRCNN and EDSR in terms of both PSNR and SSIM on the cartoon dataset. This indicates that Real-ESRGAN produces higher-quality super-resolution images with better structural similarity to the original high-resolution images.

Figure 1: Comparison of super-resolution results on cartoon image 1

Figure 1 illustrates the visual differences between the models. Real-ESRGAN produces the sharpest details, while SRCNN and EDSR show more blur in the finer structures.

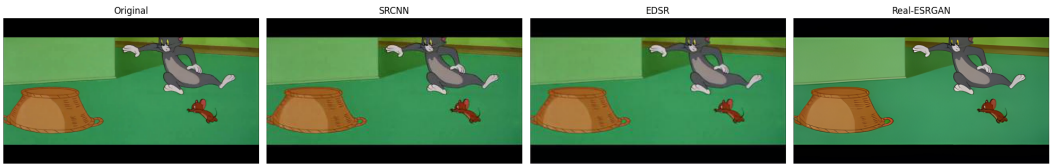

Figure 2: Comparison of super-resolution results on cartoon image 2

Similarly, Figure 2 shows that Real-ESRGAN maintains the visual integrity of the cartoon image better than the other two models.

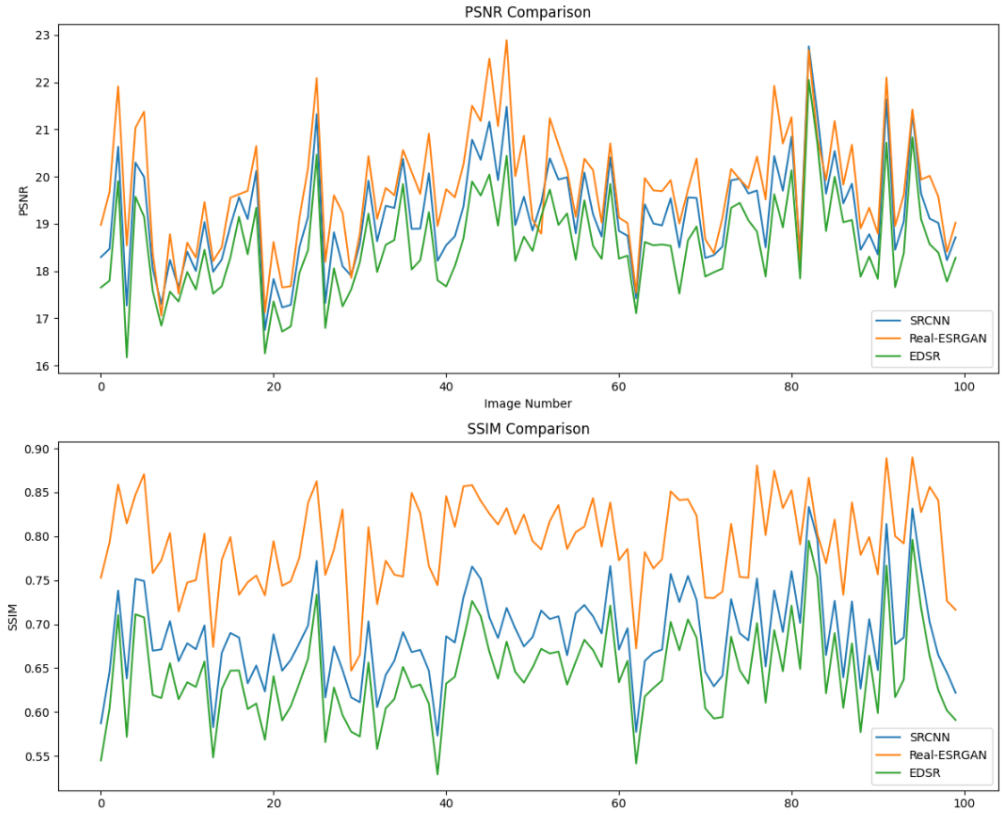

Figure 3: Comparison of 100 images performance on cartoons

Figure 3 provides a summary of the final results, clearly showing the superior performance of Real-ESRGAN in maintaining high visual quality shown as a consistently high SSIM value for ESRGAN model.

5.1.2 Performance on Anime Dataset

Model	Avg PSNR	Avg SSIM
SRCNN	19.19	0.6872
EDSR	18.55	0.6439
Real-ESRGAN	19.73	0.7933

Table 2: Performance comparison on Anime Dataset

As shown in Table 2, Real-ESRGAN again outperforms the other models, achieving higher PSNR and SSIM values on the anime dataset. This suggests that Real-ESRGAN is particularly effective at handling the unique characteristics of anime images.

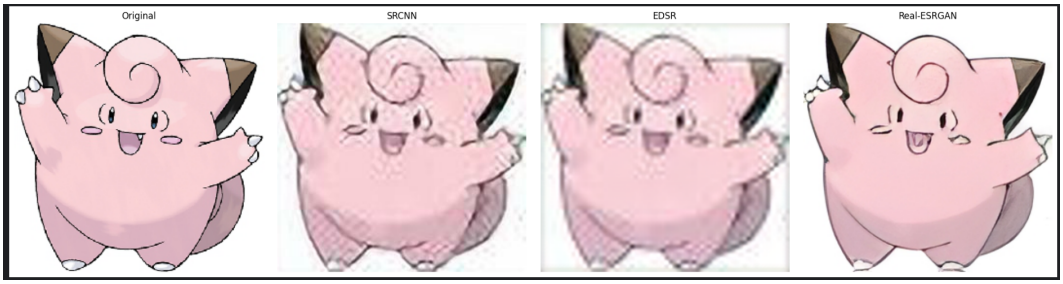

Figure 4: Comparison of SR results on anime image

Figure 4 highlights how Real-ESRGAN preserves the intricate details in the eyes and hair of the anime characters, compared to the blurrier results from SRCNN and EDSR.

Figure 5: Comparison of super-resolution results on second anime image

In Figure 5, Real-ESRGAN once again shows superior performance in maintaining the sharpness and clarity of the original image details.

Figure 6: Comparison of super-resolution results on third anime

In Figure 6, Real-ESRGAN once again shows best performance in maintaining the sharpness and clarity of the original image details.

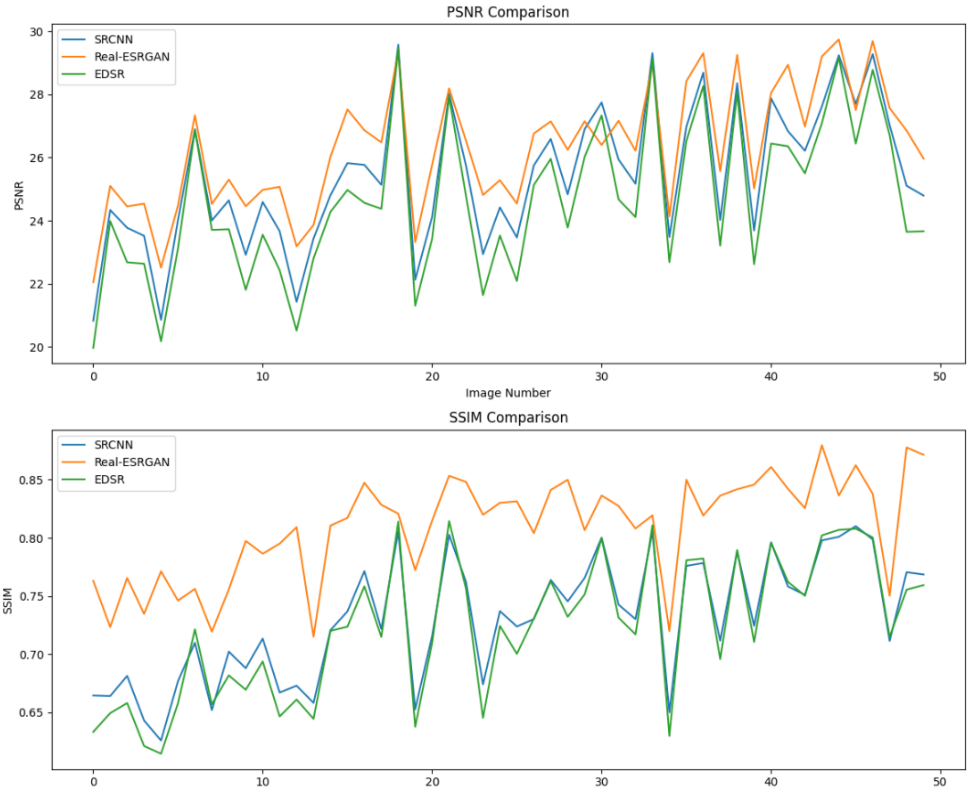

Figure 7: Summary of performance on anime dataset

Figure 7 summarizes the overall performance on the anime dataset, confirming the effectiveness of Real-ESRGAN in producing high-quality super-resolution images.

In terms of visual effect, ESRGAN outperforms SRCNN and EDSR in all images, this better performance is due to the advanced architecture of ESRGAN, specifically the use of Residual-in-Residual Dense Blocks (RRDB) and adversarial training, which allows for better feature extraction and better detail preservation. In addition, the training data also plays an important role in model performance, with more targeted training data, we can fine tune model's performance on certain area. But eventually with more training data, the difference between models will be diminishing, but the architecture advantage ESRGAN has still will make it the best performing model of all three.

6 Conclusion

The comparison of different models from supervised and unsupervised learning: SRCNN, EDSR, and Real-ESRGAN for image super-resolution reveals several critical findings:

- Real-ESRGAN consistently outperforms other models in both quantitative metrics and qualitative visual assessment, especially for anime-style and cartoon images.
- EDSR shows strong performance on general images but may require specific fine-tuning for specialized content like anime-style images [4].
- SRCNN, despite its simplicity in architecture, remains competitive, and it performed relatively well in terms of quantitative result: PSNR on anime-style images.

These results highlight the rapid progress in deep learning-based super-resolution techniques, with GAN-based models like Real-ESRGAN showing particular promise in generating high-quality, visually pleasing results for stylized content [9].

Future work could explore:

- Fine-tuning EDSR and Real-ESRGAN specifically for anime-style and cartoon images.
- Investigating the trade-off between model complexity and performance, particularly for stylized content.
- Exploring the potential of combining multiple models or techniques for even better results.
- Extending the study to include a wider variety of stylized content and real-world, low-quality images to test the models' generalization capabilities.
- Investigating the impact of different loss functions and training strategies on model performance for stylized images.

This study contributes to the ongoing research in image super resolution, providing insights into the performance of different deep learning and unsupervised learning approaches, particularly in the context of stylized content like anime and cartoon images. The findings suggest that while more complex models like Real-ESRGAN generally perform better, there is still room for improvement and specialization when dealing with images of specific domains. For the future work, we could continue to compare more advanced algorithms, and build a system that leverages the advantages of each model to generate the most visually and statistically pleasing images based on their performance.

References

[1] Jing Tian and Kai-Kuang Ma. A survey on super-resolution imaging. *Signal, Image and Video Processing*, 5:329–342, 2011.

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

[3] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 184–199, Cham, 2014. Springer International Publishing.

[4] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution, 2017.

[5] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Change Loy, Yu Qiao, and Xiaoou Tang. Esrgan: Enhanced super-resolution generative adversarial networks, 2018.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[7] 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.

[8] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan, 2018.

[9] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Real-esrgan: Training real-world blind super-resolution with pure synthetic data, 2021.

[10] Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[11] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.